



# ZomeFab: Cost-effective Hybrid Fabrication with Zometools

I-Chao Shen  Ming-Shiuan Chen Chun-Kai Hunag Bing-Yu Chen 

National Taiwan University

## Abstract

*In recent years, personalized fabrication has received considerable attention because of the widespread use of consumer-level three-dimensional (3D) printers. However, such 3D printers have drawbacks, such as long production time and limited output size, which hinder large-scale rapid-prototyping. In this paper, for the time- and cost-effective fabrication of large-scale objects, we propose a hybrid 3D fabrication method that combines 3D printing and the Zometool construction set, which is a compact, sturdy, and reusable structure for infill fabrication. The proposed method significantly reduces fabrication cost and time by printing only thin 3D outer shells. In addition, we design an optimization framework to generate both a Zometool structure and printed surface partitions by optimizing several criteria, including printability, material cost, and Zometool structure complexity. Moreover, we demonstrate the effectiveness of the proposed method by fabricating various large-scale 3D models.*

## CCS Concepts

• **Computing methodologies** → **Shape modeling**;

## 1. Introduction

The recent widespread adoption of consumer-level three-dimensional (3D) printers spurred the growth of academic and industrial fabrication applications. However, several drawbacks, including long production time, limited output size, and high material costs, persist. Many approaches have been proposed to address these shortcomings. To reduce production time and material costs, modern 3D printers allow users to vary the fill rate. Furthermore, various internal patterns have been proposed [LSZ\*14] to reduce the amount of material used without sacrificing structural soundness. In addition, some methods [LBRM12, HLZCO14, VGB\*14] have been proposed for large-scale fabrication, most of which are based on the principle of partitioning large objects into smaller components that are compatible with the printer's output size.

Large-scale fabrication using extant methods remains expensive. To substantially reduce fabrication costs, we propose a hybrid fabrication method that integrates the use of 3D printing materials for the external shape with the use of a supporting structure for the internal volume. The inner structure must satisfy several criteria; for example, the structure must be (1) easy to assemble, (2) reusable, and (3) robust. These criteria ensure that the resulting fabrication is simple to build while also being structurally reliable. *Zometool* [Dav07] satisfies these requirements for coarse fabrication. Moreover, *Zometool* has other favorable characteristics, such as stability and lightness, which facilitate large-scale fabrication, and structural modularity, which facilitates fast fabrication. In this work, we develop a computational method that integrates 3D printing with the *Zometool* structure to reduce the time and material costs of large-scale fabrication.

For a given 3D shape, we design an optimization process to synthesize the inner *Zometool* structure such that shape similarity and structural complexity are optimally traded off. Through simulated annealing, we effectively explore the large structure space. Next, using the optimized *Zometool* structure, we hollow out the shape to obtain the outer shell and partition such that several criteria, including simplicity and printability, are satisfied. These criteria are formulated as a single multiclass labeling problem and solved using a graph-cut algorithm [BK04]. We then design a particular type of connector and optimize its positions for assembling the inner *Zometool* structure and the outer shell.

This paper makes two primary contributions:

- We propose an optimization framework to synthesize the inner *Zometool* structure that replaces solid printed materials in large-scale fabrication.
- We design and print a special connector and optimize its layout for effectively combining the inner *Zometool* structure and the outer printed shell.

## 2. Related Work

### 2.1. Computational Fabrication

In recent years, computational fabrication has attracted considerable attention in the fields of computer graphics and human-computer interaction [SBM16]. Numerous approaches have been proposed for fabricating shapes that satisfy various objectives (e.g., maintaining balance [PWLSH13, BWBSH14], reducing size [LBRM12], strengthening structural soundness [ZPZ13],

and generating specific sounds [UPSW16]) while using various materials and building blocks (e.g., Lego [LYH\*15], planar slices [CPMS14], and interlocking puzzles [SFCO12, ACP\*14]). A detailed review of these approaches is available in [BCMP18].

Despite the development of assisting tools and algorithms, 3D printers still have such drawbacks as long production time, excessive material use, and limited output size. To reduce the consumption of print materials, Huang *et al.* [HZH\*16] and Wu *et al.* [WPGM16] have designed devices and algorithms that print shapes in wireframe form. In addition, several studies have developed different types of internal structures, such as the skin-frame structure [WWY\*13] and the honeycomb-like structure [LSZ\*14]. For the 3D printing of large structures, Luo *et al.* [LBRM12] developed an iterative planar-cut method with the aim of fitting decomposed parts within the 3D printing volume while considering factors such as assemblability and aesthetics. Yao *et al.* [YCL\*15] proposed a level-set framework for 3D shape partition and packing. Compared with the aforementioned works, our method fabricates a shape using both Zometool structures and 3D-printed pieces, thus reducing the fabrication time and cost given the reusability of Zometool structures.

Several studies have focused on hybrid fabrication, for example, CofiFab [SDW\*16], Universal Building Block [CLF\*18], and faBrickation [MIG\*14]. Our work primarily differs from [SDW\*16] and [CLF\*18] in the use of Zometool for fabricating the inner structure. The advantages of Zometool are as follows: (i) Zometool is easy to obtain and manipulate compared with the customized laser-cut shapes used in CofiFab; (ii) Zometool elements are reusable, making them cheaper than the laser-cut shapes; and (iii) Zometool structures are relatively sparse, thus necessitating relatively few building elements and enabling material savings. Different inner structures necessitate the use of different processes and algorithms. For example, the applications of faBrickation differ from those of our method. In faBrickation, Lego bricks are used as the main building blocks to realize shapes, whereas 3D-printed parts are used to fill the smaller components that are difficult to build using Lego bricks.

## 2.2. Zometool Design and Modeling

Zometool is a mathematically precise plastic construction set used for building a myriad of geometric structures [Dav07]; it can be used to visualize simple polygons as well as model complex structures such as DNA molecules. Zometool dates back to the 1960s, when it was first used as a simple construction system inspired by Buckminster Fuller's geodesic domes. Its application has since evolved to satisfy the requirements of versatile modeling. Although Zometool can be used to construct complex structures, it is not an intuitive tool for new users, and its use can also be time consuming. Various tools have been developed to help users design Zometool structures, such as vZome\* and ZomeCAD<sup>1</sup>; these systems provide multiple methods to grow structures. However, building a complex

shape remains difficult because these systems do not provide suggestions to the users, such as what item to use next.

