Fabricable Eulerian Wires for 3D Shape Abstraction

WALLACE LIRA, Simon Fraser University

CHI-WING FU, The Chinese University of Hong Kong

HAO ZHANG, Simon Fraser University

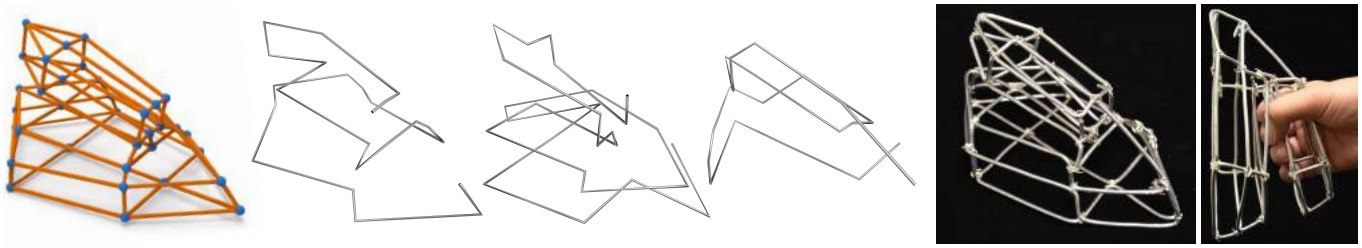


Fig. 1. We develop a *fully automatic* method to compute a small number of *machine fabricable* wires, called *Eulerian wires*, that reproduce a wire sculpture (left) with minimal overlap. Our method is able to find a globally optimal solution for the IRON, consisting of three wires (middle) with 0% overlap, in less than one minute. The wires can be physically produced in 3D using a wire bending machine and then assembled (right).

We present a *fully automatic* method that finds a *small number* of *machine fabricable* wires with *minimal overlap* to reproduce a wire sculpture design as a 3D shape abstraction. Importantly, we consider *non-planar* wires, which can be fabricated by a wire bending machine, to enable efficient construction of complex 3D sculptures that cannot be achieved by previous works. We call our wires *Eulerian wires*, since they are as Eulerian as possible with small overlap to form the target design together. Finding such Eulerian wires is highly challenging, due to an enormous search space. After exploring a variety of optimization strategies, we formulate a population-based hybrid metaheuristic model, and design the join, bridge and split operators to refine the solution wire sets in the population. We start the exploration of each solution wire set in a bottom-up manner, and adopt an adaptive simulated annealing model to regulate the exploration. By further formulating a meta model on top to optimize the cooling schedule, and precomputing fabricable subwires, our method can efficiently find promising solutions with low wire count and overlap in one to two minutes. We demonstrate the efficiency of our method on a rich variety of wire sculptures, and physically fabricate several of them. Our results show clear improvements over other optimization alternatives in terms of solution quality, versatility, and scalability.

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

Additional Key Words and Phrases: fabrication, wire, Eulerian, sculpture

ACM Reference Format:

Wallace Lira, Chi-Wing Fu, and Hao Zhang. 2018. Fabricable Eulerian Wires for 3D Shape Abstraction. *ACM Trans. Graph.* 37, 6, Article 240 (November 2018), 13 pages. https://doi.org/10.1145/3272127.3275049

Authors' addresses: Wallace Lira, Department of Computing Science, Simon Fraser University, wpintoli@sfu.ca; Chi-Wing Fu, Department of Computer Science and Engineering, The Chinese University of Hong Kong, cwfu@cse.cuhk.edu.hk; Hao Zhang, Department of Computing Science, Simon Fraser University, haoz@cs.sfu.ca.

0730-0301/2018/11-ART240 $15.00
https://doi.org/10.1145/3272127.3275049

1 INTRODUCTION

Wire sculptures [Wikipedia 2018b] are art works created out of line and curve structures. To design this form of art work, artists often use a minimal set of lines and curves to outline the shape or appearance of a target object. The resulting wireframe sculpture presents itself as a shape abstraction, exhibiting a tasteful minimalist feel; see Figure 2. However, existing wire sculptures have so far been created mostly through the hands of skilled artists and professional craftsmen. Both the design and physical construction processes are

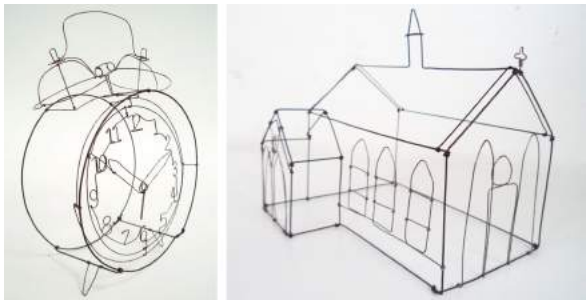


Fig. 2. Example wire sculptures that exemplify a minimalist feel. Designed by Louise Dawn Wilson.

time consuming and demand highly sophisticated skills. Even with contemporary *wire bending machines* (see Figure 3) which can free humans from the low-level task of physically bending the wires, one challenging computational problem that remains is how to *lay out* the wires to reproduce a complex wire sculpture.

Given a wire abstraction of a 3D shape represented as a graph, the wire layout problem seeks a set of wires, each a simple path in the input graph, which together cover the graph. Ideally, wires should not overlap to save fabrication time and material, and to avoid thickening of the wires for aesthetics. On the other hand, making each graph edge a separate wire, while ensuring zero overlap, is a poor solution, since all the wires must be joined to form a connected wire sculpture and a high wire count complicates the assembly task.

Finding a single long wire that covers the entire input graph is the most efficient, if the wire visits every graph edge exactly once. This is the *Eulerian path* problem, one of the most celebrated problems in graph theory. However, we cannot expect a single wire to work for general graphs. At the same time, to take advantage of a wire bending machine, which can shape the wires with high precision, we must place additional constraints on the wires so that they are *fabricable*. For example, while a fabricable wire can be *non-planar*, as shown in Figure 3(e), it must not collide with itself and the machine, since the collision interferes the physical fabrication process. Therefore, when considering machine fabricability, it is unlikely that a single wire can cover an entire wire sculpture.

Our wire layout problem is a multi-path variation of Eulerian path. We wish to find *as few as possible*, *long* and *non-planar* wires, with *minimal overlap*, to cover an input graph. Each wire should be *"as Eulerian as possible"* and fabricable, e.g., using the machine shown in Figure 3. The problem, referred to as the Eulerian wire problem in our work, is computationally challenging, since it relates to the optimization version of *set cover*, which is NP-hard, as we seek a minimal set of wires to cover the input graph. Incorporating overlap minimization, which may be in conflict with wire count minimization, further leads to a multi-objective optimization. Moreover, having non-planar wires helps lower the wire count and enriches the family of wire sculptures that one may build with the machine. However, it adds complexity to fabricability analysis, as well as to the solution search. Last but not the least, fabricability test involves collision detection, which can be costly for long wires.

In this paper, we develop a fully automatic and efficient computational method for solving the Eulerian wire problem, enabling us to produce physical wire abstractions of a variety of 3D shapes. Given the immense search space involved in forming the wires, we design several strategies to approach the problem in a tractable manner; see Figure 4. First, we precompute subwires and subwire connections that are fabricable to reduce necessary collision tests for fabricability during the solution search. Second, we define an objective function in terms of wire count and overlaps to guide the solution search. Most importantly, we explore a wide variety of optimization strategies, and then formulate a population-based hybrid metaheuristic model to efficiently solve the optimization, combining the merits of Simulated Annealing and Particle Swarm Optimization. Specifically, we design the join, bridge and split operators to refine

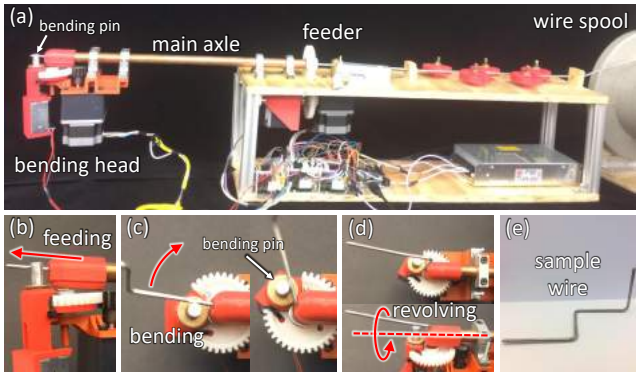


Fig. 3. We employ this 3D wire bending machine (a) for fabricating wires. (b-d) the machine operations, which together allow the shaping of non-planar wires, and (e) a sample wire fabricated by the machine.

the solution wire sets in the population. We start the solution exploration in a bottom-up manner, and adopt an adaptive simulated annealing model to regulate the exploration of each solution wire set. By further formulating a meta model on top to optimize the cooling schedule, our method can then efficiently search the vast search space and look for the optimum solution.

