LineUp: Computing Chain-Based Physical Transformation

MINJING YU, ZIPENG YE, and YONG-JIN LIU, Tsinghua University, China
YING HE, Nanyang Technological University, Singapore
CHARLIE C. L. WANG, The Chinese University of Hong Kong, China

In this article, we introduce a novel method that can generate a sequence of physical transformations between 3D models with different shape and topology. Feasible transformations are realized on a chain structure with connected components that are 3D printed. Collision-free motions are computed to transform between different configurations of the 3D printed chain structure. To realize the transformation between different 3D models, we first voxelize these input models into a similar number of voxels. The challenging part of our approach is to generate a simple path—as a chain configuration to connect most voxels. A layer-based algorithm is developed with theoretical guarantee of the existence and the path length. We find that collision-free motion sequence can always be generated when using a straight line as the intermediate configuration of transformation. The effectiveness of our method is demonstrated by both the simulation and the experimental tests taken on 3D printed chains.

CCS Concepts: • **Computing methodologies** → **Shape analysis**;

Additional Key Words and Phrases: Transforming structures, computational fabrication, folding, motion planning

ACM Reference format:
Minjing Yu, Zipeng Ye, Yong-Jin Liu, Ying He, and Charlie C. L. Wang. 2019. LineUp: Computing Chain-Based Physical Transformation. *ACM Trans. Graph.* 38, 1, Article 11 (January 2019), 16 pages.
https://doi.org/10.1145/3269979

1 INTRODUCTION

With the advancement of 3D printing technology, how to design printed models with transformable shapes has caught more and more attention in the computer graphics community (e.g., (Duncan et al. 2016; Garg et al. 2016; Guseinov et al. 2017; Konaković et al. 2016; Li et al. 2015; Pérez et al. 2017)). Existing works can be mainly classified into two groups depending on whether disassembly (and re-assembly) is allowed. Objects that allow to disassemble, such as Lego bricks (Luo et al. 2015), Mix-and-Match toys (Team 2010), interlocking puzzles (Xin et al. 2011), and free-form interchangeable components (Duncan et al. 2016), are in general flexible to construct a variety of shapes. This group of approaches has been extensively studied.

Recently, effort has shifted to a more challenging problem—transforming the configurations of a model by only shifting, folding, and twisting its components (e.g., see Zhou et al. (2014), Sun and Zheng (2015), Li et al. (2015), Garg et al. (2016), and Song et al. (2017b)). These approaches do not require disassembly, and all components are connected. As a result, they allow users to transform the shape of a model more easily. Yu et al. (2018) provide a user study based on behavioral and EEG data analysis showing that via transforming 3D shape without disassembly, tangible interaction can significantly improve users' performance related to spatial ability. However, there is no existing approach that can compute n-ary ($n \geq 3$) physical transformation among general 3D shapes. In this article, we propose a method that can generate such transformation between example models with different shapes and topology; e.g., as shown in Figures 1 and 15, five 3D models with different geometries and topologies can be physically transformed into each other via a common line configuration.

In our approach, models with different shapes are formed via folding and twisting a 3D printed chain. Models are represented by voxels in our computation and each component on a chain is corresponding to two neighboring voxels. The problem is to realize physical transformations by folding and twisting is converted to compute a line configuration to link up most voxels inside a 3D model and then convert the line configuration into a chain of pairs of voxels. Two major challenges are (1) how to compute such a line-based configuration inside the volume of each 3D model and (2) how to generate the collision-free motions to physically transform from one configuration into another. Several applications of the proposed line-based physical transformations, such as spatial ability training, self-reconfigurable modular robots, and transforming structures, are discussed in Section 8.

Given a 3D model represented by a set of voxels, we develop a layer-based approximation algorithm to compute the sequence of voxels on the line. First of all, a high-genus model will be decomposed into regions with simple topology. Each planar layer of voxels in a simple region can then be covered by a linear sequence of connected voxels, and the neighboring layers are connected by a pair of neighboring voxels. With the help of this layer-based strategy, we prove that this algorithm can generally compute such a line configuration of connected voxels for 3D models as long as a model can provide enough bandwidth of voxels to connect different layers. Collision-free motions are computed by stretching the connected pairs of voxels on a chain into a straight line. As a result, the straightened shape can be used as an intermediate configuration to physically transform a 3D printed chain from one shape

Minjing Yu and Zipeng Ye are joint first authors.

This work is supported by the National Science Foundation of China (61725204, 61432003, 61521002 and 61661130156) and the Royal Society-Newton Advanced Fellowship (NA150431).

Authors' addresses: M. Yu, Z. Ye, and Y.-J. Liu (corresponding author), Tsinghua University, MOE-Key Laboratory of Pervasive Computing, Department of Computer Science and Technology, Beijing, 100084, China; emails: {yumj14, yezp17}@mails.tsinghua.edu.cn, liuyongjin@tsinghua.edu.cn; Y. He, Nanyang Technological University, 50 Nanyang Ave, 639798, Singapore; email: YHe@ntu.edu.sg; C. C. L. Wang, The Chinese University of Hong Kong, Department of Mechanical and Automation Engineering, Hong Kong, China; email: cwang@mae.cuhk.edu.hk.

0730-0301/2019/01-ART11 $15.00
https://doi.org/10.1145/3269979

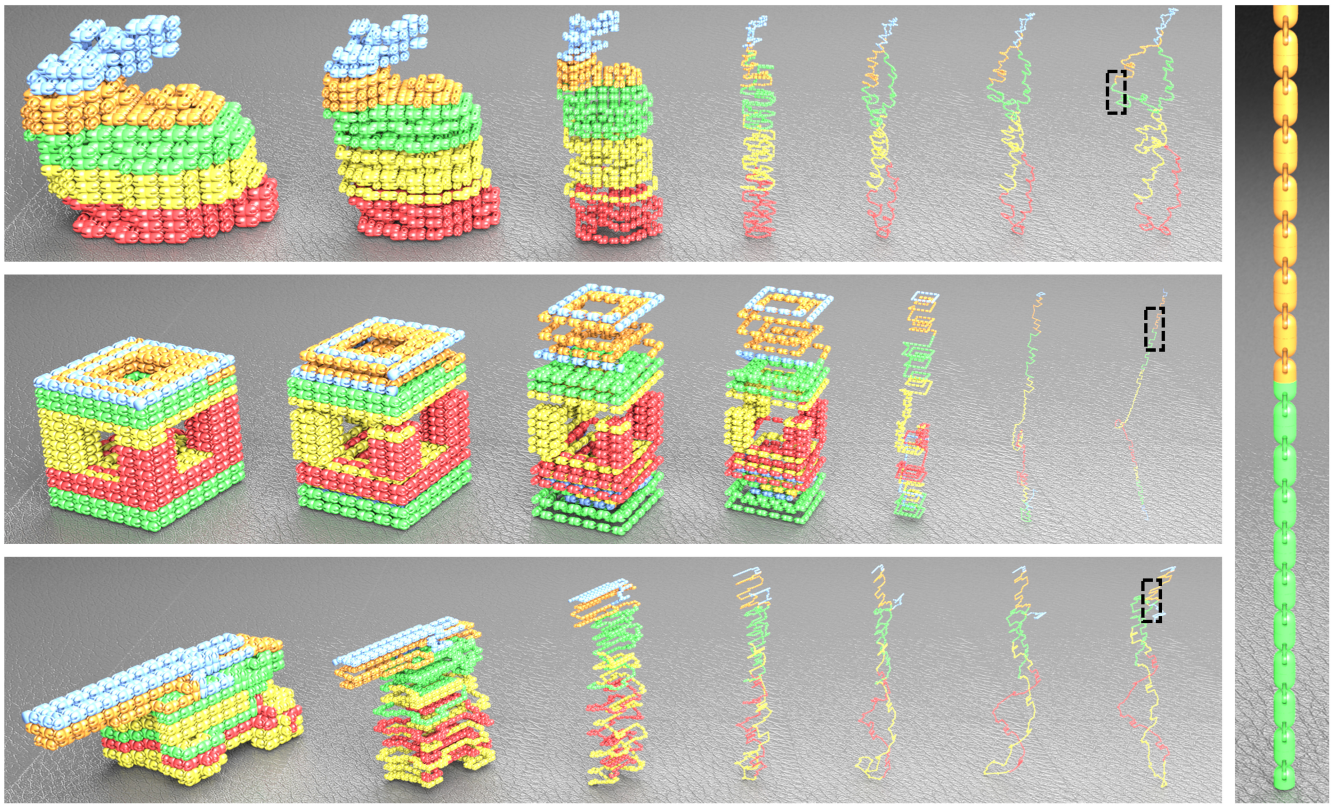


Fig. 1. Models with different shape and topology can be physically formed and transformed between each other by a 3D printed chain. Employing the straight line as an intermediate configuration, our algorithm generates collision-free motion for folding based shape transformation. Here, all models consist of 754 connected building-blocks, and each building-block is corresponding to two neighboring voxels. Colors are used in this figure to indicate the same portion of a chain in different shapes. For visualization, unfolded models in each row are scaled down to fit the space. The procedural transformations can be found in the supplementary video.

into another. Here, the inverse motion of stretching is employed to fold a straight chain into a new shape.

Our technical contributions include:

— A layer-based algorithm to compute a line configuration that connects most voxels inside a general 3D model while preserving its appearance;[1] and
— An algorithm for planning collision-free motions between a folded line configuration and a straight configuration.

Experimental tests are conducted on a variety of 3D models in different shapes and topology. Physical transformations are demonstrated to realize the shape variation on 3D printed chains (see Section 7 and also the supplementary video).

2 RELATED WORK

Special effects of computer animation such as warping and morphing on images and 3D digital models have been widely studied. With the era of computational fabrication, researchers recently start to pay more attention to physical transformation of 3D models. In this section, we only review the research approaches that focus on physically forming the shapes of 3D models.

[1]To preserve the appearance of an input model, no surface voxel will be sacrificed.

2.1 Computational Fabrication

Generally speaking, design for physical transformation is a subproblem of computational fabrication, which has garnered much attention in computer graphics recently (e.g., see Bickel et al. (2018)). Many geometric modeling techniques have been proposed to take fabrication into consideration simultaneously. Mechanical strength was considered and improved in Stava et al. (2012). A worst-case structural analysis was proposed in Zhou et al. (2013), based on which possible structural instability (e.g., high stress or large deformation) for 3D printed models can be identified. By further considering loading conditions in the real world, a context-aware design and fabrication technique with stochastic structural analysis has been recently proposed in Langlois et al. (2016).

To increase the fabrication stability or satisfy other physical properties such as spinnability or aerodynamics, the geometry of an existing shape can be deformed or optimized for fabrication (e.g., see Bächer et al. (2014), Hu et al. (2015), Musialski et al. (2015), and Umetani et al. (2014)). Not only static but also articulated models are able to be fabricated by using the ability of assembly-in-fabrication provided by 3D printing (Calì et al. 2012; Bächer et al. 2012). To generate characters that can be fabricated, mechanical friction joints that satisfy joint types, ranges, and inter-joint non-penetration constraints are designed from skinned input meshes (Bächer et al. 2012). Joints that have internal friction to withstand gravity are considered in Calì et al. (2012) to ease the fabrication of articulated models. However, the shape variation that can be performed on an articulated model is limited.

Starting from the work of Bickel et al. (2010), effort has been made to design the deformation behavior of 3D printed models with specified target shapes under actuation (e.g., the approach presented in Skouras et al. (2013) by multi-material printing and Zhang et al. (2016) by changing the thickness of shells). The behavior of deformation is also designed and optimized by using different microstructures in Schumacher et al. (2015) and Panetta et al. (2015). Again, the deformations produced by these approaches are also limited. Unlike our work presented in this article, they cannot conduct physical transformations between models with completely different shapes and topology (i.e., changing genus-number).

2.2 Shaping by Assembly

Different shapes can be produced by assembling basic building-blocks (e.g., the well-known Lego Bricks (Luo et al. 2015) and Mix-and-Match toys (Team 2010)). Early research targets on fast construction of new digital 3D models by finding and compositing parts of interest searched from a database of 3D models, including the methods of intelligent scissoring (Funkhouser et al. 2004), automatic recommendation (Chaudhuri and Koltun 2010), and probabilistic reasoning for semantic and stylistic compatibility (Chaudhuri et al. 2011) and underlying causes of structural variability in the shape (Kalogerakis et al. 2012). Nevertheless, none of these modeling tools considers the physical transformation between different shapes.

Recently, an interesting work was presented in Duncan et al. (2016) to realize the physical variation of shapes by hands-on disassembly and re-assembly. A set of interchangeable components can be fabricated and are capable of being assembled into several possible shapes. In order to make physical transformation feasible, target models should be co-segmented into meaningful multi-pieces. Early methods of shape co-segmentation (such as in Huang et al. (2011) and Sidi et al. (2011)) cannot be directly applied here as the constraint of exchangeable is hard to be incorporated. Differently, Duncan et al. (2016) jointly deform and partition multiple models together to guarantee that the segmented parts (called interchangeable components) from different shapes can be smoothly assembled into different new shapes. However, the possible shapes obtained from interchange are limited.

