Synthesizing Patterned Surfaces for 3D Printing

Weikai Chen

Department of Computer Science
The University of Hong Kong

A thesis submitted for the degree of
Doctor of Philosophy

August 2018

Dedicated to my parents and my wife.

Declaration

I declare that this thesis represents my own work, except where due acknowledgement is made, and that it has not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

Signed: _____

Date: _____

Acknowledgements

And I would like to acknowledge ...

Abstract

Recent years have witnessed the advancement of 3D printing in fabricating objects with sophisticated and highly-customized geometries. The print services are now widely available through online orders, home printers and local FabLabs. Nevertheless, it remains difficult for most users to create interesting objects, even more so when the intended design has complex geometry details. To circumvent this issue, in this thesis we present approaches to automate the task of designing and fabricating artistic patterned surfaces.

We firstly present a novel approach to synthesize fabricable filigrees over target surfaces. As thin patterns widely found in jewelry, ornaments and lace fabrics, filigrees are often manually designed by composing repeated base elements. Our technique aims to automate this challenging task. Our technique covers a target surface with a set of input base elements, forming a filigree strong enough to be fabricated. We leverage the fact that as traceries, filigrees can be well captured by their skeletons. This affords for novel energy function that measures the matching quality between base elements. In addition, instead of seeking for a perfect packing of base elements, we relax the problem by allowing appearance-preserving partial overlaps. The formulation is optimized by a stochastic search, which is further improved by a boosting step that records and reuses good configurations discovered during process. Our technique affords for multi-class synthesis and several user controls, such as scale and orientation of the elements.

Second, we extend the method to generate complex – yet easy to print – tile decorations. The user only provides base surface and a set of tiles. Our algorithm automatically decorates the base surface with the tiles. However, rather than being simple decals, the tiles *become* the final object, producing shell-like surfaces that can be used as ornaments, covers, shades and even handbags. Our technique is designed to maximize print efficiency: the results are printed as independent flat patches that are articulated sets of tiles. The patches could be assembled into the final surface through the use of snap-fit connectors. Our approach proceeds in three steps. First, a dedicated packing algorithm is proposed to compute a tile layout while taking into

account fabrication constraints, in particular ensuring hinges can be inserted between neighboring tiles. A second step extracts the patches to be printed and folded, while the third step optimizes the location of snap-fit connectors. Our technique works on a variety of objects, from simple decorative spheres to moderately complex shapes.

[401 words]

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 3D Printing

3D printing is a manufacturing process where physical objects are fabricated by forming successive layers of materials one upon another (Figure 1.1) under the control of computer program. It was firstly developed in early 1980s and commonly known as *rapid prototyping* or *additive manufacturing* in the commercial industry.

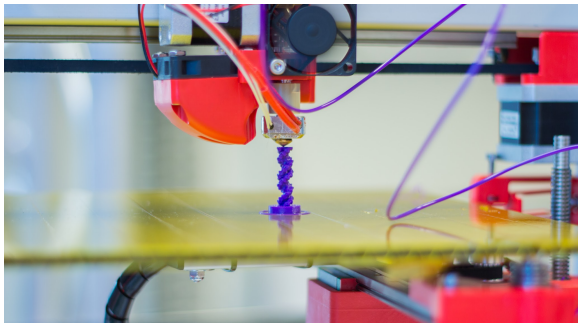


Figure 1.1 3D Printing. Object is printed by solidifying successive layered materials.

A typical 3D printing process starts from a digital model, which could be either created by computer-aided design (CAD) softwares or acquired from 3D scanners. Before fabrication, the input model will be examined for geometric errors, e.g. self-intersections, holes, flipped face normals or manifold errors. If passed, the obtained model will be processed by *slicer*, a software designed to turn a 3D model into a series of thin layers that would be sequently substantialized by 3D printer. The slicer also generates a *G-code* file that contains operational introductions tailored for a specific type of 3D printer.

The type of a 3D printer is mainly defined by the technology that it uses to deposit layers. One major type of printing methods seeks to melt or soften the material so as to extrude layers, for instance, fused deposition modeling (FDM), selective laser sintering (SLS), selective laser melting (SLM), electronic beam melting (EBM) or fused filament fabrication (FFF). Some methods add layers via solidifying liquid materials, e.g. stereoradiography (SLA) and digital light processing (DLP) while the others cut thin layers to desired shape and join them together, e.g. laminated object manufacturing (LOM). An exhaustive investigation of all printing processes is out of the scope of this thesis. We therefore only briefly introduce 3 popular types of printing technologies: FDM, SLS and SLA.

FDM is the most widely-used printing method in desktop 3D printers. Thermoplastic filament is heated and extruded throughout nozzle onto the building plate. The dimensions of an object is translated by computer into X, Y and Z coordinates which are used to calculate the moving paths of nozzle and platform. The object is essentially built from the bottom up. Therefore, if the target object contains overhanging parts, supporting structures are required during printing and need to be removed when printing is completed. FDM printers is the most cost-effective means for home-DIY'ers, education sectors and small business to prototype designs and develop products. Figure 1.1 demonstrates an FDM printer printing in progress.

SLS printers build object from powdered material in the build area. Layers of granules are selectively sintered by laser and then bound together to form solid structures. The object is left to cool in the building volume when it is fully constructed. Cleaning of attached powders is required after the object is removed from machine. As the object is printed constantly surrounded by unsintered powder, SLS doesn't require any support structures during printing. SLS has been widely used in rapid prototyping in commercial industries. However, as SLS printer requires the use of expensive high-powered lasers, it is a bit beyond the reach of average consumers.

SLA creates object by exposing photosensitive liquid resin to a laser-beam so that the resin hardens and solidfies in target shape. Similar to FDM, each new layer in SLA is built on top of preceding one. Hence, support structure is required when objects with overhanging parts are printed with SLA printer. SLA can produce smooth surface with extreme details and is thus popular in jewelry fabrication and cosmetic dentistry for building castable molds.

The advent of 3D printing opens a new era of manufacturing. It enables designers to create and fabricate complicated parts and shapes, many of which are hard to be produced (left one in Fig. 1.2) by conventional manufacturing technologies. In addition,

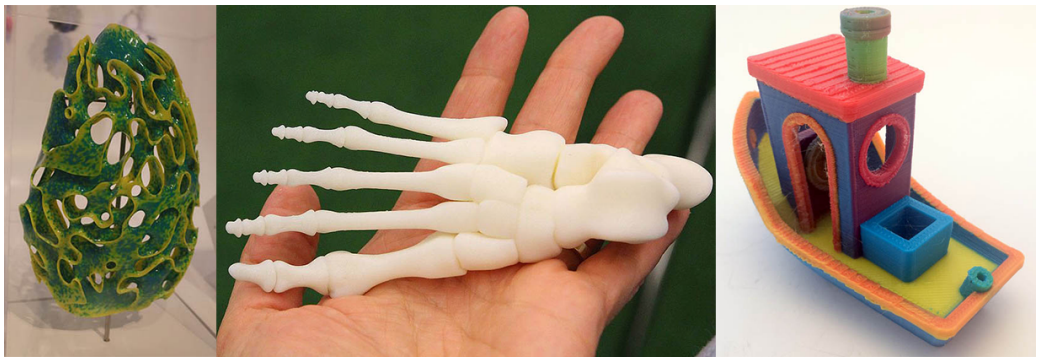


Figure 1.2 Objects fabricated by 3D printing. *Images in courtesy of WikiMedia.*

3D printing allows for great flexibility in customizing the fabrication. Changes of design are performed digitally without additional tooling or other manufacturing process required for the final product. Each item can be easily customized according to user's needs in terms of both shapes and colors (middle and right figures of Fig. 1.2). The flexibility offered by 3D printing has significantly accelerated the speed of prototyping and soon become the prior option for designers and entrepreneurs who seek to test production runs. Common users could also have easy access to 3D printing, through online print services, home printers and local Fablabs.

Despite the rapid advances in 3D printing technology, it remains difficult for most users to design interesting objects, especially when the intended design contains complex geometries. There is a significant research effort dedicated to automating the customization of shapes. For instance, researchers have developed tools on balancing shapes [35], creating spinbale objects [2], designing wind instruments [24, 46], lampshades [59] and helping users maintain fabricability during modifications [41]. These research works have greatly broaden the applications of 3D printing and made it more accessible to common users.

1.2 Surface Patterning

Thanks to 3D printing, sophisticated geometries can be printed with ease, paving the way to customize personalized designs. Among the needs of customization, designing decorative patterns over surface has been keenly pursued by designers, architects and even home-users. Patterned surface has been carefully designed and 3D printed as curtain wall of buildings, lampshades and object covers (Figure 1.3). The intriguing details turn a bland base surface into a unique piece of artwork. However, customizing 3D objects is still a difficult task for non-expert users, for all but very

simple modifications. Several modeling software, e.g. AutoCAD, Autodesk Maya, 3DS Max, MeshMixer, Rhinoceros *et al.*, have been developed to assist the job of creating and modifying extended designs. Professional skills are required in order to exploit the functions of the above tools. However, even for an expert in modeling, it would need hours or even days to complete a design as complex as those in Figure 1.3.

Figure 1.3 Surface customization is highly demanded in artistic design. *Images in courtesy of Pinterest.*

Methods to automate the design of fabricable patterned surface are rarely explored in both industry and academia. Recenlty, Dumas et al. [10] proposed a method to synthesize a fabricable pattern along a surface. It starts from a stochastic pattern defining solid/void regions. The approach modifies textures synthesis to account for structure and rigidity and synthesizes printable 3D patterns with similar appearance with exemplar. Martinez et al. [28] investigated automatic 2D shape design under rigidity and appearance objectives. The user inputs a pattern example. Their method generates a rigid shape with a specified quantity of materials while allowing the appearance of output structure to resemble the input pattern.

The related work mentioned above pioneers in producing intricate surface with user-specified exemplar. However, while these techniques excel at generating stochastic, organic patterns, they cannot properly capture relatively large individual elements. In addition, they only accept one kind of input exemplar, limiting the design space that user can explore. In this thesis, we will present works that aim to synthesize a wider variety of elements and provide methodology to fabricate sophisticated geometry efficiently with low-end filament 3D printers.

1.3 Contribution

The contributions of this thesis are summarized as below:

- We present a novel approach to automate synthesizing filigree elements over target surface with optimized structural rigidity. We exploit two properties

of filigrees to make it possible. First, as filigrees form delicate traceries, they are well captured by their skeleton. This affords for a simpler definition of operators such as matching and deformation. Second, instead of seeking for a perfect packing of base elements, we relax the problem by allowing appearance preserving partial overlaps. We optimize a filigree by a stochastic search, further improved by a novel algorithm that records and reuses good configurations discovered during process. Our technique affords for multi-class synthesis and several user controls, such as scale and orientation of the elements. This work has been published in SIGGRAPH'16. Please refer to http://i.cs.hku.hk/~wkchen/projects/proj_sig16.html for more details.

- We propose a method to generate tile decoration on surface that could be fabricated by low-end filament printers. The user only provides base surface and a set of tiles. Our algorithm automatically decorates the base surface with the tiles. However, rather than being simple decals. the tiles *become* the final object, producing shell-like surfaces that can be used as ornaments, covers, shades and even handbags. Our technique is designed to maximize print efficiency: the results are printed as independent flat patches that are articulated sets of tiles. The patches could be assembled into the final surface through the use of snap-fit connectors. A dedicated packing algorithm is proposed to compute a tile layout while taking into account fabrication constraints, in particular ensuring hinges can be inserted between neighboring tiles. A patch generation step extracts the patches to be printed and folded, which is followed by a third step that optimizes the location of snap-fit connectors. Our technique works on a variety of objects, from simple decorative spheres to moderately complex shapes.

1.4 Organization

We organize the remainder of this thesis as follows: Chapter 2 introduces the algorithm of synthesizing filigree patterns over target surface while the details of generating fabrication tile decorations are stated in Chapter 3. In particular, we firstly discuss the background and previous work in each chapter and then introduce the method, results and implementation details. Finaly, we conclude the thesis and future work in Chapter 4.

Chapter 2

Synthesis of Filigrees for Digital Fabrication

2.1 Introduction

Filigrees are fascinating ornamental patterns, forming delicate and intricate traceries in space. Their unique aesthetics emerge from the repetition of similar elements arranged in larger patterns, suggesting shapes and volumes. Filigrees often appear as jewels made of thin gold or silver wires, bended and soldered together; but they also appear as laces and finely engraved drawings on glass panels and metal plates. Due to their intricate and delicate nature, fabricating filigrees is an art reserved to few artists mastering highly specialized crafting skills.

The advent of digital fabrication technologies such as 3D printing and laser cutting holds the promise to make these traditional art forms more accessible, and to apply them in contexts that would not be achievable by traditional means. Three such examples are the sculptures *Crania Anatomica Filigree* by Joshua Harker [12], the magnificent Seashell dress by Fashion House SHIGO [6], and the concrete filigree enclosing the MuCEM museum [36].

While digital fabrication simplifies the physical realization of filigrees, a digital model is required before fabrication. In this paper we aim at providing algorithms that can assist the process of modeling filigrees. In particular we focus on the most time consuming task, which is to cover a target surface with a large number of basic elements, while enforcing connectivity and fabrication constraints.

At a high level our approach replicates the traditional filigree design process: The user inputs a set of basic filigree elements that are then automatically repeated, distributed and assembled into a larger pattern covering the target surface. Our

algorithm strives to preserve the appearance of each individual element, joining them in natural ways. The produced filigrees are fully connected and optimized to minimize fragilities, and are ready for 3D printing. In addition our technique provides many user controls: multiple base elements can be specified; orientation and scale fields can be defined along the target surface.

At a technical level, we solve for an element packing problem along the target surface. Tightly packing elements of arbitrary shapes along surfaces is extremely challenging. To achieve this task we exploit two specific properties of our problem. First, filigrees are well described by their skeletons, allowing for simple manipulations such as detecting overlaps, performing local deformations or pruning spurious branches. Second, the repetitive nature of filigrees affords for additional degrees of freedom. Elements can be partially overlapped in inconspicuous ways, relaxing the packing problem. We formalize the quality of pattern overlaps through a novel partial Pattern Matching Energy (PME) based on the modified Hausdorff distance. We exploit this energy in a stochastic optimization scheme. Starting from a dense packing of many elements — guaranteeing full connectivity but having many overlaps — we progressively refine the result. Refinements are performed through local optimization of element positions, accepting overlaps having a low matching energy. Through a limited amount of deformation our technique preserves the global connectivity of the pattern while resolving overlaps. A key component of our technique is a novel approach for recording and reusing good configuration between pairs of elements. We call this approach *boosting* as it progressively encourages good matches and good overlaps to appear. Structural properties are jointly optimized by encouraging additional connections to appear in fragile areas.

2.2 Previous Work

Texture synthesis The goal of by-example texture synthesis is to reproduce a colored pattern resembling a small exemplar given as input. This is typically done for texture images, generating larger extents of pattern while avoiding obvious periodicities [53]. Several algorithms are able to synthesize a texture from an example along a surface, e.g. producing per-vertex colors along a dense mesh [45, 54], directly in texture maps [57, 22], or by covering the model with patches [34]. Mesh Quilting [60] extend these techniques to geometry: the input texture is a patch of geometric *texture* that is used to cover a target surface. It is expected that the left/right top/bottom boundaries contain similar, repeating content. The surface is covered by placing each

patch in sequence, cutting its content to best match previously placed patches. The approach is not designed to work for individual elements, and without repeating features it cannot find good cuts to resolve seams. It therefore does not apply to the context of our approach. Zhou et al. [62] consider the synthesis of patterns along curves while constraining the topology of the result. This allows the fabrication of singly connected ornaments along curves, but does not extend to higher dimensions (2D/3D).

Closer to our purpose, the work of Dumas et al. [10] synthesizes a fabricable pattern along a surface. The input is a stochastic pattern defining solid/empty regions. The approach takes into account structural properties and fabrication constraints. Martinez et al. [28] investigate automatic shape design under rigidity and appearance objectives. The appearance is defined by an input exemplar pattern. While these techniques excel at producing stochastic, organic patterns, they cannot properly capture relatively large individual elements. This stems from the Markov Random Field assumption that limits appearance to a local definition. Li et al. [25] use field-guided shape grammars to synthesize geometric patterns. The grammar rules are manually designed from input patterns. However, this approach would generate artifacts when patterns are densely placed, limiting its application for digital manufacturing.

Several approaches have been proposed that focus specifically on the synthesis of element arrangements, e.g. [27, 14]. The input captures both a set of disjoint elements and their spatial relationships. A similar distribution of non-overlapping elements is synthesized. In contrast our work inputs only a set of *independent* elements — there is no target spatial arrangement specified in the input for our algorithm to mimic — and our technique exploits potentially large overlaps between elements for generating fully connected filigree patterns. In addition, as we target fabrication the connectivity between patterns is crucial to ensure the rigidity of the final printouts.

In concurrent work Zehnder et al. [55] synthesize filigrees by packing curved elements along surfaces. The approach provides an interactive authoring tool that can automatically generate an initial packing. The curves elastically deform, and their positions are optimized to reduce deformations while enforcing contact and sizing constraints. The system reveals weak areas that the user can reinforce by interactive editing. The elements are not allowed to overlap and may be large compared to the surface curvature. In contrast we focus on fully automatic synthesis with large numbers of curve elements, exploiting overlaps and reinforcing the structure automatically.

Both approaches are complementary and the deformation analysis and optimization in Zehnder et al. [55] would benefit our work.

Structural analysis for fabrication Our technique considers the structural properties of the final object. Several techniques have been proposed to help user identify and fix weaknesses of an input object. Stava et al. [43] automatically add struts to an object after identifying weakness by the finite element method. Zhou et al. [61] perform a worst case analysis to identify weak regions of a 3D print without prior-knowledge of external forces. Umetani and Schmidt [47] perform a fast, interactive cross sectional analysis to present the user with a weakness map. Our work performs a structural analysis that is specifically tailored to our needs, simulating the external surface with shell elements. Instead of adding visible struts, we locally change the thickness of the filigree and add more elements to locally reinforce the filigree.

2.3 Filigree Synthesis

2.3.1 Terminology

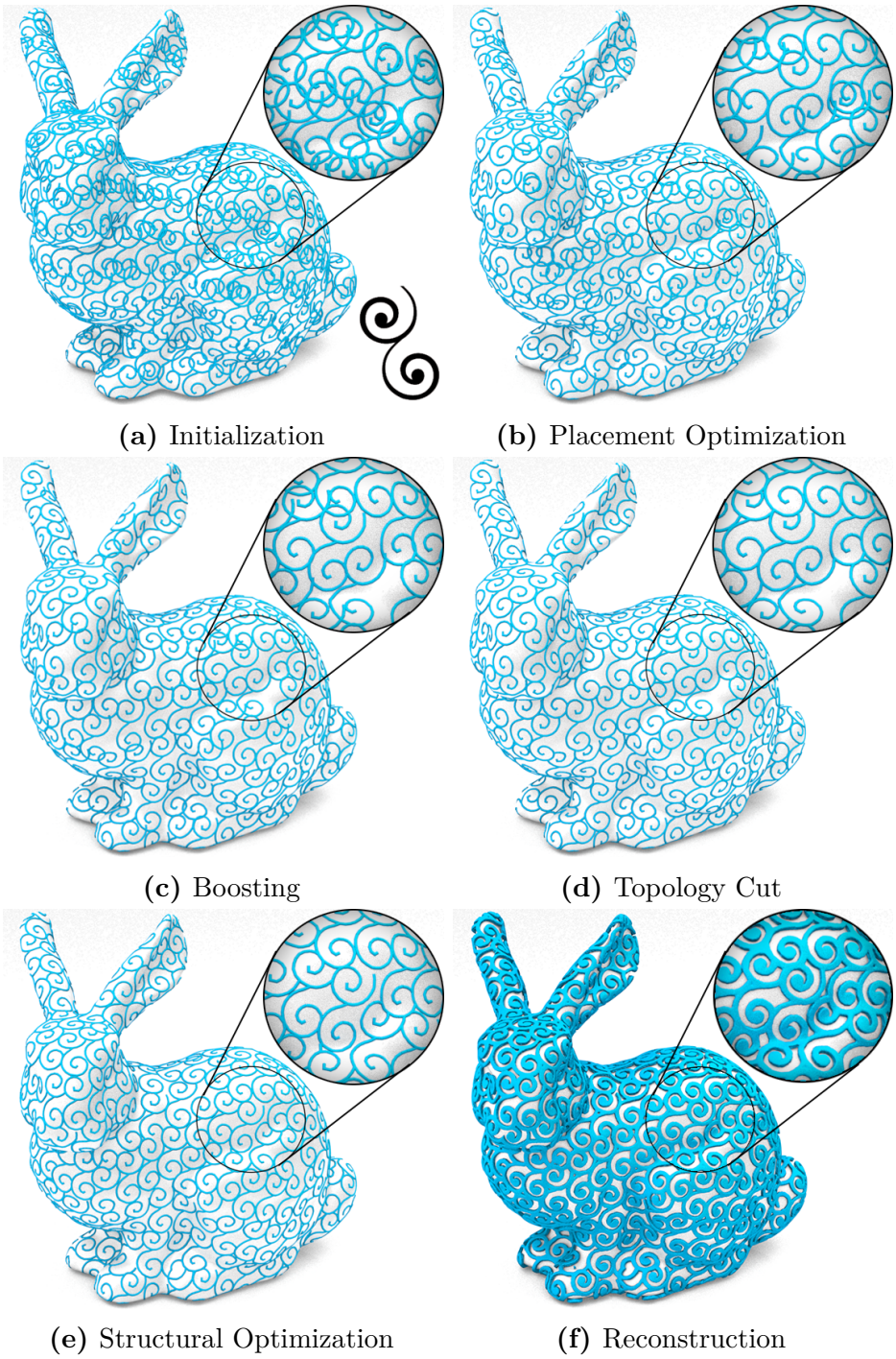
First we fix the terminology. A *filigree pattern* refers to a design layout of curvilinear strips, often called *traceries*. A basic filigree pattern used to compose a large pattern is called a *filigree pattern element*, or *element* for short. An element often again consists of individual curvilinear strips, which will be called *branches*.

2.3.2 Input and Output

The input to our synthesis algorithm consists of some 2D exemplar filigree elements and a base surface serving as the output domain for filigree synthesis. The output is a visually pleasing filigree layout over the base surface which is composed of well-connected and partially overlapping duplicates of the exemplar element, thus making the synthesized pattern have a similar style to the input exemplar elements. The base surface may be equipped with a user-specified control field which dictates how the composing elements should be oriented and scaled over the base surface.