Our current pursuit of the Eulerian wire problem draws inspiration from two recent works. In WrapIt, Iarussi et al. [2015] decomposed a *2D* wire jewelry piece into few segments, which are *manually* produced by bending wires with the assistance of a 3D printed jig. Miguel et al. [2016] assembled cross-sectional *planar-rod* structures to abstract 3D shapes, where the planar wires that make up the 3D frame sculpture were given as part of the input. In contrast, the core computational step in our work involves a search for an optimal set of, possibly non-planar, wires to build the input *3D* wire abstraction. With non-planar wires, our method can produce efficient constructions of general and complex 3D wire sculptures, in particular, freeform structures, which may not admit suitable planar-rod representations such as a simple helix. As shown in Figure 1, while the input design spans many planes, our method produces a construction using just three Eulerian wires with a 0% overlap, which turns out to be a globally optimal solution.

Contributions. Our overall contribution is the first fully automatic method that constructs 3D wires for fabricating wire sculptures. This is a novel solution for fabricating general and complex 3D wire sculptures beyond planar cross-sections. Another contribution is our computational method that can efficiently find few Eulerian wires with minimal overlap to cover a wire sculpture design. This is made possible by means of our "*Annealing Swarm*" optimization model and the subwire mechanism. We evaluate our computational method through various experiments and fabricated results for wire abstractions of a variety of 3D shapes, and demonstrate clear improvements of our method in overall quality, versatility, and scalability, in comparison to other optimization alternatives.

2 RELATED WORK

Physical fabrication of 3D objects has been an emerging topic in computer graphics research in recent years. While most works focused on creating solid 3D models via additive manufacturing [Liu et al. 2014; Shamir et al. 2016; Umetani et al. 2015], some were devoted to the design and fabrication of *wireframe* models, for low-fidelity prototyping [Huang et al. 2016; Mueller et al. 2014; Wu et al. 2016; Yue et al. 2017], artistic expression [Chen et al. 2016; Zehnder et al. 2016], or achieving certain deformation behavior [Pérez et al. 2015]. What is common in these works is that the wireframe models were meant to closely approximate the given 3D shapes. Hence, their results tend to densely cover the surface of the given shape, or were fabricated layer by layer using conventional 3D printers. The wire mesh of Garg et al. [2014] also features a dense cover, but the meshes were built using many warped pieces of planar and woven wires arranged in a dense regular grid. These woven wires are pre-manufactured and available in large quantities. Another related work is Zimmer et al. [2014], which computes a Zometool structure with balls and sticks to form a given surface approximation. In contrast to these works, the wire sculptures we build are minimalist abstractions of freeform 3D shapes. In addition, the physical wires are fabricated using a 3D wire bending machine.

A recent work by Miguel et al. [2016] produces 3D shape abstractions using a set of planar wires. Given a planar wire set, their main task is to compute an ordering for wire assembly and optimize the final wire shapes for structural stability under frictional contacts. A strong merit of their physical products is that no thin tying wires are needed to stabilize the assembled wires, unlike WrapIt [Iarussi et al. 2015] and our work. In WireFab, Liu et al. [2017b] develop an interactive tool for users to extract and edit skeletal wires of 3D shapes, and to arrange 3D printed connectors to join the wires into an articulated design. Although the wire bending machine adopted by WireFab is exactly the same as ours, their method was designed to output a large quantity of wire segments that are joined by connectors for producing articulated movement. Compared to [Liu et al. 2017b; Miguel et al. 2016], our work tackles an entirely different problem. We aim for an automatic method to compute an almost Eulerian cover of a wire sculpture design with minimum number of long, and possibly non-planar, wires, while considering machine fabricability of the wires in our method formulation.

On a technical level, the work most closely related to ours is WrapIt, by [Iarussi et al. 2015]. Its input is a line drawing of a 2D jewelry design represented by a graph. The core computational step involves a decomposition of the graph into non-overlapping sub-graphs, each being a smooth curve preferred to have at least two contacts with others. The decomposition is formulated as a graph labeling problem and solved using Simulated Annealing. Like their work, we also seek a minimum number of wires to cover the sculpture. However, we consider wire fabricability in 3D, which adds a key new dimension to the problem. Fabricability is a *global* property of an entire wire, since a wire, after extended by an edge segment, its fabricability may be invalidated by any edge in the wire, not just the new edge or the one next to it. In contrast, wire smoothness, considered in WrapIt [Iarussi et al. 2015], is a local property. Moreover, we consider and allow overlap between wires, so that our solution wires may overlap with one another to minimize the overall wire count; yet we also seek to minimize the wire overlap in our formulation.

Further, WrapIt generates a 3D-printed jig with support walls to assist users to manually bend the wires, while in our work, the wires are automatically bent using a 3D wire bender. Hence, we need to consider *machine fabricability* extensively in our computational framework. Shifting from 2D input graphs, as WrapIt, to 3D input graphs, as our problem, significantly increases the *computational complexity* of the problem, especially when one considers the fabricability constraints inherent to our approach (see Figure 1 for some sample complex wires). Putting together the other challenges, we must *efficiently* explore wires in the vast search space.

Some other recent works aim to facilitate the creation of wire art and sculptures in various problem settings. Torres et al. [2016] improve the WrapIt system by considering wire forging, assembly and weaving. Liu et al. [2017a] present a method to reconstruct the geometry of a 3D wire art piece from multi-view images. Yue et al. [2017] develop WireDraw, an interactive system to assist users in creating wireframe models using a 3D drawing pen, where visual guidance in the form of virtual strokes is provided in a mixed reality display for users to follow and fabricate more accurate models.

3 PROBLEM FORMULATION

Input and output representations. The input to our method is a wire sculpture design. Such a design can be obtained via automatic or semi-automatic curve feature extraction from a 3D shape [Gal et al. 2009; Pan et al. 2015], or through an artistic design process, e.g., from design sketches [Xu et al. 2014]. Moreover, one may create a design manually by using conventional 3D modeling tools, e.g., by sketching on a 3D shape's surface, which serves as a reference.

Technically, we represent the input design as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; see Figure 4(a). Initially, each node in $\mathcal{V}$ is a 3D junction point in the design, and each edge in $\mathcal{E}$ is a segment joining two junctions; if any edge in $\mathcal{E}$ is not fabricable, we recursively split it into smaller edges and update $\mathcal{G}$ accordingly. The output from our method is a set of n wires, denoted as $\mathcal{W}$. Each wire is represented as a simple path in $\mathcal{G}$ without revisiting any node. The union of all the wires in $\mathcal{W}$ should cover all the edges in $\mathcal{E}$ for $\mathcal{W}$ to be a valid solution. Moreover, each wire in $\mathcal{W}$ must be machine fabricable.