2.3 Designing Foldable Structures

Different from the structures and shapes constructed by assembly, a foldable structure can be transformed from a compact shape into another complex shape without disassembly and re-assembly in two ways. The first class of approaches (e.g., Guseinov et al. (2017) and Pérez et al. (2017)) use pre-stretched/auxetic materials. In particular, 3D complex shape can be fabricated on flat sheets, which are pre-stretched. After fabrication, the restoring forces drive the flat sheets back into designed 3D shapes. The second class (e.g., Zhou et al. (2014), Li et al. (2015), Sun and Zheng (2015), and Garg et al. (2016)) is mainly based on shifting, folding, and twisting joints. 3D models with foldable structures are easier to operate but much more difficult to be obtained. An interface is developed in Garg et al. (2016) to interactively design a feasible folding sequence without collision. Infeasible configurations are resolved with the help of user interaction.

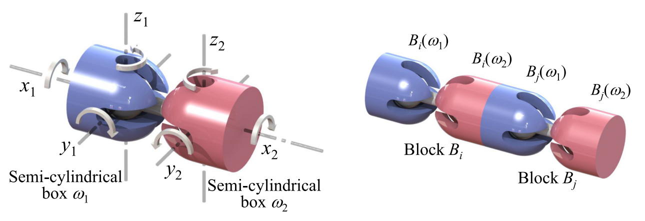
Foldable design has been well studied for some special man-made objects, such as furniture (Li et al. 2015), umbrellas, and bikes (Garg et al. 2016), which can be folded to save space. Another recent work, Boxelization (Zhou et al. 2014), generates a foldable structure that can transform a 3D printed shape into a box. The method is based on segmenting a given 3D model into voxels at a very coarse level and searching a tree structure of connected voxels for installing hinge joints. A folding sequence can be computed at the same time.

As types of foldable structures, paper folding (a.k.a. Origami) and pop-up design have been well studied in computational geometry (O'Rourke 2011). Specifically, paper folding is to find a crease pattern on a piece of paper, along with which the paper can be folded into a 3D shape. Both straight line segments (McArthur and Lang 2012) and curved folds (Kilian et al. 2008) are studied in the crease pattern. In a pop-up design, a complex 3D shape can be closed down to a flat surface. Both folding and cutting are allowed in multi-style pop-up designs (Jr. et al. 2014; Li et al. 2011). Again, a target shape that can be transformed into is fixed—i.e., a planar layout.

2.4 Reconfigurable Structures

Foldable structures usually have two states: a folded (compact) and an unfolded (non-compact) state. By contrast, reconfigurable structures can transform between complex shapes and thus are much more difficult to design. Reconfigurable assemblies were proposed in Song et al. (2017a) such that a common set of parts can be assembled into different forms of furniture. However, disassembly and re-assembly are required in this method. A group-theoretic approach was proposed in Sun and Zheng (2015) to effectively create complex twisty joints in the shape and represent the shape by a twisty puzzle that generalizes the mechanism of Rubik's Cube. The method can effectively obtain contorted poses of a complex and free-form shape. To transform between two arbitrary shapes only by rotating or translating their component parts, a morphological embedding method was proposed in Huang et al. (2016). This method was further improved in an elegant computational framework (Yuan et al. 2018), which includes a user-controllable shape segmentation for allowing users to incorporate their personal preferences into the design process.

How to compute a feasibly physical transformation among multiple (i.e., more than two) models with any 3D shape is still an open problem to be solved. We simply denote all the operations on the joints (e.g., shifting, folding, and twisting) by *transforming* and refer to *configurations* as a model's shape in different transformed statuses. In our approach, voxel representation is used to reduce the difficulty of such computation. However, different from the tree structure of voxels used in Zhou et al. (2014), we compute a line configuration of connected voxels in the form of a chain of 3D printed and connected components as our physical implementation (see Figure 15 for an illustration). Collision-free motions between different configurations of a chain can also be generated



(a) A building-block (b) Connected building-blocks

Fig. 2. The building-block. (a) Each building block consists of two cylinders with hemispherical end ω_1 and ω_2, connected by a hinge. Each cylinder represents a voxel of the input solid. Both cylinders can rotate around the axes x_1 (coincident to x_2), y_1, z_1, y_2, and z_2. (b) Two adjacent building-blocks B_i and B_j are connected by fabricating $B_i(\omega_2)$ and $B_j(\omega_1)$ as a whole piece. For any cylinder (e.g., $B_i(\omega_2)$) in a chain, both of the relative orientations between this cylinder and its two neighbors can be changed. Specifically, one can change the orientation $(B_i(\omega_2), B_i(\omega_1))$ by rotating $B_i(\omega_2)$ around the axes of $B_i(\omega_1)$, and adjust the orientation $(B_i(\omega_2), B_j(\omega_1))$ by rotating $B_i(\omega_2)$ around its own axes.

by our approach; therefore, physical transformation between general 3D shapes is realized.

3 OVERVIEW

We deal with void-free 3D solids, which are represented by voxels. One can easily convert other formats such as polygonal meshes, and implicit and parametric surfaces into voxels using the standard voxelization methods (e.g., Yngve and Turk (2002) and Zhou et al. (2014)). We assign an upright orientation to every input model so that the voxels can be organized in a layered structure. Note that each layer can be either simply or multiply connected.

Problem Statement: Given a set of 3D void-free solids $\widetilde{\Theta} = \{\widetilde{S_i}\}_{i=1}^m$ $(m \geq 2)$, compute another set of voxel-based solids $\Theta = \{S_i\}_{i=1}^m$ such that

(1) $\forall i \in [1, m]$, S_i preserves the appearance of $\widetilde{S_i}$;
(2) All solids $S \in \Theta$ can be physically formed by the same number of building-blocks connected in a linear chain, where each building-block (as shown in Figure 2) is corresponding to two voxels; and
(3) For any two solids $S_i, S_j \in \Theta$ $(i \neq j)$, they can be physically folded into each other (without disassembly and re-assembly).

To solve the problem of n-ary transformation between shapes in Θ, a 3D-printed chain consisting of connected building-blocks is used (see Figure 2). Then, the problem is converted to how to use such a chain to form different voxel-based solids. The key idea of our method is to find a *simple* path $\widetilde{P_i}$ that links as-many-as-possible voxels in each input shape $\widetilde{S_i} \in \widetilde{\Theta}$. The simple path traverses the voxels in a layer-by-layer manner (detailed in Section 4). We call the resulting path a *layered* path. Then, we trim all the paths $\{\widetilde{P_i}\}_{i=1}^m$ into $\{P_i\}_{i=1}^m$ so that all the paths have the same number of voxels (see Section 5). By this way, we can represent each solid $S_i \in \Theta$ by a sequence of voxels lying on the layered path P_i.

Thanks to the layered path $\{P_i\}_{i=1}^m$, all the solids in Θ can be unfolded into a common line configuration, implying that all shapes in Θ can be physically transformed into each other by folding (see Section 6 for more details). As all shapes can be physically formed by a 3D-printed chain for a line configuration, we name this algorithm as LineUp. The overall process is illustrated in Figure 3 and the corresponding pseudo-code is given in Algorithm 1. Main notations are summarized in Table 1.

ALGORITHM 1: LineUp algorithm

```
input: A set of $m(\geq 2)$ solids represented by closed 3D surfaces
output: A chain model consisting of nonseparable building-blocks that
    can be fabricated by 3D printing
 1: Voxelize each surface with similar number of voxels using Zhou et al.
    (2014)
 2: Store $m$ voxelized solids into a set $\widetilde{\Theta} = \{\widetilde{S_i}\}_{i=1}^m$
 3: for each solid $\widetilde{S_i} \in \widetilde{\Theta}$ do
 4:     Choose an upright orientation and compute the Reeb graph $\mathcal{G}$
        (Section 4)
 5:     Compute a traversing path $p(\mathcal{G})$ from $\mathcal{G}$ (Section 4)
 6:     Find a layered path $\widetilde{P_i}$ from $(\widetilde{S_i}, p(\mathcal{G}))$ using Algorithm 3
 7: end for
 8: Trim the paths $\{\widetilde{P_i}\}_{i=1}^m$ into $\{P_i\}_{i=1}^m$ so that all paths have the same
    number of voxels (Section 5)
 9: for each path $P_i$ do
10:     Compute the collision-free motion to unfold $P_i$ into a line
        configuration and the inverse motion for folding (Section 6)
11: end for
12: return
```

4 GENERATING LAYERED PATHS

Let $\widetilde{S} \in \widetilde{\Theta}$ be a void-free solid consisting of n voxels. We denote the connectivity graph by $G_c(\widetilde{S}) = (V_c, E_c)$ of $\widetilde{S}$, where each node in V_c corresponds to a voxel in $\widetilde{S}$ and each edge in E_c links two neighboring voxels.

Our algorithm aims at finding the longest path in $G_c(\widetilde{S})$. Note that (1) the Hamiltonian path problem—determining whether there exists a path that visits each vertex exactly once—is NP-complete (Korte and Vygen 2006), and (2) the longest path problem—finding a simple path of maximum length in a given graph—is NP-hard (Korte and Vygen 2006). Karger et al. (1997) showed that for an arbitrary graph of n nodes, unless $P = NP$, the problem of finding a path of length $n - n^{\varepsilon}$, $\forall \varepsilon < 1$, is NP-hard.

In this section, we propose a novel heuristic method with polynomial time complexity to solve the longest path problem for the connectivity graph $G_c(\widetilde{S})$. Under mild assumptions, our algorithm can find a path of length at least $\frac{5}{6}n + 2$. Furthermore, the generated path is guaranteed to be physically unfoldable, i.e., it can be unfolded into a straight line configuration without self-intersection.

The algorithm consists of the following steps, which are also illustrated in Figure 4:

— First, given a shape $\widetilde{S}$ with upright orientation, we build a *Reeb* graph $\mathcal{G}$ (Fomenko and Kunii 1997) to encode the topology of $\widetilde{S}$ as follows: a node in $\mathcal{G}$ represents a horizontal layer and two nodes are connected by an edge if their corresponding layers are adjacent vertically (Figure 4(b)).

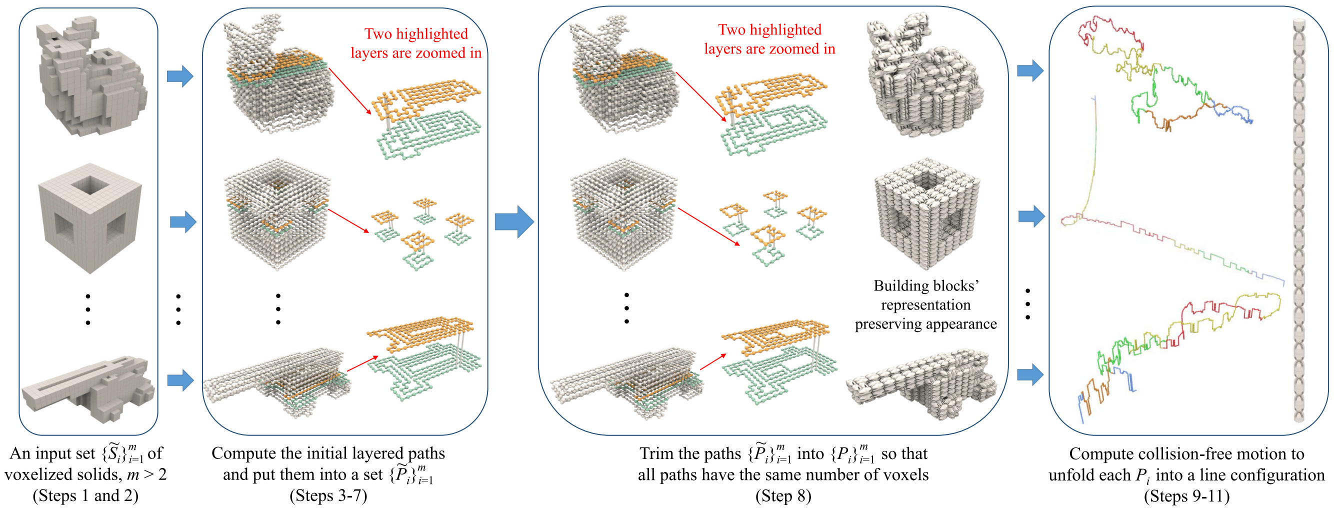

Two highlighted layers are zoomed in
Two highlighted layers are zoomed in
Building blocks' representation preserving appearance
An input set $\{\widetilde{S}_i\}_{i=1}^m$ of voxelized solids, $m > 2$ (Steps 1 and 2)
Compute the initial layered paths and put them into a set $\{\widetilde{P}_i\}_{i=1}^m$ (Steps 3-7)
Trim the paths $\{\widetilde{P}_i\}_{i=1}^m$ into $\{P_i\}_{i=1}^m$ so that all paths have the same number of voxels (Step 8)
Compute collision-free motion to unfold each P_i into a line configuration (Steps 9-11)

Fig. 3. An overview of the proposed method, where the corresponding steps of the pseudo-code given in Algorithm 1 are also indicated.