2.3.3 Medial Axis Representation

A typical filigree pattern consists of connected curvilinear strips, as shown in Figure 2.2a. For any filigree pattern, because of the curvilinear nature of its constituent

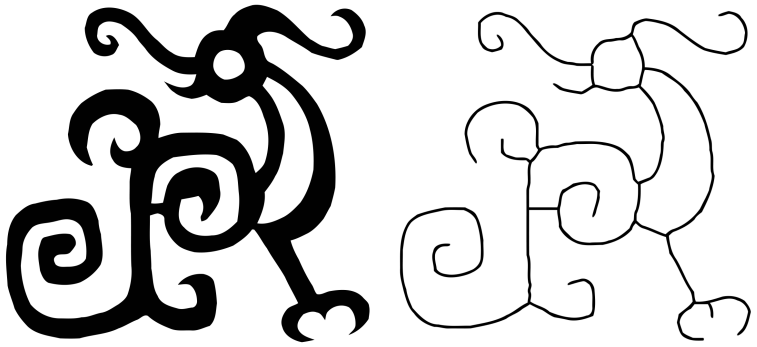


(a) Initialization **(b)** Placement Optimization

(c) Boosting **(d)** Topology Cut

(e) Structural Optimization **(f)** Reconstruction

Figure 2.1 Filigree synthesis pipeline. We first (a) generate an initial element distribution over the base model (the input element is shown on the right bottom); (b) the placement of element is first optimized using stochastic search with connectivity constraints; then followed by (c) a boosting step to improve overall element distribution. (d) Topology cut is applied to trim off conflicting branches. (e) Structural optimization strengthens the weak regions via adding new elements. (f) A final model is reconstructed that is ready for digital fabrication.



(a) (b)

Figure 2.2 (a) A typical filigree pattern consists of connected curvilinear strips. (b) The skeleton graph of the filigree element in (a).

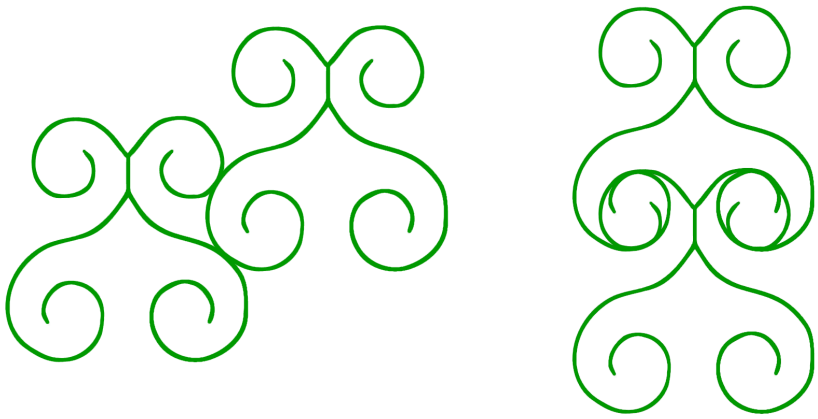
strips, we will use the medial axis of the pattern to represent its main skeletal structure, and call its medial axis the *skeleton graph* of the pattern. To simplify geometric processing tasks during filigree synthesis, we will mainly work with the skeleton graphs of filigree patterns and approximate the skeleton graphs by polygonal curves with user-specified accuracy, as shown in Figure 2.2b. Towards the end of our algorithm, we will obtain a large skeleton graph that can be viewed as the medial axis of the synthesized filigree design. By recovering width to convert this large skeleton graph into connected strips, we yield the final synthesized filigree design on the base surface (see Figure 3.12d).

2.3.4 Requirements

It is required that the synthesized filigree elements need to be connected into a single piece with sufficient mechanical strength. We encourage two types of connections to ensure preservation of the style of the exemplar filigree element: (1) *tangential connection*, which means smooth and natural contacts between the branches of adjacent elements (see Figure 2.3a); and (2) *partial overlapping with matching shapes*, in which case two adjacent elements overlap partially, and the geometrical shapes of the two elements match well within the overlapping region (see Figure 2.3b). The quantitative measurement of shape matching will be elaborated later.

2.3.5 Pipeline Overview

The pipeline of our synthesis algorithm has the following main steps, as shown in the flowchart in Figure 2.1.



(a) **(b)**

Figure 2.3 (a) Tangential connection. (b) Partial overlap.

Step 1: Initialization In the initialization step, the duplicates of the exemplar filigree element are distributed over the base surface using a blue-noise sampling method. This initial distribution takes into consideration the control field as well as the unbiased selection of input elements in the case where multiple basic filigree elements are provided. The resulting distribution of the filigree elements at this stage ensure sufficient coverage of the base surface and close connection between the elements, but are visually unsatisfactory since the pattern elements overlap in a random manner.

Step 2: Placement optimization In this step we locally adjust the positions and shapes of all the filigree elements to achieve visually more pleasing connections between elements, while preserving the surface coverage and inter-element connectivity. Two kinds of operations are performed in this stage to improve the visual quality of the connections between adjacent elements: (a) *Position adjustment*: Two elements that overlap partially will have their relative positions adjusted by small translational displacement to achieve a better matching of their shapes within the overlapping region. (b) *Forcing tangency*: If two elements are nearly in tangential contact between their curvilinear branches, then the elements are brought into tangent contact via non-rigid deformation.

Step 3: Boosting The local search method employed in the preceding step is easy to get stuck in a local minima. A simple but effective modification is to iteratively call the local search routine, but starting from a different initial configuration at each time. We, therefore, propose a boosting step to improve the overall pattern

distribution via learning good relative positions between element pairs. The candidate pairs are progressively recorded in a database when numerous position adjustments are attempted in Step 2 on placement optimization. Any pair of neighboring elements that cannot be put in a good connection will be replaced by a better-connected pair that is learnt from the database. The output element distribution will be fed back to the second step in an iterative manner.

Step 4: Topology cut While satisfactory connections can be achieved for most pairs of adjacent elements, some pairs of adjacent elements may still have unacceptable connections with each other despite the efforts on improving partial overlap in step 2 and 3, as shown as Figure 2.1d. This is typically manifested by some conflicting branches of the elements that intersect each other, causing undesired visual artifacts. To fix this, we apply an *topology cut* operation to trim off the conflicting branches with lower importance.

Step 5: Structure optimization The skeleton graph obtained after topology cut is first converted into surface mesh representation by adding the width to the curve segments of the graph. The filigree design obtained is connected and visually satisfactory, but may lack needed mechanical strength to endure normal or specified handling. We apply structural optimization to iteratively improve pattern coverage and enhance element connections until sufficient model rigidity is reached. Finally, the improved skeleton graph is converted to the 3D mesh representation that is ready for fabrication.

Discussion We wish to emphasize that we allow complete overlap between elements in the second and third step if our solver find such movement is helpful to improve the matching quality. We only keep one copy of the entirely overlapped elements when the iterations stop at each step. This strategy helps us to control the element number in an implicit way. Our method will end up with a proper number of elements regardless an initialization with excessive elements. Note that in the steps before structure optimization, all the filigree elements are represented merely by their skeleton graph.

2.4 Pattern Synthesis

In this section we will present the details of the core algorithms of filigree synthesis, that is the first four steps in the pipeline: (1) initialization (2) placement optimization

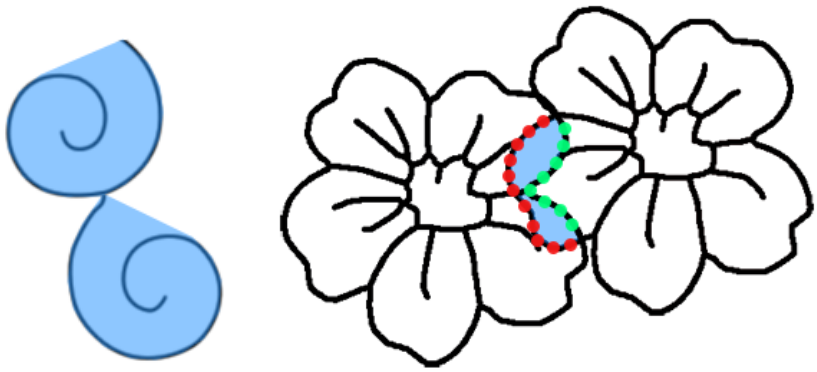
(3) boosting and (4) topology cut. As these four steps mainly determine the appearance of final output, we call them pattern synthesis in the rest context of this paper.

2.4.1 Initialization

The goal of initialization is to distribute the instances of the exemplar elements such that there is sufficient coverage of the base surface. Furthermore, it is required that each element is well connected with its neighboring elements. Our initialization method follows the dart-throwing approach in blue-noise sampling problem [52]. A fixed point of each exemplar filigree element is picked as its reference point. We repeatedly generate sample points on the base surface by dart throwing around the boundary of the existing elements. For each sample point generated, the instance is kept only if it overlaps with some existing elements and the overlapping area satisfies certain conditions to be elaborated below; otherwise it is rejected and another sample point is generated. This is repeated until the entire base surface is sufficiently covered and no more element can be added.

Specifically, let S_i denote the current instance of the element that newly generated. Then S_i is accepted if and only if all the following conditions are met: (1) The overlapping area of S_i with the union of all the existing elements is no more than 60% of the area of S_i; (2) The overlapping area of S_i with any of the existing elements is no more than 30% of the area of S_i; and (3) The overlapping area of S_i with at least one of the existing instances is no less than 10% of the area of S_i. These conditions ensure that the newly accepted element covers some previously uncovered regions of the base surface, while having sufficient overlap with the existing elements to ensure close connection between adjacent pieces. If the base surface has a control field, each generated element should be rotated and scaled as dictated by the control field before testing its overlap with the existing elements.

To ensure unbiased distribution of the exemplar filigree elements in the case that multiple exemplars are provided as input, we follow the strategy in [52] for multi-class blue noise sampling to pick the next trial pattern from the class that is currently most under-filled. Furthermore, a pattern element of a larger size is given a lower priority to be picked. Here, the size of an element pattern is defined to the diagonal length of the bounding box of the pattern.



(a) (b)

Figure 2.4 (a) Covering region of non-convex element. (b) Overlapping region (shown in blue) between two neighboring elements. The vertices of the polygons of the two elements (shown in red and green) are the input point sets to compute the modified Hausdorff distance.

2.4.2 Pattern Matching Energy

The randomly distributed pattern elements generated by initialization often have highly undesirable visual artifacts, although they are connected and cover the base surface well, as shown in Figure 2.1a. These artifacts are mainly due to that most adjacent patterns overlap in a random manner, without having their geometric features aligned with each other. We now propose some measures on the visual quality of the connection between adjacent elements.

First we discuss how to quantitatively measure the quality of alignment of two partially overlapping pattern elements. For a filigree pattern element, we need to define its *covering region*. If the element is convex or nearly convex, we simply take the convex hull of the pattern to be its covering region. Otherwise, we decompose the pattern element into several convex or nearly convex components and take the union of the convex hulls of these components to be the covering region of the original element (see Figure **??** as an example, the covering region is marked in blue). We developed a user interface to allow the user to easily perform the convex decomposition of a non-convex element.

For two partially overlapping pattern elements, we define the intersection of their covering regions to be their *overlapping region* (see Figure 2.4b). Then we measure alignment quality of these two elements by the Hausdorff distance between the subparts of the two pattern elements *within* their overlapping region. Since each filigree element is represented by its skeleton graph with edges being polygonal curves, the evaluation of this Hausdorff distance is reduced to the computation of

the Hausdorff distance between two finite sets of points which are the vertices of the polygons from the two skeleton graphs that lie in the overlapping region (see Figure 2.4b).

We adopt the *modified Hausdorff distance* [9], which has proven to be more effective for shape matching purpose than the standard Hausdorff distance. For two finite point sets U and V, their modified Hausdorff distance is defined to be

$$\text{dist}_{MH}(U,V) = \max(\frac{1}{N_U}\sum_{u\in U}\text{dist}(u,V), \frac{1}{N_V}\sum_{v\in V}\text{dist}(v,U)) \tag{2.1}$$

where N_U and N_V are the number of points of U and V, respectively. $\text{dist}(u,V)$ is the closest distance from the point u to the point set V, i.e. $\text{dist}(u,V) = \min_{v\in V} d(u,v)$, where $d(u,v)$ is the Euclidean distance between the points u and v. The term $\text{dist}(v,U)$ is similarly defined.

For two overlapping elements P_i and P_j, let U_i' and V_i' denote the set of the vertices of the skeleton graphs of P_i and P_j, respectively, that lie in the overlapping region of P_i and P_j. Then the matching quality of the patterns P_i and P_j is defined to be

$$\text{dist}(P_i,P_j) = \text{dist}_{MH}(U_i', V_i') \tag{2.2}$$

Intuitively, this term measures how well the shapes of the two filigree elements P_i and P_j match each other within their overlapping region. Note that if P_i and P_j do not overlap, that is U_i' and V_i' are both empty set, we penalize $\text{dist}(P_i,P_j)$ to be infinity. This setting helps to enforce pattern connections during minimizing the global matching energy that defined below.

Now we are ready to define a global energy function, called *Pattern Matching Energy* (PME), to measure the overall quality of the synthesized filigree pattern. Let P_i denote a pattern element. Let Γ be the set of the index pairs (i,j) such that the elements P_i and P_j are connected to each other. Γ can also be viewed as the edge set of the connectivity graph of all the pattern elements. Let $\mathcal{P} = \{P_i\}$ denote the set of existing patterns. Then the PME function is defined to be

$$E(\mathcal{P},\mathcal{O}) = \sum_{(i,j)\in\Gamma}\text{dist}(P_i,P_j) + \Theta(\mathcal{P},\mathcal{O}) \tag{2.3}$$

where the first term measures the overall appearance by considering the alignment quality of every pair of connected pattern elements. Θ is an optional application-specific energy term, which can be cast into connectivity constraints among $\mathcal{P}$ or field

constraint to follow the requirements of control field over the output domain $\mathcal{O}$. Our goal is to find an element distribution $\mathcal{P}$ with lowest PME value.

2.4.3 Placement Optimization

The goal of this step is to improve the initial distribution of pattern element $\mathcal{P}$ by minimizing the PME energy. However, it is non-trivial to perform a meaningful gradient descent to minimize PME in Equation 2.3 due to the non-linearity of Hausdorff distance, not mentioning that arbitrary pattern shapes could be taken as input. Therefore, we resort on a greedy strategy to iteratively refine the placement of each element. In order to maintain connectivity between pattern elements, in this step, we impose hard constraints to maintain the overlapping relationship between element pairs. That is, the initial neighbors $\mathcal{N}_i$ of element P_i should at least be the subset of its new neighbors $\mathcal{N}_i'$ after placement optimization.

There are two phases of placement optimization as shown in Algorithm 1.

2.4.3.1 Stochastic Search via Translation

Stochastic search is implemented in StochasticSearch. In this phase, each element undergoes some small translational displacement around its current position to search for a location with a lower value of the PME energy function, indicating better alignment of the element with its neighboring elements. These displacements are generated by randomly sampling a number of points as the candidate positions of the element in the neighborhood of its current location. All these sampling points are tested and the position with the lowest PME value is returned as the optimal position in current iteration. Note that for each candidate position, we need to rotate and scale the element according to the underlying control field before computing the pattern matching energy.

In order to satisfy the hard connectivity constraints, we first find the neighbors $\mathcal{N}_i$ of P_i in GetNeighbors before updating its position. During stochastic search, we observe that some new neighbors $\mathcal{N}_i'$ of P_i may result from the candidate positions. Such new neighboring relation is accepted, as long as it yields the smallest PME value. Note that during stochastic search, we record data (i.e. relative position, class ID *etc*) of all element pairs that have been traversed and output as connection database $\mathcal{D}$ for the learning purpose in the following boosting step.

Algorithm 1 PLACEOPTIM

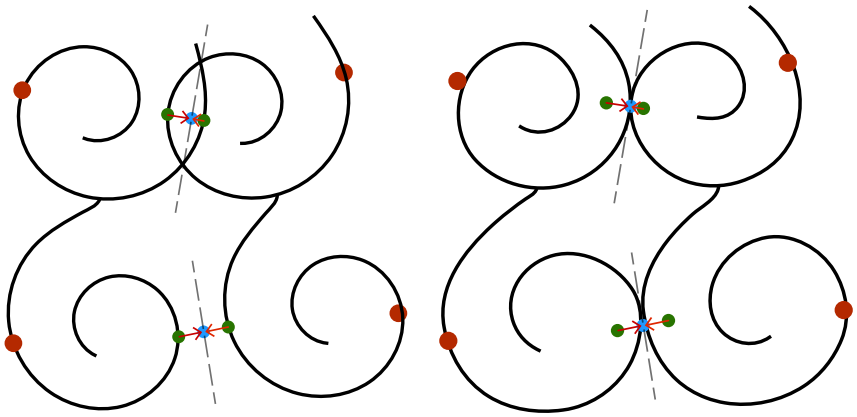
Require:

Input surface model $\mathcal{S}$; Initial placement of pattern set $\mathcal{P}$ on S; Input control field $\mathcal{F}$ on $\mathcal{S}$

Ensure:

An optimized pattern placement $\mathcal{P}_O$ following $\mathcal{F}$ on S that conforms to both input connectivity constraints and control field $\mathcal{F}$; A connection database $\mathcal{D}$ that records all element pair with their matching quality;

1: $\Pi \leftarrow$ BUILDCONNECTIVITYGRAPH($\mathcal{P}$);
2: $\{\Omega_i\} \leftarrow$ INDEPENDENTSET(Π);
3: **while** *true* **do**
4: **for each** $\Omega_i \in \{\Omega_i\}$ **do**
5: **for each** $P_j \in \Omega_i$ **do**
6: $\mathcal{N}_i \leftarrow$ GETNEIGHBORS(P_i, Π);
7: STOCHASTICSEARCH(P_i, $\mathcal{N}_i$);
8: *accept* $\leftarrow$SMOOTHCONNECTION(P_i);
9: **if** *accept* **then**
10: update the shapes of P_i and its neighboring elements;
11: **end if**
12: **end for**
13: **end for**
14: $\mathcal{P} \leftarrow$ UPDATE();
15: **if** converged **or** enough # of iterations reached **then**
16: **break**;
17: **end if**
18: **end while**
19: **return** $\mathcal{P}_O = \mathcal{P}$;



(a) (b)

Figure 2.5 (a) A double intersection and a small gap between neighboring elements are forced to be in tangent contact by deformation in our algorithm. The moving control points are shown in green while the red ones are fixed control points. (b) Tangent contacts are achieved after non-rigid deformation.

2.4.3.2 Smooth Connection via Non-rigid Deformation

During placement optimization, although each pair of overlapping elements are aligned better than before, we observe that their geometric features could often match better to be visually more pleasing if the elements can be modified by a non-rigid deformation of limited amount.

We encourage tangent connection throughout our synthesis pipeline as it is one of most visually pleasing manners to join two neighboring elements. There are two cases, as shown in Figure 2.5, that deserve special treatment. That is, (1) the two curves have two intersection points that are close to each other; and (2) the two curves segments are separated by a narrow gap. In both cases we use non-rigid deformation to bring the two curves into tangential contact with each other. In case (1), the tangential contact keeps the connection between the two elements but make them contact in a smoother way, thus improve the visual quality of the synthesized pattern. In case (2), the forced tangential contact eliminates the narrow gap, thus again improves the visual appearance of the pattern design, and introducing a new pair of connected elements, if that didn't exist before.

The details of deformation method are elaborated in Section 2.8.2. Basically, we achieve non-rigid deformation via moving/fixing a set of control points. Take Figure 2.5a for instance, we first connect the intersection points with a line. The moving control point (shown in green) is picked on the skeleton graph that has the closet tangent with the line slope. The fixed control points (shown in red) are those

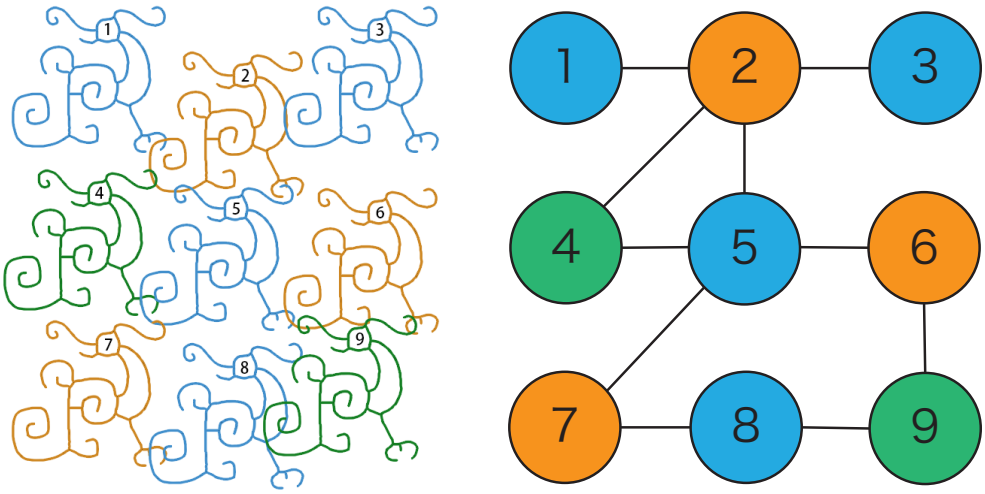


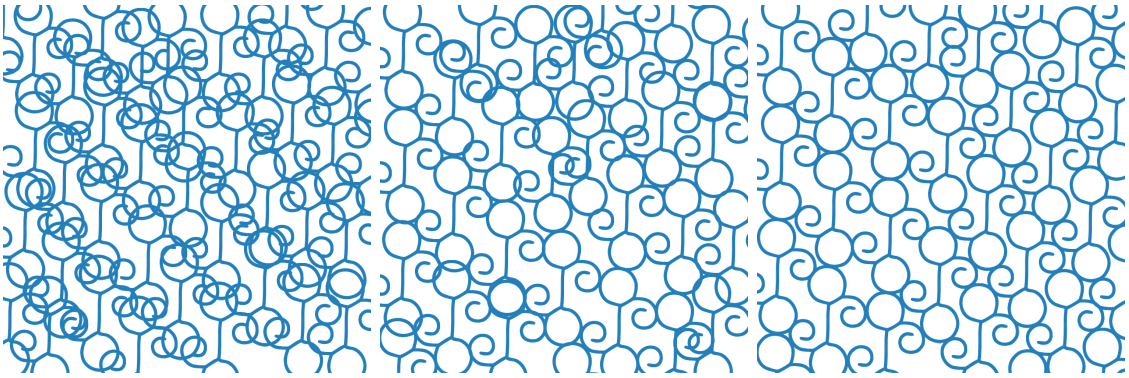
Figure 2.6 (a) Pattern distribution and (b) its corresponding node graph.

intersections points with other neighboring elements that do not need to be changed. We then apply the deformation to achieve tangential contacts by matching the moving control points to their projections on the connecting line while letting the fixed control points remain still. See Figure 2.5b.

Distortion Measurement. As non-rigid deformation may introduce great distortion, we define a cost function to measure the degree of distortion. Given a pattern element P that is discretized into N dense sample points $\{p_i \in P, 1 \leq i \leq N\}$. We use a distance matrix $D_P = (d_{ij})_{N \times N}$ to encode the shape of P, where d_{ij} is the distance between p_i and p_j. Obviously, D_P is independent of translation and rotation. Suppose P is transformed into P' and we can compute distance matrix $D_{P'} = (d'_{ij})$ in a similar fashion. Then we measure the distortion of P' with regard to P using the standard deviation of $\{\frac{d'_{ij}}{d_{ij}}, i \neq j\}$ as the cost function, which gives the measurement on the acceptance/rejection of the deformation.