Objective function. Letting the machine produce an individual wire for each edge segment and then manually connecting the wire segments into the sculpture would involve a highly tedious fabrication and assembly process. Instead, we seek a minimum number of *long* and *fabricable* wires, which we called *Eulerian wires, possibly in 3D*, to cover the input graph with *minimized overlap*. To guide the process of searching for the solution wire set $\mathcal{W}$, we aim to minimize the following objective function:

$$E(\mathcal{W}) \;=\; \left\lceil\, (1-\alpha)\cdot \frac{n}{|\mathcal{E}|} \;+\; \alpha \cdot \frac{\mathrm{olp}(\mathcal{E},\mathcal{W})}{L_0} \,\right\rceil, \qquad (1)$$

where α is a weight, which is empirically set as 0.03; n is normalized by the edge count $|\mathcal{E}|$; $L_0 = \sum_{e\in\mathcal{E}} |e|$ is the total length of all the

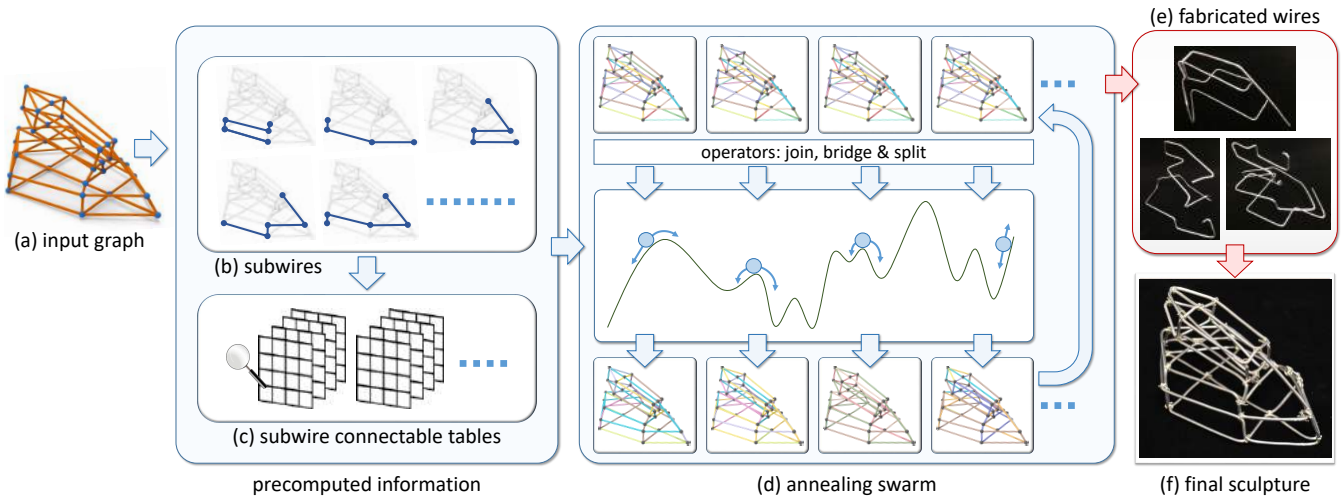


Fig. 4. Overview of our approach. From the input design (a), we first find all fabricable short wires, namely subwires (b), in the design, and then find all subwire pairs that are connectable and fabricable together (c). Then, we model an annealing swarm framework to simultaneously evolve a set of wire solutions, namely particles: each particle has its own annealing parameters and a wire set, which is initially the set of individual edge segments in the input, and we design three operators (split, join, and bridge) to update the particles' wire set accordingly. Upon termination, we pick the particle with the best objective value, fabricate its wire set using a wire bender (e) and compose the wires into the final sculpture (f).

edges in $\mathcal{E}$ ($|e|$ denotes the length of edge e), and the olp function sums up the amount of overlaps among the wires in $\mathcal{W}$:

$$\text{olp}(\mathcal{E}, \mathcal{W}) \;=\; \sum_{e \in \mathcal{E}} \left[\, \text{cnt}(e, \mathcal{W}) - 1 \,\right]^2 \;\cdot\; |e| \,, \tag{2}$$

where the cnt function counts the number of wires in $\mathcal{W}$ that pass through e. Note that we design the olp function in this form to penalize excessive overlaps for aesthetic concerns, and we further normalize it by L_0 in Eq. (1), since edge lengths (i.e., $|e|$ in L_0 and $|e|$ in Eq. (2)) are all computed in virtual 3D space.

Wire fabrication constraints. After setting up the machine shown in Figure 3(a), it can perform three operations to bend 3D wires:

i) *feed* the wire through the bending head (see Figure 3(b));

ii) *bend* the wire using the bending pin (see Figure 3(c)); and

iii) *revolve* the bending head around the main axle to change the orientation of the bending plane (see Figure 3(d)).

Bending-and-shaping wires is a mechanical process that involves two kinds of fabrication constraints. First, the portion of the wire that has extruded out of the bending head should not collide with the machine during any operation. Second, there is a mechanical limit in the bending angle (i.e., 120°) that the machine supports. See Section 5.1 for the details of these constraints.

4 BACKGROUND AND OVERVIEW OF APPROACH

Initial attempts. We tried several different approaches to find Eulerian wires. The first approach finds a large pool of fabricable and preferably long paths (10^3 to 10^6) in $\mathcal{G}$ as candidates, and computes a greedy set cover [Kleinberg and Tardos 2005] to find minimal wires with small overlaps. The advantage of the approach is that all candidate wires are prepared to be fabricable, so we do not need to test wire fabricability during the solution search. However, we cannot effectively reduce the wire count and overlaps, except for simple inputs. Since enumerating all possible paths in a graph is already an NP complete problem, we unlikely have sufficient candidate wires that well match one another for forming good solutions.

The second approach makes use of the subwire formulation in Section 5: subwires are precomputed short wires that are ensured to be fabricable. The approach picks the longest unvisited subwire and iteratively extends it by joining a subwire, as long as the combined wire is fabricable; when the wire is no longer extensible, we pick an unvisited subwire and repeat the process until the entire graph ($\mathcal{G}$) is covered. The approach can find better results in shorter time compared to the first approach, but it cannot effectively lower the wire count, since it *greedily* picks and extends subwires.

Our third attempt is a beam search model [Wikipedia 2018a], where we build a search tree with partial wire set solutions as internal nodes and full solutions as leave nodes. Compared to the second approach, using beam search can more accurately evaluate the quality of each decision: extend an existing wire or start a new wire. Hence, its results usually have lower wire count and overlap, compared to previous approaches. However, it requires a considerable amount of computing time and resource, and it *does not scale well* to produce good solutions when the model complexity increases.

From the initial attempts, we learn that it is hard for top down methods (first approach) to pre-determine a set of good candidates due to the vast search space. From the bottom up methods (second and third approaches), we learn that greedy strategies can easily trap in local minima. In addition, while evolving a pool of solutions certainly helps the method explore a larger search space, we find

that more efficient search strategies are yet required, particularly for handling more complex sculpture models.

Our approach. We start this work by exploring a wide variety of optimization strategies. Beyond the initial attempts described above, we further explore several metaheuristic methods, and then formulate a population-based hybrid metaheuristic model, we called *annealing swarm*, to solve the optimization. The model is developed by carefully adopting strategies in an adaptive simulated annealing model [Azizi and Zolfaghari 2004] and a particle swarm optimization model [Zhang et al. 2015] (see Section 6 for the details) for maximizing the solution quality and search efficiency.

Figure 4 presents an overview of our approach. To start, we find all machine fabricable short wires (subwires), and connectable subwire pairs, in the input design; see Figure 4(b&c). These one-time steps precompute wire fabricability information to speed up the search. Next, it comes to our hybrid metaheuristic model; see Figure 4(d). Our model initializes each solution wire set in the population as individual edge segments in the input design, and starts the search in a bottom-up manner. To refine the wires, three operators are designed: (i) *join* two wires end-to-end; (ii) *bridge* two distant wires by a subwire; and (iii) *split* a wire and remove its overlap parts. These operators provide essential and sufficient capabilities to refine wires for our problem, since they enable us not only to form longer wires (join) or break them down (split), but also to introduce an overlap to reduce the wire count (bridge). Further, we adopt an adaptive simulated annealing model with fast cooling to refine wires in each solution instance, and construct a meta model on top to optimize the cooling schedule of each solution instance and to perform multiple runs to adaptively evolve the solution instances. As a result, our method can efficiently explore the search space and find promising solutions in one to two minutes; see Figure 1 and Table 1.