Table 1. Main Notations

$\{\widetilde{S}_i\}_{i=1}^m$	Input solids
$\{S_i\}_{i=1}^m$	3D printed objects
$\mathcal{G}_i$	Reeb graph of $\widetilde{S}_i$
$p(\mathcal{G}_i)$	a traversing path of $\mathcal{G}_i$
$\widetilde{P}_i$	a layered path in all voxels of $\widetilde{S}_i$
$\{P_i\}_{i=1}^m$	trimmed paths with the same number of voxels
o_i	a horizonal layer of voxels in S
Ω_i	a set of connected voxels in a layer o_i
$\partial\Omega$	the boundary of Ω
Ω°	the interior of Ω
$p(o)$	a long planar path in the layer o
$v_s(o)$	the starting voxel of layer o
$v_e(o)$	the ending voxel of layer o
C_i	a maximal sub-cycle in $p(\mathcal{G})$
(a,b)	directed edge from a to b

—Second, by applying depth-first search to $\mathcal{G}$, we find a spanning tree of the Reeb graph that provides a path $p(\mathcal{G})$ traversing all the nodes (Figure 4 (c) right).
—Third, for each node o_i in $\mathcal{G}$, we build an approximate longest path called *long planar path* $p(o_i)$ in the corresponding layer of connected voxels (Section 4.1).
—Finally, we merge the long planar paths between adjacent nodes in $\mathcal{G}$ according to the number of visits in the traversal (Section 4.2).

4.1 Computing Approximate Longest Path for Each Layer

Each node o in the Reeb graph $\mathcal{G}$ corresponds to a set of voxels Ω that are connected at a horizontal layer. Whenever there is no risk of confusion, we denote the corresponding horizontal layer also by o. We compute an approximate longest path in Ω that links most of the voxels, meanwhile preserving its shape.

ALGORITHM 2: Computing a long planar path in Ω

```
input: A set of connected voxels $\Omega$ in a horizontal layer (corresponding
    to a node $o$ in the Reeb graph $\mathcal{G}$), a starting voxel $v_s$, and an ending
    voxel $v_e$ in $\Omega^\circ$
output: A long planar path $p(o)$ in $\Omega$ connecting $v_s$ and $v_e$
1: Find boundary voxels $\partial\Omega$ and remove all dangling voxels.
2: Generate a cycle $C_{bndy}$ passing through all voxels in $\partial\Omega$ by a linear
   marching scheme (Figure 7(b)).
3: Use $v_s$ and $v_e$ as starting and ending nodes, respectively, to generate
   an approximate longest path $p_{inter}$ in $\Omega^\circ$ (Zhang and Liu 2011) (Figure
   7(c)).
4: Merge $C_{bndy}$ and $p_{inter}$ into a long planar path $p(o)$ (Figure 7(d)).
5: return $p(o)$
```

We can simply view Ω as a 2D domain. Borrowing the terminology of image processing, we define that the 4-connected (resp., 8-connected) voxels are the neighbors in the *same* layer to every voxel that touches one of their faces (resp., one of their faces or corners). To ease the presentation of our method, below, we assume[2] that there are no holes in Ω.

A voxel $v \in \Omega$ is called *boundary voxel* if at least one of its 8-connected neighbors is not in Ω (see Figure 5). We denote the set of boundary voxels by $\partial\Omega$. A boundary voxel $v \in \partial\Omega$ is *dangling* if it has only one 4-connected neighbor in $\partial\Omega$. Algorithm 2 aims at finding a path passing through all the boundary voxels in $\partial\Omega$. After removing all the dangling voxels, one can adopt a linear marching scheme to generate a simple cycle passing all remaining voxels in $\partial\Omega$. We denote by $\Omega^\circ \triangleq \Omega \setminus \partial\Omega$ the interior of Ω.

Given two disjoint cycles c_1 and c_2 in the same layer o, we say c_1 and c_2 are *mergeable* if there exist two adjacent voxels v_i^1 and v_{i+1}^1 in c_1 and two adjacent voxels v_j^2 and v_{j+1}^2 in c_2, such that v_j^2 is a

[2]Actually, our method can be easily adapted to handle holes in Ω, by simply replacing the *boundary voxels* by the *voxels of outmost boundary*.

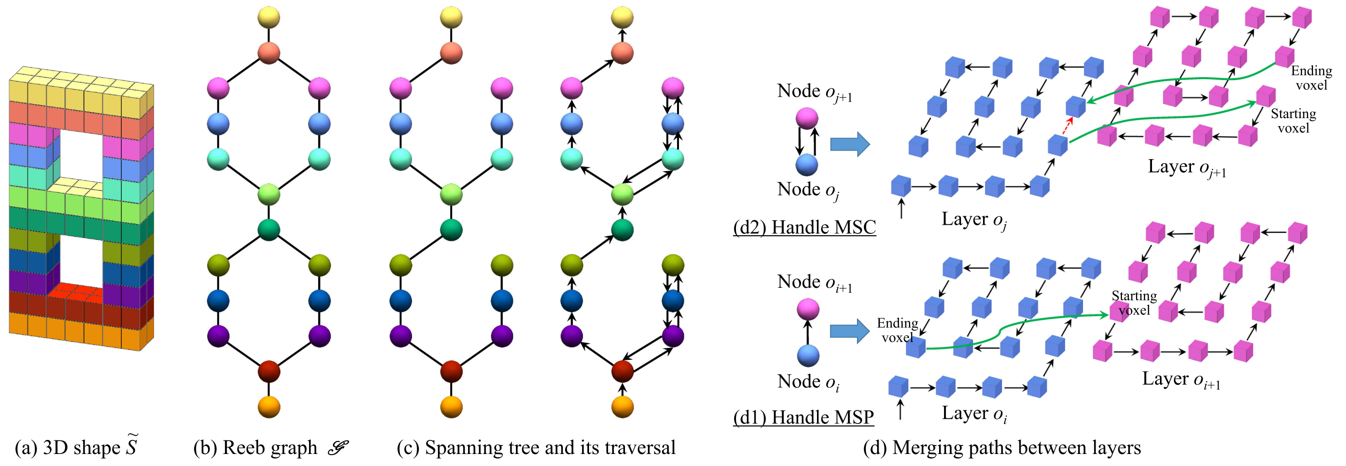
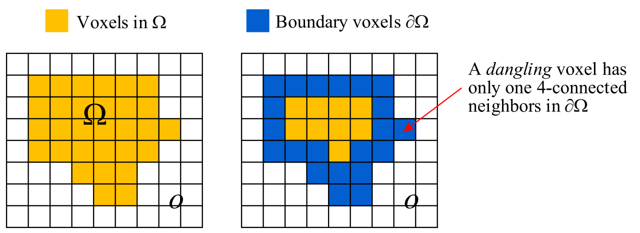


Fig. 4. Overview of the layer-based approximation algorithm for the longest path problem. Given an input 3D solid $\widetilde{S}$ represented by voxels (a), we take each connected component of a horizontal layer as a node and construct a Reeb graph $\mathcal{G}$, which encodes the topological structure of $\widetilde{S}$ (b). Applying depth-first search to $\mathcal{G}$, we obtain a spanning tree, which is a traversing path $p(\mathcal{G})$ (c). Then we compute a long planar path $p(o)$ for each layer o. Finally, we merge the planar paths into a single 3D path by considering two cases (MSP or MSC), depending on whether a node is visited once or twice during the spanning tree traversal (see (d)).



Voxels in Ω
Boundary voxels $\partial\Omega$
Ω
o
A *dangling* voxel has only one 4-connected neighbors in $\partial\Omega$
o

Fig. 5. Voxel types. Left: Ω is a set of connected voxels (shown in yellow) in a horizontal layer o. Right: the set of boundary voxels in Ω (shown in blue) is denoted by $\partial\Omega$. A dangling voxel is a boundary voxel with only one 4-connected neighbor.

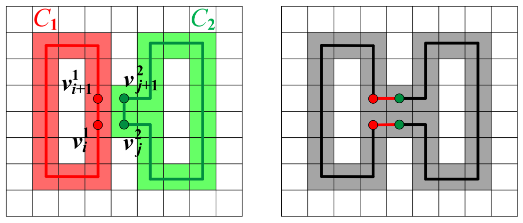


Fig. 6. The merge operation connects two mergeable adjacent cycles c_1 and c_2 (left) into one big cycle (right).

4-connectivity neighbor of v^1_i and v^2_{j+1} is a 4-connectivity neighbor of v^1_{i+1} (Figure 6, left).

We define the merge operation to join adjacent cycles c_1 and c_2 into one big cycle as illustrated in Figure 6, right. Note that if one of the two cycles is a path, a merge operation will join a cycle and a mergeable path into a longer path. We assume Ω° is connected.[3] Then the graph defined by 4-connectivity on voxels is also connected. We apply the algorithm in (Zhang and Liu 2011) to find an

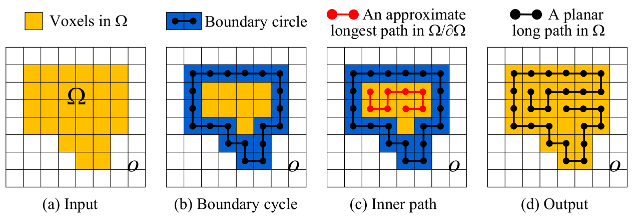


Fig. 7. Given a horizontal layer o (a), Algorithm 2 finds a long planar path (d) by connecting the boundary cycle $\partial\Omega$ (b) and an approximate longest path in Ω° (c). See the text for details.

approximate longest path in voxels Ω°. Then we merge the interior path and the boundary cycle into a single path, which is called a *long planar path* $p(o)$, where o is the layer in which Ω lies. We show a toy example in Figure 7 and present the pseudo-code in Algorithm 2.

After voxelizing the shape, we remove all dangling voxels so that all boundary voxels have two 4-connected neighbors. Denote the number of voxels in Ω and Ω° by n_Ω and n_{Ω°, respectively. We adopt Zhang and Liu's algorithm (2011), which is able to find a path of length at least $\frac{5}{6}n_{\Omega^\circ} + 2$ in $O(n^2_{\Omega^\circ})$ time. Since all the boundary voxels lie on the boundary cycle, we have

Lemma 4.1. *Algorithm 2 takes $O(n^2_\Omega)$ time to find a path $p(o)$ of length at least $\frac{5}{6}n_\Omega + 2$.*

Algorithm 2 takes a starting voxel and an ending voxel in Ω° as input, which are used to control the merging of paths in adjacent layers; see Section 4.3 for details.

4.2 Merging Paths between Layers

After determining a long planar path at each node $o_i \in \mathcal{G}$, we merge paths between adjacent nodes according to directed edge (o_a, o_b) in the traversing path $p(\mathcal{G})$ of the Reeb graph $\mathcal{G}$. We consider two cases (see Figure 4(d)):

[3] All mild assumptions that 3D models need to be satisfied are summarized in Section 4.4.

ALGORITHM 3: Generating layered path

```
input: A void-free 3D solid $\widetilde{S}$, a Reeb graph $\mathcal{G}$, and a traversing path $p(\mathcal{G})$
output: A layer-based long path $\widetilde{P}$, which links as many as possible voxels in $\widetilde{S}$
1:  Decompose $p(\mathcal{G})$ into a set of MSCs $\{C_i\}_{i=1}^K$ and an MSP.
2:  for each node $o_i$ in the MSP do
3:      Find $v_s(o_i)$ and $v_e(o_i)$ using the method in Section 4.3.1;
4:      Build a long planar path $p(o_i)$ with two endpoints $v_s(o_i)$ and $v_e(o_i)$ using Algorithm 2
5:  end for
6:  Merge all $p(o_i)$, $o_i \in MSP$, into a long path $\widetilde{P}$
7:  for each MSC in $\{C_i\}_{i=1}^K$ do
8:      for each node $o_j$ in $C_i$ do
9:          Find $v_s(o_j)$ and $v_e(o_j)$ using the method in Section 4.3.2;
10:         Build a long planar path $p(o_j)$ with two endpoints $v_s(o_j)$ and $v_e(o_j)$ using Algorithm 2
11:     end for
12:     Update $\widetilde{P}$ by merging all $p(o_j)$, $o_j \in C_i$, and merge them with $\widetilde{P}$
13: end for
14: return $\widetilde{P}$
```

— Case 1: there is only one directed edge (o_i, o_j) between nodes o_i and o_j in $p(\mathcal{G})$;
— Case 2: there are two directed edges (o_p, o_q) and (o_q, o_p) in $p(\mathcal{G})$.