To address these concerns, several studies have proposed automatic construction through the use of computational methods. Zimmer *et al.* approximated and realized freeform surfaces automatically by using Zometool. Zimmer and Kobbelt [ZK14] adopted a growth strategy that entails the use of incremental panels to approximate the desired surface without self-collisions.

## 3. Overview

Given a 3D shape  $M$ , the proposed method automatically generates an inner Zometool structure and outer 3D-printed shells. This method has the following features:

1. **Large Objects.** The proposed method is designed for fabricating large-scale objects whose volume exceeds the printing volume of most current consumer-level 3D printers.
2. **Fabricability.** Each segment of the outer shell can be printed such that it fits inside the volume of consumer-level 3D printers.
3. **Assemblability.** The inner Zometool structure can be easily assembled and connected to the outer printed shells by using specifically designed connectors.
4. **Cost-effectiveness.** Our method maximizes the volume of the inner structure and minimizes the amount of printing materials used.

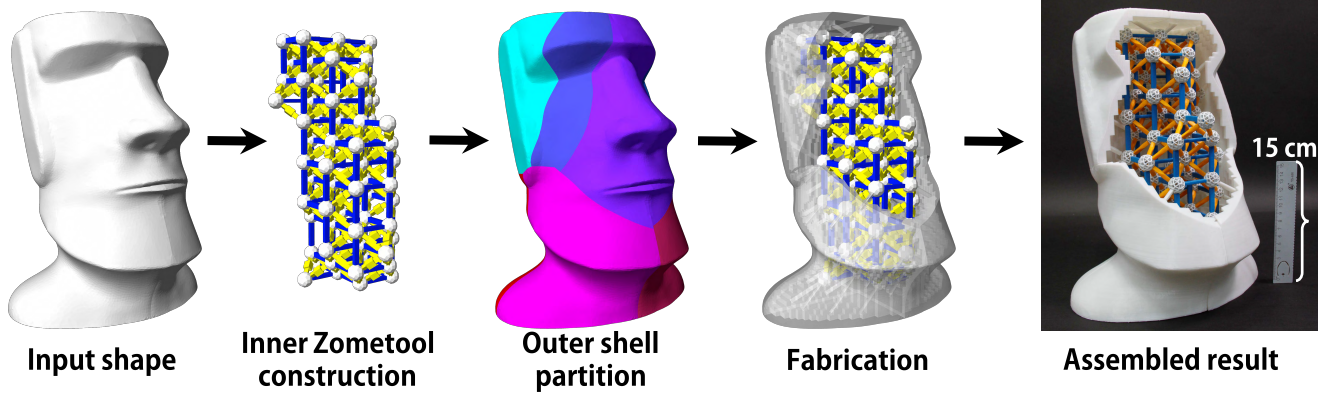
Figure 1 illustrates our method. For a given input shape, we voxelize the inner volume of the input mesh to realize an initial Zometool structure. The objective is to grow the Zometool structure such that it maximizes the inner volume, thus reducing the amount of printing materials necessary. To this end, we design an optimization framework through simulated annealing and design several local operations to explore the optimized Zometool structure space (see Section 4).

Next, to print the outer shell with the appropriate shape, we (i) partition the shape into pieces such that each piece fits inside the printing volume, (ii) place connectors at feasible locations so that the inner Zometool structure and the outer shell can be connected robustly, and (iii) maintain the salient regions intact. We formulate this partition problem as a multi-label problem and solve it by using a graph-cut algorithm under the following considerations: each triangle must be connected to its closest Zomeball, but the number of partitions must also be reduced without sacrificing the integrity of the salient regions. To further regularize the resulting partitions, we apply a support vector machine (SVM) algorithm to determine the hyperplane between different labels. We then use this hyperplane as the cut plane to separate the mesh (see Section 5).

Before separating the mesh using the aforementioned cut plane, we generate the inner surface by voxelizing the inner volume and combine it with the outer mesh as the solid mesh. Subsequently, we apply all the cut planes to the solid mesh and obtain the distinct pieces. Finally, we design a special connector and connect the printed pieces to the inner Zometool structure to form the designed large-scale object (Section 6).

\* <http://vzome.com>

<sup>1</sup> <http://www.softpedia.com/get/Science-CAD/ZomeCAD.shtml>

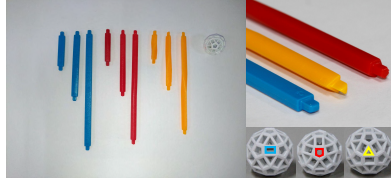


**Figure 1:** For a given input shape, we first optimize the inner Zometool structure (Section 4). Guided by the Zometool structure, we then partition the outer shell (Section 5) and generate connectors for assembling them (Section 6). The final fabricated result is obtained by assembling both assembled Zometool structure and printed outer shell.

## 4. Zometool construction

### 4.1. Introduction to Zometool

Zometool is widely used as an educational toy that replicates complex scientific structures, such as chemical structures. The rods in the standard Zometool system have three types of struts: blue for rectangle, red for pentagon, and yellow for triangle. Each strut comes in three lengths, with length here defined as the distance from the center of a ball on one end to the center of a ball on the other end (see inset). We denote  $(b_0, b_1, b_2)$  as the three lengths of the blue struts; similarly,  $(r_0, r_1, r_2)$  and  $(y_0, y_1, y_2)$  represent the red and yellow struts, respectively. The ratio of the lengths follows the golden ratio,  $\gamma = \frac{1+\sqrt{5}}{2}$ . For example, for the blue struts,  $b_1 = b_0 \cdot \gamma$  and  $b_2 = b_0 + b_1$ . Moreover, the relative length ratio of the yellow and blue struts and that of the red and blue struts differ:  $y_i = \frac{\sqrt{3}}{2} \cdot b_i$  and  $r_i = \frac{\sqrt{2+\gamma}}{2} \cdot b_i$ . Each Zomeball has 62 slots, namely 30 rectangular, 12 pentagonal, and 20 triangular slots. Please refer to [Dav07] for more details on the mathematical model of Zometool.