This phase is implemented in SmoothConnection (Algorithm 1) which performs the deformation and returns the flag of accepting the such deformation based on distortion measurement. If the cost of deformation is lower than a tolerance threshold, we update the shapes of P_i and its neighboring elements accordingly.

Updating Sequence If all elements are updated simultaneously, one problem is that elements updated according to neighbors are also changing. We use an updating sequence based on independent set. First, all the elements are divided into several



(a) Initialization **(b)** After placement optim. **(c)** After boosting

Figure 2.7 Illustration of the effect of each step.

independents sets such that any two elements in the same set are not connected (as illustrated in Figure 2.6b). Then the elements in each set are updated before processing those in the next set. This idea is similar to the *subpass* strategy in [21], which greatly improves their correction performance. Our experiments showed that the convergence of this strategy is much better than using a depth-first or breadth-first processing order in the connectivity graph of the elements. Indeed, one may also adopt a random order to traverse all the elements to optimize their positions; however, our scheme has the potential advantage of allowing parallel processing in this time-consuming task of optimizing all the elements in multiple rounds, since it is ensured that any two elements in the same set are always not connected so can be updated simultaneously without producing conflicting updated positions. In Algorithm 1, BuildConnectivityGraph computes the connectivity graph among input elements. IndependentSet then generates the independent sets afterwards.

Figure 2.7 shows how the initial layout of the pattern elements is improved after the step of placement optimization.

2.4.4 Boosting

Because the minimization of the PME function is a difficult nonconvex and combinatorial optimization problem, the resulting pattern layout after placement optimization may still leave some pairs of connected elements with unsatisfactory alignment. Specifically, while the hard constraint in the preceding step is helpful to maintaining the connectivity of the synthesized filigree pattern, it makes our search prone to getting trapped in a poor local minimum, thus hampering the ability of finding a better pattern layout with an even smaller PME energy value. For example, the

Algorithm 2 BOOSTING

Require:
Input surface model $\mathcal{S}$; Input pattern set $\mathcal{P}$ on S; Input control field $\mathcal{F}$ on $\mathcal{S}$; Connection Database $\mathcal{D}$;

Ensure:
An output pattern placement $\mathcal{P}_O$ that resembles the good spacing example in $\mathcal{D}$ while maintaining sufficient connectivity;

$\mathcal{D}' \leftarrow$ FILTERCANDIDATEPAIRS($\mathcal{D}$);
while TRUE **do**
  $\Pi \leftarrow$ BUILDPATTERNCONNECTIONMAP($\mathcal{P}$);
  $\{\Omega_i\} \leftarrow$ INDEPENDENTSET(Π);
  // Selection Phase
  for each pattern $P_i \in \Omega_i$ **do**
    candidate set $C_i := \emptyset$;
    $\mathcal{N}_i \leftarrow$ find neighbors of P_i;
    for each neighbor $P_j \in \mathcal{N}_i$ **do**
      $b_j \leftarrow$ FINDBOOSTINGVECTOR(P_i, P_j, $\mathcal{D}'$);
      $C_i := C_i \cup b_j$;
    end for;
    $C_i = C_i \cup$ AVERAGE(C_i);
    // Learning Phase
    Attempt all boosting vector $b_j \in C_i$ and assign P_i with the one with lowest PME value.
  end for
  if converged **or** enough # of iterations reached **then**
    return $\mathcal{P}_O = \mathcal{P}$;
  end if
end while

element placement shown in Figure 2.7b cannot be further optimized by placement optimization.

We remedy this issue by using a boosting strategy that replaces a pair of elements with unsatisfactory alignment by the same pair with better alignment, while relaxing the constraint that the connectivity of each involved element with its neighboring element be the same as before. Hence, this operation enables us to search for a better pattern layout starting from a new local initialization. In other words, the boosting step aims to improve pattern distribution by providing better local pattern connectivity so that subsequent application of placement optimization can jump out of a poor local minimum. The boosting step is further followed by another round of placement optimization. These steps are iterated until convergence or a satisfactory result is obtained.

The pseudo-code of boosting is illustrated in Algorithm 2. Specifically, there are two phases of implementing this boosting strategy.

Selection Phase During the massive search in preceding step of placement optimization, nearly all pairs of adjacent elements with good alignment have been collected and kept in the database $\mathcal{D}$. The goal of selection phase is to select candidate boosting vectors that for the learning in the next phase. Before selection phase, we filter out those element pairs with low matching quality in FilterCandidatePairs (Algorithm 2). Specifically, we only kept those pairs with best matching quality (typically top 5% out of all the items in our implementation) and obtained a filtered database $\mathcal{D}'$. While updating P_i, we collect boosting vectors from all its neighbors $\mathcal{N}_i$ in the candidate set C_i. We denote the relative position between (P_i, P_j) as $\nu_{ij} = p_j - p_i$, where p_i and p_j are the local coordinates of P_i and P_j within a locally parameterized surface domain. Similarly, for a candidate pair $(\bar{P}_i, \bar{P}_j) \in \mathcal{D}'$, we can define its relative position vector $\bar{\nu}_{ij}$. Then the similarity metric between (P_i, P_j) and $(\bar{P}_i, \bar{P}_j)$ is formulated as $|\nu_{ij} - \bar{\nu_{ij}}|^2$. For each $P_j \in \mathcal{N}_i$, we query $\mathcal{D}'$ with the element pair (P_i, P_j) to find its best matching pair $(\bar{P}_i, \bar{P}_j)$ with the smallest similarity metric value (i.e. the most similar one). The relative position $\bar{\nu_{ij}}$ of the best match pair $(\bar{P}_i, \bar{P}_j)$ is returned as the boosting vector b_j corresponding to P_j. Note that P_i and $\bar{P}_i$ are the same element (so are P_j and $\bar{P}_j$), but the pairs (P_i, P_j) and $(\bar{P}_i, \bar{P}_j)$ may have different relative vectors. We also append the average vector of all the candidate boosting vectors in C_i as an additional boosting vector (Line 11 in Algorithm 2).

Learning Phase The second phase is to learn from candidate boosting vectors. For each learning attempt, we translate each element with the boosting vector. Note that, since the displacement by the boosting vector does not have to preserve the previous connectivity, there may be significant loss of connection between patterns. When that occurs, we will enhance the connectivity by forcing some nearby pairs of elements to be connected to each other using non-rigid deformation (using identical method in Section 2.4.3.2). We compute the PME value resulting from each boosting vector and keep the one with highest score.

Stopping Criteria We stop the iteration of *Placement Optimization* and *Boosting* if there is no significant change in the synthesized layout. Specifically, we compute the sum of element translation in each iteration and terminate the iterations if it is below a threshold.

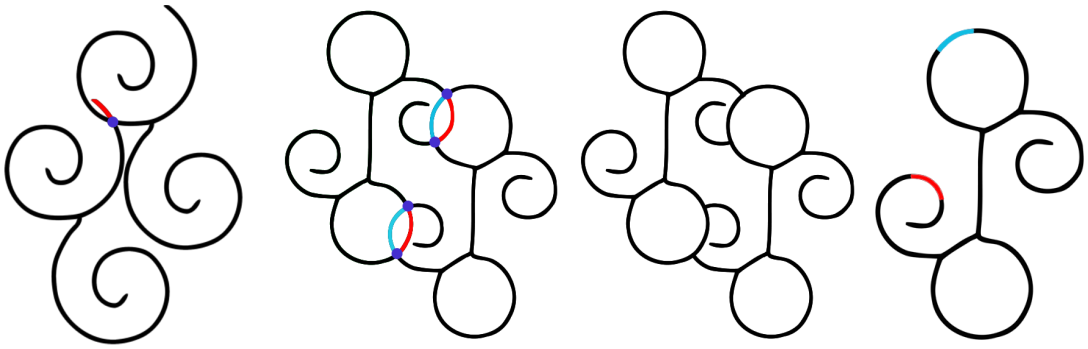


Figure 2.8 (a) Short protruding segment (in red) will be trimmed off. (b) Overlapping of similar features will lead to conflicting branches (blue and red). (c) Topology cut result of (b). (d) The blue subgraph has higher importance than the red one as it is inside a loop.

2.4.5 Topology Cut

After the steps of placement optimization and boosting, there may still exist misalignment between some connected elements and such misalignment are mostly manifested as the intersection of conflicting branches from the two involved patterns. To resolve this issue, we develop an effective scheme, called *topology cut*, for trimming off some conflicting branches within the overlapping region of such two pattern elements. Figure 2.8 shows how this kind of misalignment is resolved by topology cut to improve the visual quality of the output.

We consider two cases in our topology cut scheme: (1) trimming off a short protruding curve segment when two curves intersect at one single intersection; and (2) trimming off some branches when there are more than one intersection points that are close to each other. In case (1), as shown in Figure 2.8a, we simply detect the protruding curve segment that is shorter than some threshold (shown in red) and trim it off.

However, a more elaborate treatment is needed to deal with case (2), where the branches for the two pattern elements intersecting each other in a more complicated manner. Consider two connected patterns, denoted P and Q, whose skeleton graphs intersect at a number of points, t_i, $i = 1, 2, \ldots, m$ (purple dots in Figure 2.8b). Using the intersection points t_i as cutting points, we decompose each skeleton graph into a number of subgraphs. Then the conflicting branches from P and Q are two of these subgraphs. For example, each pair of blue and red branches in Figure 2.8b are two curve segments sharing the same endpoints. We shall next assign an importance score to each of these two subgraphs and remove the subgraph with lower importance score.

The importance score of a subgraph in this context is defined as follows. Suppose that the skeleton graph $G(P)$ of the pattern P is decomposed by the intersection points t_i into a collection of connected subgraphs G_k, $j = 1, 2, \ldots, n$. Intuitively, a subgraph G_k receives a lower importance score, i.e., is less important, if (i) G_k has a small size, measured in the total length of all its edges, denoted as $L(G_k)$; or (ii) G_k is located near "frontier" of its supergraph $G(P)$. (The "frontier" of a skeleton graph is understood to be the set of all the valence-1 vertices.) Regarding the latter consideration, we observe that, usually, G_k is near the "frontier" of $G(P)$ if one of the subgraphs of $G(P)$, excluding G_k itself, has a small size. Hence, we measure the proximity of G_k to the "frontier" of $G(P)$ by $F(G_k) = \min_{j \neq k}\{L(G_j)\}$.

Finally, summarizing the two considerations above, we define the importance score of the subgraph G_K as

$$I(G_k) = \alpha L(G_k) + \beta F(G_k), \tag{2.4}$$

where α and β are weighting coefficients. We use $\alpha = \beta = 1$ in our implementation. Figure 2.8d provides an example to compare the importance scores of two subgraphs of a skeleton graph. Hence, given two conflicting branches that are represented by two subgraphs from the two connected pattern P and Q, we first evaluate the importance scores of the two subgraphs by Equation (2.4), with respect to their own supergraphs $G(P)$ and $G(Q)$, respectively. Then we keep the subgraph with the higher importance score, and trim off the other.

2.5 Structural Optimization

During structural optimization, we iteratively detect weak regions and reinforce these areas. This iterates until the shape is strong enough. The user can manually specify the initial thickness of model and external force profiles. The external forces are pressure forces from the outside, applied on all shell elements. The force profiles could be easily changed to match different scenarios.

Our structural optimization contains two phases. In the first phase, we apply structural analysis to detect mechanically weak regions of the reconstructed surface mesh (see Section 2.6 for more details on mesh reconstruction) based on the synthesized filigree design. New pattern elements are then inserted into the weak regions to create denser coverage and connections. We then re-synthesize the result throughout previous stages based on the new pattern layout. We define the weak node as the one with Von Mises stress larger than yield strength σ_{yield}. The first phase is repeated until the portion of weak nodes is below a threshold (typically 0.5% in our implementation).

See Figure 2.9 for an example showing how the mechanical strength of a synthesized filigree pattern is enhanced after the first phase.

After strengthening local weak parts, we obtain a balanced structure with respect to both gravity and external forces. We then iteratively increase the thickness of model in the second phase, by 10% in each iteration, until no weak nodes are detected.

Our structural optimization tries to strengthen the weak parts first before increasing the thickness of model. This pipeline is based on our observations in experiments that if we increase the model thickness first, the weak regions cannot be totally removed even when the thickness is increased significantly. However, we observe that if we strengthen weak regions first to obtain a uniformly balanced structure, the strength of the model can be greatly improved with a small increase in thickness, which leads to faster convergence.

Finite Element Simulation In order to accelerate structural simulation, we apply numerical simulation on open surface mesh with shell element analysis. This treatment leads to faster simulation for two reasons: (1) The generation of volume mesh with specified thickness is time-consuming, while the thin shell model based on a 2D surface mesh can be generated much faster. In fact, a surface mesh with a preset thickness is a good approximation to a closed mesh for the purpose of structural simulation. (2) The computation for analyzing shell elements is much faster than conventional methods based on volume elements [63].

We simulate the elastic material with the properties of commonly used printing materials, typically ABS or PLA plastic. We use Von Mises for stress simulation which is formulated as

$$\bar{\sigma} = \frac{1}{\sqrt{2}}\sqrt{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2} \tag{2.5}$$

where $\sigma_i (i = 1, 2, 3)$ represents the eigenvalue of the stress tensor. The stress tensor field is calculated from the displacements and rotations at all nodes of input model, following a standard procedure of finite element analysis [3].

Figure 2.9 shows the intermediate results of structural optimization. As seen from the results, the weak regions (shown in red) are greatly reduced after structural optimization. We also present a complete comparison in Section 2.8.3.1 that demonstrates the mechanical properties of all tested models before and after structural optimization.

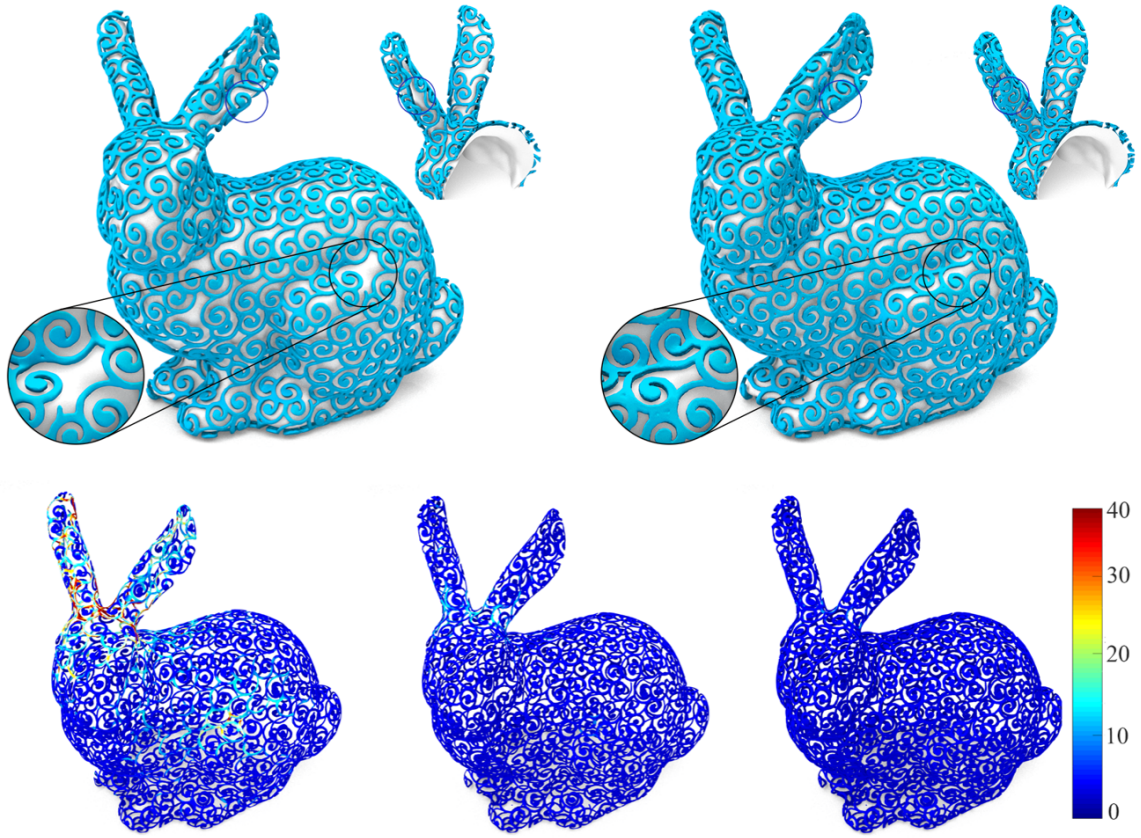


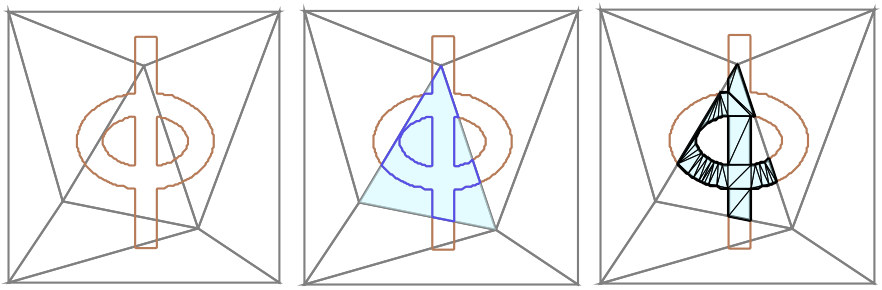
Figure 2.9 ***Top:*** *From left to right, synthesis result before and after adding more element connections to strengthen weak parts.* ***Bottom:*** *Color coded stresses results during structural optimization. Left: Before structural optimization. Middle: After strengthening local weak regions. Right: After increasing model thickness. The bunny model is courtesy of of Stanford 3D scanning repository.*

2.6 Reconstruction

We reconstruct the printout model, represented as a polygonal mesh, in two steps: (1) generate a hallowed surface mesh according to the composite skeleton graph and the width function defined on it; and (2) reconstruct a printable mesh with specified thickness via an offsetting operation.

2.6.1 Surface Mesh Reconstruction

Based on the skeleton graph and the width function, we can recover a set of contours $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ bounding the solid area that should be kept; See Figure 2.10a. For the i-th triangle T of the base triangle mesh, we perform the 2D boolean



(a) **(b)** **(c)**

Figure 2.10 Pattern baking: (a) The input contours, (b) The boolean intersection between a triangle and the input contours, and (c) Constrained Delaunay triangulation of the intersecting region.

intersection between T and the composite contour $\mathcal{C}$; See Figure 2.10b. After that, we build the constrained Delaunay triangulation of the region $T \bigcap \mathcal{C}$ and get a collection of subtriangles $\mathcal{T}_i = \{T_1 \in T, T_2 \in T, \cdots, T_{n_i} \in T\}$; See Figure 2.10c. Performing the boolean operation all over the base mesh yields a triangle pool $\bigcup_i \mathcal{T}_i$ that actually gives the baked mesh $\mathcal{M}_b$ defined by the synthetic pattern elements. Note that $\mathcal{M}_b$ serves as both the input of stress analysis and the base mesh for generating a printable mesh with specified thickness.

2.6.2 Final Mesh Generation

Suppose that a thickness function $\tau(\cdot)$ has been specified on a surface $\mathcal{M}_b$. We compute the bounding surface $\mathcal{M}_b^\tau$ of the volume induced from the pair $(\mathcal{M}_b, \tau)$. For a point p in $\mathbb{R}^3$, there always exists a closest point $q \in \mathcal{M}_b$. We say p lies inside the volume $\mathcal{M}_b^\tau$ if and only if

$$\|p - q\| \leq \frac{\tau(q)}{2} \tag{2.6}$$

and

$$\frac{(p - q) \cdot \text{Normal}(q)}{\|p - q\|} = 1, \tag{2.7}$$

where $\text{Normal}(q)$ is the surface normal at q. Equation (2.7) serves only when q is located on the open boundary of $\mathcal{M}_b$. Or alternatively, we can use

$$F(p) \triangleq \|p - q\|^2 - \frac{\tau(q)}{2}(p - q) \cdot \text{Normal}(q) \leq 0 \tag{2.8}$$

to define the volume of $\mathcal{M}_b^\tau$. Sometimes we prefer inflating the point q to all directions, rather than only along $\text{Normal}(q)$, when q is located on the open boundary. And in

(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 2.11 With the filigree synthesis technique proposed in this paper applied to input surfaces (b) and (f), the generated models (c-e) generally present more fascinating and aesthetic appearance than the base models. Note that the primitive filigree patterns used for (c-e) are shown in (a), whose backgrounds are respectively painted in blue, orange and green. The input models in (b), (d) and (f) are courtesy of Kevin Xu, Open3DModel and PinShape, respectively.

this case, $F(p)$ reduces to

$$F(p) \triangleq \|p - q\|^2 - \frac{\tau^2(q)}{4} \leq 0. \tag{2.9}$$

We use the Poisson surface reconstruction [17] to extract the triangle mesh approximating the surface $F(p) = 0$, which is the boundary surface of the virtual model for fabrication.

2.7 Results

2.7.1 Basic Synthesis

We first present some synthesis results in 2D planar domain for validation. Figure 2.12 shows a variety of results generated by our method. As seen from the results, our algorithm generates smooth connections between patterns via either tangent connections or overlapping their similar features.

Since we formulate filigree synthesis as dense packing problem with appearance constraints, we compare our results with the state-of-art packing method [13] in Figure 2.14. As shown in Figure 2.14, conventional packing method cannot guarantee full connections between adjacent elements thus fails to satisfy the printable criteria. Moreover, the packing method tends to randomly place the element since it cannot

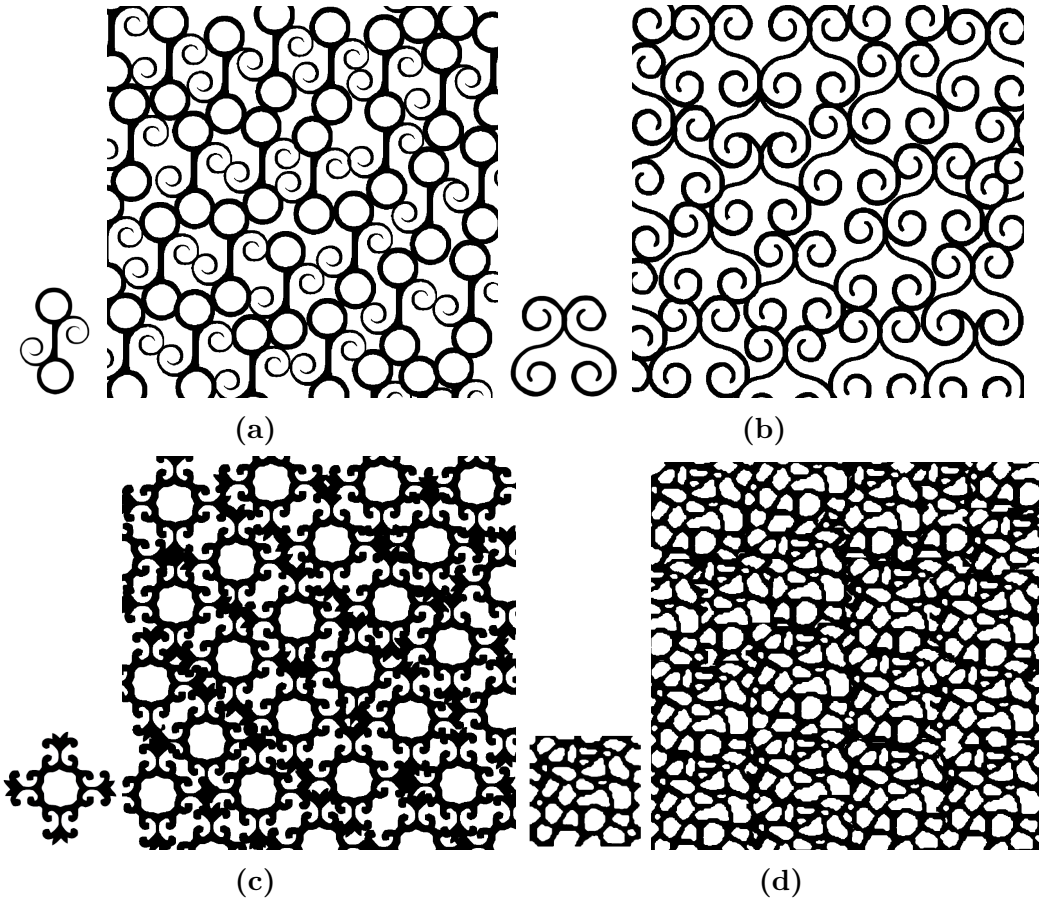


Figure 2.12 Single-class synthesis results on 2D in which a variety of different elements are used as input.