We denote by $v_s(o)$ and $v_e(o)$ the starting and ending voxels of a long planar path $p(o)$ at a node o. The position of each voxel is represented by its center's integer coordinate (x, y, z). We adopt two strategies to merge the paths for the two different cases.

For Case 1, we require that $v_e(o_i)$ and $v_s(o_j)$ have the same horizonal coordinate (x, y) (Figure 4(d1)). Then, we simply add a link $(v_e(o_i), v_s(o_j))$ to connect paths $p(o_i)$ and $p(o_j)$.

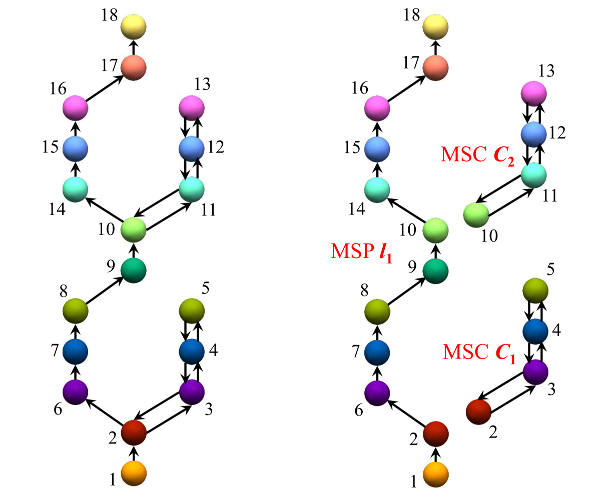
For Case 2, we require that (i) $v_s(o_q)$ and $v_e(o_q)$ are face-adjacent; (ii) each of $v_s(o_q)$ and $v_e(o_q)$ has a face-adjacent voxel in the layer o_p, denoted by v_1 and v_2 respectively; and (iii) there is an edge linking v_1 and v_2 in the path $p(o_p)$. Without loss of generality, we assume the linking direction is (v_1, v_2). We merge paths $p(o_p)$ and $p(o_q)$ by removing the link (v_1, v_2) from $p(o_p)$ and adding two directed edges $(v_1, v_s(o_q))$ and $(v_e(o_q), v_2)$ (Figure 4(d2)).

Pseudo-code is summarized in Algorithm 3. Note that the above merging strategies put some constraints on the position of starting and ending voxels in the path at each layer. In Section 4.3, we present a solution that satisfies these constraints.

4.3 Feasible Starting and Ending Voxels

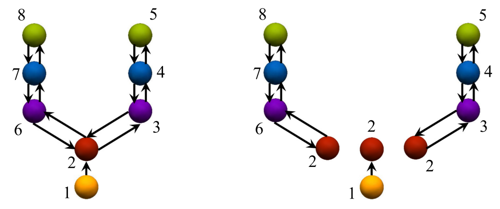
We decompose the traversing path $p(\mathcal{G})$ of the Reeb graph $\mathcal{G}$ into maximal sub-cycles and a maximal sub-path, such that feasible starting and ending voxels at the nodes of each maximal sub-cycle or sub-path can be found separately.

A *maximal sub-cycle* (MSC), denoted by C, is a closed walk (along directed edges) of $p(\mathcal{G})$ consisting of a maximal sequence of nodes starting and ending at the same node $o(C)$, with the constraint that $o(C)$ is visited exactly twice in C. Figures 8 and 9 show two examples of MSC.



(a) A traversing path $p(\mathcal{G})$ of the Reeb graph $\mathcal{G}$ (b) Decompose $p(\mathcal{G})$ into maximal sub-cycles and a sub-path

Fig. 8. (a) A traversing path $p(\mathcal{G}) = (o_1, o_2, o_3, o_4, o_5, o_4, o_3, o_2, o_6, o_7, o_8, o_9, o_{10}, o_{11}, o_{12}, o_{13}, o_{12}, o_{11}, o_{10}, o_{14}, o_{15}, o_{16}, o_{17}, o_{18})$. (b) Decompose $p(\mathcal{G})$ into two MSCs C_1 and C_2, and one MSP l_1. $C_1 = (o_2, o_3, o_4, o_5, o_4, o_3, o_2)$, $C_2 = (o_{10}, o_{11}, o_{12}, o_{13}, o_{12}, o_{11}, o_{10})$, and $l_1 = (o_1, o_2, o_6, o_7, o_8, o_9, o_{10}, o_{14}, o_{15}, o_{16}, o_{17}, o_{18})$.



(a) $p(\mathcal{G})$ (b) $p(\mathcal{G}) = \{C_1, C_2, l_1\}$

Fig. 9. The cycle $(o_2, o_3, o_4, o_5, o_4, o_3, o_2, o_6, o_7, o_8, o_7, o_6, o_2)$ in $p(\mathcal{G})$ is not a maximal sub-cycle, because o_2 is visited more than twice. The proper decomposition of $p(\mathcal{G})$ contains two MSCs C_1 and C_2, and one MSP l_1.

We denote the set of all nodes in an MSC C_i by $O(C_i)$ and define $\widetilde{O}(C_i) = O(C_i) \setminus \{o(C_i)\}$. Let $\{C_i\}_{i=1}^m$ be all MSCs in $p(\mathcal{G})$. Removing all the nodes in $\{\widetilde{O}(C_i)\}_{i=1}^m$ and related edges from $p(\mathcal{G})$, we call the remaining path the *maximal sub-path* (MSP) in $p(\mathcal{G})$. See Figures 8 and 9 for examples.

Our key observation is that finding feasible starting and ending voxels at each node can be realized in two steps (Algorithm 3):

— Step 1. Finding feasible starting and ending voxels at all nodes in the MSP (step 3 in Algorithm 3);
— Step 2. Finding feasible starting and ending voxels at all nodes in each MSC one by one (step 9 in Algorithm 3).

To describe Steps 1 and 2 in Sections 4.3.1 and 4.3.2, we define an orthogonal projection of a voxel's position $v = (x, y, z)$ onto a layer o_i (i.e., a horizonal plane $z = z_i$) as $\pi(v, z_i) = (x, y, z_i)$. For each layer, we assign a color to each voxel by mapping a two-color checkerboard based on voxels' contacted relations (Figure 10, left).

4.3.1 Handling MSP. We re-index all nodes in the MSP and sort them along the path as $\{o_j\}_{j=1}^{n_{MSP}}$. There is only one directed edge

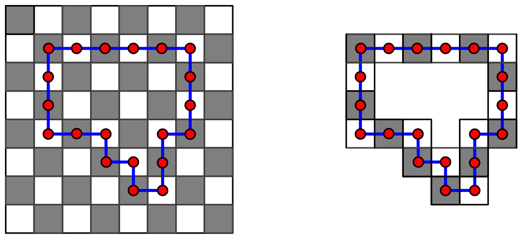


Fig. 10. Left: the 2-color checkerboard. Right: any boundary cycle (blue lines) has an even number of voxels (red dots), since any two adjacent boundary voxels are 4-connected neighbors and thus must be assigned to different colors.

$\{o_i, o_{i+1}\}$ between two sequential nodes o_i and o_{i+1} in the MSP. As mentioned in Section 4.2, we require that $v_e(o_i)$ and $v_s(o_{i+1})$ have the same horizonal coordinates (x, y); see also Figure 4(d1).

Starting from the first node o_1, we sequentially determine the starting and ending voxels at each node in the following way. For each layer o_i, $i = 1, 2, \ldots, n_{MSP}$, we determine its starting voxel by the ending voxel at o_{i-1}, i.e., $\pi(v_s(o_i), z_{i-1}) = v_e(o_{i-1}) = (x_{i-1}, y_{i-1}, z_{i-1})$. For $i = 1$, we randomly pick up a voxel in $\Omega^\circ(o_1)$ as the starting voxel. We denote the set of voxels at the node o as $\Omega(o)$. Let $I_i = \pi(\Omega^\circ(o_{i+1}), z_i) \bigcap \Omega^\circ(o_i)$ be the overlap between the projected interior $\pi(\Omega^\circ(o_{i+1}), z_i)$ and the interior $\Omega^\circ(o_i)$. We assume $I_i \setminus \{v_s(o_i)\} \neq \emptyset$ for all i. Then we randomly pick up a voxel in $I_i \setminus \{v_s(o_i)\}$ with a different color as $v_s(o_i)$ as the ending voxel.

Since two voxels form a building block for 3D printing (Figure 2), there must be an even number of voxels for each path.

Theorem 4.2 (Even Number of Voxels in MSP). *For each node in the MSP, the long planar path generated with the above specified starting and ending voxels has an even number of voxels.*

Proof. Since the adjacent voxels in the boundary cycle are 4-connected neighbors, the boundary cycle has an even number of voxels (thanks to the property of the 2-color checkerboard; see Figure 10). Also note that the starting and ending voxels have different colors, therefore, any path connecting them has an even number of voxels. □

4.3.2 *Handling MSCs.*

We can determine the starting and ending voxels in layers for each MSC C separately. Starting from the node $o(C)$, we traverse the cycle C and re-index the nodes in $\widetilde{O}(C) \setminus \{o(C)\}$ into $\{o_j\}_{j=1}^{n_{MSC}}$. As mentioned in Section 4.2, for each pair of nodes o_j and o_{j+1} in C, which are connected by two directed edges, we require that (1) $v_s(o_{j+1})$ and $v_e(o_{j+1})$ are contacted, and (2) each of $v_s(o_{j+1})$ and $v_e(o_{j+1})$ has a face-adjacent voxel in the layer o_j, which we denote as c_1 and c_2, and there is an edge linking c_1 and c_2 in the path $p(o_j)$; see also Figure 4(d2).

For each layer o_i, $i = 1, 2, \ldots, n_{MSC}$, we sequentially project two adjacent planar paths $p(o_{i-1})$ and $p(o_i)$ into a common plane and find the starting and ending voxels at o_{i-1} inside the overlapped segments of the two paths. In more details:

- $\pi(p(o_{i-1}), z_i)$ projects the long planar path $p(o_{i-1})$ at the node o_{i-1} onto the layer o_i. When $i = 1$, we set $o_0 = o(C)$. Let $\widetilde{I_i} = (\pi(p(o_{i-1}), z_i) \bigcap \Omega^\circ(o_i)) \setminus \{\pi(v_s(o_{i-1}), z_i), \pi(v_e(o_{i-1}), z_i)\}$. We assume $\widetilde{I_i} \neq \emptyset$ for all i. Then we randomly pick up two neighboring voxels in $\widetilde{I_i}$, which has a directed edge in $\pi(p(o_{i-1}), z_i)$ as the starting and ending voxels.

Theorem 4.3 (Even number of voxels in each MSC). *For each node in the MSC, the long planar path generated with the above-specified starting and ending voxels has an even number of voxels.*

Observe that the starting and ending voxels are 4-connected and they are of different colors. So the proof is almost identical to that of Theorem 4.2.

4.4 Theoretical Guarantees & Complexity Analysis

Algorithm 3 works under a few mild assumptions stated as follows:

- in each node o of the Reeb graph, the set of interior voxels Ω° are connected (Section 4.1);
- the dangling voxels have been removed (Section 4.1);
- $I_i \setminus \{v_s(o_i)\} \neq \emptyset$ for all i (Section 4.3.1); and
- $\widetilde{I_i} \neq \emptyset$ for all i (Section 4.3.2).

In practice, all the assumptions are easily met if the input $\widetilde{S}$ is a high resolution solid.

Theorem 4.4. *For a void-free solid $\widetilde{S}$ with n_S voxels satisfying the above assumptions, Algorithm 3 finds a long path $\widetilde{P}$ of length $l(\widetilde{P}) \geq \frac{5}{6} n_S + 2$ in $O(n_S^2)$ time. Moreover, the path $\widetilde{P}$ has an even number of voxels.*

Proof. The approximation ratio and the even number of voxels are the direct consequences of Lemma 4.1, and Theorems 4.2 and 4.3.

To analyze the time complexity, we note that building the Reeb graph takes $O(n_S)$ time and depth-first search takes $O(n_S)$ time to find the traversing path. It also takes $O(n_S)$ time to decompose the Reeb graph into MSCs and MSP. By Lemma 4.1, finding the starting and ending voxels and merging the paths into a layered path take $O(n_S^2)$ time. Putting it all together, Algorithm 3 runs in $O(n_S^2)$ time. □

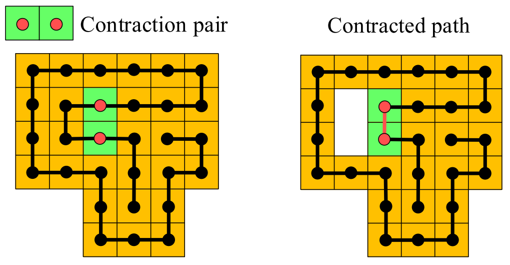
5 PATH TRIMMING

To transform shape $\widetilde{S}_1$ to $\widetilde{S}_2$ and vice versa, we trim their corresponding layered paths $\widetilde{P}_1$ and $\widetilde{P}_2$ into the same number of voxels. Let $n_{\widetilde{P}_1}$ and $n_{\widetilde{P}_2}$ be the number of voxels in $\widetilde{P}_1$ and $\widetilde{P}_2$, respectively. Without loss of generality, assume $n_{\widetilde{P}_1} > n_{\widetilde{P}_2}$. Our strategy is to shorten the longer path by removing "unnecessary" voxels. Here, the meaning of "unnecessary" is of course application-dependent. As our application aims at preserving appearance, removing interior voxels is acceptable.