### 4.2. Initialization

Many complicated structures can be assembled using Zometool. Nevertheless, the assembly complexity and time increase rapidly with the number of Zometool items (i.e., struts and balls) used. A simple and repeating unit structure is used to fabricate the initial structure, following which the rest of the structure is built outward from the initial structure to achieve a good fit with the outer surface. After experimenting with different basic structures (e.g., cubes, triangular pyramids, square pyramids, and pentagonal pyramids), we select the cube as the unit building block for this study because it

has the shortest assembly time. The major difference between our study and that of Zimmer *et al.* [ZLAK14] is that we choose  $b_0$  (the length of the shortest blue rod) as the edge length of the cube; this is because the smaller is the cube, the higher is the fitting rate, and the closer are the inner and outer surfaces (inset shows sample initialization).

### 4.3. Problem Formulation

We measure the quality of the Zometool structure  $\mathbf{Z}$  with energy  $E$  in terms of four quality measurements:

$$E(\mathbf{Z}) = w_{\text{fid}} \cdot E_{\text{fid}}(\mathbf{Z}) + w_{\text{reg}} \cdot E_{\text{reg}}(\mathbf{Z}) + w_{\text{val}} \cdot E_{\text{val}}(\mathbf{Z}) + w_{\text{sim}} \cdot E_{\text{sim}}(\mathbf{Z}), \quad (1)$$

We set  $w_{\text{fid}} = 1.0$ ,  $w_{\text{reg}} = 100.0$ ,  $w_{\text{val}} = 1.0$ ,  $w_{\text{sim}} = 1.0$  for all examples in this paper.

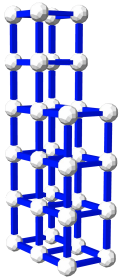
#### 4.3.1. Shape fidelity

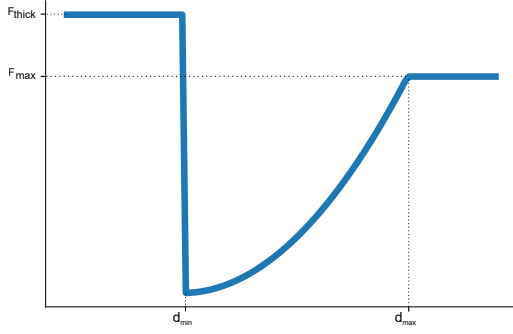
To better represent the input shape  $S$ , the outermost nodes must stay close to the surface of  $S$ ; that is, the distance from the outermost nodes to the surface of  $S$  should be minimized. Thus, the distance from  $\mathbf{Z}$  to  $S$  is integrated over the outermost nodes:

$$E_{\text{fid}}(\mathbf{Z}) = \frac{1}{|N_{\text{out}}| \cdot d_{\text{norm}}^2} \sum_{i=1}^{|N_{\text{out}}|} \|p_i - \delta(p_i)\|^2 \cdot (1 + F(p_i)), \quad (2)$$

where  $N_{\text{out}}$  is the set of outermost nodes and  $\delta(p_i)$  is the point on  $S$  nearest to the outermost nodes. The normalization factor  $d_{\text{norm}}$  is set as  $b_0$  to retain the energy within  $[0, 1]$ . We follow [ZLAK14] and define the term  $F(p)$  as the forbidden zone that penalizes node points lying too far away from the surface.

**Forbidden Zone**  $F(p)$  is defined as a quadratically increasing function that is dependent on the distance between the node  $n$  and the nearest triangle centroid on surface  $M$  when the distance is





**Figure 2: Forbidden zone function  $F(p)$ .** Because Zometool elements that stay close to  $d_{min}$  are preferred, elements farther from  $d_{min}$  are penalized.

within the range from  $d_{min}$  to  $d_{max}$  (as shown in Figure 2). We choose appropriate parameters by conducting tests on different 3D shapes. We thus identify the following set of parameters that produce elements that do not extrude out the forbidden zone across all the tested shapes.  $F(p)$  is set as (i) a constant large penalty ( $F_{thick}$ ) when the distance is less than  $d_{min}$  to prevent the Zometool elements from growing between the outer surface and  $d_{min}$ , and (ii)  $F_{max}$  when the distance exceeds  $d_{max}$ . By testing various 3D shapes, we found that the following set of parameters produce results that do not extrude out of the forbidden zone for all the tested shapes:  $d_{min} = 16.0$  ( $\frac{1}{3}$  length of  $b_0$ ),  $d_{max} = 47.3$  (length of  $b_0$ ),  $F_{max} = 70.0$ , and  $F_{thick} = 90.0$ ; these values are thus used all examples in this paper.

#### 4.3.2. Regularity

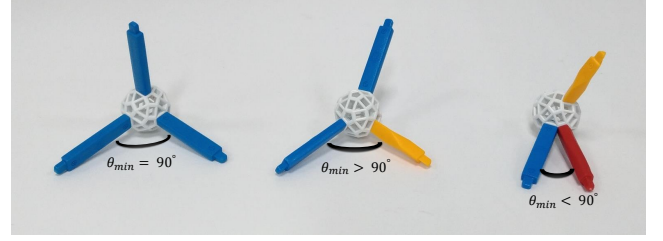
On observing the assembling process of several complicated Zometool structures, we discover two major states that slow down the process: (i) the struts on a Zomeball are too close to each other (i.e., the angles between the struts are too small), and (ii) the slots used to connect the struts are spread out irregularly on the Zomeball. To address these concerns, we regularize the angles between struts to be exactly  $90^\circ$  and penalize any angle that is too small or too large (Figure 3)

$$E_{reg}(\mathbf{Z}) = \sum_{i=1}^S \frac{1}{|\mathcal{N}_i|} \sum_{s_j \in \mathcal{N}_i} (\min(\theta_{ij}) - \frac{\pi}{2}), \quad (3)$$

where  $S$  represents the number of struts and  $\mathcal{N}_i$  represents the struts adjacent to strut  $i$ . The resulting Zometool structure has a relatively high number of repeated Zomeball patterns (the struts on each Zomeball are placed at similar slots), and this considerably reduces the assembling time.