Lastly, after the annealing swarm model terminates, we pick the best solution instance with the lowest objective value, post-process its wires, use a 3D wire bender to fabricate the wire set (see Figure 4(e)), and assemble the wires into the final sculpture model; see Figure 4(f) for a result and Sections 5 & 6 for the technical details.

5 CONSTRUCTING SUBWIRES WITH FABRICABILITY

5.1 Machine Fabricability

The procedure to test the fabricability of a wire is as follows. First, we check if any local bending angle along the wire exceeds the machine's bending limit. If so, we further see if we can use a double bending strategy to stretch the limit to 160°; see the inset figure on the right. Second, we check if the wire collides with the machine when the machine fabricates it. Here, we use a rectangular bounding box to model the bending head (see Figure 3(a)), and successively simulate each machine operation planned for the wire to detect if any portion of the wire that has extruded out of the bending head collides with the bending head during the operation; see Figure 3(b)-(d). For the bending operation, besides the bending head, we need to additionally detect collision between the extruded wire and the main axle; see Figure 3(c). Note also that a wire may be fabricated from either ends, so we need to perform the test for each end in general.

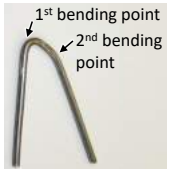


Fig. 5. We may fabricate this spiral-shaped wire (left) from inside to outside, but if we reverse the fabrication direction, the resulting wire (right) may become ill-shaped due to wire-machine collision.

Below, we summarize and define rules for wire fabricability:

- Rule 1: *a wire is fabricable if it can be fabricated by the machine, starting from one of its two ends.*
- Rule 2: *there are three cases for a fabricable wire: (i) forward fabricable, (ii) backward fabricable, and (iii) fabricable in both directions.* Note that a wire fabricable from one of its ends *may not* be fabricable if we start its fabrication from the other end. See Figure 5 for an example; we can fabricate the wire from inside to outside, but not the other way around, due to wire-machine collision.
- Rule 3: *if a wire is fabricable in a certain direction, any sub-portion of the wire is also fabricable in the same direction.* Conversely, *if a wire is not fabricable in a certain direction, any longer wire that contains it is also not fabricable in the same direction.*

In the solution search process, we maintain for each fabricable wire in our data structure on whether the wire is forward fabricable, backward fabricable, or fabricable in both directions.

5.2 Optimizing Fabricability Evaluation

Given an input design, we first identify all fabricable subwires (short wires) in the graph representation ($\mathcal{G}$) to minimize necessary fabricability tests in the solution search process; see Section 6. As a one-time pre-process, we first perform a breadth first search (BFS) from each node in $\mathcal{G}$ to exhaustively find all fabricable short paths started from the node. In the BFS, we stop to extend a path when the path has four edge segments in $\mathcal{G}$, or when it is found to be not fabricable (by Rule 3). After we perform BFS from every node in $\mathcal{G}$, each path of four or fewer edge segments should have been explored twice, once from each end, so the procedure has considered both forward and backward fabricability for each fabricable subwire. We denote $\mathcal{S}$ as the resulting set of all fabricable subwires in $\mathcal{G}$.

The next one-time pre-process is to find subwires in $\mathcal{S}$ that can be connected end-to-end as a single fabricable wire. In detail, for each node in $\mathcal{G}$, we first lookup the subwires in $\mathcal{S}$ that start or end at the node. Then, we examine each pair of them, say s_1 and s_2, and test if they are connectable at the shared end node, say v. To do so,

we first test if s_1 is forward fabricable toward v and s_2 is forward fabricable away from v, and vice versa. If the test is passed, we further test for collisions when fabricating s_2 after the bending head extrudes s_1, and vice versa. If no collision is found, the two subwires are connectable, and we store their fabricability type (i.e., forward, backward, or both) in the (per-node) lookup table.

6 ANNEALING SWARM MODEL

There are two factors that contribute to the size of the search space in our problem. The first factor is the number of simple paths in the input graph, since the associated wires, if fabricable, are candidates in our solution. Note that the requirement of not having repeated nodes actually relates to machine fabricability, since a fabricable wire should not self-intersect. As an upper bound on the number of simple paths, we consider a complete graph with $|\mathcal{V}|$ nodes, and deduce that the graph has $\frac{e \cdot |\mathcal{V}|!}{2}$ unique simple paths; see the supplemental material for the derivation. Having said that, for a complete graph with just 15 nodes, it already has $\sim 1.78 \times 10^{12}$ simple paths. The second contributing factor is that a solution usually requires multiple simple paths, rather than a single simple path. The reason behind is that when considering machine fabricability, wires should not self-intersect, so we often need multiple wires to cover the input graph. Therefore, the problem is intrinsically to find a few simple paths in a really huge set of simple paths; such set is unknown and intractable when we search for the solution.

Given the vastness of the search space, we need approximation heuristics to find close-to-optimal solutions in a reasonable time using limited resources. Concerning this, we explored a wide variety of optimization strategies besides those we discussed in Section 4, including simulated annealing, tabu search, ant colony optimization, and particle swarm optimization, and then formulate a metaheuristic model with the following characteristics:

- First, rather than local search-based methods, which can easily trap at local minima, we need metaheuristics, such as simulated annealing, to allow the search to get out of local minima.
- Second, rather than maintaining and improving on a single solution, we need a population-based search that processes multiple solution instances to explore the vast search space, where we can further parallelize and accelerate the exploration.
- Lastly, rather than using simulated annealing parameters that induce slower cooling, we use parameters for fast cooling and adapt the cooling schedule of each solution instance in a metaheuristic model, aiming at enabling each instance to search faster and go over a larger search space in multiple runs.

Population, or the solution instances. We employ a small population of N solution instances in our model, where N is empirically set in the range from 8 to 64, depending on the input model complexity; see Section 7. Each solution instance has two parts, $\mathbf{W}_i$ and $\mathbf{A}_i$, where i denotes the i-th solution instance; $\mathbf{W}_i$ denotes the wire set of the i-th solution instance; and $\mathbf{A}_i = (T_i^s, C_i, T_i^e)$ denotes its

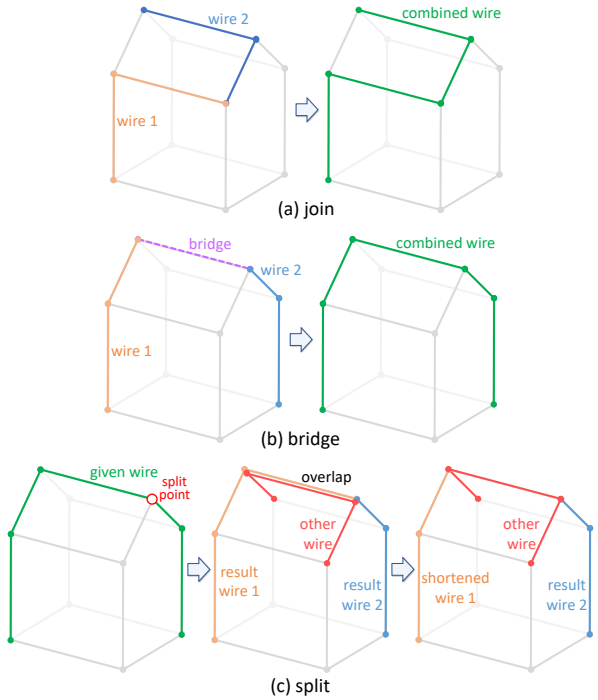


Fig. 6. Operators for refining the wires in the solution instances.

annealing parameters: T_i^s, C_i, and T_i^e are the starting temperature, cooling factor, and ending temperature, respectively.