To reduce the number of voxels in $\widetilde{P}$, we define *contraction pair* of voxels in $\widetilde{P}$, which satisfy the following conditions (see also the illustration in Figure 11(a)):

- the two voxels lie in the same layer o and are 4-connected; and
- the segment of the path inside o between them consists of all unnecessary voxels.

We define the *index* of a contraction pair as the number of voxels in the segment of the path inside o between the two voxels.



(a) Before contraction (b) After contraction

Fig. 11. Path contraction. (a) A contraction pair of index 2. (b) The contracted path with the path length reduced by 2.

Due to the property of two-color checkerboard, all indices are even numbers.

Given a contraction pair, we can contract the path by directly linking the voxels in pair and removing the segment of path between the pair (Figure 11(b)). After contraction, the length of the path is reduced by the index of the contraction pair.

Applying a linear scan on the paths, we find all possible contraction pairs in $\widetilde{P}_1$ and $\widetilde{P}_2$, and put their indices into two sets I_1 and I_2, respectively. In most cases, there are enough contraction pairs of index 2 in $\widetilde{P}_1$ (whose sum equals to $n_{\widetilde{P}_1} - n_{\widetilde{P}_2}$). Therefore, we simply trim $\widetilde{P}_1$ into P_1 such that $n_{P_1} = n_{\widetilde{P}_2}$. If this simple trimming scheme fails in rare cases, we find minimum subsets $I'_1 \subset I_1$ and $I'_2 \subset I_2$, such that $\sum_{i \in I'_1} i - \sum_{j \in I'_2} j = n_{\widetilde{P}_1} - n_{\widetilde{P}_2}$. As a variant of the subset sum problem, it can be solved in pseudo-polynomial time by dynamic programming (Martello and Toth 1990).

6 COLLISION-FREE MOTION PLANNING

After trimming, all layered paths $\{P_i\}_{i=1}^m$ have the same number of building blocks. Now we compute the collision-free unfolding sequence for each shape so that all paths can be straightened into a common line configuration. Since the unfolding sequence is collision-free and reversible, it allows us to transform one layered path into the other and vice versa.

Thanks to its layered structure, the output of Algorithm 3 can be mapped to a planar linkage of rigid bars connected at flexible joints (O'Rourke 2011). It is known that any tangled but non-crossing planar polygonal linkages (i.e., polygonal chains) can be straightened under a sequence of non-colliding motions (Connelly et al. 2003; Streinu 2000). Therefore, to lineup the layered path, we need to map it to a planar domain. We present such a simple yet novel mapping below.

Recall that each layer in the path P is horizontal, i.e., its z coordinates are a constant. We refer to Figure 12. For each voxel $v = (x, y, z)$ in P, we project it onto the xz plane to obtain a square (x, z). We obtain a projected 2D path, denoted as $P_{xz} = \Pi_{(x,z)}(P)$, by mapping the square centers as flexible joints and placing a rigid bar between any two contiguous joints along the path P. The 2D path P_{xz} of joints and bars corresponds to a planar polygonal linkage of rigid bars. We call a joint of P_{xz} *composite* if it corresponds to two or more voxels in P. Due to composite joints, some rigid bars may overlap but they do not cross.

ALGORITHM 4: *Collision-free motion planning*

```
input: A shape $S$ with a layered path $P$, in which the $z$ coordinates of
    each layer are a constant
output: A collision-free motion sequence that unfolds $S$ into a line
    configuration
1: Project the path $P$ into a 2D path $P_{xz} = \Pi_{(x,z)}(P)$
2: Apply the energy-driven approach (Cantarella et al. 2004) to unfold
   $P_{xz}$ into a line configuration $l_{xz}$; denote the obtained motion
   sequence as $\Upsilon_{xz}$
3: Apply the rigid body transformation to make $l_{xz}$ parallel to $z$-axis in
   the $xz$ plane
4: Compute the projected 2D path $P_{yz} = \Pi_{(y,z)}(\Pi^{-1}_{(x,z)}(l_{xz}))$
5: Apply the approach (Biedl et al. 2001) to unfold $P_{yz}$ into a line
   configuration; denote the obtained motion sequence as $\Upsilon_{yz}$
6: return $\Upsilon_{xz} \cup \Upsilon_{yz}$
```

We apply the simple yet effective energy-driven approach (Cantarella et al. 2004) to straighten planar polygonal linkages. The method is based on the gradient flow of a repulsive energy function and its output motion is free from self-intersection. Note that in the original definition of linkage, rigid bars do not have a width, so they can rotate with each other in any angle. However, the width of the unit square requires that the angle between two joined bars must be in $[\frac{\pi}{2}, \pi]$ for avoiding self-intersection. Fortunately, given a valid initial configuration of P_{xz}, the energy-driven approach (Cantarella et al. 2004) never increases the angle between joined bars and thus can apply to our case.

It is worthy of pointing out that the energy-driven approach starts with a valid polygonal linkage, i.e., a planar open chain without self-intersection. Our projected 2D path $P_{xz} = \Pi_{(x,z)}(P)$ can be regarded as a degenerate open chain, in which some rigid bars overlap due to composite joints. To remove degeneracy, we can apply infinitesimal perturbations on the joints of P_{xz} (Edelsbrunner and Mücke 1990). If there exists such a perturbation to make P_{xz} a general (i.e., non-degenerate) valid polygonal linkage (e.g., Figure 12(b)), the energy-driven approach will successfully output an unfolding motion free from self-intersection. However, in many cases, such a self-intersection-free perturbation does not exist; an example is shown in Figure 13. Fortunately, the energy-driven approach still works in our situation due to the following key observation. For each joint in P_{xz}, we assign to it a depth attribute, which is the y coordinate value of that joint in P. According to different attributes, we can decompose P_{xz} into a set of subpaths: all joints in each subpath have the same attribute. See Figure 14 for an example. A subpath may have several disjoint components, but they must be free of self-intersection. Then the same proof in Cantarella et al. (2004) can be easily adapted to each subpath and show that the output straightening motion is free of self-intersection.

After straightening P_{xz}, up to rigid body motion, we can assume it is a line parallel to the z-axis in the xz plane. Then the mapping $P_{yz} = \Pi_{(y,z)}(\Pi^{-1}_{(x,z)}(P_{xz}))$ is again a planar polygonal linkage on the yz plane (Figure 12(d)). Applying the approach of Cantarella et al. (2004) again on P_{yz}, we can finally unfold the 3D path P into a straight line configuration. In the non-colliding motion generated by Cantarella et al. (2004), each step takes $O(n_P^2)$ time, and the

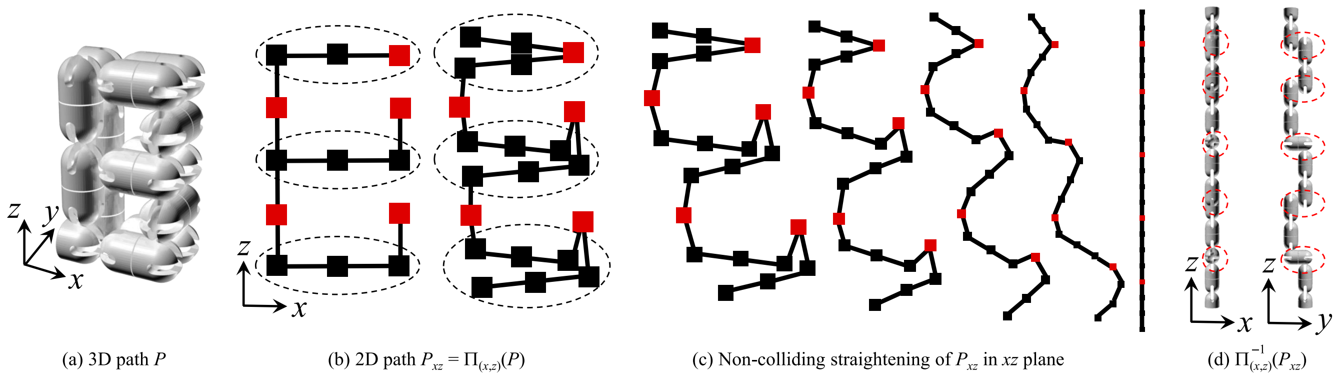

z
y
x
z
x
z
x
z
y
(a) 3D path P
(b) 2D path $P_{xz} = \Pi_{(x,z)}(P)$
(c) Non-colliding straightening of P_{xz} in xz plane
(d) $\Pi^{-1}_{(x,z)}(P_{xz})$

Fig. 12. Unfolding a layered path to a straight line. Projecting the 3D layered path P (a) to the xz plane, we obtain a 2D path $P_{xz} = \Pi_{(x,z)}(P)$, in which each voxel $v = (x, y, z)$ becomes a unit square (x, z) (b). Due to composite squares (colored in red in (b) and (c), and circled in (d)), some rigid bars (inside the dashed circles) are overlapping. We apply the energy-driven approach (Cantarella et al. 2004) to straighten the 2D path P_{xz} in the xz plane (c). The 3D path by inversely mapping $\Pi^{-1}_{(x,z)}(P_{xz})$ is shown in (d), which again is a planar polygonal linkage on the yz plane. All transformations are rigid motions and they are physically feasible. In (c) and (d), we scale down the linkages in order to fit the space.

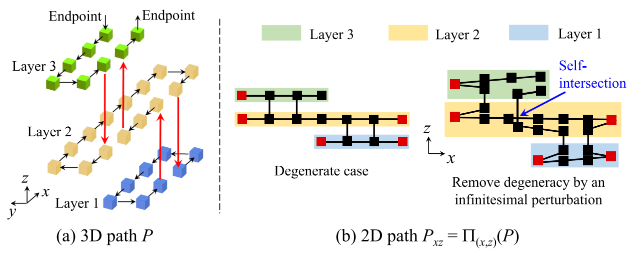


Fig. 13. Self-intersection of a projected 2D path. (a) A 3D path P consisting of three layers. (b) The projected 2D path $P_{xz} = \Pi_{(x,z)}(P)$, which can be regarded as a degenerate open chain with some overlapping rigid bars due to composite joints (shown in red square). By applying infinitesimal perturbations on the joints of P_{xz}, the degeneracy can be removed; and in this case, a self-intersection occurs inevitably in the projection plane.

number of steps is a polynomial of n_P, where n_P is the number of blocks in P. The pseudo-code is presented in Algorithm 4. In fact, we can use a much faster method (Biedl et al. 2001) to straighten P_{yz} (Step 5 in Algorithm 4), which takes $O(n_P)$ rotations in $O(n_P)$ time. Note that the method (Biedl et al. 2001) requires that the 3D path P has a simple orthogonal projection on a plane, e.g., P_{yz} on the yz plane, which is only satisfied on P_{yz} but not P_{xz} (before applying the method (Cantarella et al. 2004)).

7 EXPERIMENTS AND DISCUSSION

We implemented the LineUp algorithm in C++ and evaluated it on a workstation with 2 Intel Xeon E5-2698V3 CPUs and 128GB RAM running Windows. The line configuration output from our method can be directly used for 3D printing. Diverse 3D shapes with various geometric features and topological complexity are tested.