#### 4.3.3. Valence

For ease of assembly, we regularize the optimized Zometool structure such that it has good valence for simple structures (see Figure 4). The primary reason for regularizing valence number is that an undesired Zomeball valence number (Figure 4 (c) and (d)) increases the possibility of element collision, which considerably

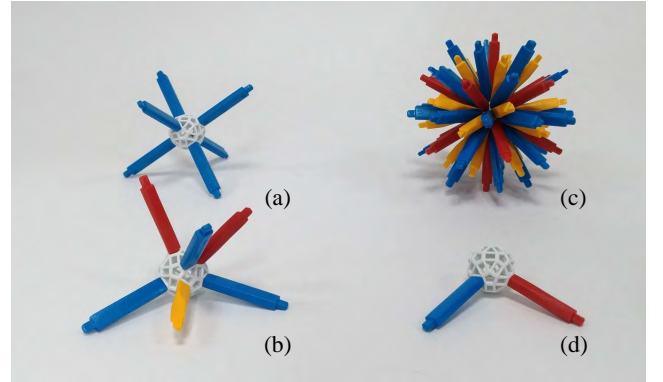


**Figure 3: Regularity.** We penalize configurations where the angle between the struts stray far from  $90^\circ$ .

reduces the optimization efficiency. Collided structures are considered to be invalid (because they can not be assembled) in our method (this is discussed subsequently herein). We set the target valence as 6 according to the initial cube structure, which minimizes the complexity and maximizes the utility of each Zomeball:

$$E_{val}(\mathbf{Z}) = \sum_{i=1}^{N_{in}} \frac{(V_i - 6)^2}{6}, \quad (4)$$

where  $|N_{in}|$  denotes the number of internal nodes and  $V_i$  denotes the valence of node  $n_i \in N_{in}$  (Figure 4).



**Figure 4: Valence.** We encourage the valence of each Zometool node to be 6 (as in configuration (a) and (b)). We penalize the valence that is not 6 (configuration (c) and (d)).

#### 4.3.4. Shape simplicity

To reduce the complexity of the Zometool structure, we limit the number of Zometool elements. However, oversimplification negates the effects of the shape fidelity energy discussed in Section 4.3.1; that is, the desired shape cannot be adequately replicated. For the optimal trade-off of shape fidelity and simplicity, we determine the target number of Zometool elements before optimization. To this end, we run a random Zometool assembling process ten times within the volume of the input shape. For each run, we record the number of Zometool elements required to reach a predefined shape similarity threshold between the Zometool structure and the target shape  $M$ . We set the 90% of the average number of Zometool elements as the final target item number ( $\tau$ ). For most cases, this target number serves as the upper bound of the number



of elements, and the simulated annealing optimization process is always terminated before this target number is reached. Let  $|N|$  represent the total number of nodes, and  $|S|$  represent the total number of struts. Simplicity is measured as the difference between the actual and target item numbers:

$$E_{\text{sim}}(\mathbf{Z}) = \frac{1}{\tau} (|N| + |S| - \tau)^2. \quad (5)$$

#### 4.4. Exploration Mechanism

Determining the Zometool structure that minimizes the energy  $E(\mathbf{Z})$  (Eq. 1) is a nontrivial optimization problem because  $E(\mathbf{Z})$  is nonconvex and contains global terms. Because an exhaustive search is impractical, we adopt a more scalable strategy based on the simulated annealing algorithm [SFS02]. In a nutshell, this algorithm executes a random exploration of the solution space by iteratively perturbing the current solution with a certain probability depending on the energy variation between the two solutions and a relaxation parameter  $T$ . We describe our local perturbation operators and relaxation scheme as follows. Algorithm 1 details our optimization algorithm.

---

##### Algorithm 1 Exploration mechanism

---

```

1: Input: Initialized Zometools  $\bar{\mathbf{Z}}$ ,
2: relaxation parameter  $T = T_{\text{init}}$ 
3: Output: Optimized Zometools  $\mathbf{Z}$ 
4: procedure EXPLORATION( $\mathbf{Z}$ )
5:   repeat
6:     generate  $\mathbf{Z}'$  from  $\mathbf{Z}$  with a random local operation.
7:     draw a random value  $p \in [0, 1]$ 
8:     if  $p < \exp(\frac{E(\mathbf{Z}) - E(\mathbf{Z}')}{T})$  and CollisionFree( $\mathbf{Z}$ ) then
9:       update  $\mathbf{Z}' \leftarrow \mathbf{Z}$ 
10:    end if
11:    Update  $T \leftarrow C \times T$  ▷ Update temperature.
12:  until  $T < T_{\text{end}}$ 
13: end procedure

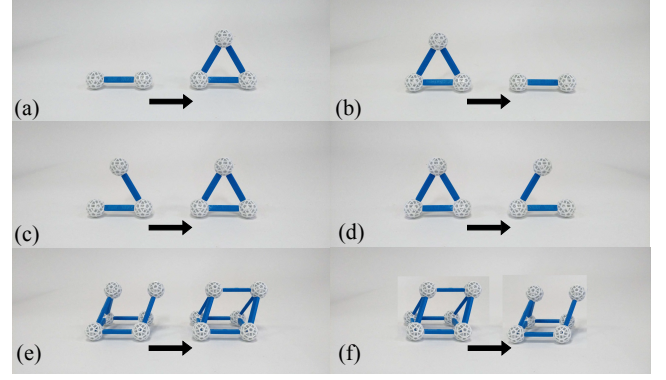
```

---

**Local Perturbation Operation** During the exploration, we proposed six local perturbation operations (Figure 5) to construct the Zometool structure by minimizing Eq. 1.

- **InsNode** This operator inserts a new node and two struts to split the original strut.
- **DelNode** This operator deletes a node and two struts.
- **InsStrut** This operator inserts a single strut to connect two nodes that are not directly linked.
- **DelStrut** This operator deletes a strut between two nodes that are directly linked.
- **InsBridge** This operator inserts a new strut to merge two nodes that are not directly linked (two nodes with connected path length more than two).
- **DelBridge** This operator deletes a strut between two nodes with connected path length more than two.

**Operation validity** The chosen operation at each iteration updates the structure from the previous iteration. However, some updates might introduce invalid structures, such as structures with item collision. We detect collision by checking the (1) Zomeball



**Figure 5:** We use six local operations during structure perturbation. (a) InsNode, (b) DelNode, (c) InsStrut, (d) DelStrut, (e) InsBridge, and (f) DelBridge. Each operation is performed from the left configuration to the right configuration.

to Zomeball, (2) strut to Zomeball, and (3) strut to strut distances; if the distance is less than a certain threshold, the update is rejected.