When our method starts, each solution instance is initialized as follows: (i) $\mathbf{W}_i$ is simply $\mathcal{E}$, meaning that each individual edge segment in input graph $\mathcal{G}$ is a wire in the initial $\mathbf{W}_i$, so the search proceeds in a bottom-up manner, and (ii) T_i^s, C_i, and T_i^e are randomized in ranges $[20, 30]$, $[0.02, 0.002]$, and $[0.02, 0.002]$, respectively. These ranges facilitate fast convergence when we improve a solution instance in the adaptive simulated annealing model.

Operators to modify $\mathbf{W}_i$. We design the following three operators to modify the wire sets in the solution instances:

- *Join* randomly picks two wires in a solution instance that share an endpoint node, and connects them into a single wire, if the combined wire is fabricable; see Figure 6(a) for an illustration.
- *Bridge* randomly picks two wires in a solution instance, finds the shortest path (measured in terms of total path length in 3D) as a bridge between the endpoint nodes of the wires, and connects the wires with the bridge into a single wire, if (i) the bridge is a fabricable subwire in $\mathcal{S}$ and (ii) the combined wire through the bridge is also fabricable; see Figure 6(b) for an illustration.
- *Split* randomly picks a wire in a solution instance, and randomly picks an interior (non-endpoint) node in the wire to split it into two separate wires. For each resulting wire, we further check its starting and ending edge segments, and successively shorten it by removing segments that overlap with other wires in the solution instance; see Figure 6(c) for an illustration.

Algorithm 1 Adaptive Simulated Annealing (ASA) procedure

Require: $\mathbf{W}_i$, T_i^s, C_i, and T_i^e
  $T = T_i^s$ // initialize temperature T
  $\mathbf{S}_E = \{ E(\mathbf{W}_i) \}$ // store obj. value: use Eq. (1)
  $\mathbf{W}_{\text{best}} = \mathbf{W}_{\text{current}} = \mathbf{W}_i$ // current and best wire sets
  while $T > T_i^e$ **do**
    $\mathbf{W}_{\text{candidate}}$ = perturb($\mathbf{W}_{\text{current}}$)
    $\mathbf{S}_E = \mathbf{S}_E \cup \{ E(\mathbf{W}_{\text{candidate}}) \}$ // use Eq. (1)
    $\Delta = E(\mathbf{W}_{\text{candidate}}) - E(\mathbf{W}_{\text{current}})$
    σ = standard deviation of values in $\mathbf{S}_E$
    $T = \frac{T}{1+T \cdot In(1+C) \cdot / 3\sigma}$ // update temperature T
    $P = \min(1, exp(-\frac{\Delta}{T}))$
    if $P < T$ **then**
      $\mathbf{W}_{\text{current}} = \mathbf{W}_{\text{candidate}}$ // update current wire set
    end if
    if $E(\mathbf{W}_{\text{best}}) > E(\mathbf{W}_{\text{candidate}})$ **then**
      $\mathbf{W}_{\text{best}} = \mathbf{W}_{\text{candidate}}$ // update best wire set
    end if
  end while
  return $\mathbf{W}_{\text{best}}$

See the supplemental material for the technical procedures that make use of the precomputed subwires to test or update wire fabricability in each operator. Note also that to avoid re-testing the fabricability of long wires, we maintain a hash table to store the fabricability of long wires that we have tested so far.

Adaptive simulated annealing. In our metaheuristic model, we perform an adaptive simulated annealing procedure to iteratively refine wires in each solution instance. Unlike standard simulated annealing, we use an adaptive model [Azizi and Zolfaghari 2004] and set a very fast cooling schedule, so that the procedure can complete in a split second on each solution instance. As a result, we can quickly perform multiple runs on multiple solution instances, and explore a much larger search space to improve the solutions.

Algorithm 1 presents the procedure. Given a solution instance as the input, the procedure first initializes temperature T, $\mathbf{S}_E$ to store the objective values (Eq. (1)) of all the explored wire sets, as well as the current and best solution wire sets. Next, the procedure enters the main loop, where it first randomly generates a candidate wire set using the *perturb* function; *perturb* randomly chooses with equal probability to apply the join, bridge, or split operator to the current wire set. Lastly, Algorithm 1 uses an adaptive simulated annealing model to modify the temperature, updates the current and best solution wire sets, and repeats the loop until T falls below T_i^e.

Overall procedure. Algorithm 2 presents the overall metaheuristic procedure to look for the global best wire set. After the various initializations at the beginning, the procedure performs N_{run} runs of Algorithm 1 on the solution instances to refine their wire sets. We empirically set N_{run}=10, since we found it sufficient for producing good solutions for the input models presented in Section 7.

Algorithm 2 Overall procedure (Annealing Swarm Model)

Require: $\mathcal{G}$=($\mathcal{V}, \mathcal{E}$), N, N_{run}
  find all fabricable subwires (i.e., $\mathcal{S}$) in $\mathcal{G}$
  find all connectable pairs of subwires in $\mathcal{S}$
  for i = 1 to N **do**
    $\mathbf{I}_i$ = random $\{\mathbf{W}_i, (T_i^s, C_i, T_i^e)\}$ // initialize solution instance
    $\mathbf{I}_i^*$ = $\mathbf{I}_i$ // initialize personal best
    $\mathbf{v}_i = [0, 0, 0]$ // initialize param. velocity
  end for
  $\mathbf{G}^*$ = $\text{argmin}_{\mathbf{I}_i} E(\mathbf{W}_i)$ // initialize global best: Eq. (1)

  for run = 1 to N_{run} **do**
    for i = 1 to N **do**
      $\mathbf{W}_i = ASA(\mathbf{W}_i, T_i^s, C_i, T_i^e)$ // refine $\mathbf{W}_i$ by Algorithm 1
      update $\mathbf{v}_i$ using Eq. (3)
      $[T_i^s, C_i, T_i^e] = [T_i^s, C_i, T_i^e] + \mathbf{v}_i$
      if $E(\mathbf{W}_{\mathbf{I}_i^*}) > E(\mathbf{W}_i)$ **then**
        $\mathbf{I}_i^* = \mathbf{I}_i$ // update personal best
      end if
      if $E(\mathbf{W}_{\mathbf{G}^*}) > E(\mathbf{W}_i)$ **then**
        $\mathbf{G}^* = \mathbf{I}_i$ // update global best
      end if
    end for
  end for
  return $\mathbf{W}_{\mathbf{G}^*}$

In Algorithm 2, rather than a single run of the adaptive simulated annealing model (Algorithm 1) on each solution instance, we run Algorithm 1 on each solution instance multiple times, without re-initializing it after each run. Since temperature T in Algorithm 1 reverts back to T_i^s at the beginning of each run, a solution instance may get out of the local best obtained in previous run, and reach out to find a better solution in subsequent runs. Moreover, Algorithm 2 updates and refines the annealing parameters of each solution instance after each run by using the parameter velocity ($\mathbf{v}_i$) updated by the following equation (which is a strategy modified from the particle swarm optimization (PSO) model [Zhang et al. 2015]):

$$\mathbf{v}_i = \omega \cdot \mathbf{v}_i + (1 - \omega) \cdot r \cdot (\mathbf{A}_{\mathbf{I}_i^*} - \mathbf{A}_{\text{random}}), \quad (3)$$

where ω is a weight, which is empirically set as 0.2; r denotes a one dimensional random variable distributed uniformly in the interval $[0, 1]$; $\mathbf{A}_{\mathbf{I}_i^*}$ denotes the annealing parameters of the solution instance's personal best; and $\mathbf{A}_{\text{random}}$ denotes a three-dimensional random variable, where the three values are randomized in the ranges $[20, 30]$, $[0.02, 0.002]$, and $[0.02, 0.002]$, following the ranges to initialize T_i^s, C_i, and T_i^e at the beginning of Algorithm 2.