Physical prototype. Figure 1 shows three example models (i.e., Bunny, Cannon, and Decocube) generated by Algorithm 1, all of which have 754 blocks (corresponding to 1,508 voxels) and can be transformed into each other continuously. We fabricate the common line configuration of these shapes on an iSLA-650 Pro Stereolithography 3D Printer, as shown in Figure 15. To fix the

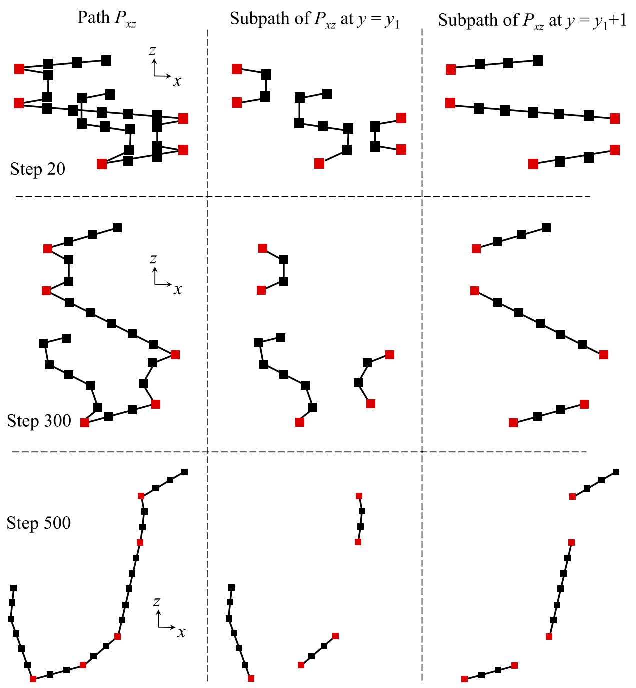

Path P_{xz}
Subpath of P_{xz} at $y = y_1$
Subpath of P_{xz} at $y = y_1+1$
z
x
Step 20
Step 300
Step 500

Fig. 14. Straightening of 2D path P_{xz} shown in Figure 13 and their subpaths at different depth values y by the approach of Cantarella at el. (2004).

positions of building-blocks during physical shape transformation, the surface of building-blocks are covered by thermosol and strong velcros—this is motivated by the work of a reconfigurable robot called *YaMoR* (Moeckel et al. 2006). The computer animation of collision-free motion sequence for unfolding the shapes into the common line configuration can be found in the supplementary video.

In any folded 3D model, the physical strength of the structure depends on the maximal stresses that can be supported by all building blocks. There are three different types of maximal stresses in our application:

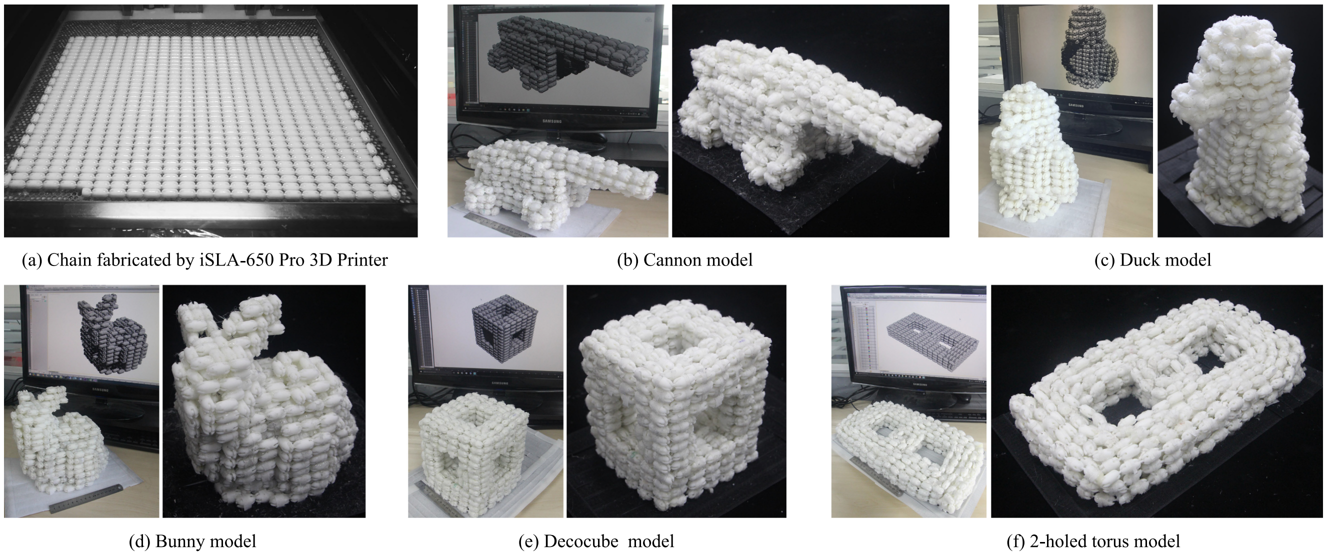


Fig. 15. Experimental tests to form the shapes of five models by a chain-structure, where models can be physically transformed into each other. See supplementary video for transformation process of physical folding.

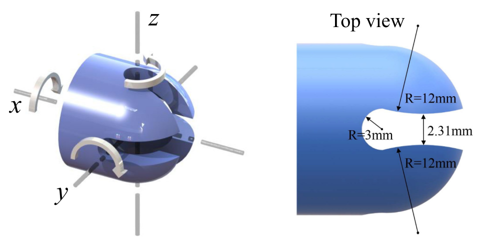


Fig. 16. For each hemispherical-end in a building block, four slots are used to facilitate the end to rotate with respect to different orientations of the hinge. The two sides of each slot are arcs with a large radius, such that the narrowest width between two sides is slightly smaller than the diameter of the circular arcs at the end of the slot. Therefore, a certain force is needed to drive the hinge to go through the slot by making elastic deformation of the slot's sides.

— *The normal compressive stress*: it depends on the 3D printing material. We use photopolymer resin UV9400 whose compressive strength is 38–56MPa.
— *The normal tensile stress*: it depends on the strength of thermosol and strong velcros between the two building blocks.
— *Critical buckling stress*: when the building block with the orientation as shown in Figure 17 (see critical buckling stress test) is subjected to compressive stress, buckling may occur. This is characterized by a sudden sideways deflection of the hinge connecting two cylinders with hemispherical-end (Figure 2(a)). As shown in Figure 16, we design the slot's width to be slightly smaller than the diameter of the circular arcs at the end of the slot, so that a certain force is needed to drive the hinge to go through the slot. This force, together with the thermosol, devotes to the critical buckling load.

To evaluate the normal tensile stress σ_{max} and the critical buckling force F_c, we test the fabricated build blocks on a Zwick/Roell Z020 universal testing system (Figure 17, left) with the results $\sigma_{max} = 206.7KPa$ and $F_c = 2042N$. We also evaluate the maximal stress in each of the five folded 3D models designed by our method. The finite element analysis software Abaqus (version 6.12) is used. The evaluation results as shown in Figure 17 indicate that the folded models generated by our method have enough physical strength.

Physical transformation. Thanks to the layered path, the motion planing algorithm can guarantee the existence of a collision-free motion sequence, which can be computed in polynomial time. In practice, one can further improve the performance by a simple heuristic. For each path P_i representing a fabricated shape S_i, we set the starting and ending blocks of P_i at the top layer. Then the shape S_i can be straightened into a straight line configuration by pulling the blocks upward, and the pulled blocks are propagated progressively from the starting and ending blocks to the remaining blocks along the path P_i. Due to the internal support provided by the hinges in each block (xy directions) and strong velcros (z direction), this unfolding process is physically invertible, such that each physical shape can be folded from a line configuration—see the supplementary video for details.

We observe it is challenging for novice users, especially children, to unfold a line configuration into a desired 3D shape. To assist them, we provide an operation manual for each 3D model. The user manual documents the sequential indices of the building blocks in the physical chain, $(B_0, B_1, \ldots, B_i, \ldots, B_{754})$, and shows the global coordinate system to the first cylinder with hemispherical-end $B_1(\omega_1)$ (Figure 2(b)). We iteratively show the global orientation of $B_{i+1}(\omega_1)$ in the manual, $i = 1, 2, \ldots, 754$, which is determined by the relative orientation of $B_i(\omega_2)$ and $B_{i+1}(\omega_1)$ (they are printed as a whole) with resect to $B_i(\omega_1)$. It is worthy to note that this unfolding process calls up childrens' and teenagers' distinct cognitive processes to analyze a spatial representation to make a particular structure by structured block building. This practice can improve their spatial cognition that is

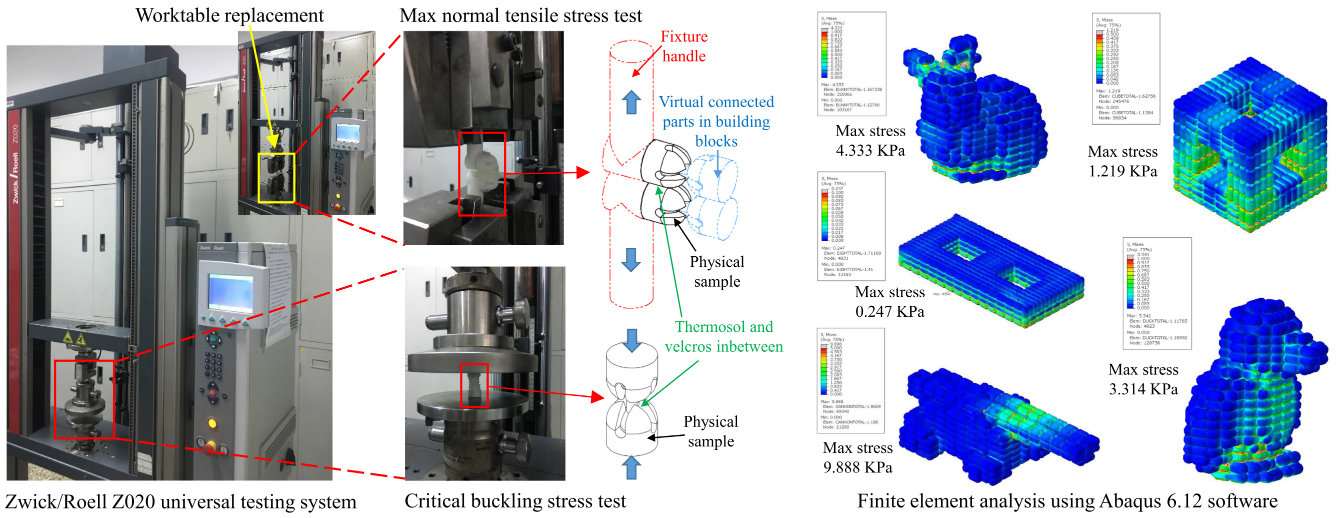

Worktable replacement
Max normal tensile stress test
Fixture handle
Virtual connected parts in building blocks
Physical sample
Thermosol and velcros inbetween
Physical sample
Max stress 4.333 KPa
Max stress 1.219 KPa
Max stress 0.247 KPa
Max stress 3.314 KPa
Max stress 9.888 KPa
Zwick/Roell Z020 universal testing system
Critical buckling stress test
Finite element analysis using Abaqus 6.12 software

Fig. 17. Physical strength test on building blocks and maximal stress estimation by finite element analysis. Left: Zwick/Roell Z020 universal testing system was used in our tests. For building blocks, the maximal normal tensile stress is 206.7KPa and the critical buckling load is 2042N. Right: Abaqus 6.12 software was used to analyze the stress distribution in five models, i.e., Bunny ($\sigma_{max} = 4.333$KPa), Cannon ($\sigma_{max} = 9.888$KPa), Decocube ($\sigma_{max} = 1.219$KPa), Duck ($\sigma_{max} = 3.314$KPa), and 2-holed torus ($\sigma_{max} = 0.247$KPa), with σ_{max} denoting the maximal stress. All the stresses and force loads applied to each building block in all five models are less than the affordable maximal normal tensile stress and the critical buckling load.

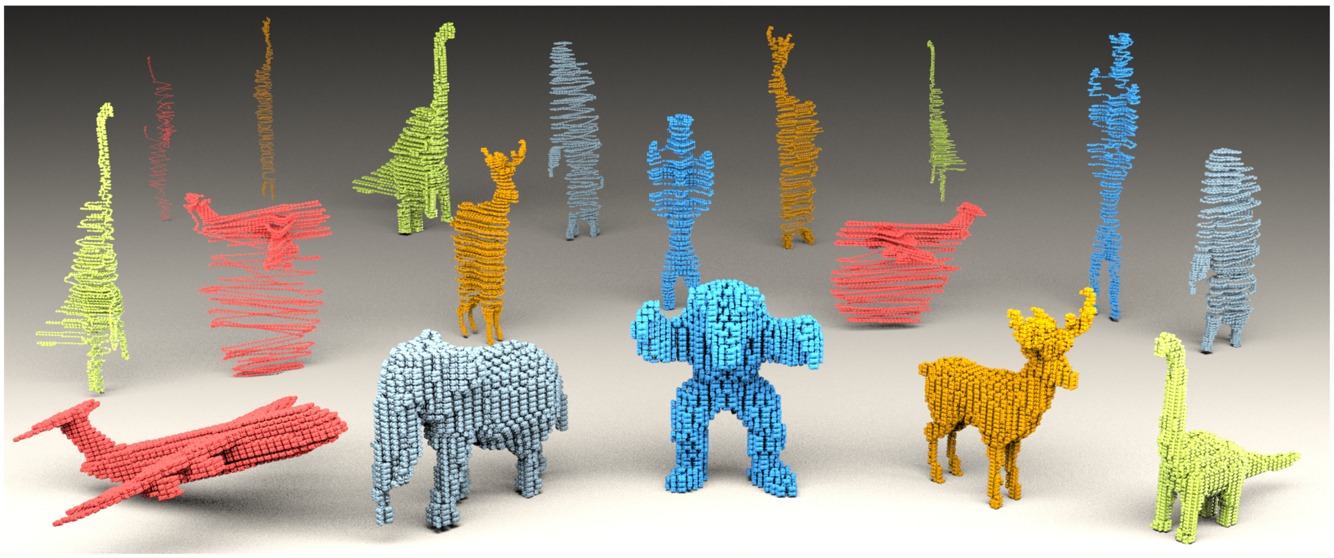


Fig. 18. The physical world consisting of 3D models that can be transformed by folding a common line configuration, including Armadillo, Dinosaur, Aircraft, Elephant, and Deer, and some of their folded versions during the transformation process. See supplementary video for detailed transformation processes.

associated with creative design processes (Stiles and Stern 2001; Yu et al. 2018).