##### 4.4.1. Cooling schedule

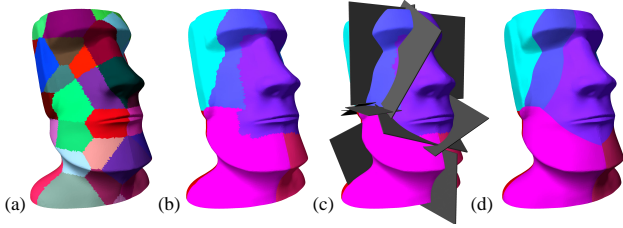
The relaxation parameter  $T$ , referred to as temperature, controls both the speed and quality of the exploration. Starting from an initial temperature  $T_{\text{init}}$ , we decrease the temperature, approaching zero as the iteration tends to infinity. This process is referred to as cooling, and various cooling schedules are available. Although the logarithmic cooling schedule [SFS02] guarantees global minimum convergence, we implement a geometric cooling schedule [HJJ03]. In our experiment, we set the initial temperature  $T_{\text{init}} = 1$  and apply the decrease rate  $C = 0.99$  after every 100 iterations.

#### 5. Object Partition

Most consumer-level 3D printers have limited printing volumes. To print large-scale objects, the objects must be decomposed into smaller partitions. Conventional surface partition methods only account for surface features; however, we must also consider the relationship between the outer surface partitions and the inner optimized Zometool structure. Specifically, in this work, we place connectors between the outer and inner structures in order to connect them (see Section 6). We can simply compute the distance between each triangle  $t$  and all nodes in  $\mathbf{Z}$  and assign  $t$  to the nearest node as its label. However, inconsistencies may arise among adjacent triangles, engendering unsatisfactory visual effects and assembly complexities because numerous small partitions might exist (Figure 6(a)).

To address this concern, we formulate the problem as a multi-label graph-cut minimization problem. As each triangle  $t$  can potentially correspond to different Zometool nodes, it gets assigned data costs for different corresponding nodes. Given  $n$  elements,  $k$  labels, and  $n \cdot k$  costs, finding the minimum assignment is a typical NP-hard combinatorial problem, which we solve using Boykov [BK04].

After partitioning the structure, we further regularize the boundaries between the partitions by performing multi-class classification; this process of seeking smooth boundaries ensures easy assembly. This problem has been previously addressed by Wang *et al.* [WZK16] and Alderighi *et al.* [AMG\*18]. We follow [WZK16] and use an SVM [CV95] classifier to determine our cut plane. We describe the formulation and implementation of a Markov random field (MRF) problem and how to find the cut planes in the following paragraph.



**Figure 6:** (a) Result of nearest node classification (b) result of graph cut, (c) cut-plane generated using an SVM classifier, and (d) cutting result.

## 5.1. Surface Partition

### 5.1.1. Optimization energy

We compute the assignment function  $f$  that assign labels to each triangle  $t$ , where  $t \in T$ , such that the labeling  $f$  minimize the following energy  $E(f)$ :

$$E(f) = w_{data} \sum_{t \in T} D(t, f_t) + w_{smoothness} \sum_{t,s \in \mathcal{N}} \Psi_{t,s}(t, s, f_t, f_s), \quad (6)$$

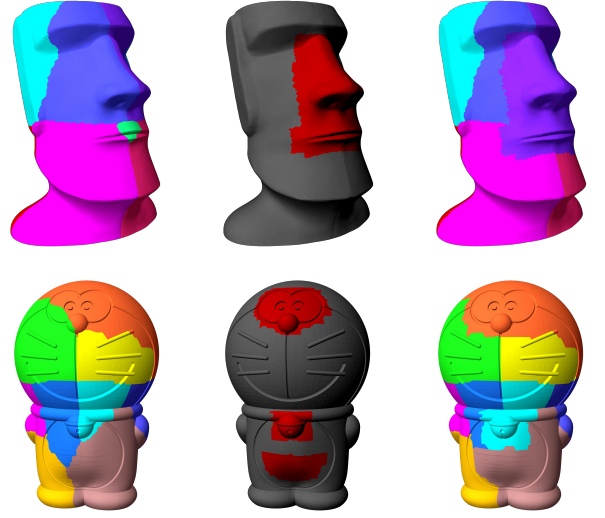
where  $f_t$  and  $f_s$  are labels assigned to triangle  $t$  and  $s$ , and  $\mathcal{N}$  is the set of all pairs of triangles sharing edges. We set  $w_{data} = 1.0$  for all the shapes shown in this paper, and we use  $w_{smoothness}$  to control the size of the partitions, which is crucial for limiting the size of the partitions such that they fit within the 3D printing volume. Specifically, we initialize  $w_{smoothness}$  as 10 and examine whether we can fit all the partitions within the 3D printing volume. If not, we multiply  $w_{smoothness}$  by 0.1 and repeat this process until all the partitions fit into the desired volume. We optimize this function by using the multilabel graph-cut algorithm proposed by Boykov [BK04]. In our setting, the entire outer nodes of  $\mathcal{Z}$  form the complete possible label set  $L$ . This  $E(f)$  comprises two terms: data cost and smoothness.

**Data cost.** We measure how well a triangle  $t$  covers an outermost node  $n \in N_{out}$  as data cost. This cost is simply defined as the distance between the centroid of the triangle  $t$  and the outermost node  $n \in N_{out}$ .

$$D(t, f_t) = \log(d(t, n)). \quad (7)$$

Noted that the set of possible  $f_t$  is the set of node id of outermost nodes.

**Smoothness cost.** This term measures the spatial consistency of



**Figure 7:** In the left-most column, we show optimized partitions without introducing the user-guided saliency term. We can observe that the partition cut through some salient features on the mesh. To address this problem, the user annotates the region that they wish to preserve (red region in the middle column). We then incorporate an additional term in the original smoothness term to prevent the annotated regions from separating into multiple parts (right-most column).

neighboring elements.

$$\Psi_{t,s}(t, s, l_t, l_s) = \begin{cases} 0, & \text{if } l_t = l_s, \\ -\log(\theta_{t,s}/\pi)\phi_{t,s} + w_{saliency} * E_{saliency}(t), & \text{otherwise} \end{cases} \quad (8)$$

where we set  $w_{saliency} = 5.0$ , and  $\theta_{p,q}$  and  $\phi_{p,q}$  are the dihedral angle and the centroid distance between triangles  $p$  and  $q$ , respectively. With the smoothness term, two adjacent triangles are likely to have consistent labels. Each object has many salient regions that the the partition seam must not go through (e.g., the eyes and nose on the face in Figure 7). To preserve the integrity of the salient regions, we ask users to draw the region that they wish to preserve, and we formulate this requirement as part of the smoothness cost to prevent the partition seams from cutting through the salient region (see last column of Figure 7).

$$E_{saliency}(t) = \begin{cases} 1, & \text{triangle is marked as salient region,} \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

### 5.1.2. Optimized partitions

We solve the labeling problem in Eq. 6 by using a graph-cut algorithm and obtain 11 labels on the Maoi head object (Figure 6(b)). Compared with the method that directly assigned the triangle to the closest outermost node, our approach yields much smaller partition

numbers (11 vs. 60 (Figure 6(a)), and each partition is of relatively better shape and size.