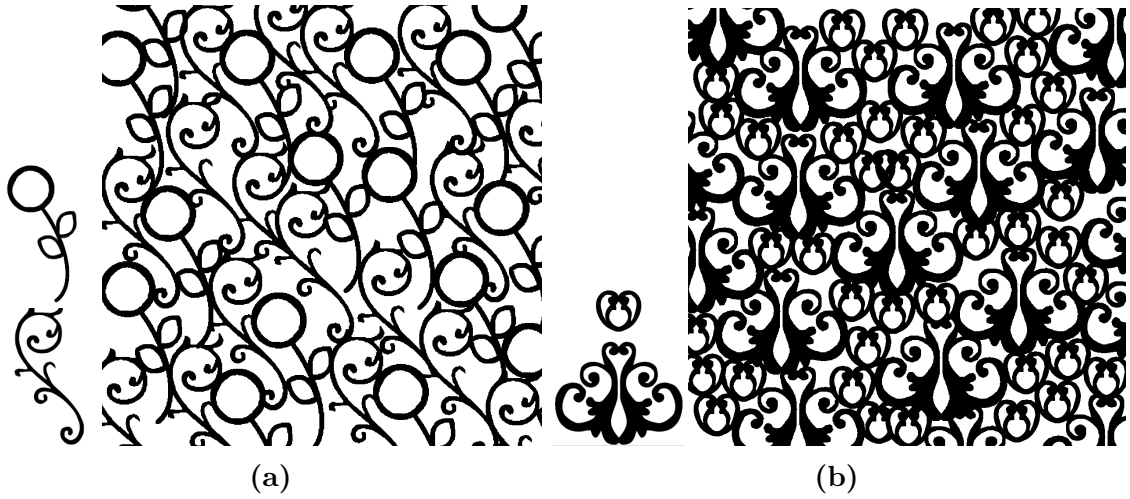


Figure 2.13 Multi-class synthesis results on 2D planar domain.

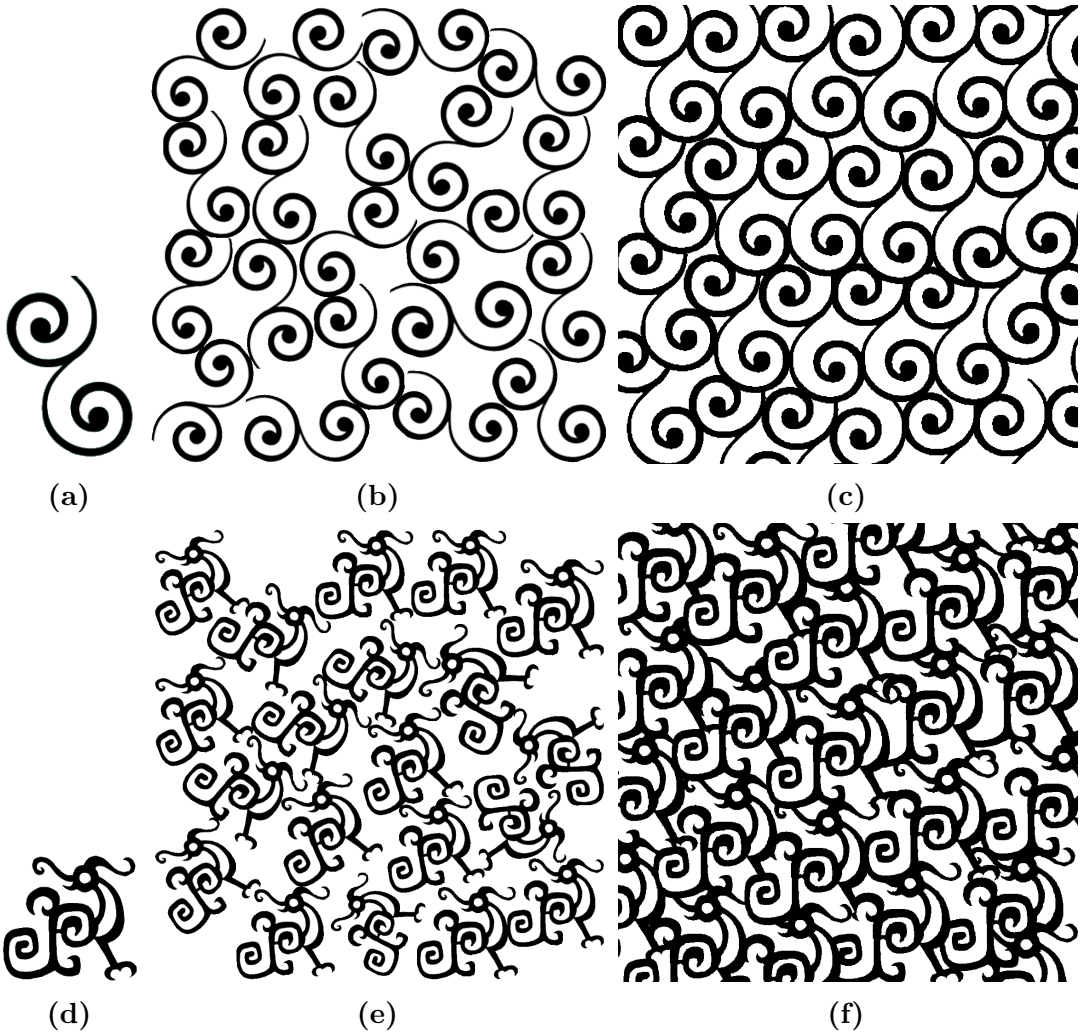


Figure 2.14 Comparisons with the packing method in [Hu *et al. 2016*].

exploit the partial shape similarity between elements to form smooth connections. Our method, on the other hand, is capable to ensure all the elements are connected in one piece while naturally join them without noticeable artifacts.

By allowing partial overlapping with good shape alignment, our method smoothly connects adjacent patterns of different shapes. Figure 2.13 demonstrates our synthesis results when multiple-class patterns are specified as input. Here, our method automatically detects the similar parts between different patterns and join them in a natural way. In Figure 3.12c, different filigree elements are used for synthesis on different parts of a vase. All these exemplar pattern elements are from actual filigree jewels. These results show that our method is capable of producing high-quality filigree decorations.

2.7.2 Control Field Editing

Our method can adapt the size and orientation of pattern elements according to an underlying control field on the base surface. Figure 2.15 shows different control fields on a 2D domain and the corresponding synthesized filigree pattern. Clearly, the pattern elements change their size and orientation according to the control field, while maintaining natural connections between adjacent elements.

Figure 2.16 shows a field-controlled synthesis result on a dress model. The control field here consists of an orientation field and a scaling field. We have developed a user interface to let the user specify the orientation field by sketching lines on the surface mesh, as shown in yellow curves in Figure 2.16a. We then generate a smooth orientation field by treating the sketch lines as constraints. Here the code from [8] is used to build the orientation field. The user can specify the scaling field by drawing closed regions on the base surface and assign scaling values to the designated regions. Then the scaling values are propagated to the entire domain via error diffusion.

Our method also supports boundary-aware synthesis in that we only synthesize a filigree pattern within a selected region. We observe that proper boundary handling is important in boundary-aware synthesis in order to produce satisfactory results.Hence, we add additional constraints to encourage pattern elements next to the domain boundary to have tangential contact with boundary curves. This can easily be achieved within our framework by applying proper translation and limited deformation during pattern synthesis.

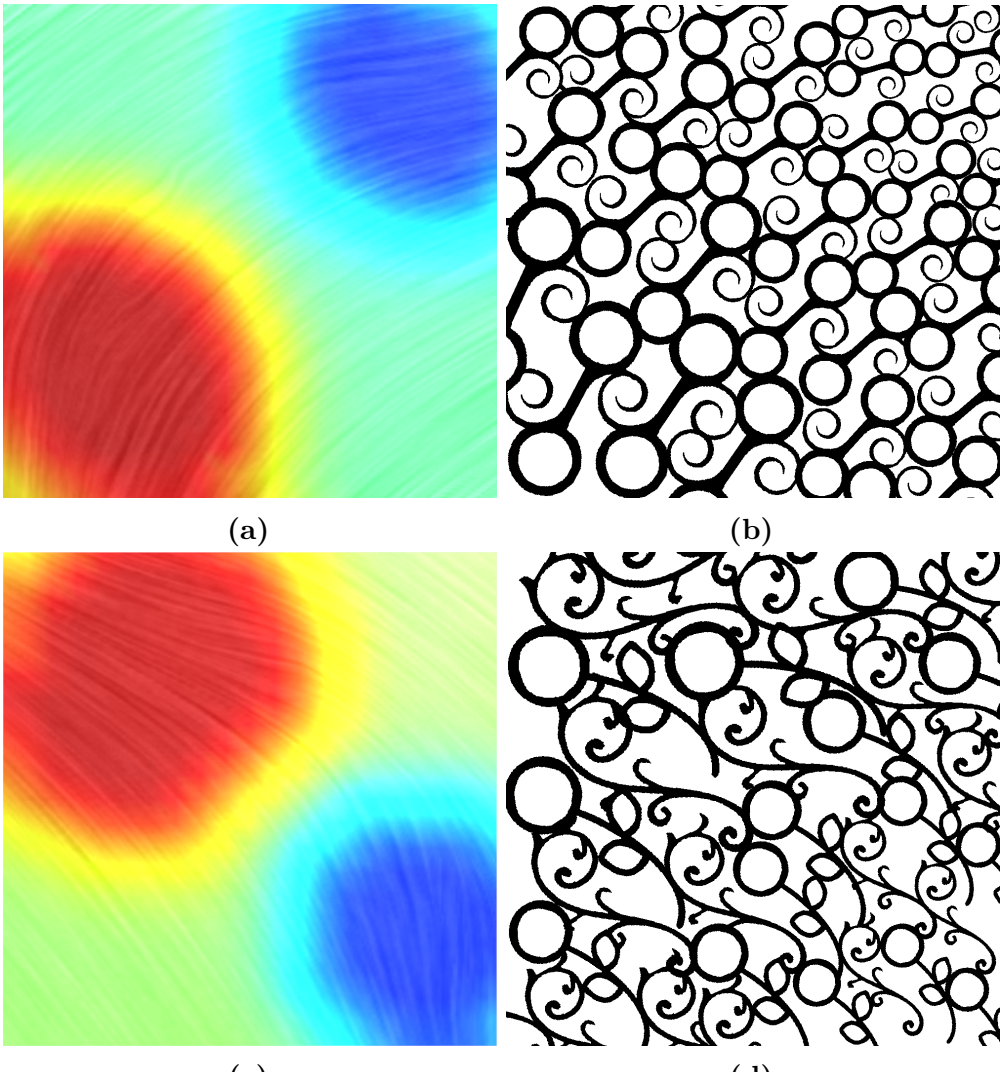


Figure 2.15 Results of field controlled synthesis in 2D planar domain. For each row, the input control field is shown on the left. The control field is visualized using line integral convolution (LIC), where the color encodes the scaling of the elements (red - larger scale; blue - smaller scale) and the orientation is illustrated by streamlines.

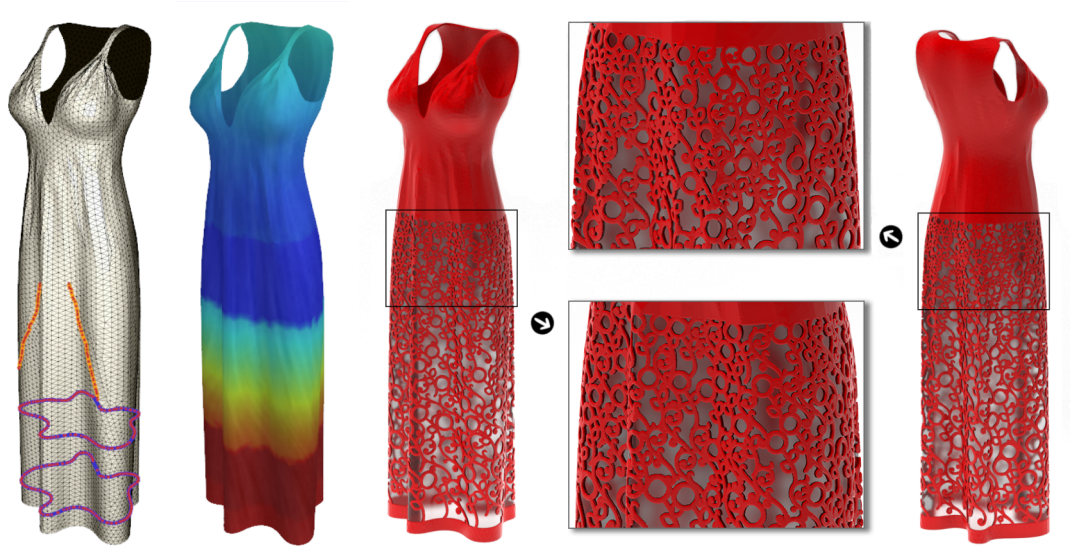


Figure 2.16 **(a)** User interface. User can easily design the control field via sketching lines (yellow with red dots) on the surface mesh to define the orientation field and assigning scale values to hand-drawn (blue lines with red dots) regions to design the scaling field. **(b)** Control field. An example control field generated by user which is the input field for the synthesis result in **(c)**. **(c)** We support constrained synthesis within the user-specified regions. The synthesized elements are well aligned to the boundary and transformed according to the input control field. The input dress model is courtesy of Open3DModel.

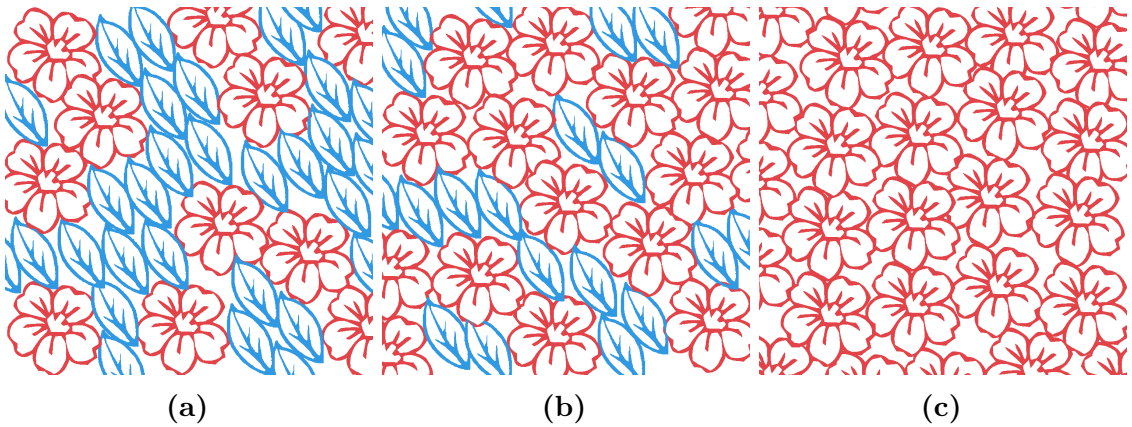


Figure 2.17 2D Synthesis results with different percentages of each element class.

(a) (b) (c)

Figure 2.18 Synthesis results with different percentages of each element class in 3D.

2.7.3 Class Number Control

For multi-class synthesis, our method is capable of controlling the percentages of different classes of elements that appear in the final output. Figure 2.17 shows a series of results with an increasing number of flower patterns until it becomes a case of single-class synthesis. This is mainly achieved in the initialization step, in which the initial pattern distribution is generated. We adopt the class control strategy in [52]. According to Equation (1) in [52], the pattern with larger size will have lower priority to sample from. We control the percentage of each class by multiplying a scaling factor to the size of the pattern. If the scaling factor of an element is set to infinity, then that element will not appear in the output domain which makes single-class synthesis possible.

To achieve uniform distribution among different classes of elements, we penalize the overlap area if one element overlaps with an instance of the same element. Specifically, overlapping with the same class of elements will lead to higher overlapping ratio, thus will be rejected with higher probability. Figure 2.18 shows three examples of decorating the outer shell of a vase with different percentages of the flower and leaf patterns.

Figure 2.19 Prototypes printed by Stratasys Fortus 400mc and EOS FORMIGA P 110.

2.7.4 Fabrication

Figure 2.19 shows the printouts of our synthesis results. Note that the base elements remain easily identifiable on the surfaces while the element connections are smooth and robust even in highly curved regions, e.g. the bunny ears. The powder-based 3D printers present challenges on the model as the printout is very fragile when removed from the powder basin. However, our model is successfully printed with thin features, due to effective structural optimization.

2.8 Implementation and Performance

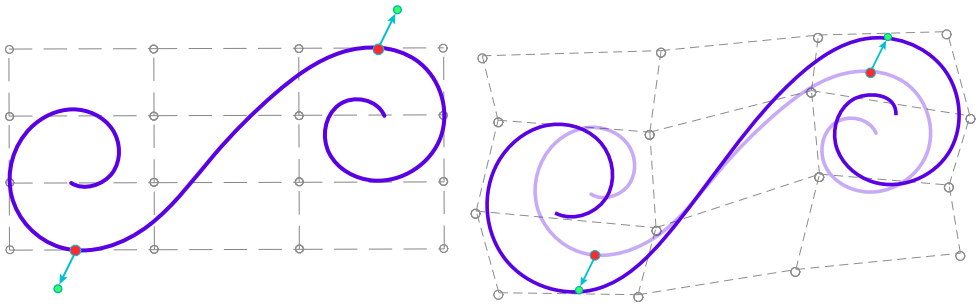
2.8.1 Input and Parameters

The input model of our method is a triangular mesh. All computations (e.g. modified Hausdorff distance) are done on a locally parameterized 2D surface patch. Each pattern has an up vector, which we maintain aligned with the underlying vector field to rotate elements along the surface. The patterns are mapped back to the surface mesh, so they naturally bend with the input model.

Our method is fully automatic but there are two main parameters that would affect the final results: (1) the threshold σ_f to filter element pairs based on matching quality. We typically use 5%. Larger values would lead to more randomization in the

Parameters	Bunny	Lamp	Dragon Vase	Vase	Dress
σ_f	5.0%	4.5%	5.0%	5.0%	4.5%
σ_d	0.15	0.13	0.15	0.17	0.16

Table 2.1 Parameters used in all tested models.



(a) **(b)**

Figure 2.20 (a) A 2D bicubic Bézier surface patch covers the skeleton, with its control points regularly distributed. The specified points (in red) must be moved to corresponding destinations (in green). (b) The soft constraints to avoid large distortions are also taken into account by moving the control points.

output; (2) the control of the distortion σ_d during deformations. A larger tolerance leads to larger deformations. We typically use 0.15. A complete list of the values employed for each model is shown in Table 2.1.

2.8.2 Deformation

For filigree synthesis problem, there is no strict requirement that the filigree element be kept in shape and size. Hence, we take the flexibility of slightly deforming filigree elements and develop non-rigid deformation method that is extensively used in our framework to enable smooth tangential contact and improve alignment of partial overlaps. Our method is fast because it needs only to solve a linear system of equations without iterative computation.

In the context of filigree synthesis, we need to deform a given pattern element P into another pattern shape P' with the following constraints: (1) Hard constraints. Some selected points p_k, $k = 1, 2, \ldots, m$, on P are mapped to some designated corresponding points p'_k on P'. (2) Soft constraints. The shape of P' is similar to that P as much as possible, not allowing a large distortion in any local or global subpart

of the pattern element. Note that the deformation is applied to skeleton graphs, the internal representation of filigree patterns.

Our deformation method follows the spirit of 2D free-form deformation using a bicubic Bézier surface patch [38] to deform the region containing a filigree element. Specifically, given a pattern element S to be deformed, we use a rectangular bounding region to cover P and define a bicubic Bézier surface patch $S^0(u, v)$ with its array of 4 by 4 control points p^0_{ij}, $i, j = 0, 1, 2, 3$, being regularly positioned within the bounding rectangular region, as shown in Figure 2.20. Since its control points p^0_{ij} are regularly distributed, $S^0(u, v)$ reduces to an affine mapping between the parameter space (u, v) and the output domain space (x, y) in which the filigree element S lies. Hence, for an arbitrary point $q^0_k = (x_k, y_k)$ of the output domain, it is easy to find its corresponding unique parameter values (u_k, v_k), with $q^0_k = S^0(u_k, v_k)$. With the fixed parameter values (u_k, v_k), the initial control points p^0_{ij} will be replaced by variable control points p_{ij} during the deformation process, so the point $q^0_k = (x_k, y_k)$ will be mapped to $q_k = S(u_k, v_k)$, where S is the new bicubic Bézier surface patch.

We now consider the following four types of constraints applied to the deformation.

Type 1. Suppose that we want to map some selected points q^0_k on the pattern P to their corresponding points q_k, $k = 1, 2, \ldots, m$. Then we have the constraints

$$a_k \equiv S(u_k, v_k) - q_k = 0, \quad k = 1, 2, \ldots, m. \tag{2.10}$$

Note that $S(u_k, v_k)$ is a linear combination of the unknown control points $\{p_{ij}\}$.