Discussion. Ideally, a swarm optimization (such as particle swarm optimization and ant colony optimization) would leverage some forms of communication between the solution instances to update the solution instances, e.g., steering the solution instance towards the global best in the search space. We have attempted to implement such a strategy in our metaheuristic model by computing the global best wire solution and then steering the solution instances towards it; in short, a transform operator was created to apply join, bridge

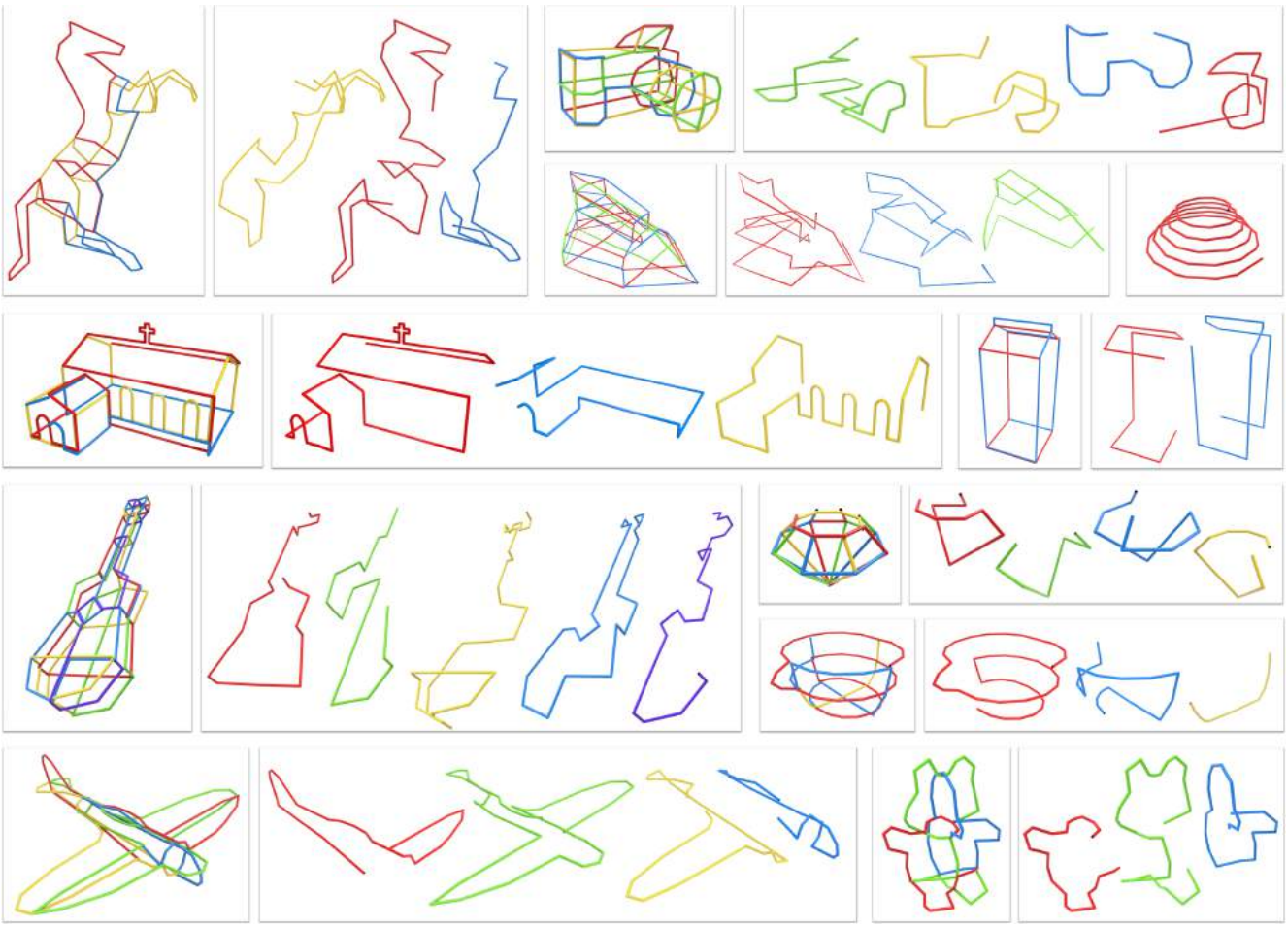


Fig. 7. A gallery of wire sculptures, which abstract a rich variety of 3D shapes, and the corresponding fabricable Eulerian wires computed by our fully automatic method. We show both assembled wire models and the individual Eulerian wires laid out to exhibit their complexity.

and split (following the graph editing distance) to modify the wire set in a solution instance and make it closer to the global best wire set. However, the strategy did not improve the results nor the running time. It is particularly challenging in the setting, given the fact that the search domain is discrete rather than continuous.

Post-processing & Fabrication. After a solution is found, it is necessary to perform two post-processing steps on the solution wire set. First, if a wire in the set stops right away on some other wires, like a T-junction, we extend the wire slightly by 1cm and bend its extension part to facilitate wire assembly by tying; see the inset figure shown on the right for an example. Second, we generate the specification of each wire and apply the double bending strategy to modify the machine instructions at each necessary bending location. After that, we fabricate the wires, and then assemble them using tying wires to produce the final sculpture.

7 RESULTS AND EVALUATION

We ran our experiments on a desktop PC with a four-core Intel i5 6400 processor and 8GB of DDR4 2800 SDRAM, where multi-threading was enabled for improved performance. Our method has the following tunable parameters. The first is α in the objective function that trades off between minimizing the wire count and overlap. It is set as 0.03 for all experiments, which conveys our focus on finding minimal wire counts. Even at a relatively small value, the parameter can effectively differentiate solutions of the same wire count but different amount of overlap. Second, the ranges for T_i^s, C_i, and T_i^e are set as $[20, 30]$, $[0.02, 0.002]$, and $[0.02, 0.002]$, respectively, so that the solution instances can quickly cool down, while being able to avoid the local optima. Furthermore, we initialize the population size N based on the number of subwires in the input graph. Specifically, we set N as 8, 16, 32, and 64 for graphs with approximately 500, $1k$, $2k$, and $4k$ or more subwires, respectively. Higher values potentially increase the quality of the solution but at the expense of more running time. We argue that the number of subwires is a better measure of a model's complexity rather than

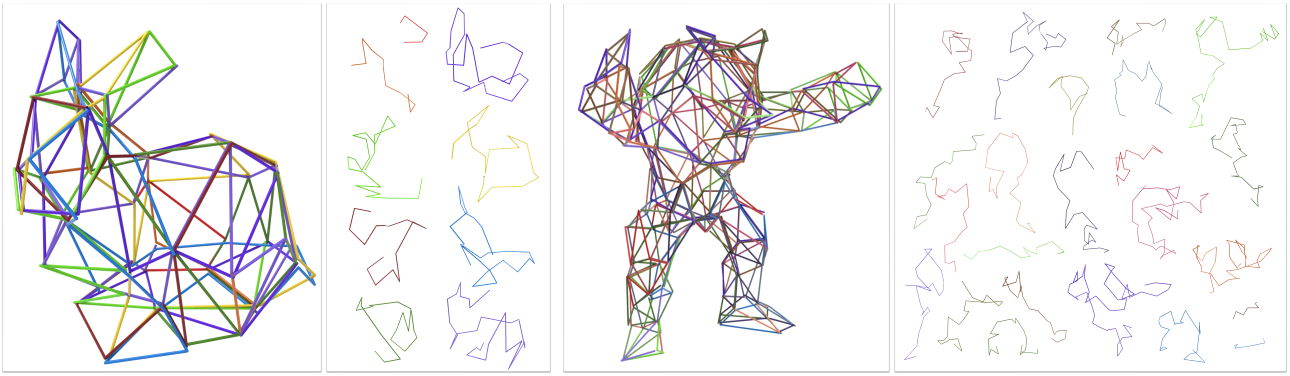


Fig. 8. The input Bunny and Armadillo models were obtained by decimating the original meshes to 100 and 300 faces, respectively. Note the challenges of constructing Eulerian wires for these models: the wires have to zigzag along the mesh edges; yet they have to be machine fabricable.