## 5.2. Object Cut

To cut the physical object into pieces, we must find the cut planes that separate the space occupied by the object. However, the boundaries between the optimized partitions are not regular enough to be directly used as the separating plane (see Figure 6(b) and the inset). It is very difficult to find the plane in 3D space because of the varied plane normal vectors. Moreover, the zig-zag partition boundaries introduce additional complexities in the 3D printing process; thus, we find a clean separating plane by following [WZK16] and analogizing our problem as a multiclass classification problem, which is then solved using SVM. We use each triangle as a data sample, with its location as the feature vector and the optimized label as its class. Briefly, the SVM algorithm finds the best hyperplane, which is the one that represents the largest separation, or margin, between the two adjacent classes. Because we found that the hyperplanes obtained using SVM suffice to cut the planes, we did not proceed with the subsequent step suggested described in [WZK16] (Figure 6(c)).



## 6. Fabrication

The object to be printed should be a solid mesh. In our method, we use the Zometool structure to fill in most of the inner volume while retaining a predefined thickness of the outer shell. The minimum thickness is dependent on the requirements of the 3D printer used. To print each solid piece, we must obtain the inner surface and use the original surface as the outer surface. The inner surface can be generated in many approaches, such as by shrinking the mesh along the vertex normals. However, these approaches often generate surfaces with flipped triangles that stick out of the outer surface. To prevent this problem, we instead voxelize the original object and remove the voxels whose center falls within the minimum thickness from surface. We then use the outermost surface of the remaining voxels as our inner surface solid piece. As discussed in Section 4.3.1, we penalize the Zometool elements that grow within the range of  $d_{\min}$  (which is our default thickness). For all the shapes shown in this paper, no Zometool elements poked out beyond the surface.

### 6.1. Generate connector

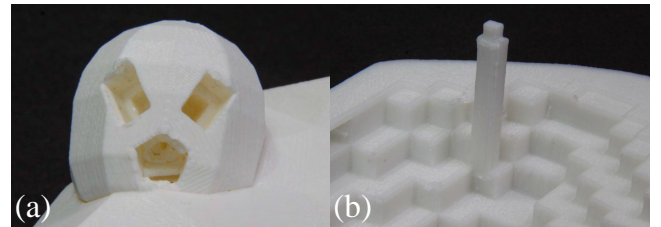
With the inner surface, we need to place connectors on it to connect between inter Zometool structure and outer shells. Two potential designs for building these connectors are:

1. Dig holes on the surface and use the Zometool struts to connect both inter and outer structures (Figure 8 (a)).
2. Grow Zometool tenons on the inner surface (Figure 8 (b)).

We tried both designs, and as we experimented, we observed that the generated support structures from the 3D printers hugely reduce the quality of the digged holes. The reason is the printed holes are

usually filled with the support materials, and it is difficult to remove all of them. Hence, we choose to grow tenons on the inner surface with following method.

**Grow tenons on the surface.** Given it's easier cleanup and more robust structure, we use this design to connect the inner Zometool structure and outer shell (see Figure 8 (b)). We restricted the grown tenons have to be perpendicular to the inner surface in order to achieve better structural robustness. And we decide how many tenons on each outer shell with the following process: We shoot rays from each admissible slot on a single Zomeball in the optimized Zometool structure, and record whether it intersects with the tested shell. Since the tenons are only generated perpendicular to the inner voxel surface, its direction is basically axis-aligned. This means that for each Zomeball, only 6 slots are admissible for each tenon. We repeated this examination on all of 6 slots on each of the Zomeball covered by this shell, and we find the directions that are perpendicular to the inner surface and grow tenons along those directions.



**Figure 8:** Connector designs: (a) dig holes and (b) grow tenons on inner surface. The materials in the dug holes can not be removed entirely in (a), so the struts can not be inserted well.

## 7. Result

### 7.1. Experiment environment

We implement ZomeFab in C++ and Python on desktop PC with 3.4GHz CPU and 16GB memory. The computation time of Zometool structure usually takes around 0.5 - 2.5 hrs due to the convergence of the simulated annealing. The multi-label graph cut algorithm takes around 2 mins for all shapes shown in the paper. The outer surfaces are printed by Ultimaker 3, a low-cost FDM 3D printer with 0.2m x 0.2m x 0.2m printing volume and PLA material.

### 7.2. Evaluation

We evaluate the material cost and fabrication time between "Zomefab" and solid-printed object (denoted as "baseline" method). We use CURA<sup>2</sup>, a slicer software for 3D printing, to simulate the fabrication time and used materials for solid-printed object. Meanwhile, we subdivide the shape using a octree, and stop the subdivision once each sub-shape fit into the printing volume. To evaluate our method, we report two different infill methods called "hollow" and "solid". Shapes printed under "hollow" use only 20% infill rate,

<sup>2</sup> <https://ultimaker.com/en/products/ultimaker-cura-software>

where “solid” use 100% infill rate. Note that we only use 20% infill rate to fabricate the results shown in this paper, and the “solid” results are simulated using CURA.

**Material cost.** As shown in Table 1, our method greatly saves materials from 24% to 64% under “hollow” setting, and from 68% to 85% under “solid” setting. As expected that our method brings more benefits when infill rate grows. The material cost are listed as follow: 0.56 USD/meter, Zometool strut: 0.19 USD/strut and Zomeball: 0.29 USD/ball for our experiments.

**Fabrication time.** We report printing time evaluation under single 3D printer scenario and assembling time of zometool structure in Table 1. The statistics show that the overall fabrication time, i.e. printing time of outer pieces plus the assembly time of Zometool structure, is on average 30% shorter than the baseline method under “hollow” setting, and 74% under “solid” setting.