Type 2. We wish to keep the domain containing the pattern P to be deformed as little as possible. Therefore, we impose the "soft" constraint that any three consecutive control points that are equally spaced in the initial setting to still keep being equally spaced. Let $p_{i,j}$, $p_{i',j'}$ and $p_{i'',j''}$ denote three such control points, with p_{ij} lying between the other two. Then this constraint is expressed as

$$c_{ij} \equiv p_{i,j} - \frac{1}{2}(p_{i',j'} + p_{i'',j''}) = 0. \tag{2.11}$$

Type 3. Recall that the skeleton graph of a filigree pattern is represented by polygonal curves. To keep the shape and orientation of the skeleton graph under deformation, we first require that each straight-line segment of the skeletal element keep its original direction as much as possible. Let v^0_i, v^0_j be two consecutive vertices of the skeleton graph before deformation and let v_i, v_j be their corresponding points

after deformation. Then we impose

$$d_{ij} \equiv (v_i - v_j) - \tau_{ij}(v_i^0 - v_j^0) = 0 \tag{2.12}$$

where τ_{ij} is the scaling factor between the two parallel vectors. Note that the points v_i and v_j are not new variables but are expressed as functions of the variable control points p_{ij}. However, the scaling factors are new variables and they need to be regularized as follows to prevent excessive shape distortion of the skeleton graph.

Type 4. We assume that the scaling factors $\{\tau_{ij}\}$, imposed on the segments of the skeletal element, smoothly change on the domain of interest. Suppose the segment $\overline{v_i^0 v_j^0}$, with a scaling factor τ_{ij}, has h neighboring segments, i.e. sharing an endpoint with $\overline{v_i^0 v_j^0}$, whose corresponding scaling factors are $\tau_{ij}^1, \tau_{ij}^2, \cdots, \tau_{ij}^h$, we have

$$g_{ij} \equiv \tau_{ij} - \frac{1}{h}(\tau_{ij}^1 + \tau_{ij}^2 + \cdots + \tau_{ij}^h) = 0. \tag{2.13}$$

Then we form a constrained linear least squares problem by including the hard constraints of type 1 with Lagrange multipliers, and the other soft constraints of types 2, 3 and 4 using penalty terms. Let S denote the set of control points p_{ij} and T denote the set of scaling factors τ_{ij}. Then the resulting objective function to minimize is

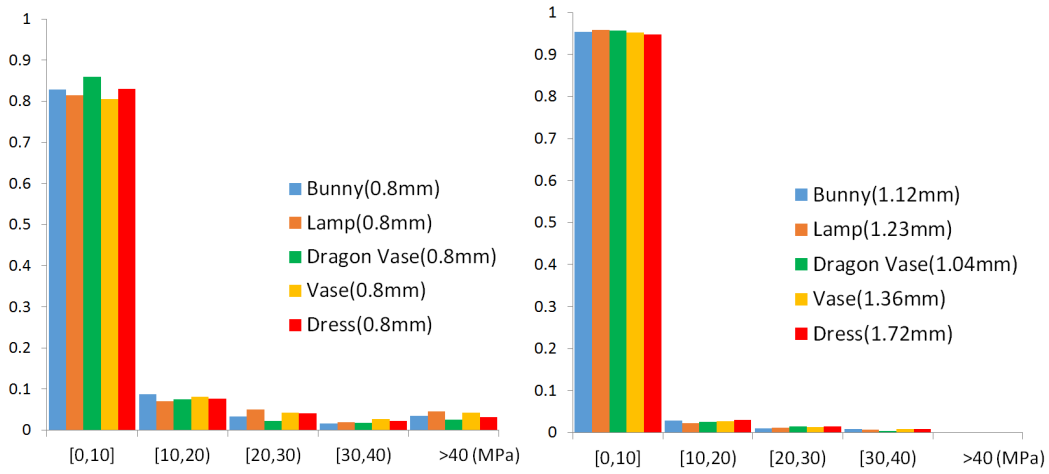
$$F(S,T) = \sum_k \theta_k a_k + \lambda_1 \sum_{ij} c_{ij}^2 + \lambda_2 \sum_{ij} d_{ij}^2 + \lambda_3 \sum_{ij} g_{ij}^2 \tag{2.14}$$

where θ_k are Lagrange multipliers for the constraints $a_k = 0$ of type 1, and $\lambda_1, \lambda_2, \lambda_3$ are weighting coefficients of the other penalty terms. In our implementation we take $\lambda_1 = 2, \lambda_2 = 60$ and $\lambda_3 = 0.5$. Because $F(S,T)$ is quadratic with regard to the variables S and T with linear constraints $a_k = 0$, it can be minimized efficiently by solving a linear system of equations with a sparse coefficient matrix.

2.8.3 Quantitative Analysis

2.8.3.1 Stress

We compare the stress distribution of all input models before and after structural optimization in Figure 2.21. The horizontal axis shows the intervals of Von Mises stress while the vertical axis is the percentage of nodes lying in each interval. The yield stress of the plastic material used in our models is 40MPa. As seen from Figure 2.21a, before structural optimization, there exist weak nodes (stress larger than 40MPa) in



(a) Before structural optimization **(b)** After structural optimization

Figure 2.21 Stress distribution of all tested models before and after structural optimization.

resulting models. However, no weak nodes are detected after optimization thanks to stronger pattern connection and increase of model thickness (as shown in Figure 2.21).

2.8.3.2 Matching Energy

Figure 2.22 shows the energy curve of appearance optimization with respect to iterations. The y axis is the pattern matching energy normalized by pattern number. Note that each iteration contains both placement optimization (Section 2.4.3) and boosting (Section 2.4.4). The matching energy generally decreases as the number of iterations increases, thus gradually improve the appearance. The jumps seen in the curves are due to new pattern connections added in structural optimization. The iterations would stop if the stopping criteria (Section 2.4.4) is satisfied. Normally, models covered by less and simpler patterns will converge faster. As seen from Figure 2.22, all tested models are stably converged to low PME values.

2.8.4 Timing

Table 2.2 summarizes the statistics of some synthesis results in 2D. When the number of patterns runs between 20 and 60, all the results are produced within 3 iterations. The computing time of 3D synthesis results are shown in Table 2.3. Unlike the 2D case, most of the computation time is spent on computing the local parameterizations of local surface patch. Thus, the computing time is highly related to the number of vertices and faces on the mesh surface. The results reported above were produced on a PC with Intel i7-4770 CPU and 16GB RAM.

2D Result	#Ele	#Iter	t_{total}	2D Result	#Ele	#Iter	t_{total}
Fig. 2.12a	31	2	10.2	Fig. 2.13a	34	2	13.7
Fig. 2.12b	25	2	6.6	Fig. 2.13b	57	3	25.3
Fig. 2.12c	28	2	8.3	Fig. 2.15b	32	2	11.3
Fig. 2.12d	20	2	11.4	Fig. 2.15d	32	2	11.9
Fig. 2.14c	28	2	7.9	Fig. 2.17a	41	2	16.2
Fig. 2.14f	23	2	7.3	Fig. 2.17b	35	2	14.2
Fig. 2.17c	26	2	8.0				

Table 2.2 Statistics of all the 2D synthesis results shown in the paper. The iteration number refers to the number of iterations for pattern synthesis, including both placement optimization and boosting. The timings are in seconds.

3D Model	# V	#F	# E	$\#I_{syn}$	$\#I_{sct}$	t_{syn}	t_{sct}
Bunny	57154	114304	242	6	2	630	186
Lamp	4679	9098	75	5	2	138	36
Dragon Vase	29891	59786	521	8	3	1182	246
Vase	5419	10787	129	5	2	252	72
Dress	8172	16117	316	7	2	792	198

Table 2.3 Statistics of some 3D synthesis results, showing the number of vertices, the number of faces, the number of elements, the number of iterations of pattern synthesis, the number of iterations of structural optimization, timings for pattern synthesis, and timings for structural optimization (in seconds).

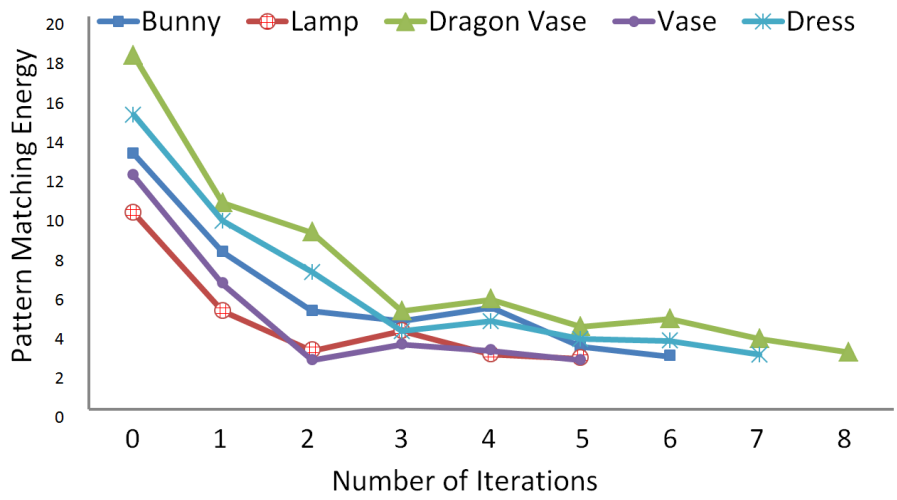


Figure 2.22 Energy curve of appearance optimization of all input models.

2.9 Conclusions

We have proposed a new method for filigree synthesis based on a Pattern Matching Energy (PME) function, equipped with a stochastic optimization strategy, to guarantee that the output model is sufficiently connected, visually artifact-free and structurally strong for fabrication. Our method also allows for flexible user controls, such as the scale and orientation of pattern elements on the base surfaces.

Our method, in its current form, cannot handle those pattern elements that lack an obvious skeletal representation, e.g. the chessboard texture, or the case where a good local alignment between the input pattern elements cannot be found. In addition, the optimization phase is still time consuming due to the mesh reconstruction required by mechanical simulator. Future works include how to support a greater variety of pattern types and investigation into the possibility of conducting structural optimization directly on the skeleton graph.

Chapter 3

Fabricable Tile Decorations

3.1 Introduction

Additive manufacturing is paving the way to mass customization, enabling anyone to create her own version of a base product, customizing it in surprising and beautiful ways. In addition, with the wide availability of 3D printers in FabLabs and homes, users can fabricate their customized objects directly, in a wide variety of colors and materials. However, customizing objects is a difficult task for non-expert users, for all but very simple modifications. Thus, there is a significant research effort dedicated to enabling computational support for customization of shapes. For instance, researchers have focused on balancing shapes [35], creating spinable objects [2], turning surfaces into physical filigrees [4], designing wind instruments [24, 46], lampshades [59] and helping users maintain fabricability during modifications [41].

In this paper we focus on bringing a popular type of decoration to customization for 3D printing: patterns obtained by packing together distinct decorative elements, or *tiles*. Such patterns are ubiquitous in fashion and design, but are also popular among hobbyists as they are obtained by juxtaposing existing decorations, e.g. stickers or decals. Our objective is to enable non-expert users to decorate surfaces with a set of flat tiles. Taken together, the tiles form a 3D network that outlines the same volume as the initial surface, and can be fabricated using a home printer (see Figure 3.12). The problem is challenging as the tiles must be closely packed such that connectors can be inserted, while approximating the original surface well.

In addition to the core packing problem, it is worth considering how the output can be fabricated. One could imagine printing the result as a single piece. However, the final shape is a shell-like object that typically occupies a large volume compared to its actual material usage, and has many overhanging features. This makes the

parts quite inefficient to print. In particular, on the printers most widely available for home users – filament printers – large amounts of additional support structures would be required. Support removal and cleaning makes printing much less enjoyable, and increases time and material costs. While some printers do not require supports (e.g. powder bed systems like selective laser sintering), print efficiency on large hollow objects is less than optimal: such devices are most efficient when printing a batch of parts at once. Hollow shapes occupy a large volume while using little material, and thus severely reduce the efficiency of batch printing.

We design our approach to take into account these fabrication issues. Instead of producing a single, final part, we exploit the fact that the final object is a network of tiles joined by small connectors. In particular, these connectors do not have to be rigid. Neighboring tiles can be allowed to rotate with respect to one another, as long as they finally lock into a stable shape. The connectors are natural places where we can cut the design, and later assemble it back through the use of *snap-fit joints*. Therefore, our approach produces the final design as a set of disjoint *flat* patches, that are printed independently and later assembled. This completely removes the need for support, alleviates print bed size limits, and significantly improves print efficiency. The final assembly is left to the user, however it is a much more enjoyable operation than cleaning support.

3.2 Previous Work

Art and design. As early adopters of additive manufacturing, artists are constantly pushing the boundaries of fashion and design. Among the many examples of this trend, *Nervous System* has produced beautiful flexible designs made of many interconnected triangles – the resulting objects are bracelets, necklaces or even full dresses. The flexibility is obtained by placing hinges in between the triangles of a densely tessellated initial flat surface. As a result, the designs can be printed flat, pre-assembled, on low-cost printers.

While this is a key inspiration for us – in particular showing the importance of taking into account printability concerns in the design itself – our goal is different. We seek to produce 3D structures made of packed, arbitrary shaped tiles. We do not have a base flat contour to work from, and we approximate a target 3D surface.

Pattern synthesis. Several recent works consider the problem of decorating surfaces to turn them into printable objects, with the same purpose of customization for

3D printing. Zhou et al. [62] synthesize connected patterns along curves, and use their approach to model fabricable 3D objects. Dumas et al. [10] modify texture synthesis to account for structure and rigidity, and synthesize printable 3D patterns that cover a base surface. Chen et al. [4] synthesize filigree-like structures along surfaces. Their approach is closest to ours since the input is a set of base elements and a target surface. However, they relax the packing problem by allowing appearance-preserving overlaps between elements. This produces very appealing patterns, but cannot guarantee their base shape is preserved, and thus forbids the use of patterns with a semantic meaning (e.g. a fish, a heart, a letter). Zehnder et al. [56] explore the interactive design of curve networks onto surfaces. The user positions curve elements (visually similar to a bended wire) onto a surface. The curves are simulated as elastic rods, giving a very natural feel to their deformations, while intersections are disallowed to preserve their appearance. Tight packings are achieved thanks to deformations. This approach produces beautiful, airy curve networks that can be fabricated on high-end printers.

All the aforementioned techniques output complex yet hollow 3D geometries that are challenging to print. In contrast, our approach strives to produce an easy and efficient to print output.

Packing onto surfaces. Packing of arbitrary shaped elements into the plane is an extensively researched topic important to many industries (e.g. textile). However, packing on surfaces is less explored. Lai et al. [20] and Dos Passos et al. [33, 32] proposed methods for creating mosaics on surfaces using convex planar tiles. Hu et al. [13] recently proposed an approach for surface mosaics that supports *irregular* planar tiles. It operates through iterated continuous and combinatorial steps. Other techniques consider pattern distributions by locally mapping 2D elements (i.e. *decals*) onto surfaces, e.g. [23, 37, 51]. These are less suited to our needs as we seek to preserve the planarity of the packed elements.

In this work we use the approach of Hu et al. [13] as a comparison baseline regarding packing quality. It is worth noting, however, that the problem settings are different. The aforementioned techniques are geared towards mosaicking, where large numbers of relatively small tiles are placed. We instead have to use fewer tiles, large with respect to object curvature, to accommodate for fabrication constraints.

Fabrication from sheets and wires. Several recent works focus on helping users fabricate shapes from simpler materials. Miguel et al. [29] optimize wire sculptures that approximate input surfaces. Iarussi et al. [15] present an approach for wire

jewelry design. Other approaches consider fabrication from paper or similar planar sheets. The original surface is unfolded into charts [40] or strips [30, 44] which can be cut, folded and assembled to approximate the input. Fabrication from flat surfaces has also been studied for modeling large objects from laser cut wood pieces [7], for modeling tight-fit cloths from 3D scanners by automatic flattening [58], to fabricate surfaces that self-fold under the action of heat [19], and for fabrication from wire mesh sheets [11] and auxetic planar materials [18].

While our approach shares the idea of assembling from planar patches, our problem setting is very different as we consider networks of rigid planar tiles along the surface, and exploit 3D printed hinges to fold patches into shape.

3D printing large objects. Solutions have been proposed for printing large objects, which either do not fit the printer, or occupy a large volume compared to their material use.

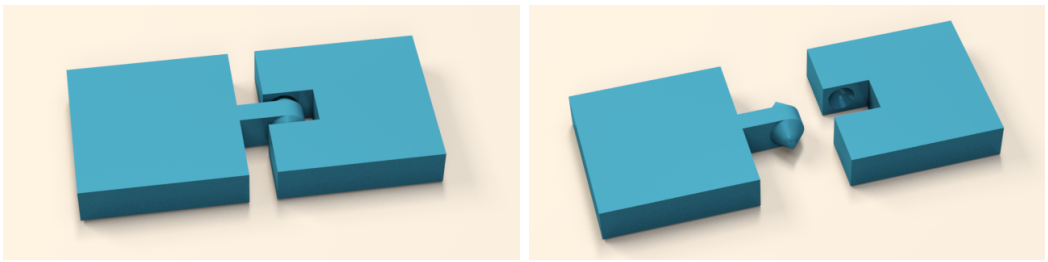
A first set of methods decompose a shape into smaller parts, for later assembly [26]. Other methods additionally consider how to pack the parts together for printing [48, 5, 1]. Wang et al. [50] decompose objects into smaller parts for maximizing print quality, changing the print direction of each sub-part. Finally, Song et al. [42] decompose large objects into small pieces that are 3D printed and then fixed to an internal, laser-cut structure.

The tile packings generated by our approach are not an ideal input for the aforementioned methods, as cross sections are thin everywhere. By proposing a method specifically tailored to our outputs, we are able to introduce rotational degrees of freedoms between tiles and can fully unfold large patches, maximizing print efficiency.

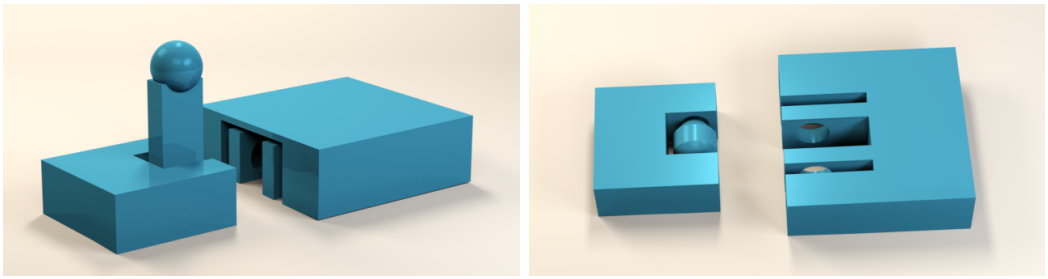
3.3 Overview

Our approach starts from a set of user-specified irregular tiles, and a target surface represented by a triangular mesh $\mathcal{S}$. The output is a set of foldable patches that could be printed flat without support. Each is made of interconnected tiles and can be assembled with other patches into an approximation of $\mathcal{S}$ (see Figure 3.12).

We refer to the surface assembled from the synthesized tile patches as the *tile network* (as illustrated in Figure 3.8a). The tile network is optimized to approximate the target surface with minimal error. The user can optionally specified a desired tile



(a) Top view, attached **(b)** Top view, detached



(c) Top view **(d)** Bottom view

Figure 3.1 (a,b) Hinge joint. (c,d) Snap-fit joint. Note how the connector of snap-fit joints is printed vertically.

scale along the surface. Orientation is not controllable as it is optimized for better approximation of $\mathcal{S}$.

Fabrication Constraint. In order to connect adjacent tiles, parts of the tiles need to be carved out so as to insert joints (as shown in Figure 3.1). The tiles may contain narrow features and concavities. Therefore we take care to position the tiles such that they face each others along sufficiently large areas, allowing for the joints to be inserted. Please note that our approach does not support tiles that are thin everywhere, e.g. thin wire–like structures, as they offer no space for connectors.

3.3.1 Pipeline overview

Our framework mainly consists of two steps: *tile packing* and *patch extraction*. During tile packing, tiles are densely packed over the base surface. The initial surface fits entirely below the output of our algorithm, so that the result can cover it. Each tile is aligned with the tangent plane passing through its centroid.

Unlike a conventional packing problem which solely focuses on maximal surface coverage [13], we seek to produce a dense packing of irregular tiles that 1) satisfies fabrication constraints and 2) follows the target surface as closely as possible. This results in a complex optimization problem, with both continuous and combinatorial

aspects. We therefore propose a dedicated optimization that can jointly optimize the positions, scales and orientations of the tiles in order to meet fabrication constraints, maximized approximation and packing objectives. We detail the tile packing in Section 3.4.

After packing, we divide the tile network into several flat, foldable patches. We resort on two types of joints for connecting the tiles: hinge joints (Figure 3.1a), which allow rotations between two connected tiles, keeping their inter-distance constant; and snap-fit hinge joints (Figure 3.1c), that are printed disconnected and later assembled to connect tiles within and across patches. For the sake of clarity, we refer to the later type of joints simply as snap-fits. After assembly, hinges and snap-fits result in distance constraints between the tiles, locking the assembled patches into a stable surface in most cases. The details of patch extraction are discussed in Section 3.5.

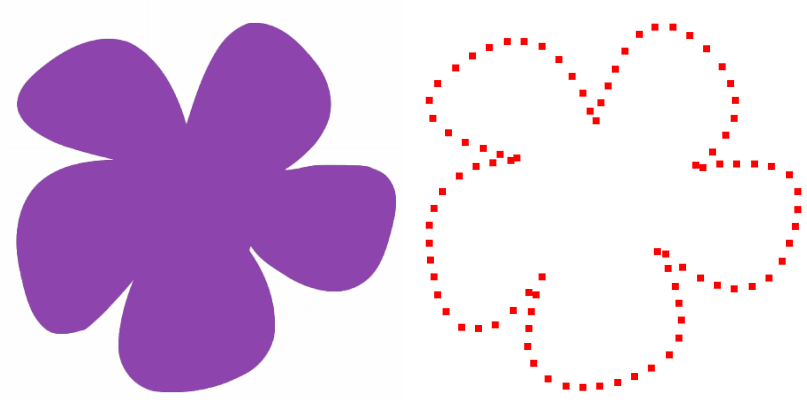
3.4 Tile Packing

In this section, we detail our approach for tile packing. Dense packings of irregular flat tiles (mosaicking) cannot be optimized via traditional gradient based approaches [13]. Compared to mosaicking, we also face additional constraints: fabrication imposes the use of fewer tiles that are relatively large compared to surface curvature (this stems from limitations in minimal printable feature size and maximum object size). This makes packing even more difficult, especially as the tiles remain flat and rigid. We also have to ensure that the tiles are neighboring in a way that allows the insertion of hinges and snap-fits.

Our approach is based on a two-phase *attract-and-repulse* mechanism that iteratively refines an initial layout towards our goal. During the *attraction* phase adjacent tiles are attracting each others to form tangent contacts among themselves, while tiles are added to cover the surface. During the *repulsion* phase, each tile is encouraged to repel its neighbors and adaptively scale its size until a uniform inter-spacing among tiles is achieved.

3.4.1 Tile representation

We input tiles as 2D closed boundary polygons in the XY plane, with the origin at their centroid. During optimization, we only consider a sampling of the contours, as shown in Figure 3.2b. We use two different sampling for performance reasons: the finest is a sampling with a spacing of 2 mm, the coarsest is a sampling with a constant



(a) Input tile **(b)** Sampled contour points

Figure 3.2 The tiles are represented by samples along their contour.

number of 20 samples per tile. The 3D geometry of the tiles is reconstructed during post-processing. The tiles may only be uniformly scaled during optimization, and are otherwise kept rigid.