Incremental design. Algorithm 1 can be easily adapted to incremental design. For example, for the line configuration L shown in Figure 1, which can be folded into three shapes, we can incrementally design the fourth and fifth shapes such that L can be folded into them (see Figure 15). In this example, we incrementally add the 2-holed torus and Duck models into the shape pool by the following steps:

(1) Voxelize each model into the shape with n_{new} voxels; n_{new} is slightly larger than 1,508;
(2) Generate a path $\widetilde{P}_{new}$ that preserves the appearance of $\widetilde{S}_{new}$ and go through as many interior voxels as possible;
(3) Trim the path $\widetilde{P}_{new}$ into the path P_{new}, which passes through exactly 1,508 voxels;
(4) Plan a collision-free motion that unfolds P_{new} into L.

Complex examples. Our method is automatic and efficient. More complex transformable shapes can be easily generated. Figure 18

Table 2. Statistic Data of Generating Two Sets of Physically Achievable Transformations {Bunny, Decocube, Cannon, 2-holed torus, Duck} and {Armadillo, Dinosaur, Aircraft, Elephant, Elk}

Model name	Voxelization		Build $\widetilde{P}$		Build P		Motion planning	
	#voxels	time	#voxels	time	#voxels	time	time/step	#steps
Bunny	1643	0.19s	1536	1.40s		0.03s	0.72s	240K
Decocube	1868	0.43s	1504	0.68s		0.03s	0.12s	240K
Cannon	2105	0.59s	1508	0.74s	1,508	0.04s	0.15s	240K
2-hole torus	1520	0.42s	1512	0.38s		0.06s	0.09s	80K
Duck	1586	0.40s	1484	0.70s		0.02s	0.36s	240K
Armadillo	6,280	0.48s	5,898	11.86s		0.12s	1.80s	70K
Dinosaur	6,522	0.34s	5,966	11.50s		0.14s	1.10s	150K
Aircraft	7,088	0.99s	6,482	27.47s	5,890	0.19s	0.90s	180K
Elephant	6,478	0.55s	5,892	11.94s		0.01s	1.32s	180K
Elk	6,425	0.42s	5890	20.42s		0.13s	2.04s	180K

We report the numbers of voxels (#voxels), the number of steps (#steps), time (seconds) for voxelization, the layer-based path $\widetilde{P}$ generation (Build $\widetilde{P}$), the trimmed path P generation (Build P), and non-collision motion planning for path straightening.

shows the transformations generated for the other five models (i.e., Armadillo, Dinosaur, Aircraft, Elephant, and Elk), each of which have 2,945 connected components—corresponding to 5,890 voxels. They all can be transformed into each other by folding. The details of collision-free motion planning of these shapes for unfolding into a line are shown in the supplementary video. The examples shown in Figures 1, 15, and 18 demonstrate that when the number of voxels increases, the appearance of models that can be transformed into each other are more and more realistic. The statistic data of generating these two sets of shapes are summarized in Table 2.

Comparison with Boxelization. Boxelization (Zhou et al. 2014) is the state-of-the-art method that can physically fold 3D shapes of complex geometry into a box. Compared to Boxelization, our method has three unique advantages:

—Our method can transform one shape into multiple targets, rather than a single box.
—Our method outperforms Boxlization in terms of running time. Note that Boxelization represents a shape by a tree structure of voxels and takes an exhaustive search on a graph for finding a tree structure that fits two shapes. Since the search is carried out by simulated annealing, which takes theoretically exponential time with lower bound $\Omega(4^{\lfloor \frac{n}{16} \rfloor})$ (Sasaki 1987), Boxelization is time consuming. For solids with 48 to 125 voxels, it takes up to a thousand hours to find a feasible solution using a cloud computing service. In contrast, our method represents shapes by a path, and with mild assumptions. Our method is guaranteed to terminate in $O(n^2)$ time. Therefore, we can efficiently handle shapes with higher resolution.
—The interactive folding strategy with user interference in Boxelization cannot guarantee to be always feasible, while, thanks to the layered path, our method is guaranteed to always have a collision-free motion planning for straightening.

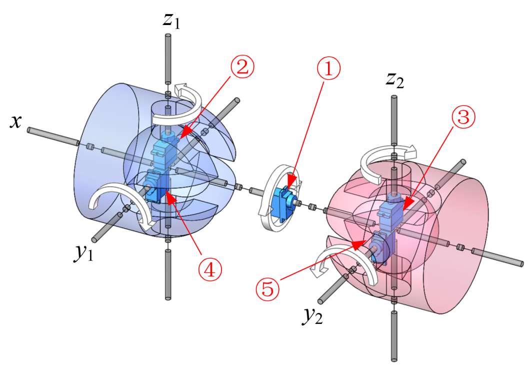


Fig. 19. An SRMR module design for realizing the rotation mechanism in a building-block. Following SuperBot (Salemi et al. 2006), a central rotation part is used, which uses a motor ① to carry out the rotation around the x-axis. Following EasySRRobot (Yu et al. 2017), two motors ② and ③ are used to carry out the rotation around the y_1- and y_2-axes, respectively, and the other two motors ④ and ⑤ are used to carry out the rotation around the z_1- and z_2-axes, respectively. Following M-TRAN II (Kurokawa et al. 2003), the electromagnetic connector is used. Overall, we can use the method of Yu et al. (2017) to design an optimal structure for assembling all these components.

We would like to also point out that the reason that LineUp is much more computationally efficient than Boxlization is due to the use of building blocks that have full rotational degrees of freedom.

8 APPLICATIONS

This section discusses a few applications of LineUp.

Spatial ability training: Spatial ability (a.k.a. visuo-spatial ability) is a category of human reasoning skills that plays an important role in affecting a child's or teenager's development in science, technology, engineering, and mathematics (Wai et al. 2009). Manually transforming a 3D physical model from one shape into the other by using the 3D printed chain is a task related to mental rotation and mental folding, which are widely used to evaluate spatial ability (Harris et al. 2013). LineUp provides a tangible interaction, through which users can improve their spatial ability by performing the reconfiguration task (Yu et al. 2018).

Self-reconfigurable modular robots (SRMRs): Our method sheds light upon the reconfiguration planning of SRMRs (Stoy et al. 2010). An SRMR is usually constructed by modules and a widely studied type is double-cube modules (e.g., M-TRAN III (Kurokawa et al. 2008), SuperBot (Salemi et al. 2006), iMobot (Ryland and Cheng 2010), and EasySRRobot (Yu et al. 2017)). Each module is equipped with actuators, sensors, inter-module communication/power-transmission devices, and microprocessors. As a result, the rotation of a module can be controlled by computer programs and any two modules can be connected via electronic/magnetic connectors or strong velcros. In particular, we can use a combination of the mechanical structure of SuperBot, the electromagnetic connector of M-TRAN II (Kurokawa et al. 2003), and the optimal design methodology in EasySRRobot to design a new SRMR module to realize all rotation functions in a building-block (see Figure 19). The details of mechano-electronics will be implemented in our future work. By replacing the building-blocks of a 3D printed chain

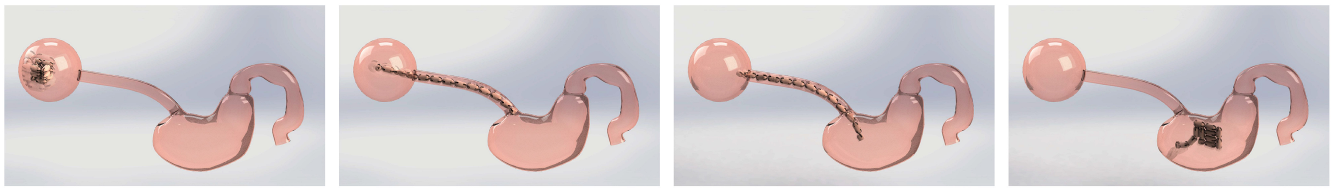


Fig. 20. Micro-robotic surgery using transforming structure generated by LineUp. A folded endoscopic device was placed into the mouth of a patient. One can unfold the device into a line configuration and then deliver it through the esophagus. After reaching the stomach, it can be folded into another endoscopic structure and is ready for operation. See also the accompanying video.

with the new SRMR modules, the connected modules can automatically transforms the robot's shape into any other one in the shape pool, behaving akin to the robots in the featured movies such as *Transformers*. The current bottleneck of autonomous transformation taken on these SRMRs is the small power-to-weight ratio on actuators. Nevertheless, this problem does not exist in the environment with small or even no gravity such as underwater or in outer space.

Transforming structures: Transforming structures (a.k.a. reconfigurable or folding structures) have received considerable attention in computational geometry (Demaine et al. 2002), computer graphics (summarized in Section 2.3), and robotics field (Rubenstein et al. 2014; You 2014). Common configurations include chains (i.e., tree structures), lattices, and hybrid (Stoy et al. 2010). We develop an algorithm that allows physical transformation among 3D models with different geometry and topology by using lines as the intermediate configuration. Such a line configuration has two unique advantages that are not available on other configurations in robotic applications: (1) A line configuration can be easily driven by a single actuator placed at the head as a snake, and (2) it can pass through narrow holes/channels for applications with limited spaces. We list three application scenarios that can benefit from such transforming structures as follows.

— *Micro-robotic surgery* (e.g., endoscopy): Tools in the configurations and motion computed by LineUp can be used in the surgery with the help of an endoscopic-type device. For example, through the channel placed in the patient's mouth and esophagus, the chain can be folded into a new structure/shape for operations in the stomach. An example has been illustrated in Figure 20. This idea can be similarly applied to other minimally invasive surgery.
— *Archeology*: One can form the probing device using LineUp and send it inside the target (e.g., sealed tomb) through a tiny hole. After that, it can be transformed into a probing device.
— *Aerospace*: To send equipment to a space station, it is highly demanded to pack it in a compact form so that it can fit into a small capsule of the launch vehicle. LineUp can also help to generate ideal forms.

More demonstrations can be found in the accompanying video.

9 CONCLUSIONS & FUTURE WORK

We propose the LineUp algorithm that can physically transform 3D models into a chain structure, thereby allowing the n-ary transformation between 3D models with different shape and topology. Our technical contributions include (1) a novel method to convert voxel-based void-free solid into a simple path that goes through most of the voxels and (2) a collision-free motion planning method that can fold/unfold the model physically. Our methods are theoretically sound and computationally efficient. We demonstrated the efficacy of our method via both computer simulation and 3D printing.

Shape orientation plays a critical role in LineUp. This is mainly because the Reeb graph is sensitive to orientation. Obviously, different orientations can lead to different traversing paths of the Reeb graph, which, in turn, produces different MSP and MSC decompositions. We experience that the fewer number of MSCs in a shape, the easier the physical transformation can be achieved. In our current implementation, the upright orientation is specified by the user. In the future work, we will formulate an optimization framework to find the optimal orientation. Our main objective of this work is to develop a *general* approach to enable transforming between a wide range of 3D models via line configurations. Therefore, we adopt the voxel-based representation, which sacrifices the visual appearance of models. A naïve way to improve the visual appearance is to increase the resolution with the additional cost of computing and manufacturing. Another possible improvement to balance the cost and visual appeal is to compute a shell of each object with varying thickness (Musialski et al. 2015). These alternatives will be explored in our future work.

ACKNOWLEDGMENTS

The authors thank Mr. Guoxin Fang to help prepare the finite element analysis using Abaqus 6.12 software.

REFERENCES

Moritz Bächer, Bernd Bickel, Doug L. James, and Hanspeter Pfister. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph.* 31, 4, Article 47 (2012), 47:1–47:9.
Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. 2014. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Trans. Graph.* 33, 4, Article 96 (2014), 96:1–96:10.
Bernd Bickel, Moritz Bächer, Miguel A. Otaduy, Hyunho Richard Lee, Hanspeter Pfister, Markus Gross, and Wojciech Matusik. 2010. Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graph.* 29, 4, Article 63 (2010), 63:1–63:10.
Bernd Bickel, Paolo Cignoni, Luigi Malomo, and Nico Pietroni. 2018. State of the art on stylized fabrication. *Computer Graphics Forum* (2018), DOI: 10.1111/cgf.13327.
T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, I. Streinu, G. Toussaint, and S. Whitesides. 2001. Locked and unlocked polygonal chains in three dimensions. *Discrete Comput. Geom.* 26, 3 (2001), 269–281.
Jacques Calì, Dan A. Calian, Cristina Amati, Rebecca Kleinberger, Anthony Steed, Jan Kautz, and Tim Weyrich. 2012. 3D-printing of non-assembly, articulated models. *ACM Trans. Graph.* 31, 6, Article 130 (2012), 130:1–130:8.