### 7.3. Zometool Use

Table 2 shows the quantity of Zometool struts and balls used in each result. In Section 4, we use smallest blue struts to make the zomecube as the unit structure in order to make the best-fit initial structure. The bigger the object is, more smallest blue struts ( $b_0$ ) will be used. We also observed that, blue struts and yellow struts are more likely to be used interchangeably since we encourage regularity during simulated annealing optimization. Meanwhile, the rule of Zometool indicates that the longest strut only can be replaced by one shortest and middle strut in the same color. So it seldom introduces red struts into the structure, which resulted in the used number of struts imbalance shown in Table 2. To ease the assembling process of Zometool structure, we wrote a simple interface that can highlight the connected struts for each ball. This can greatly improve the efficiency of the assembling process.

## 8. Conclusion

In this paper, we propose ZomeFab, a hybrid fabrication method that combines Zometool and 3D printing to fabricate a large-scale 3D object. We aimed at decomposing an input 3D object into a inner structure and pieces of outer surface. By replacing the huge inner space with Zometool from 3D printed materials, we can greatly reduce the cost of 3D printing. And we retain the fine geometric detail by printing the outer shell with 3D printer. With the reusability of Zometool, the long-term cost of fabrication is greatly decreased. We have demonstrated that our method is able to fabricate large-scale object by physically replicating 5 objects.

**Limitations and Future Work.** There are many remaining challenges and opportunities for future research on large-scale fabrication. While our initial unit structure is simple and easy-to-assemble, it's size (4.7cm x 4.7cm x 4.7cm) prevents us to fabricate 3D object with thin structure. Meanwhile, since Zometool is designed under formal and strict mathematical formulation, the connection between two separated Zometool sub-structure might not exist within the 3D object inner space. As a result, our method can not fabricate 3D object with two or more parts connected by thin structure that our initial unit structure can not fit in. In the future, we would love to investigate how to shrink the size of the initial unit structure, in order to enable more complicated large-scale 3D object fabrication.

## 9. Acknowledgements

We would like to thank Sheng-Jie Luo, Hsin-I Chen and anonymous reviewers for insightful suggestions and discussions. During this work, I-Chao Shen was supported by the MediaTek Fellowship. This research was supported in part by the Ministry of Science and Technology of Taiwan (MOST105-2218-E-002-011) and National Taiwan University.

## References

- [ACP\*14] ALEMANNO G., CIGNONI P., PIETRONI N., PONCHIO F., SCOPIGNO R.: Interlocking pieces for printing tangible cultural heritage replicas. In *Proceedings of the Eurographics Workshop on Graphics and Cultural Heritage* (2014), Eurographics Association, pp. 145–154. 2
- [AMG\*18] ALDERIGHI T., MALOMO L., GIORGI D., PIETRONI N., BICKEL B., CIGNONI P.: Metamolds: Computational design of silicone molds. *ACM Trans. Graph.* 37, 4 (July 2018), 136:1–136:13. URL: <http://doi.acm.org/10.1145/3197517.3201381>, doi:10.1145/3197517.3201381. 6
- [BCMP18] BICKEL B., CIGNONI P., MALOMO L., PIETRONI N.: State of the art on stylized fabrication. *Computer Graphics Forum* 37 (2018). URL: <http://vcg.isti.cnr.it/Publications/2018/BCMP18.2>
- [BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 9 (2004), 1124–1137. 1, 5, 6
- [BWBSH14] BÄCHER M., WHITING E., BICKEL B., SORKINE-HORNUNG O.: Spin-It: Optimizing moment of inertia for spinnable objects. *ACM Trans. Graph. (Proc. SIGGRAPH 2014)* 33, 4 (2014), 96:1–96:10. 1
- [CLF\*18] CHEN X., LI H., FU C.-W., ZHANG H., COHEN-OR D., CHEN B.: 3d fabrication with universal building blocks and pyramidal shells. *ACM Trans. Graph.* 37, 6 (Dec. 2018), 189:1–189:15. URL: <http://doi.acm.org/10.1145/3272127.3275033>, doi:10.1145/3272127.3275033. 2
- [CPMS14] CIGNONI P., PIETRONI N., MALOMO L., SCOPIGNO R.: Field-aligned mesh joinery. *ACM Trans. Graph.* 33, 1 (2014), 11:1–11:12. 2
- [CV95] CORTES C., VAPNIK V.: Support-vector networks. *Mach. Learn.* 20, 3 (1995), 273–297. 6
- [Dav07] DAVIS T.: The mathematics of zome, 2007. 1, 2, 3
- [HJJ03] HENDERSON D., JACOBSON S. H., JOHNSON A. W.: *The Theory and Practice of Simulated Annealing*. Springer US, 2003. 5
- [HLZCO14] HU R., LI H., ZHANG H., COHEN-OR D.: Approximate pyramidal shape decomposition. *ACM Trans. Graph.* 33, 6 (2014), 213:1–213:12. 1
- [HZH\*16] HUANG Y., ZHANG J., HU X., SONG G., LIU Z., YU L., LIU L.: Framefab: Robotic fabrication of frame shapes. *ACM Trans. Graph.* 35, 6 (2016), 224:1–224:11. 2
- [LBRM12] LUO L., BARAN I., RUSINKIEWICZ S., MATUSIK W.: Chopper: Partitioning models into 3D-printable parts. *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2012)* 31, 6 (2012). 1, 2
- [LSZ\*14] LU L., SHARF A., ZHAO H., WEI Y., FAN Q., CHEN X., SAVOYE Y., TU C., COHEN-OR D., CHEN B.: Build-to-last: Strength to weight 3D printed objects. *ACM Trans. Graph.* 33, 4 (2014), 97:1–97:10. 1, 2
- [LYH\*15] LUO S.-J., YUE Y., HUANG C.-K., CHUNG Y.-H., IMAI S., NISHITA T., CHEN B.-Y.: Legolization: Optimizing LEGO designs. *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2015)* 34, 6 (2015), 222:1–222:12. 2