3.4.2 Initialization

We initialize the tile distribution following multi-class blue noise sampling [52]. New tiles are placed around the boundary of existing tiles, in a process that resembles dart-throwing. We accept a new tile position if it satisfies both of the following criteria: (1) it does slightly overlap with existing tiles; and (2) the overlapping ratio with each existing tile is below a given threshold. This encourages a dense initialization – overlaps are resolved later. We also optimize the orientation of each newly positioned tiles so as to increase the possibility of placing hinges between neighbors. We postpone the discussion of this optimization to Section 3.4.3.2.

When using different classes of tiles, we found it necessary to explicitly encourage mixing. Specifically, for tiles belonging to different classes the overlap area should not exceed 20% of the area of the tiles, while the threshold is reduced to 5% for tiles of the same class. This encourages tiles from different classes to be neighbors while prohibiting tiles of a same class from staying too close, which is demonstrated in Figure 3.6a.

This process is repeated until a maximum number of trials has been reached (1500 in our implementation).

3.4.3 Attraction phase

This first phase distributes and optimizes the position of the tiles to bring them closer together. The variables are the number of tiles, the position of their centroid along the surface, their size (uniform scaling), and their orientation (angle around normal).

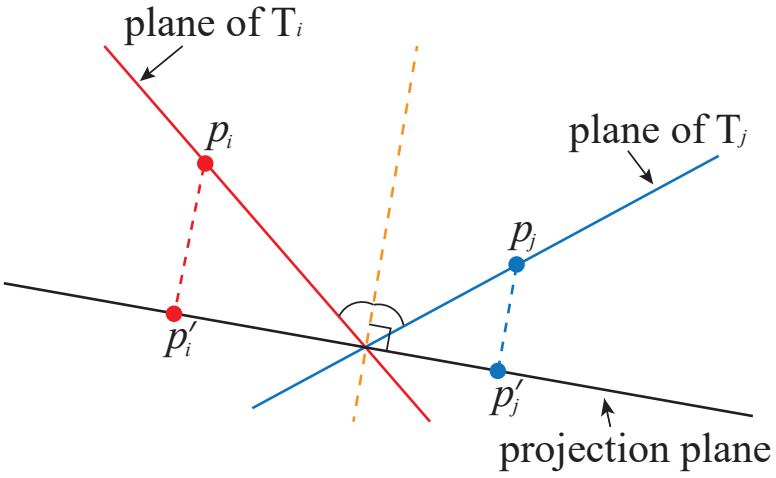


Figure 3.3 Common projection plane of T_i and T_j (side view).

3.4.3.1 Objective function

The objective function is a combination of three terms: neighborhood distance, local surface approximation and orientation.

Neighborhood distance. We define the set of direct neighbors $\mathcal{N}_i$ of a tile T_i as the set of tiles that are overlapping T_i, e.g. $\mathcal{N}_i = \{T_j | T_i \cap T_j \neq \emptyset, j \neq i\}$. The intersection is tested after projecting each tile to a common projection plane, which is orthogonal to the bisector plane of the support planes of T_i and T_j (dashed line in Figure 3.3).

We define the neighborhood distance between a tile T_i and its neighborhood $\mathcal{N}_i$ as:

$$\text{dist}(T_i, \mathcal{N}_i) = \max_{p_j \in \mathcal{P}_{\mathcal{N}_i}} (\text{dist}(p_j, T_i)) \tag{3.1}$$

where $\mathcal{P}_{\mathcal{N}_i}$ is the point set formed by the union of all the sample points of the tiles in $\mathcal{N}_i$ that lie inside T_i (green dots in Figure 3.4). $\text{dist}(p_j, T_i)$ returns the shortest distance of point p_j to the boundary of T_i. Since T_i and the tiles in $\mathcal{N}_i$ are not

coplanar, we project the points in $\mathcal{P}_{\mathcal{N}_i}$ onto the plane of T_i to perform all the distance computations. $\text{dist}(T_i, \mathcal{N}_i)$ evaluates to a large value if $\mathcal{N}_i = \emptyset$ (i.e. T_i has no overlaps).

Intuitively, minimizing $\text{dist}(T_i, \mathcal{N}_i)$ encourages T_i to be attracted to adjacent tiles while preserving a minimum overlap (tangential contact) when tiles come together.

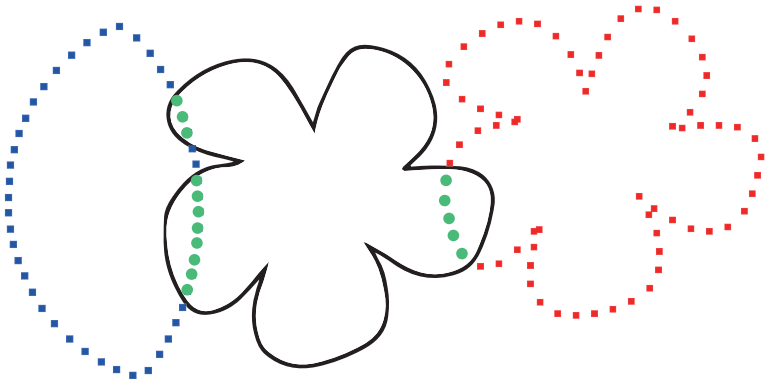


Figure 3.4 Samples for the neighborhood distance (green dots).

Local surface approximation. The tiles are aligned with the tangent plane of the base surface $\mathcal{S}$ at their centroid. In high-curvature regions this quickly leads to large deviations from $\mathcal{S}$. We penalize such configurations by defining the following approximation error for a tile T_i:

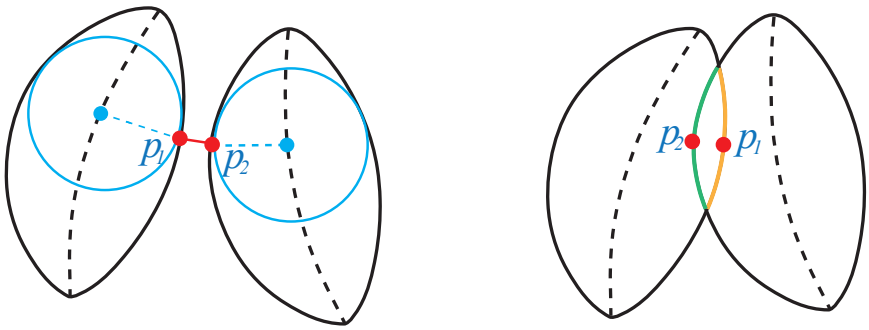
$$\text{approx}(T_i, \mathcal{S}) = \max_{p_i \in T_i} (\text{dist}(p_i, \mathcal{S})) \tag{3.2}$$

where $\text{dist}(p_i, \mathcal{S})$ returns the (Euclidian) distance from point p_i to the closest point on $\mathcal{S}$. Low values of $\text{approx}(T_i, \mathcal{S})$ indicate a better local approximation of $\mathcal{S}$ by T_i.

Orientation. Tile orientation is important both for the surface approximation and for fabrication. In particular, this will determine where the hinges can be added between neighboring tiles.

The capacity of holding an hinge is measured using the *local shape thickness* – a notion similar to the shape diameter defined in [39]. For a 2D shape, we define the shape thickness at a boundary point p as the diameter of the maximal ball centered at the medial axis and tangential to p (as shown in Figure 3.5a). Larger values indicate that it is more likely that an hinge can be placed at this location.

Whenever considering adjacent tiles (overlapped or not), we would like them to be oriented such that the potential connection points have large local shape thicknesses.



(a) Non-overlapping tiles **(b)** Overlapping tiles

Figure 3.5 Potential joint positions. (a) Local shape thickness. The medial axis of each shape is outlined by a dashed line. The medial balls at p_1 and p_2 are drawn in blue. (b) In case of overlap p_1 and p_2 are the midpoints of the orange and green segment, respectively.

For non-overlapping tiles (Figure 3.5a), the connection points to place an hinge are the two nearest points between the tile contours. As tiles may be overlapping during optimization, we predict the connection points of such cases as the middle point of the curve segment contained in the other tile (Figure 3.5b). We denote the orientation score for a tile T_i as $\Theta(T_i, \mathcal{N}_i)$: it is computed as the sum of the local shape thicknesses for all potential contact points between T_i and its neighbors.

Objective function. We finally define the global attraction objective function $E_{attract}$:

$$E_{attract} = \alpha \sum_i \text{dist}(T_i, \mathcal{N}_i) + \beta \sum_i \text{approx}(T_i, \mathcal{S}) - \Theta(T_i, \mathcal{N}_i) \tag{3.3}$$

$E_{attract}$ is a weighted sum of the distance, local approximation and orientation terms, where α and β are controlling the tradeoff. We set $\alpha = 1.0$ and $\beta = 1.5$ in our implementation. β is set larger than α to penalize positions in high-curvature regions.

During the first phase, our goal is to find a tile configuration with the lowest value of $E_{attract}$.

3.4.3.2 Minimization

$E_{attract}$ is a non-linear, combinatorial objective function. We therefore rely on a greedy strategy to locally optimize the configuration of each tile. The pseudo-code for minimization is detailed in Algorithm 3. It takes the following steps:

Position update. In PositionUpdate each tile is translated to a number of candidate positions (including the current position), searching for a displacement producing a lower value of $E_{attract}$. The candidate positions are randomly sampled within the minimal circle centered on the centroid and fully enclosing the tile. We test 400 positions in our implementation. The position with lowest value is used to update the tile in the current iteration. Note that if the user provided a scale control field, the tile is resized before computing $E_{attract}$ at each tested position.

In regions of high-curvature it is likely that no good update can be found, as the tile plane misaligns with the surface. When all the candidate positions for a tile T_i produce an approximation error $\text{approx}(T_i, \mathcal{S})$ greater than a threshold τ_{approx}, the tile is shrunk at its current position. This favors smaller tiles in high curvature regions. A minimum size constraint prevents tiles from becoming too small. τ_{approx} is determined before starting the attraction phase. The approximation errors of all tiles are computed and sorted. τ_{approx} is the average error of the top 20%.

Angle update. Function AngleUpdate optimizes the orientation of each tile. Each tile is tested with different orientations while its centroid remains fixed. Among the 360 tested rotation angles, we consider the top n with the lowest value of $E_{attract}$ as the candidate orientations ($n = 10$ in our implementation). Among these, we select the angle that provides the best opportunity to insert an hinge, that is the angle that maximizes the sum of local shape thicknesses between the tile and its neighbors.

Updating sequence. We process tiles in parallel, updating the tile positions in independent sets similarly to the mechanism in [4].

Increasing the surface coverage. The minimization of $E_{attract}$ leads to a compact packing of existing tiles, uncovering other parts of the surface. Therefore, after position and orientation optimization we add tiles to uncovered regions in AddMoreTiles. This is done similarly to the initialization. The attraction phase stops if no more tiles can be added.

3.4.4 Repulsion phase

The second phase of the tile packing focuses on resolving overlaps that remain after the attraction phase. In particular, we adaptively scale each tile according to its distance to adjacent tiles, and encourage repulsion among neighboring tiles to evenly distribute tile inter-spacing. A target gutter size d is given as input to the algorithm.

Algorithm 3 ATTRACTION

Require:
Target Surface $\mathcal{S}$; Initial placement of tile set $\mathcal{T}$ on S; Input scale field $\mathcal{F}$ on $\mathcal{S}$.
Ensure:
Optimized tile configuration $\mathcal{T}_O$ that follows $\mathcal{F}$ on S.

```
while true do
    T ← PositionUpdate(T, F);
    T ← AngleUpdate(T);
    stop ←AddMoreTiles();
    if stop then
        break;
    end if
end while
return T_O = T;
```

(a) Initialization **(b)** Attraction phase **(c)** Repulsion phase

Figure 3.6 Intermediate results in different steps of our algorithm.

Our algorithm iterates until there is no overlap between any two tiles and the minimal gutter distance is larger than or equal to d (1 mm in our implementation).

3.4.4.1 Objective function

We first define a repulsion term between two tiles T_i and T_j:

$$\text{repulse}(T_i, T_j) = \min\{\min_{p_i \in T_i} \text{sdist}(p_i, T_j), \min_{p_j \in T_j} \text{sdist}(p_j, T_i)\} \tag{3.4}$$

where $\text{sdist}(p_i, T_j)$ is the *signed* closest distance from point p_i to the point set sampled from T_j's contour:

$$\text{sdist}(p_i, T_j) = \begin{cases} -\min_{p_j \in T_j} \|\Gamma(p_i) - \Gamma(p_j)\| & \text{if } p_i \text{ inside } T_j \\ \min_{p_j \in T_j} \|\Gamma(p_i) - \Gamma(p_j)\| & \text{otherwise} \end{cases} \tag{3.5}$$

where $\Gamma(p)$ projects the point onto the common projection plane of T_i and T_j, as illustrated in Figure 3.3.

The sign of $\text{repulse}(T_i, T_j)$ indicates whether T_i and T_j overlap with each other (negative values imply an overlap). The absolute value of repulse measures how deeply T_i and T_j penetrate each other if they overlap, or how far they are separated from each other if they do not.

During the repulsion phase we seek to maximize the value of repulse between each tile T_i and its *adjacent neighbors* $\mathcal{A}_i$. Note that this neighborhood is different from the one used during the attraction phase, denoted by $\mathcal{N}_i$ (Section 3.4.3.1). $\mathcal{A}_i$ is defined as $\mathcal{A}_i = \{T_j | \text{repulse}(T_i, T_j) < \sigma, j \neq i\}$, that is, the set of tiles closer to T_i than a threshold distance σ. This is illustrated in Figure 3.7 ($\sigma = 6$ mm in our implementation).

We then define the final objective function as:

$$E_{repulse} = \sum_i \min_{T_j \in \mathcal{A}_i} \text{repulse}(T_i, T_j) \tag{3.6}$$

Maximizing this objective increases the distance between each tile and its adjacent neighbors.

3.4.4.2 Maximization

We resort on a similar strategy as in Section 3.4.3.2 to maximize $E_{repulse}$. Algorithm 5 details the procedure, which executes the following steps:

Scale update. To eliminate gaps and overlaps between tiles we adaptively scale them according to the distances to their adjacent neighbors. The pseudo code for ScaleUpdate is given in Algorithm 4. We measure the size of a tile as the diameter of the smallest enclosing circle centered on the centroid.

ComputeDistance returns d_i, the smallest value of repulse (Equation 3.4) between a tile and its adjacent neighbors. Assuming that the current size of T_i is s_i, we compute in ComputeNewScale the expected size of T_i in the next iteration as

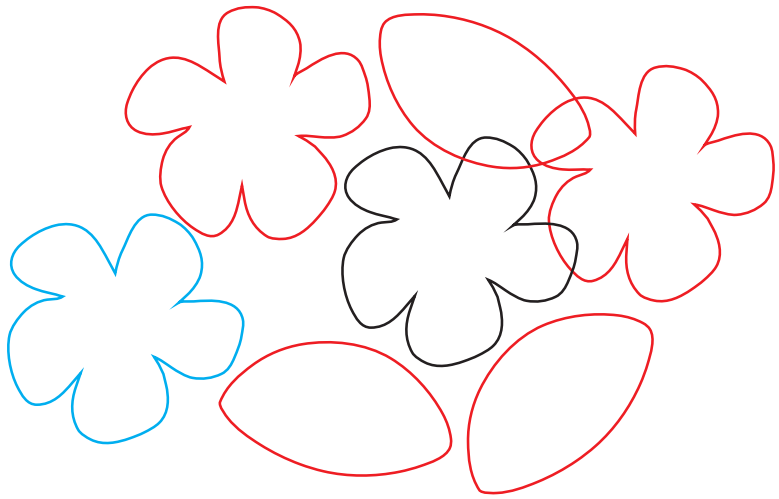


Figure 3.7 The adjacent neighborhood (in red) of the black tile are all the tiles surrounding it while being closer than a threshold.

$k_i = s_i + (d_i - d)/2.0$, where d is the target gutter distance. We divide $(d_i - d)$ by two as T_i and its nearest neighbor change in size simultaneously. After the expected size is computed for all tiles, ScaleToExpectedSizes applies the actual scaling operations.

Algorithm 4 ScaleUpdate

Require:
Input tile layout $\mathcal{T}$ on S, target interspacing distance d;
Ensure:
Updated tile layout $\mathcal{T}_O$ with size adaptively scaled.
$K := \emptyset$; // *vector that stores expected size of each tile*
for each tile $T_i \in \mathcal{T}$ **do**
  $\mathcal{A}_i \leftarrow$ FindAdjNeighbors(T_i);
  $d_i \leftarrow$ ComputeDistance(T_i, $\mathcal{A}_i$);
  $k_i \leftarrow$ ComputeNewScale(T_i, d_i);
  $K \leftarrow K \cup \{k_i\}$;
end for
$\mathcal{T}_O \leftarrow$ ScaleToExpectedSizes($\mathcal{T}$, K);
return $\mathcal{T}_O$;

Position and orientation update. We follow a similar strategy as in Algorithm 3 to locally update the positions and orientations of the tiles. However, during each iteration, we seek to search for updates that *maximize* Equation 3.6.

Termination. At each iteration we track the minimum distance between closest neighbors d_{min}. If this value is larger than or equal to the target gutter distance d the process terminates – all tiles are far enough from their neighbors. Otherwise, the iterations continue up to a maximum value. The overall process is illustrated in Figure 3.6c.

While there is no strict guarantee that the process converges, we never observed cases where the maximum number of iterations is reached: after initialization and attraction phase there is only a limited amount of overlap, and these can be resolved without having to apply excessive scaling to the tiles.

Algorithm 5 REPULSION

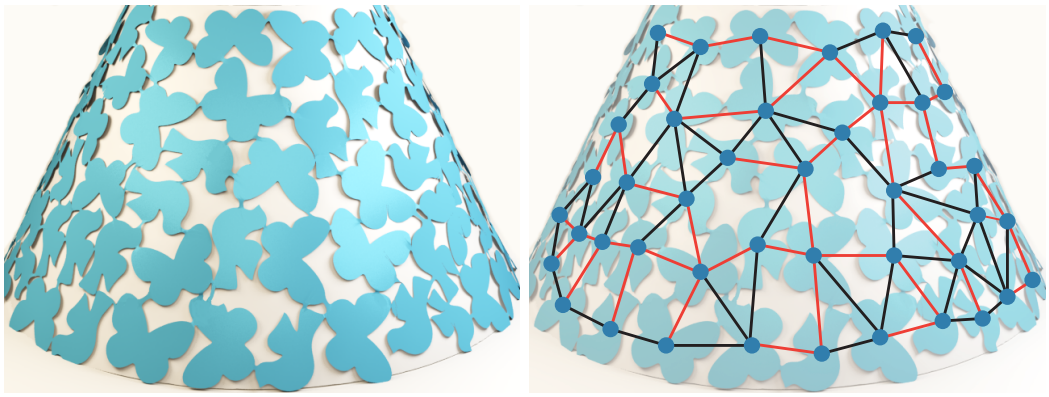
Require:
Target Surface $\mathcal{S}$; Input tile distribution $\mathcal{T}$ on S; Target interspacing distance d; Input magnitude field $\mathcal{F}$ on $\mathcal{S}$.

Ensure:
Optimized tile configuration $\mathcal{T}_O$ that follows $\mathcal{F}$ on S.

while *true* **do**
  $\mathcal{T} \leftarrow$ SCALEUPDATE($\mathcal{T}$);
  $\mathcal{T} \leftarrow$ POSITIONUPDATE($\mathcal{T}$, $\mathcal{F}$);
  $\mathcal{T} \leftarrow$ ANGLEUPDATE($\mathcal{T}$);
  $d_{min} \leftarrow$CHECKDISTANCE();
  if $d_{min} \geq d$ **or** maximum iterations reached **then**
    break;
  end if
end while
return $\mathcal{T}_O = \mathcal{T}$;

3.5 Patch Extraction

We now consider the problem of connecting tiles with hinges and snap-fit joints, thus forming patches that can be fabricated flat, folded and assembled to form the final 3D shape. The set of potential hinges are the connections between a tile T_i and its adjacent neighbors in $\mathcal{A}_i$ (see Section 3.4.4.1 and Figure 3.7). For two neighboring tiles, the potential location of a hinge is in-between the two closest points along the contours. We use the neighborhood information to create a graph G that captures the tile network. Each tile T_i is a node, and one edge is added for each adjacent neighbors T_j in $\mathcal{A}_i$ (see Figure 3.8b). This graph is very densely connected since the tiles are packed.



(a) Tile network **(b)** Tile graph

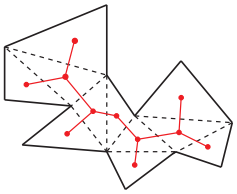
Figure 3.8 Tile network and its corresponding graph. Note that (b) only shows the graph for the tiles that are visible in (a). The red edges in (b) form a spanning tree of the graph.

This section focuses on two questions: (1) how to segment the tile network into foldable patches (Sections 3.5.1 and 3.5.2) and (2) how to assign and optimize the hinge placement so that the final assembled printout is fabricable and possibly stable (Section 3.5.3).

What we mean by *stable*, is that the tiles are all locked in place with respect to one another. This possibility stems from the fact that taken all together the joints create a dense set of distance constraints between the rigid tiles. These constraints hold the tiles into a stable configuration – a property which is expected for a closed convex polyhedron through Cauchy's rigidity theorem, but may not generally be true (e.g. a flat surface). Our algorithm attempts to preserve this property while trying to reduce the number of joints (Section 3.5.3.1). Note that if the shape is not stable even when inserting all joints, the final result can still be fabricated but will exhibit unconstrained degrees of freedom.

3.5.1 Extracting foldable patches

Let us consider a case where the tile graph G is a tree – it is easy to observe that it could be unfolded: Each hinge separates the graph in two distinct parts which can rotate freely around the hinge axis with respect to one another. This notion is similar to the foldability of the *triangulation dual*, where triangles are nodes and edges are hinges [31], as illustrated in the inset. The unfolding might however produce overlaps in the plane.



We build upon this idea and formulate patch extraction as a *graph partitioning* problem: we seek to partition the graph G into a set of spanning trees such that each tree can be unfolded.

3.5.2 Graph partitioning

We now describe how to partition the graph G (Figure 3.8b) into a set of spanning trees. There are two fabrication requirements for the graph partitioning problem. First, the size of the unfolded geometry of each patch cannot exceed the extent of the printer bed; second, the unfolding should not produce overlaps between tiles. We propose a method to grow each patch in sequence while considering the properties of its unfolded geometry.

We decorate the graph with edge weights. Each edge weight indicates the capability $c(i, j)$ to insert a hinge between T_i and T_j at two points p_i and p_j as described in Section 3.4.3.2 and Figure 3.5. $c(i, j)$ is equal to the average local shape thickness.

In practice the hinges require smaller footprints than the snap-fits (see Figure 3.1). Therefore, we encourage edges with lower $c(i, j)$ to become hinges. We thus seek to extract a *minimal spanning tree* from G such that the edges with lower $c(i, j)$ belong to the tree.