Table 1. Statistics of the model complexity of the inputs, results from beam search, results from Simulated Annealing using the model in WrapIt [Iarussi et al. 2015], and results from our method. From left to right: $|\mathcal{V}|$ and $|\mathcal{E}|$ denote the number of vertices (nodes) and edges in the input graph; $|\mathcal{S}|$ is the number of fabricable subwires found in the input; $|\mathcal{W}|$ denotes the wire count in a solution; *overlap* denotes the amount of wire overlap as a percentage of the total wire length; T denotes the time taken in seconds to generate a solution; and N denotes the population size in our annealing swarm model. Note that we mark the best solution for each input model in boldface, and put asterisks (*) next to the wire count numbers for the globally optimum solutions with the fewest possible number of wires to cover the input model (see discussion for the details).

		Model Complexity			Beam Search			Simulated Annealing (WrapIt)			Annealing Swarm			
Models	Fig.	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|\mathcal{S}\|$	$\|\mathcal{W}\|$	overlap	T	$\|\mathcal{W}\|$	overlap	T	$\|\mathcal{W}\|$	overlap	T	N
Helix	7	2	1	1	**1***	**0.0%**	0.0s	**1***	**0.0%**	0.0s	**1***	**0.0%**	0.0s	8
Cube	12(a),14	8	12	156	**2***	**16.7%**	0.4s	**2***	**16.7%**	20.1s	**2***	**16.7%**	6.4s	8
Bed	14	16	20	200	4	0.0%	0.6s	**3***	**18.2%**	21.9s	**3***	**18.2%**	4.6s	8
Milk Carton	7,14	12	18	256	3	5.4%	0.6s	**2***	**10.8%**	23.2s	**2***	**10.8%**	3.8s	8
Teddy	7,14,13	10	20	599	**3***	**0.0%**	2.4s	**3***	**0.0%**	109.2s	**3***	**0.0%**	21.3s	8
Horse	7	19	32	605	3*	8.7%	22.8s	4	1.2%	224.8s	**3***	**7.8%**	48.7s	8
Church	7	31	47	615	3*	28.4%	25.8s	3*	16.9%	131.1s	**3***	**16.8%**	32.4s	8
Cup	7	13	25	714	**3***	**0.0%**	7.8s	**3***	**0.0%**	51.0s	**3***	**0.0%**	8.1s	8
Diamond	7	17	32	1174	4*	3.6%	13.2s	4*	1.8%	32.2s	**4***	**0.0%**	10.5s	16
Airplane	7,13	20	42	1426	4	8.7%	25.8s	**4**	**2.6%**	51.0s	4	3.6%	33.2s	16
Camera	7	32	60	1777	4	15.1%	94.2s	5	6.6%	246.2s	**4***	**5.8%**	103.9s	32
Iron	1,4,7,13	35	69	2362	4	12.1%	91.8s	4	2.4%	83.2s	**3***	**0.0%**	51.8s	32
Guitar	7,13	54	96	2890	6	23.8%	221.4s	6	16.0%	141.1s	**5**	**8.2%**	68.4s	32
Bunny	8	52	150	20406	-	-	-	9	15.6%	211.0s	**9**	**13.1%**	67.9s	64
Armadillo	8	152	449	66595	-	-	-	25	21.6%	1067.17s	**23**	**22.4%**	396.6s	64

the number of nodes and edges in the input graph, since it accounts for the graph topology and wire fabricability constraints.

Ground truth. For most input graphs, there are no known ways of determining the ground truth wire count or the optimal overlap percentage. However, a 0% is clearly the best possible value for overlap, and we can also provide a theoretical *lower bound* on the wire counts, owing to fabricability constraints. Since a fabricable wire cannot self intersect, any graph vertex can only be visited by a wire at most once. Therefore, it follows that the wire count for any Eulerian wire problem should at least be $\lceil deg(v^*)/2 \rceil$, where v^* is the vertex with maximum valence in the input graph $\mathcal{G}$.

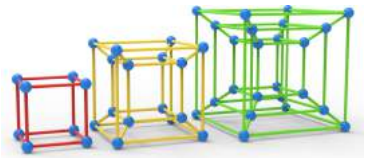
Eulerian wire results. Figures 7 and 8 present the visual galleries of the wire layout results computed by our method, together with the constituting Eulerian wires. As we can see, the input wire sculptures show abstractions of both man-made and organic 3D objects, as well as shapes of varying complexity. The Bunny and Armadillo models shown in Figure 8 are particularly challenging. These models were generated by decimating the original Bunny and Armadillo meshes to 100 and 300 faces, respectively.

Table 1 reports the input model statistics, and then the timing, wire count, and overlap for our method, as well as Beam Search and Simulated Annealing (SA). We implement Beam Search with a beam width of 75, and implement SA by adopting the optimization scheme in WrapIt [Iarussi et al. 2015]; however, since WrapIt assumes no wire overlap, we add our bridge operator to the SA implementation. Solutions giving the lowest wire count and lowest overlap are shown

in boldface, and if a wire count is optimal, based on the theoretical lower bound described above, we mark it with an asterisk.

Our method is able to find Eulerian wire solutions with few wires and small overlap, outperforming or matching solutions attained by the other two alternative approaches for most tested models; the only exception is the Airplane. For the Teddy, Cup, Diamond, and Iron models, our method is able to find the global optima, since in these cases the wire count equals the theoretical lower bound and the overlap is 0%. In terms of running time, Beam Search often wins for simpler input models, but it does not scale well. For example, for the two most complex models, i.e., Bunny and Armadillo (see the last two rows in the table), the search was unable to converge to a solution after 30 minutes. Comparing between SA and our Annealing Swarm optimization, we can generally see a two to five times speed-up of our method over SA. Also, note that the results for these two models have comparatively higher number of Eulerian wires and wire overlaps for two reasons. First, long wires in these models are geometrically more complex than long wires in other models, since zigzagging wires have a higher chance of colliding with the bending head, which in turn conflicts with the machine fabricability constraints. Second, their graph structures are more complex, requiring more fabricable wires to cover the entire graph.

Scalability. To examine the scalability of the optimization schemes for our Eulerian wire problem, we must first find out how the problem complexity grows. It turns out that the problem does not become computationally more expensive if one merely increases the vertex/edge count in the input graph. This is confirmed by Figure 9 (top-left), where we show that the timing plots for the three optimization alternatives, Beam Search, Simulated Annealing (SA), and our Annealing Swarm, all remain essentially flat as we progressively subdivide the edges in a wired cube.



On the other hand, the complexity of the problem does grow when there is a larger and more diverse set of graph paths to explore. For example, this would happen when the vertex valences increase in the graph. To this end, we have devised a recursive schema to generate interconnected cubes (or any other surface wire sculptures) via embedding, as shown in the above inset figure. This provides us with a simple means to programmatically generate arbitrarily complex graphs, contingent on the number of recursions employed to generate them.

Figure 9 (top-right) shows the timing plots for the three optimization schemes as more and more cubes are embedded inside a regular cube. We also show the wire counts and overlap percentages obtained for the different inputs in the figure. As one can see, Beam Search has poor scalability and its running time grows exponentially, while SA and Annealing Swarm exhibit significantly better scalability. The Annealing Swarm optimization scheme clearly outperforms the other two alternatives in terms of both wire count and overlap at all complexity levels, with only one exception, for the simplest case with only one embedded cube. Furthermore, the running time

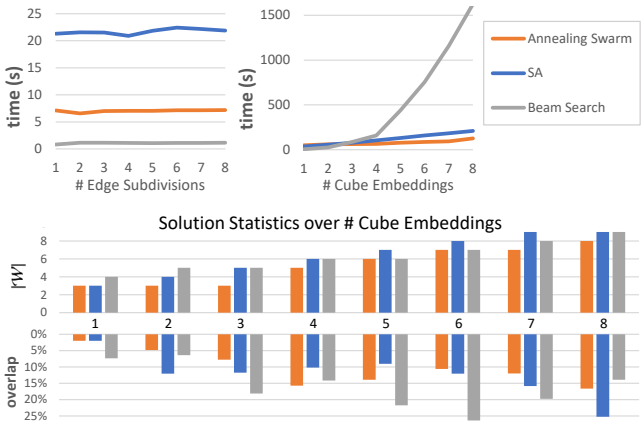


Fig. 9. Results of the scalability test for the three optimization alternatives (Beam Search, Simulated Annealing (SA), and our Annealing Swarm) with varying edge subdivisions and cube embeddings. Note that "#Cube embeddings" denotes the number of additional cubes embedded inside a regular cube. Top left: timing plot over progressively refined cubes via edge subdivision. Top right and bottom: timing plot and wire count + overlap statistics, over increasing cube embeddings.