Mesh	Infill Method	Fabrication Method	Single 3D Printer Fabrication Time (hours)			Material Cost (USD)			Efficiency (Saved)	
			3D printing	Zometool (assemble time)	Overall (sum)	3D printing	Zometool	Overall (max)	Time	Material
Moai	Hollow	Zomefab	293.97	2.5	296.47	82.56	70.57	153.13	31.70%	13.83%
		Baseline	434.08		434.08	177.71		177.71		
	Solid	Zomefab	523.20	2.5	525.70	170.9	70.57	241.47	77.75%	65.11%
		Baseline	2362.28		2362.28	692.07		692.07		
Squirrel	Hollow	Zomefab	355.05	3.0	358.05	100.38	77.55	177.93	22.04%	21.74%
		Baseline	459.28		459.28	227.37		227.37		
	Solid	Zomefab	643.65	3.0	646.65	213.96	77.55	291.51	76.57%	66.17%
		Baseline	2759.72		2759.72	861.66		861.66		
Doraemon	Hollow	Zomefab	356.60	4.0	360.60	101.38	76.34	177.72	37.88%	21.91%
		Baseline	580.48		580.48	227.59		227.59		
	Solid	Zomefab	643.65	4.0	647.65	227.78	76.34	304.12	57.71%	64.25%
		Baseline	1531.42		1531.42	850.79		850.79		
Totoro	Hollow	Zomefab	273.68	3.0	276.68	72.32	61.48	133.80	27.76%	36.38%
		Baseline	383.02		383.02	178.88		178.88		
	Solid	Zomefab	468.82	3.0	471.82	148.26	61.48	209.74	77.68%	68.09%
		Baseline	2113.42		2113.42	657.31		657.31		
Iron Man	Hollow	Zomefab	282.30	2.0	284.30	73.99	57.76	131.75	34.98%	34.12%
		Baseline	437.25		437.25	199.97		199.97		
	Solid	Zomefab	477.03	2.0	479.03	149.67	57.76	207.43	80.65%	72.93%
		Baseline	2475.22		2475.22	766.24		766.24		

**Table 1:** ZomeFab's performance on saving time & material time as compared to a baseline method.

Mesh	Zometool										Total struts	Total balls
	Blue			Red			Yellow					
	S	M	L	S	M	L	S	M	L			
Moai	112	0	0	0	0	0	140	0	0	252	73	
Squirrel	144	22	0	8	4	0	119	3	1	301	85	
Doraemon	143	26	1	2	5	1	89	1	1	569	87	
Totoro	95	0	0	0	0	0	137	0	0	232	60	
Iron Man	93	0	0	0	0	0	124	0	0	217	57	
Owl	115	0	0	0	0	0	159	0	0	274	70	
Pig	61	12	0	10	8	0	53	6	0	150	47	
Slime	132	0	0	0	0	0	182	0	0	314	80	
Lion	78	0	0	0	0	0	101	0	0	179	50	
Bunny	215	0	0	0	0	0	266	0	0	481	126	

**Table 2:** Zometool element usage.

- [MIG\*14] MUELLER S., IM S., GUREVICH S., TEIBRICH A., PFISTERER L., GUIMBRETIERE F., BAUDISCH P.: Wireprint: 3D printed previews for fast prototyping. In *Proc. ACM UIST '14* (2014), pp. 273–280. [2](#)
- [PWLSH13] PRÉVOST R., WHITING E., LEFEBVRE S., SORKINE-HORNUNG O.: Make It Stand: Balancing shapes for 3D fabrication. *ACM Trans. Graph. (Proc. SIGGRAPH 2013)* 32, 4 (2013), 81:1–81:10. [1](#)
- [SBM16] SHAMIR A., BICKEL B., MATUSIK W.: Computational tools for 3D printing. In *ACM SIGGRAPH 2016 Courses* (2016), pp. 9:1–9:34. [1](#)
- [SDW\*16] SONG P., DENG B., WANG Z., DONG Z., LI W., FU C.-W., LIU L.: CofiFab: Coarse-to-fine fabrication of large 3D objects. *ACM Trans. Graph. (Proc. SIGGRAPH 2016)* 35, 4 (2016). [2](#)
- [SFCO12] SONG P., FU C.-W., COHEN-OR D.: Recursive interlocking puzzles. *ACM Trans. Graph. (SIGGRAPH Asia 2012)* (2012), 128:1–128:10. [2](#)
- [SFS02] SALAMON P., FROST R., SIBANI P.: *Facts, conjectures, and improvements for simulated annealing*. Society for Industrial and Applied Mathematics, 2002. [5](#)
- [UPSW16] UMETANI N., PANOTOPOULOU A., SCHMIDT R., WHITING E.: Printone: Interactive resonance simulation for free-form print-wind instrument design. *ACM Trans. Graph.* 35, 6 (2016), 184:1–184:14. [2](#)
- [VGB\*14] VANEK J., GALICIA J. A. G., BENES B., MECH R., CARR N. A., STAVA O., MILLER G. S. P.: PackMerger: A 3D print volume optimizer. *Comput. Graph. Forum* 33, 6 (2014), 322–332. [1](#)
- [WPGM16] WU R., PENG H., GUIMBRETIERE F., MARSCHNER S.: Printing arbitrary meshes with a 5DOF wireframe printer. *ACM Trans. Graph.* 35, 4 (2016), 101:1–101:9. [2](#)
- [WWY\*13] WANG W., WANG T. Y., YANG Z., LIU L., TONG X., TONG W., DENG J., CHEN F., LIU X.: Cost-effective printing of 3D objects with skin-frame structures. *ACM Trans. Graph.* 32, 6 (2013), 177:1–177:10. [2](#)
- [WZK16] WANG W. M., ZANNI C., KOBELT L.: Improved surface quality in 3D printing by optimizing the printing direction. *Comput. Graph. Forum (Proc. EG 2016)* 35, 2 (2016), 59–70. [6, 7](#)
- [YCL\*15] YAO M., CHEN Z., LUO L., WANG R., WANG H.: Level-set-based partitioning and packing optimization of a printable model. *ACM Trans. Graph.* 34, 6 (2015), 214:1–214:11. [2](#)
- [ZK14] ZIMMER H., KOBELT L.: Zometool rationalization of freeform surfaces. *IEEE Trans. Vis. Comput. Graph.* 20, 10 (2014), 1461–1473. [2](#)
- [ZLAK14] ZIMMER H., LAFARGE F., ALLIEZ P., KOBELT L.: Zometool shape approximation. *Graph. Models* 76, 5 (2014), 390–401. [3](#)
- [ZPZ13] ZHOU Q., PANETTA J., ZORIN D.: Worst-case structural analysis. *ACM Trans. Graph.* 32, 4 (2013), 137:1–137:12. [1](#)



**Table 3:** We fabricated five shapes using our method, and show the printed pieces (1st column), inner Zometool structure (2nd column), and assembled final results (3rd and 4th column). Noted that the scale of 2nd, 3rd, and 4th columns are the same.