In the following, whenever selecting an edge to become a hinge (or snap-fit) we first check whether the value of $c(i, j)$ is large enough to host the joint, and ignore the edge otherwise. Note that most edges can host a joint: the packing algorithm specifically optimizes tile orientations to achieve this.

We grow patches with the following process. We start from a tile selected randomly. We then locally grow a subgraph by breadth-first expansion. At each step of the growth, we extract a minimal spanning tree and verify whether the tree unfolding is valid, in which case we continue growing the subgraph. The unfolding is valid as long as it does not produce an overlap between tiles and it fits the printer bed size. If the unfolding is not valid, we return the last valid spanning tree as the next patch, and start extracting a new patch. This is done until all tiles have been covered.

3.5.3 Snap-fits optimization

After patch extraction the set of hinges is fully determined (edges of the spanning trees). However, the number and placement of snap-fit joints is not fixed yet. This affects both the model stability and its printability.

Ideally, we would like to minimize the number of snap-fits: they have to be manually assembled, and using too many is likely to result in non-printable configurations as they cannot be hosted in the tiles. However, each snap-fit introduces a new distance constraint in the final assembly, working towards locking the result in a stable configuration. We thus seek for a proper amount of snap-fit joints so that the final printout is both fabricable and stable.

Note that some tile networks simply cannot be made stable: this depends on the input surface properties (see discussion in the introduction of Section 3.5). In this section we assume that the tile network is stable if all possibly snap-fit joints are inserted – thus, our objective during the snap-fit joint selection process is to preserve this property. If a surface cannot be made stable, all possible snap-fit joints will be inserted.

3.5.3.1 Snap-fit edges selection

A necessary condition for the assembled printout to lock in a stable configuration is to have loops in the connection graph – otherwise any leaf tile would be able to rotate freely along its hinge. In addition, smaller loops increase the set of constraints, further reducing the number of degrees of freedom in the assembly. Based on these simple observations, we optimize the selection of snap-fits to ensure each tile is captured inside constraint loops of small size.

During the algorithm we check the stability of the assembly by using a physics simulation (based on *Bullet Physics*), verifying whether the tiles remain fixed in space under the effect of gravity. The lowest point of the object is fixed to the ground as we are not interested in checking for balance. If the largest displacement exceeds 5 mm we consider the model unstable.

Algorithm 6 details the process. The input consists of the graph G and the set of edges that are already selected as hinges. The remaining unselected edges become a pool for selecting snap-fits, see Figure 3.9a. We filter out any edge in which tiles could not host a snap-fit. We refer to edges that could become snap-fits as *available*.

The first step in Algorithm 6 is to construct initial loops so that each tile node is captured within a certain cycle. This is implemented in ConnectLeafNodes. We search all leaf nodes (degree 1) and add an additional available edge to them, favoring closest neighbors. As the graph is densely connected, and as the packing is optimized to maximize the possibility to place joints, there is a very high likelihood that such an edge exists. This is illustrated by the green dashed segments in Figure 3.9b.

Algorithm 6 SelectSnapFits

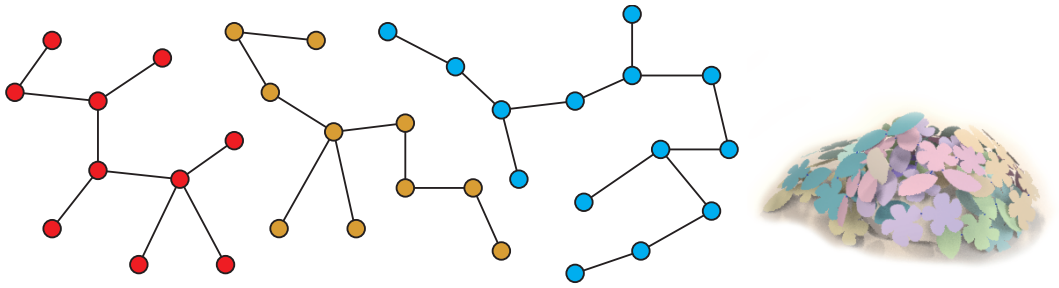
Require:
Graph G; A set of hinge edges E; Snap-fit hinge pool P;
Ensure:
Result stability f_{stb}; A set of edges F in G that will be realized by snap-fits.

```
1: k ← 15; f_stb ← true;
2: G' ← ConnectLeafNodes(E);
3: while true do
4:   {C_i} ←DetectMinimalCycles(G');
5:   (F, G') ← EnhanceLargeCycles({C_i}, k, P);
6:   stable ← TestStability(F, E);
7:   if stable then
8:     break;
9:   else
10:    if F == P then
11:      f_stb ← false;
12:      break;
13:    else
14:      k − −;
15:    end if
16:  end if
17: end while
    return (f_stb, F);
```

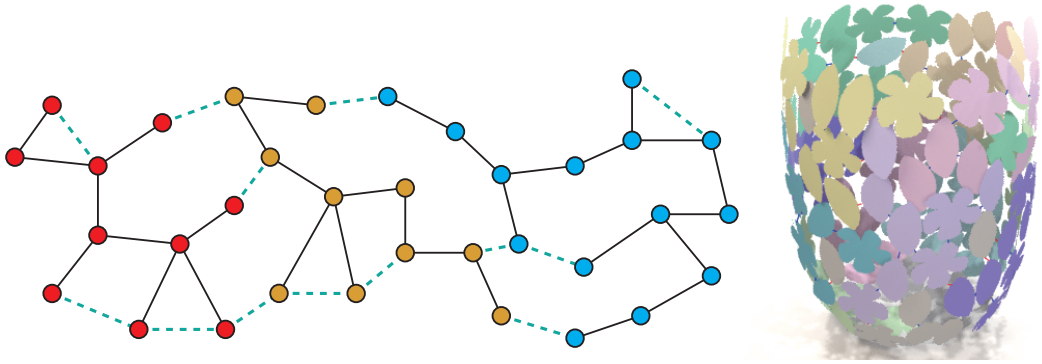
After removing leaf nodes, all tiles belong to cycles. The next step is to achieve a denser connectivity. DetectMinimalCycles detects the minimal cycles $\{\mathcal{C}_i\}$. The value of k determines the maximal accepted length for the cycles. If a cycle $\mathcal{C}_i$ is longer than k, more edges are added to it by searching for available edges connecting pairs of nodes in the cycle, favoring closest neighbors. Thanks to the high degree of connectivity in G many such choices exist. This process, performed by EnhanceLargeCycles, is illustrated in Figures 3.9c and 3.9d. Finally, TestStability checks whether the model is stable with the current set of snap-fits. If not, the value of k is decreased and another iteration adds more snap-fits. At worst the algorithm terminates when all available snap-fits are added.

3.5.3.2 Final joint assignments

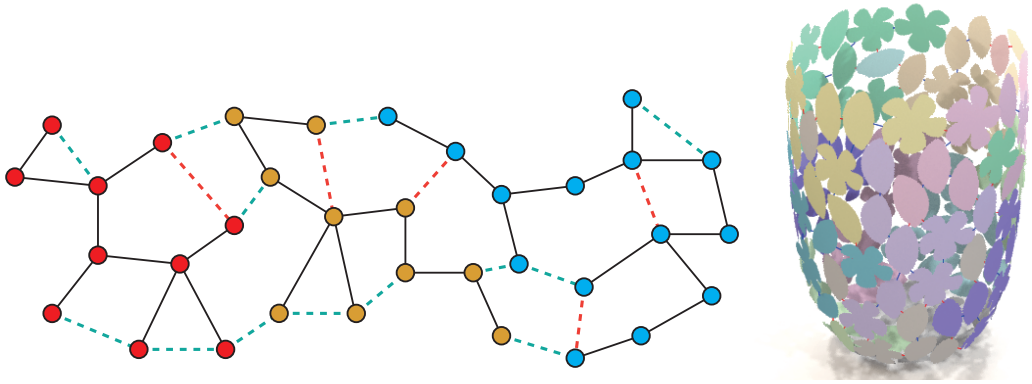
After selecting the hinges and snap-fit joints, there might exist overlaps between their geometries (see Figure 3.10a). Hinges require space only for one of their ends (the other simply protrudes out of the tile). Snap-fits require space on both ends,



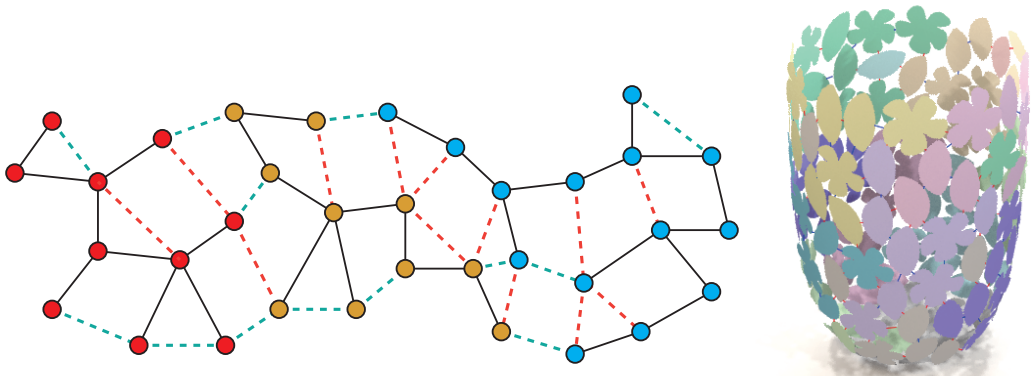
(a) Input spanning trees



(b) After connecting leaf nodes

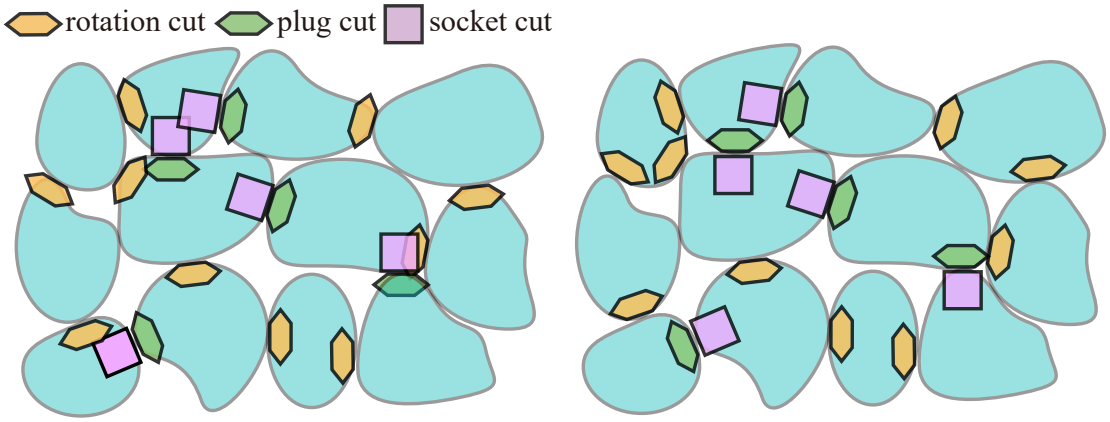


(c) After adding snap-fits, $k = 6$



(d) After adding snap-fits, $k = 4$

Figure 3.9 Adding snap-fit joints. The left column illustrates the graph, the right column shows the physical simulation as more snap-fits are added.



(a) Before optimization **(b)** After optimization

Figure 3.10 Effect of joint assignment optimization.

one being slightly larger as shown in Figure 3.1d. This results in a combinatorial problem where we attempt to resolve for all conflicts, while swapping the joint ends assignment.

We proceed in two steps described in Algorithm 7. First, the joint ends are evenly distributed by function EvenlyDistribute. We define the "load" of a tile as the ratio between the number of joint ends it hosts and its area. We distribute the joint ends while attempting to achieve an even load across all tiles. We process the tiles in a priority queue in order of decreasing load value. Each time a tile is visited, the neighboring tiles are checked to assign the joint ends such that the load is kept minimal.

This initial assignment might however create conflicts. To resolve these we process the tiles in a priority queue by decreasing order of number of conflicts. A conflict can be removed in two ways in function ResolveConflicts: the first attempted is to swap the ends of a joint, the second is to slightly move the joint attachment along the tile boundary. These changes can produce a conflict on a neighboring tile, and we therefore add any new conflict to the queue. In rare cases the conflicts cannot be resolved: this is detected whenever neither swapping nor moving the attachment works, or when a new conflict is produced on an already processed tile. In such a case we cancel the edge – a situation that occurred only on one edge in all our experiments (Lamp in Figure 3.11s). Figure 3.10 shows the effect of joint assignment optimization.

Algorithm 7 DistributeJoints

Require:
Initial placement of joints $\mathcal{H}$; graph G;
Ensure:
An optimized distribution of joints $\mathcal{H}_{out}$.
1: $\mathcal{H}' \leftarrow$ EvenlyDistribute($\mathcal{H}$, G);
2: $\mathcal{H}_{out} \leftarrow$ ResolveConflicts($\mathcal{H}'$);
3: **return** $\mathcal{H}_{out}$;

3.6 Implementation and Fabrication

This section gives details regarding implementation and fabrication.

3.6.1 3D model generation

To generate model for fabrication, we first unfold each patch into a set of 2D contours. The 3D model is constructed via adding a thickness to the 2D pattern. Joints are embedded into the model via boolean operations [49, 16].

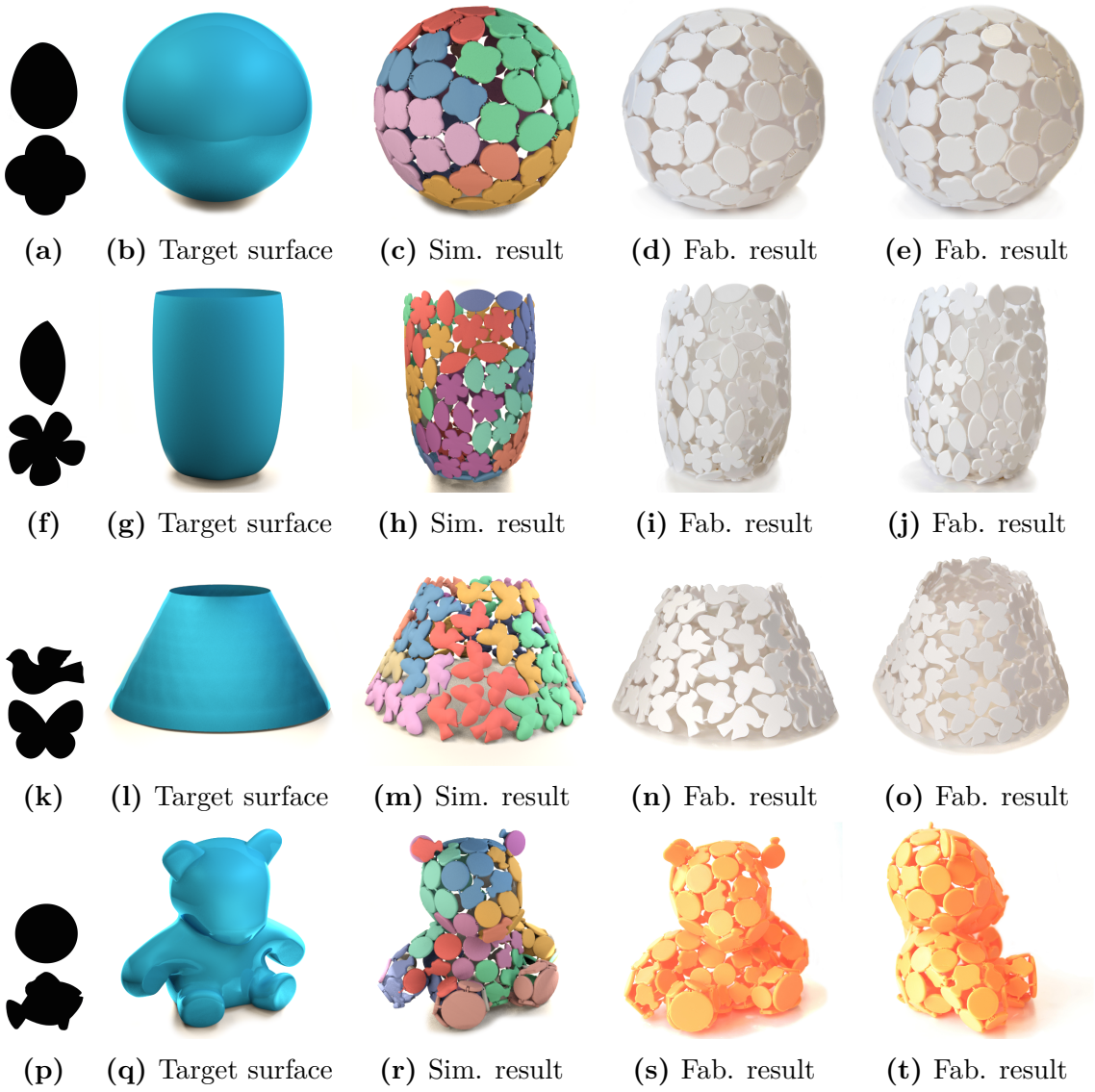
Snap-fits internal to a patch are printed vertically – which avoids having to consider potential collisions. Other snap-fits (across patches) are printed horizontally whenever possible, such that the part inserted into the tile is hidden from view (this can be seen on the printed patches in the second column of Figure 3.12).

3.6.2 Optimizations

We achieve better performance by using a coarser sampling. In particular, the attraction phase resorts on a coarse sampling of the tiles (20 samples per tile), followed by a first repulsion phase using a coarse sampling. After this point the packing is almost finalized, and we perform a final repulsion phase using the finest sampling (2 mm spacing). This second repulsion phase terminates quickly as only small overlaps – missed by the coarse pass – are resolved. We provide timings in Section 3.7.3.

3.7 Results

We show several results produced with our method in Section 3.7.1, discuss surface packing quality in Section 3.7.2 and timings in Section 3.7.3.



(a) (b) Target surface (c) Sim. result (d) Fab. result (e) Fab. result

(f) (g) Target surface (h) Sim. result (i) Fab. result (j) Fab. result

(k) (l) Target surface (m) Sim. result (n) Fab. result (o) Fab. result

(p) (q) Target surface (r) Sim. result (s) Fab. result (t) Fab. result

Figure 3.11 From left to right: input tiles, target surfaces, simulated results with each patch color coded and fabricated results (two views).

3.7.1 Fabrication

We test our algorithm on a variety of models, from a simple sphere to a teddy bear with high curvatures. Figure 3.12 and Figure 3.11 show our fabrication results and their corresponding simulations. Tiles of various shapes are used to decorate the base surfaces, including round convex shapes (e.g. sphere and egg) and concave contours with thin features (e.g. the bird and fish pattern). We fabricate all the results using a filament printer (Flash Forge *Creator Pro*) with ABS filament. All of the assembled models correspond to their simulation and approximate the target surface well. Some

Figure 3.12 Given a target surface and a set of user-specified tiles (top left), our method produces a dedicated packing of tiles that is optimized for fabrication (bottom left). The results are printed as independent flat patches (second column) with integrated hinges and snap-fit joints. The patches are folded and assembled into the final object (third column); in this case a functional handbag that can carry light objects.

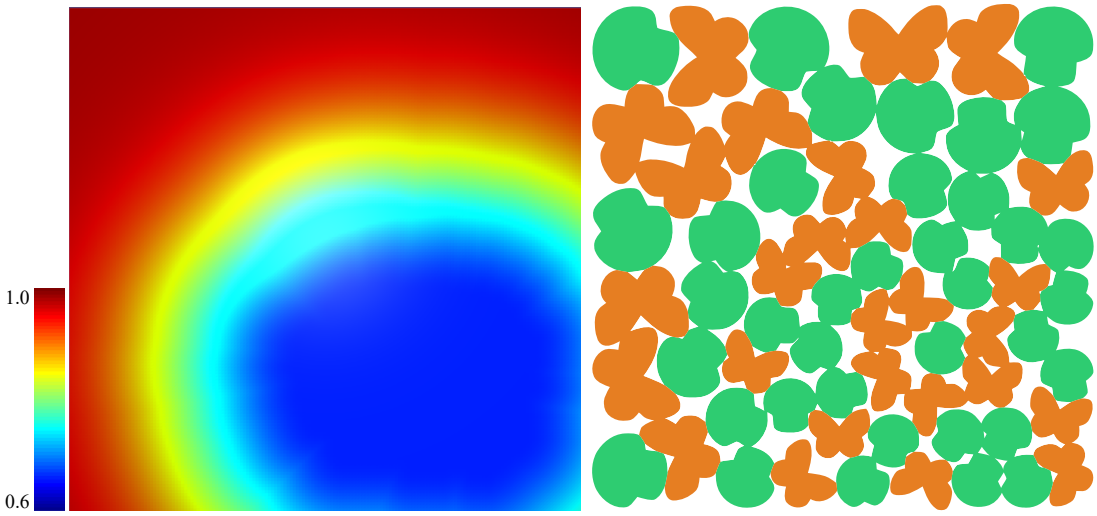
low amount of distortion can be seen, which is essentially due to necessary tolerances when fabricating pre-assembled joints.

Our fabrication results have a variety of uses, for instance acting as lamp shades, vase decor (Figure 3.16) or even as a bag (Figure 3.12) that can carry light objects. On the bag, two ring tiles are manually placed and fixed during optimization in order to attach the handles.

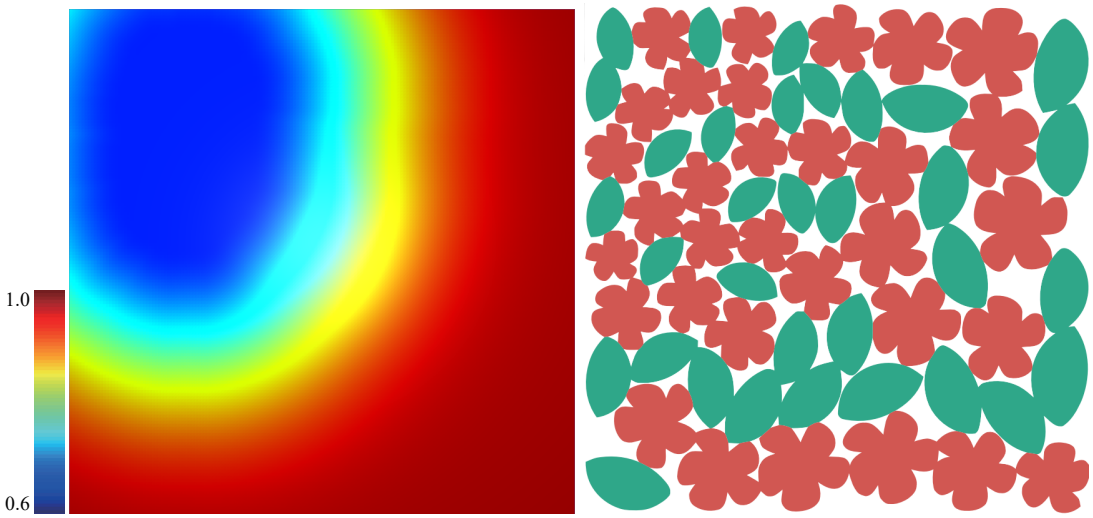
Table 3.1 summarizes the statistics for each fabricated result: size of assembled printout, number of tiles and the number of fabricated patches. The number of patches depends on both the model size and the surface complexity. Models with high curvatures require more patches (e.g. teddy bear).