Jason H. Cantarella, Erik D. Demaine, Hayley N. Iben, and James F. O'Brien. 2004. An energy-driven approach to linkage unfolding. In *Proceedings of the 20th Annual Symposium on Computational Geometry (SCG'04)*. 134–143.

Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. 2011. Probabilistic reasoning for assembly-based 3D modeling. *ACM Trans. Graph.* 30, 4, Article 35 (2011), 35:1–35:10.

Siddhartha Chaudhuri and Vladlen Koltun. 2010. Data-driven suggestions for creativity support in 3D modeling. *ACM Trans. Graph.* 29, 6, Article 183 (2010), 183:1–183:10.

Robert Connelly, Erik D. Demaine, and Günter Rote. 2003. Blowing up polygonal linkages. *Discrete Comput. Geom.* 30, 2 (2003), 205–239.

Erik D. Demaine, David Eppstein, Jeff Erickson, George W. Hart, and Joseph O'Rourke. 2002. Vertex-unfoldings of simplicial manifolds. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry (SCG'02)*. 237–243.

Noah Duncan, Lap-Fai Yu, and Sai-Kit Yeung. 2016. Interchangeable components for hands-on assembly based modelling. *ACM Trans. Graph.* 35, 6, Article 234 (2016), 234:1–234:14.

Herbert Edelsbrunner and Ernst Peter Mücke. 1990. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.* 9, 1 (1990), 66–104.

Anatolij T. Fomenko and Tosiyasu Kunii. 1997. *Topological Modeling for Visualization.* Springer.

Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. 2004. Modeling by example. *ACM Trans. Graph.* 23, 3 (2004), 652–663.

Akash Garg, Alec Jacobson, and Eitan Grinspun. 2016. Computational design of reconfigurables. *ACM Trans. Graph.* 35, 4, Article 90 (2016), 90:1–90:14.

Ruslan Guseinov, Eder Miguel, and Bernd Bickel. 2017. CurveUps: Shaping objects from flat plates with tension-actuated curvature. *ACM Trans. Graph.* 36, 4, Article 64 (2017), 64:1–64:12.

Justin Harris, Kathy Hirsh-Pasek, and Nora S. Newcombe. 2013. Understanding spatial transformations: Similarities and differences between mental rotation and mental folding. *Cognit. Process.* 14, 2 (2013), 105–115.

Kailun Hu, Shuo Jin, and Charlie C. L. Wang. 2015. Support slimming for single material based additive manufacturing. *Comput.-Aided Des.* 65 (2015), 1–10.

Qixing Huang, Vladlen Koltun, and Leonidas Guibas. 2011. Joint shape segmentation with linear programming. *ACM Trans. Graph.* 30, 6, Article 125 (2011), 125:1–125:12.

Yi-Jheng Huang, Shu-Yuan Chan, Wen-Chieh Lin, and Shan-Yu Chuang. 2016. Making and animating transformable 3D models. *Comput. Graphics* 54 (2016), 127–134.

Conrado R. Ruiz Jr., Sang N. Le, Jinze Yu, and Kok-Lim Low. 2014. Multi-style paper pop-up designs from 3D models. *Comput. Graphics Forum* 33, 2 (2014), 487–496.

Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph.* 31, 4, Article 55 (2012), 55:1–55:11.

David Ron Karger, Rajeev Motwani, and G. D. S. Ramkumar. 1997. On approximating the longest path in a graph. *Algorithmica* 18, 1 (1997), 82–98.

Martin Kilian, Simon Flöry, Zhonggui Chen, Niloy J. Mitra, Alla Sheffer, and Helmut Pottmann. 2008. Curved folding. *ACM Trans. Graph.* 27, 3, Article 75 (2008), 75:1–75:9.

Mina Konaković, Keenan Crane, Bailin Deng, Sofien Bouaziz, Daniel Piker, and Mark Pauly. 2016. Beyond developable: Computational design and fabrication with auxetic materials. *ACM Trans. Graph.* 35, 4, Article 89 (2016), 89:1–89:11.

Bernhard Korte and Jens Vygen. 2006. *Combinatorial Optimization: Theory and Algorithms.* 3rd Edition, Springer.

Haruhisa Kurokawa, Akiya Kamimura, Eiichi Yoshida, Kohji Tomita, Shigeru Kokaji, and Satoshi Murata. 2003. M-TRAN II: Metamorphosis from a four-legged walker to a caterpillar. In *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.* 2454–2459.

Haruhisa Kurokawa, Kohji Tomita, Akiya Kamimura, Shigeru Kokaji, Takashi Hasuo, and Satoshi Murata. 2008. Distributed self-reconfiguration of M-TRAN III modular robotic system. *The International Journal of Robotics Research* 27, 3–4 (2008), 373–386.

Timothy Langlois, Ariel Shamir, Daniel Dror, Wojciech Matusik, and David I. W. Levin. 2016. Stochastic structural analysis for context-aware design and fabrication. *ACM Trans. Graph.* 35, 6, Article 226 (2016), 226:1–226:13.

Honghua Li, Ruizhen Hu, Ibraheem Alhashim, and Hao Zhang. 2015. Foldabilizing furniture. *ACM Trans. Graph.* 34, 4, Article 90 (2015), 90:1–90:12.

Xian-Ying Li, Tao Ju, Yan Gu, and Shi-Min Hu. 2011. A geometric study of v-style pop-ups: Theories and algorithms. *ACM Trans. Graph.* 30, 4, Article 98 (2011), 98:1–98:10.

Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. 2015. Legolization: Optimizing LEGO designs. *ACM Trans. Graph.* 34, 6, Article 222 (2015), 222:1–222:12.

Silvano Martello and Paolo Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, Inc., New York, NY.

Meher McArthur and Robert J. Lang. 2012. *Folding Paper: The Infinite Possibilities of Origami.* Turtle Publishing.

Rico Moeckel, Cyril Jaquier, Kevin Drapel, Elmar Dittrich, Andres Upegui, and Auke Jan Ijspeert. 2006. Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with bluetooth interface. *Industrial Robot: An International Journal* 33, 4 (2006), 285–290.

Przemyslaw Musialski, Thomas Auzinger, Michael Birsak, Michael Wimmer, and Leif Kobbelt. 2015. Reduced-order shape optimization using offset surfaces. *ACM Trans. Graph.* 34, 4, Article 102 (2015), 102:1–102:9.

Joseph O'Rourke. 2011. *How To Fold It: The Mathematics of Linkages, Origami, and Polyhedra.* Cambridge University Press.

Julian Panetta, Qingnan Zhou, Luigi Malomo, Nico Pietroni, Paolo Cignoni, and Denis Zorin. 2015. Elastic textures for additive fabrication. *ACM Trans. Graph.* 34, 4 (2015), 135:1–135:12.

Jesús Pérez, Miguel A. Otaduy, and Bernhard Thomaszewski. 2017. Computational design and automated fabrication of Kirchhoff-plateau surfaces. *ACM Trans. Graph.* 36, 4, Article 62 (2017), 62:1–62:12.

Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. 2014. Programmable self-assembly in a thousand-robot swarm. *Science* 345, 6198 (2014), 795–799.

Graham G. Ryland and Harry H. Cheng. 2010. Design of iMobot, an intelligent reconfigurable mobile robot with novel locomotion. In *IEEE International Conference on Robotics and Automation (ICRA'10)*. 60–65.

Behnam Salemi, Mark Moll, and Weimin Shen. 2006. SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06)*. 3636–3641.

Galen Hajime Sasaki. 1987. *Optimization by Simulated Annealing: A Time-Complexity Analysis.* Ph.D. Dissertation, University of Illinois at Urbana-Champaign.

Christian Schumacher, Bernd Bickel, Jan Rys, Steve Marschner, Chiara Daraio, and Markus Gross. 2015. Microstructures to control elasticity in 3D printing. *ACM Trans. Graph.* 34, 4, Article 136 (2015), 136:1–136:13.

Oana Sidi, Oliver van Kaick, Yanir Kleiman, Hao Zhang, and Daniel Cohen-Or. 2011. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *ACM Trans. Graph.* 30, 6, Article 126 (2011), 126:1–126:10.

Mélina Skouras, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, and Markus Gross. 2013. Computational design of actuated deformable characters. *ACM Trans. Graph.* 32, 4, Article 82 (2013), 82:1–82:10.

Peng Song, Chi-Wing Fu, Yueming Jin, Hongfei Xu, Ligang Liu, Pheng-Ann Heng, and Daniel Cohen-Or. 2017a. Reconfigurable interlocking furniture. *ACM Trans. Graph.* 36, 6, Article 174 (2017), 174:1–174:14.

Peng Song, Xiaofei Wang, Xiao Tang, Chi-Wing Fu, Hongfei Xu, Ligang Liu, and Niloy J. Mitra. 2017b. Computational design of wind-up toys. *ACM Trans. Graph.* 36, 6, Article 238 (2017), 238:1–238:13.

Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Měch. 2012. Stress relief: Improving structural strength of 3D printable objects. *ACM Trans. Graph.* 31, 4, Article 48 (2012), 48:1–48:11.

Joan Stiles and Catherine Stern. 2001. Developmental change in spatial cognitive processing: Complexity effects and block construction performance in preschool children. *J. Cognit. Dev.* 2, 2 (2001), 157–187.

Kasper Stoy, David Brandt, and David J. Christensen. 2010. *Self-Reconfigurable Robots: An Introduction.* The MIT Press.

Ileana Streinu. 2000. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS'00)*. 443–453.

Timothy Sun and Changxi Zheng. 2015. Computational design of twisty joints and puzzles. *ACM Trans. Graph.* 34, 4, Article 101 (2015), 101:1–101:11.

Disney Storybook Art Team. 2010. *Toy Story 3 Mix & Match.* Disney Pr, ISBN13 9781423121411.

Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. 2014. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph.* 33, 4, Article 65 (2014), 65:1–65:10.

Jonathan Wai, David Lubinski, and Camilla P. Benbow. 2009. Spatial ability for STEM domains: Aligning over 50 years of cumulative psychological knowledge solidifies its importance. *Journal of Educational Psychology* 101, 4 (2009), 817–835.

Shiqing Xin, Chi-Fu Lai, Chi-Wing Fu, Tien-Tsin Wong, Ying He, and Daniel Cohen-Or. 2011. Making burr puzzles from 3D models. *ACM Trans. Graph.* 30, 4, Article 97 (2011), 97:1–97:8.

Gary Yngve and Greg Turk. 2002. Robust creation of implicit surfaces from polygonal meshes. *IEEE Trans. Visual. Comput. Graphics* 8, 4 (2002), 346–359.

Zhong You. 2014. Folding structures out of flat materials. *Science* 345, 6197 (2014), 623–624.

Minjing Yu, Yong-Jin Liu, and Charlie C. L. Wang. 2017. EasySRRobot: An easy-to-build self-reconfigurable robot with optimized design. In *IEEE International Conference on Robotics and Biomimetics (IEEE-ROBIO'17)*. 1094–1099.

Minjing Yu, Yong-Jin Liu, Guozhen Zhao, Chun Yu, and Yuanchun Shi. 2018. Spatial ability improvement by tangible interaction: A case study with EasySRRobot. In *Proceedings of the 2018 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA'18)*. ACM, LBW013:1–LBW013:6.

Ye Yuan, Changxi Zheng, and Stelian Coros. 2018. Computational design of transformables. *Comput.theor Graphics Forum* 37 (2018), 103–113.

Wen-Qi Zhang and Yong-Jin Liu. 2011. Approximating the longest paths in grid graphs. *Theor. Comput. Sci.* 412, 39 (2011), 5340–5350.

Xiaoting Zhang, Xinyi Le, Zihao Wu, Emily Whiting, and Charlie C. L. Wang. 2016. Data-driven bending elasticity design by shell thickness. *Comput. Graph. Forum* 35, 5 (2016), 157–166.

Qingnan Zhou, Julian Panetta, and Denis Zorin. 2013. Worst-case structural analysis. *ACM Trans. Graph.* 32, 4, Article 137 (2013), 137:1–137:12.

Yahan Zhou, Shinjiro Sueda, Wojciech Matusik, and Ariel Shamir. 2014. Boxelization: Folding 3D objects into boxes. *ACM Trans. Graph.* 33, 4, Article 71 (2014), 71:1–71:8.

Received May 2018; revised October 2018; accepted November 2018