3.7.1.1 Scale editing

Our method allows some user control, in particular over the scale of the tiles. However, due to fabrication constraints and surface approximation the final scale may override the user requests. Scale control is demonstrated on 2D examples in Figure 3.13. Figure 3.15 shows scale control on a surface with a small number of tiles. This makes it more challenging for the algorithm to follow the user intent, for instance tiles in the top left corner have to shrink to resolve overlaps. However, the result still preserves the overall scale trend indicated by the user. The fabricated result is in Figure 3.12.



(a) Input scale field **(b)** Packing result



(c) Input scale field **(d)** Packing result

Figure 3.13 Packing results in 2D controlled by a scale field.

	Sphere Lamp	Vase	Lamp	Handbag	Bear
Size	(17,17,16)	(17,17,27)	(27,27,17)	(25,13,18)	(22,13,21)
#T	102	126	83	122	147
#P	10	13	14	17	22

Table 3.1 Statistics of fabricated results. From top row to bottom are size of assembled printout, number of tiles packed and number of patches used for assembly. The size is represented as (*length*, *width*, *height*), all measured in centimeters.

3.7.2 Surface packing

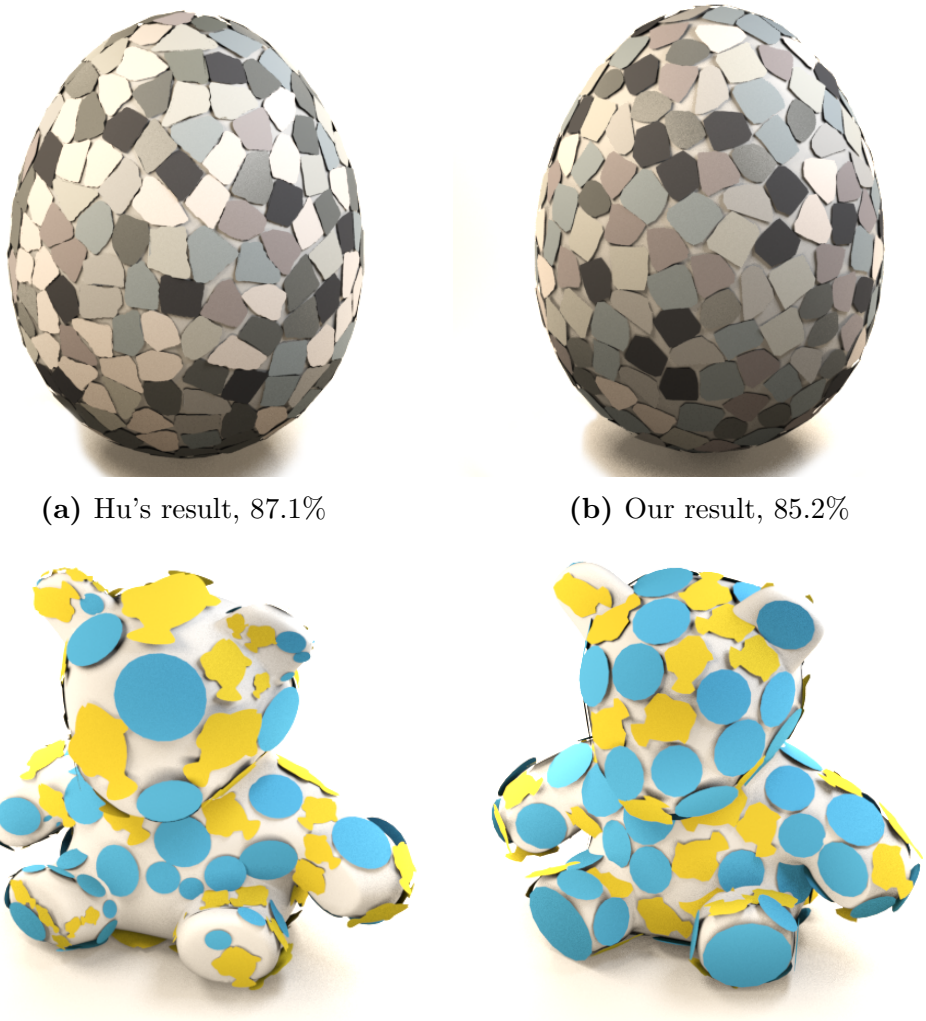
We now consider the quality of our packing result with respect to the recent work of Hu et al. [13]. In particular we want to verify that taking into account the fabrication constraints does not penalize packing quality significantly. We perform two comparisons: the first is based on the inputs from [13] (egg results in Figure 3.14) while another uses one of our test cases (teddy bear). We use the same amount of tiles when comparing both algorithms.

As seen in Figure 3.14, our algorithm produces similar surface coverage (85.2% vs 87.1%) compared to the result found in [13]. Thus, fabrication constraints do not heavily penalize the packing. As expected, our approach performs significantly better on the challenging packing of the teddy bear model (coverage of 77.5% vs 60.1%). Such cases are not the focus of [13] as the method targets mosaicking with relatively small tiles. When this is not the case, [13] tends to generate smaller tiles in high curvature regions and ends up with few tiles to cover smoother domains. This leads to an imbalance of tile scales and leaves gaps along the surface. Our approach achieves a more uniform distribution of scales and deals better with coarser surface approximations. As fabrication constraints are taken into account explicitly (uniform tile spacing and insertion of joints), the result is fabricable.

The results of [13] in Figure 3.14 use the implementation provided by the authors.

3.7.3 Timing

Table 3.2 summarizes the timing of our results. All the results are produced on computer with an Intel i7-4770 CPU and 16GB RAM. The tile packing dominates the runtime. The timing is mostly influenced by the number of tiles and the complexity of the target surface. However, thanks to the coarse sampling optimization, our packing



(a) Hu's result, 87.1%

(b) Our result, 85.2%

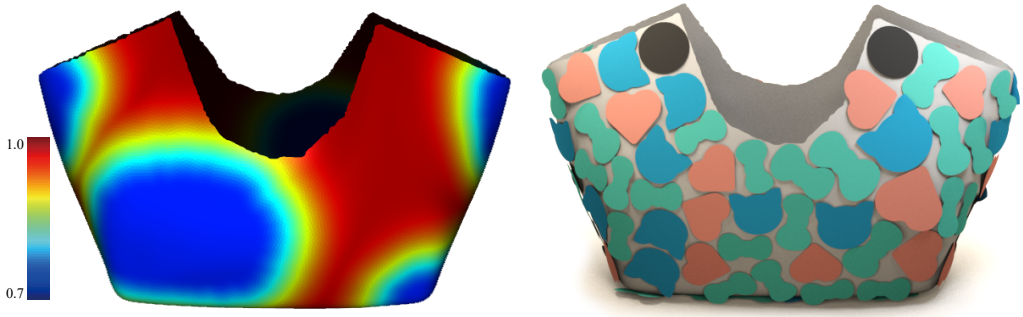
(c) Hu's result, 60.1%

(d) Our result, 77.5%

Figure 3.14 Comparison of surface packing with [*Hu et al.* 2016]. The surface coverage rate of each result is indicated below each image. Results use the same number of tiles (287 for the egg, 147 for the teddy bear).

Time	Sphere Lamp	Vase	Lamp	Handbag	Bear
t_{pack}	8.15	9.23	6.57	11.32	15.16
t_{patch}	0.25	0.27	0.28	0.30	0.35
t_{print}	36.1	52.0	37.1	37.8	42.6
t_{asm}	8.5	13.2	9.1	10.6	18.7

Table 3.2 Timing of each result. For each result, we show the timing of tile packing t_{pack}, patch extraction t_{patch}, printing t_{print} and assembling t_{asm}. All timings are listed in minutes expect t_{print} which is in hours.



(a) Input scale field **(b)** Packing result

Figure 3.15 Packing result on a surface controlled by a scale field.

Figure 3.16 Fabrication results used for home decor.

algorithm is relatively fast compared to [13]. For the bear result in Figure 3.14, our algorithm finished in 15.2 minutes while [13] took 1 hour.

3.8 Conclusions

Our approach lets anyone fabricate visually interesting objects by decorating a surface with tiles. This mimics a popular way of improving a surface appearance by applying stickers and decals. Rather than synthesizing a complex 3D model difficult to print, our technique is designed to allow for efficient fabrication: the final surface is assembled from articulated patches that print flat, without support. This makes them fabricable on home filament printers, and easy to pack which maximizes utilization of powder-based printers, and reduces shipment costs.

While the assembly step is left to the user – and it does take some time – we find the assembly to be an enjoyable process, that gives the user a better sense of ownership on the part she customized. However, it would be interesting, as future work, to attempt to further simplify this stage. As the scale of an input surface grows, the number of patches increases – since the printer bed size limits the maximal extent of a patch. Tolerances required in the fabrication of joints will also accumulate and lead to increasingly larger distortions in the final assembly. Finally, it would be interesting to study whether a final assembly could be made stable by gluing or constraining the motion of a small subset of hinges.

Using our technique, users without prior expertise can model objects that fully exploit advanced possibilities of 3D printing: embedding pre-assembled hinges and snap-fit joints in a model, as well as producing freeform, unusual geometries. We hope our approach will find a wide audience, and we will make the application available for everyone to enjoy.

Chapter 4

Conclusion

In summary, in this thesis we seek to automate the task of designing patterned surface for 3D printing and ease the printing job with high efficiency and low cost. To our best knowledge, little research effort has been put in synthesizing non-stochastic elements over surfaces, not mentioning the constraints of being fabricable. We present two works aiming to synthesizing two different kinds but widely used elements: filigree patterns and tiles. Both the of methodologies have taken into account the fabrication constraints.

In particular, in Chapter 2, we formulate the filigree synthesis problem as a dense packing of filigree elements with both appearance and fabrication constraints. We leverage two properties of filigrees as the key to our solution: first, filigree elements can be well represented by their skeletons; second, we relax the packing problem by allowing the base elements to be partial overlapped in inconspicuously way. A novel objection function based on Pattern Matching Energy (PME) has been proposed for measuring the quality of synthesis. The objective is optimized via a stochastic search strategy with a boosting step that records and reuses good configurations. We ensure the output structure is sufficiently strong by structural optimization, which progressively enhances the structure by adding new elements or locally thickening the weak regions. The method supports flexible user controls that allows scaling and orientating the elements on base surfaces.

The method presented in Chapter 2 has several limitations. It cannot handle well elements without obvious skeletal representations, e.g. bulky tiles, or the cases where good local alignments cannot be found. In addition, the resulting model can only be printed by high-end SLS printer as it contains too many intricate geometries, making it expensive to fabricate. The work proposed in Chapter 3 strives to resolve these issues. Our approach enables fabricating visually interesting objects that are

decorated with customized tiles. It is designed for efficient and low-cost fabrication: the final model is assembled from several flatly-printed and foldable patches. Each patch can be printed using home filament printers (e.g. FDM printer), making the technique more accessible to average users. The goal is achieved by a dedicated tile packing algorithm, which considers fabrication constraints, and a patch extraction step that generates patches to be printed and folded. The location of hinges that connect different patches has been optimized to achieve both foldability and printability.

There are several future works to improve the presented methods:

- The framework of filigree synthesis is time-consuming due to the mesh reconstruction required by mechanical simulators. Investigation into the possibility of conducting structural optimisation directly on the skeleton graph will be extremely helpful to accelerate the job.

- For the second work, it does take some time for the users to assemble the patches. Although we find the process enjoyable as building the parts we customized, it would be an interesting work to further simplify this task. As the scale of target surface grows, tolerances required in fabricating joints will accumulate and lead to larger distortion in final assembly. Improving the design of snap-fit connectors would be a potential future work to alleviate the distortion. Finally, it would be interesting to investigate how the model will be stable via gluing a minimum number of hinge connectors.

References

[1] Attene, M. (2015). Shapes in a box: Disassembling 3d objects for efficient packing and fabrication. *Computer Graphics Forum*, 34(8):64–76.

[2] Bächer, M., Whiting, E., Bickel, B., and Sorkine-Hornung, O. (2014). Spin-It: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 33(4):96:1–96:10.

[3] Bathe, K.-J. (2006). *Finite element procedures*. Klaus-Jurgen Bathe.

[4] Chen, W., Zhang, X., Xin, S., Xia, Y., Lefebvre, S., and Wang, W. (2016). Synthesis of filigrees for digital fabrication. *ACM Transactions on Graphics*, 35(4):1–13.

[5] Chen, X., Zhang, H., Lin, J., Hu, R., Lu, L., Huang, Q., Benes, B., Cohen-Or, D., and Chen, B. (2015). Dapper: decompose-and-pack for 3d printing. *ACM Transactions on Graphics*, 34(6):1–12.

[6] Cheng, D. and Wong, K. (2009). 3doodler seashell pattern lace plastic dress. http://www.s-h-i-g-o.com/#!project/c147c.

[7] Cignoni, P., Pietroni, N., Malomo, L., and Scopigno, R. (2014). Field-aligned mesh joinery. *ACM Transactions on Graphics*, 33(1):1–12.

[8] Diamanti, O., Vaxman, A., Panozzo, D., and Sorkine-Hornung, O. (2014). Designing n-PolyVector fields with complex polynomials. *Computer Graphics Forum*, 33(5):1–11.

[9] Dubuisson, M. P. and Jain, A. K. (1994). A modified Hausdorff distance for object matching. In *Pattern Recognition*, volume 1, pages 566–568.

[10] Dumas, J., Lu, A., Lefebvre, S., Wu, J., and Dick, C. (2015). By-Example Synthesis of Structurally Sound Patterns. *ACM Transactions on Graphics (TOG)*, 34(4).

[11] Garg, A., Sageman-Furnas, A. O., Deng, B., Yue, Y., Grinspun, E., Pauly, M., and Wardetzky, M. (2014). Wire mesh design. *ACM Transactions on Graphics*, 33(4):66:1–66:12.

[12] Harker, J. (2011). Crania Anatomica Filigre: Me to You. https://www.kickstarter.com/projects/joshharker/crania-anatomica-filigre-me-to-you.

[13] Hu, W., Chen, Z., Pan, H., Yu, Y., Grinspun, E., and Wang, W. (2016). Surface mosaic synthesis with irregular tiles. *IEEE transactions on visualization and computer graphics*, 22(3):1302–1313.

[14] Hurtut, T., Landes, P.-E., Thollot, J., Gousseau, Y., Drouillhet, R., and Coeurjolly, J.-F. (2009). Appearance-guided synthesis of element arrangements by example. In *Proceedings of the 7th International Symposium on Non-photorealistic Animation and Rendering*, pages 51–60. ACM.

[15] Iarussi, E., Li, W., and Bousseau, A. (2015). Wrapit: Computer-assisted crafting of wire wrapped jewelry. *ACM Transactions on Graphics*, 34(6).

[16] Jacobson, A., Panozzo, D., et al. (2016). libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

[17] Kazhdan, M. and Hoppe, H. (2013). Screened Poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3):29.

[18] Konaković, M., Crane, K., Deng, B., Bouaziz, S., Piker, D., and Pauly, M. (2016). Beyond developable: computational design and fabrication with auxetic materials. *ACM Transactions on Graphics (TOG)*, 35(4):89.

[19] Kwok, T.-H., Wang, C. C., Deng, D., Zhang, Y., and Chen, Y. (2015). Four-dimensional printing for freeform surfaces: design optimization of origami and kirigami structures. *Journal of Mechanical Design*, 137(11):111413.

[20] Lai, Y.-K., Hu, S.-M., and Martin, R. R. (2006). Surface mosaics. *The Visual Computer*, 22(9-11):604–611.

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

[22] Lefebvre, S. and Hoppe, H. (2006). Appearance-space texture synthesis. *ACM Transactions on Graphics (TOG)*, 25(3):541–548.

[23] Lefebvre, S., Hornus, S., and Neyret, F. (2005). Texture sprites: Texture elements splatted on surfaces. In *ACM Symposium on Interactive 3D Graphics and Games*. ACM SIGGRAPH, ACM Press.

[24] Li, D., Levin, D. I., Matusik, W., and Zheng, C. (2016). Acoustic voxels: Computational optimization of modular acoustic filters. *ACM Transactions on Graphics*, 35(4).

[25] Li, Y., Bao, F., Zhang, E., Kobayashi, Y., and Wonka, P. (2011). Geometry synthesis on surfaces using field-guided shape grammars. *Visualization and Computer Graphics, IEEE Transactions on*, 17(2):231–243.

[26] Luo, L., Baran, I., Rusinkiewicz, S., and Matusik, W. (2012). Chopper: Partitioning models into 3d-printable parts. *ACM Transactions on Graphics*, 31(6):1.

[27] Ma, C., Wei, L.-Y., and Tong, X. (2011). Discrete element textures. *ACM Transactions on Graphics (TOG)*, 30(4):62:1–62:10.

[28] Martínez, J., Dumas, J., Lefebvre, S., and Wei, L.-Y. (2015). Structure and appearance optimization for controllable shape design. *ACM Transactions on Graphics (TOG)*, 34(6):229:1–229:11.

[29] Miguel, E., Lepoutre, M., and Bickel, B. (2016). Computational design of stable planar-rod structures. *ACM Transactions on Graphics*, 35(4):1–11.

[30] Mitani, J. and Suzuki, H. (2004). Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Transactions on Graphics*, 23(3):259.

[31] O'Rourke, J. (1998). *Computational geometry in C*. Cambridge university press.

[32] Passos, V. A. D. and Walter, M. (2009). 3d virtual mosaics: Opus palladium and mixed styles. *The Visual Computer*, 25(10):939–946.

[33] Passos, V. D. and Walter, M. (2008). 3d mosaics with variable-sized tiles. *The Visual Computer*, 24(7-9):617–623.

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

[35] Prévost, R., Whiting, E., Lefebvre, S., and Sorkine-Hornung, O. (2013). Make it stand: balancing shapes for 3D fabrication. *ACM Transactions on Graphics (TOG)*, 32(4):81:1–81:10.

[36] Riccioti, R. (2013). Mucem museum concrete filigree. http://www.lemayonline.com/en/wow/mucem-photographed-by-edmund-sumner.

[37] Schmidt, R., Grimm, C., and Wyvill, B. (2006). Interactive decal compositing with discrete exponential maps. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 605–613. ACM.

[38] Sederberg, T. W. and Parry, S. R. (1986). Free-form deformation of solid geometric models. In *ACM SIGGRAPH computer graphics*, volume 20, pages 151–160.

[39] Shapira, L., Shamir, A., and Cohen-Or, D. (2008). Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249–259.

[40] Shatz, I., Tal, A., and Leifman, G. (2006). Paper craft models from meshes. *The Visual Computer*, 22(9-11):825–834.

[41] Shugrina, M., Shamir, A., and Matusik, W. (2015). Fab forms: customizable objects for fabrication with validity and geometry caching. *ACM Transactions on Graphics (TOG)*, 34(4):100.

[42] Song, P., Deng, B., Wang, Z., Dong, Z., Li, W., Fu, C.-W., and Liu, L. (2016). Cofifab: Coarse-to-fine fabrication of large 3d objects. *ACM Transactions on Graphics*, 35(4):1–11.

[43] 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 (TOG)*, 31(4):48:1–48:11.

[44] Takezawa, M., Imai, T., Shida, K., and Maekawa, T. (2016). Fabrication of freeform objects by principal strips. *ACM Transactions on Graphics (TOG)*, 35(6):225.

[45] Turk, G. (2001). Texture synthesis on surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 347–354.

[46] Umetani, N., Panotopoulou, A., Schmidt, R., and Whiting, E. (2016). Printone: Interactive resonance simulation for free-form print-wind instrument design. *ACM Transactions on Graphics*, 35(6):184:1–184:14.

[47] Umetani, N. and Schmidt, R. (2013). Cross-sectional structural analysis for 3D printing optimization. In *SIGGRAPH Asia 2013 Technical Briefs*, pages 5:1–5:4.

[48] Vanek, J., Galicia, J. A. G., Benes, B., Měch, R., Carr, N., Stava, O., and Miller, G. S. (2014). Packmerger: A 3d print volume optimizer. *Computer Graphics Forum*, 33(6):322–332.

[49] Wang, C. C. (2014). Ldni-based solid modeling. http://ldnibasedsolidmodeling.sourceforge.net/.

[50] Wang, W. M., Zanni, C., and Kobbelt, L. (2016a). Improved surface quality in 3d printing by optimizing the printing direction. *Computer Graphics Forum*, 35(2):59–70.

[51] Wang, X., Le, T. H., Ying, X., Sun, Q., and He, Y. (2016b). User controllable anisotropic shape distribution on 3d meshes. *Computational Visual Media*, 2(4):305–319.

[52] Wei, L.-Y. (2010). Multi-class blue noise sampling. *ACM Transactions on Graphics (TOG)*, 29(4):79.

[53] Wei, L.-Y., Lefebvre, S., Kwatra, V., and Turk, G. (2009). State of the art in example-based texture synthesis. In *Eurographics '09 State of the Art Report*, pages 93–117.

[54] Wei, L.-Y. and Levoy, M. (2001). Texture synthesis over arbitrary manifold surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 355–360.

[55] Zehnder, J., Coros, S., and Thomaszewski, B. (2016a). Designing structurally-sound ornamental curve networks. SIGGRAPH 2016, page to appear.

[56] Zehnder, J., Coros, S., and Thomaszewski, B. (2016b). Designing structurally-sound ornamental curve networks. *ACM Transactions on Graphics*, 35(4):1–10.

[57] Zhang, J., Zhou, K., Velho, L., Guo, B., and Shum, H.-Y. (2003). Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics (TOG)*, 22(3):295–302.

[58] Zhang, Y., Wang, C. C., and Ramani, K. (2016). Optimal fitting of strain-controlled flattenable mesh surfaces. *The International Journal of Advanced Manufacturing Technology*, pages 1–15.

[59] Zhao, H., Lu, L., Wei, Y., Lischinski, D., Sharf, A., Cohen-Or, D., and Chen, B. (2016). Printed perforated lampshades for continuous projective images. *ACM Transactions on Graphics*, 35(5):1–11.

[60] Zhou, K., Huang, X., Wang, X., Tong, Y., Desbrun, M., Guo, B., and Shum, H.-Y. (2006). Mesh quilting for geometric texture synthesis. *ACM Transactions on Graphics (TOG)*, 25(3):690–697.

[61] Zhou, Q., Panetta, J., and Zorin, D. (2013). Worst-case structural analysis. *ACM Transactions on Graphics (TOG)*, 32(4):137:1–137:12.

[62] Zhou, S., Jiang, C., and Lefebvre, S. (2014). Topology-constrained synthesis of vector patterns. *ACM Transactions on Graphics (TOG)*, 33(6):1–11.

[63] Zienkiewicz, O. C. and Taylor, R. L. (2005). *The finite element method for solid and structural mechanics*. Butterworth-Heinemann.