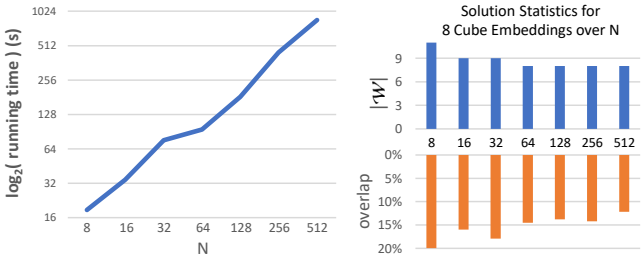


Fig. 10. Effect of population size N in our Annealing Swarm scheme on the running time (left), wire count $|\mathcal{W}|$, and overlap amount (right), using the most complex cube embedding model in Figure 9.

of our approach naturally improves as more computing cores are available, since particle updates in each iteration are parallelized.

For each method, we execute each experiment (edge subdivisions and #cube embeddings) 100 times, from which we report the median results. Annealing swarm parameters are defined as in the previous section, while N is set at 64. The parameters for the SA search were $T^s = 100$, $C = 0.001$, and $T^e = 0.001$. The reason for choosing these parameters was to keep the SA procedure in a similar running time frame as our method. Pushing C and T^e both to 0.0001 would result in better outputs but at a significantly higher running time cost. The beam width of the Beam Search procedure was kept at 75, consistent with the experiments shown in Table 1.

Impact of population size N. To assess the impact of the parameter N in our Annealing Swarm optimization, we took the Cube with eight embeddings and ran the optimization with this parameter ranging from 8 to 512. Figure 10 presents the results of this experiment. The left plot shows that the running time of our method roughly scales linearly with N. On the right, the two plots show the changes in

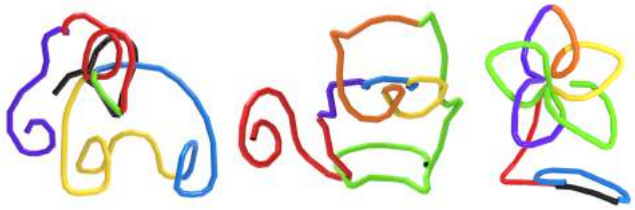


Fig. 11. Breaking a single Eulerian non-fabricable wire into fabricable wires using our algorithm. Each result was produced in about 1.5 minutes. The relatively high amount of wires for such topologically simple inputs is a direct result of the fabricability constraints. For instance, the "eyes" of the Cat model (middle) involve sharp twisting, which is not fabricable; therefore, these "eyes" become a breaking point on the wire.

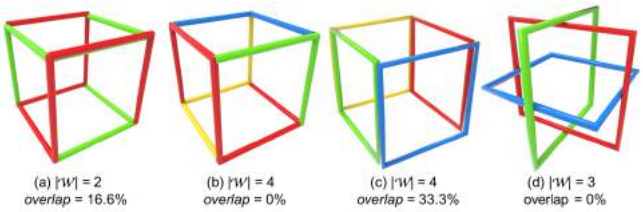


Fig. 12. Eulerian wire cover vs. planar contour abstraction, for the cube. (a) shows the result produced by our algorithm which cover the 12 edges of the cube using two wires. It is possible to obtain a non-overlapping solution (b) by modifying α in the objective function to 0.97 and disabling the bridge operator in our method. (c) Covering the cube edges using planar contours must incur significant overlap. (d) A "gift wrapping" represents the most efficient way to abstract a cube using non-overlapping planar contours.

wire count and overlap percentage for each N. Note that the solution quality only improves moderately when N goes beyond 64.

Single input wire. The Helix model is a single fabricable wire, serving as a simple sanity check to show that our method does return the expected solution (see the first row of Table 1); it is also representative of those freeform wire structures that do not admit efficient planar cross-sectional constructions.Further, the three wire models provided in Figure 11 are results produced by the image-based wire reconstruction of Liu et al. [2017a]. Each result is an unstructured long wire that is *not suitable* for planar cross-sectional constructions, like the Helix. Given the machine fabricability constraints, these wires are not fabricable without breaking them.

3D wires vs. planar contours. In the work of Miguel et al. [2016], a 3D shape is abstracted by a set of given planar contours. Planar cross-sectional abstractions are more suited to man-made objects, not general and freeform structures such as the Helix, Teddy, and Horse models, and the models shown in Figure 11. Our method produces "3D space" wires and is applicable to richer sets of inputs. Besides planarity, closed-ness of the wires is also a relevant factor. Consider a simple cube for example, using a set of planar contours to cover all its edges would necessitate significant (33%) overlap, as shown in Figure 12(c). Also, the four planar contours cannot be held together by wire friction, as they do not "cross" each other. A "gift wrapping" style abstraction would be possible, but it does not cover the cube edges. As for our method, it can produce efficient covers of the cube edges using non-planar wires, with or without overlap, by controlling parameter α in the objective function and enabling/disabling the bridge operator; see Figures 12(a)-(b).

Human performance. To obtain a sense of how difficult the Eulerian wire construction problem is for a human, we selected three simple wire models: Cube, Bed, and Milk Carton (see Table 1), and recruited ten participants who are graduate students (five males and five females) aged from 22 to 29 on a volunteer basis. In this experiment, we first introduced to each of them the concepts of 3D graph representations, simple paths and path overlaps in around five to ten minutes, and then gave each of them a maximum of 20 minutes to work on each model. None of them used up the 20 minutes in working out a solution. For each model, our instruction to them was to find as few simple paths as possible to cover the given graph with minimal path overlaps.

We recorded the solutions found by the participants as well as the time spent. For the simplest model, which is Cube, all participants were able to find the optimal solution with two wires and two overlapping edges, but it took them 8.5 minutes on average. The other two models are, however, more challenging for humans. For Bed, only three of them found the three-wire solution produced by our method, while for Milk Carton, only one participant found a two-wire solution; however, that solution has larger overlap percentage (21.7%) compared to our solution (10.8%) and all others found three-wire solutions with varying amount of overlaps. Figure 14 shows the timing and solution results, and presents the Bed model.

Clearly, even for these simple models, there is a large gap between human performance and what a computational approach such as our method could achieve. Particularly, it is challenging for humans to find solutions with lowest wire counts while having a small overlap. It is hard to imagine that humans could come up with the three-wire solution for the Iron model in a reasonable time.

Physical fabrication. Using the wire bending machine shown in Figure 3, we fabricated several wire sculptures of different shapes and varying complexity using the results produced by our method; see Figure 13. The machine bent lightweight aluminum wires of 0.126" thickness and the resulting wires were assembled using thin tying wires; besides tying, one may assemble wires by soldering or by using 3D printed connectors as in [Liu et al. 2017b]. We built the wire bending machine ourselves, after purchasing the machine parts of the DIWire Bender[1]. Wire fabrication time depends not only on the wire length, but also on the amount of bending and rotation operations. In short, complex wires take longer time to fabricate than simple wires of the same length.

Challenging models. There are several challenging situations arising from the input models, resulting in high search cost and/or high wire count or overlap by our current method. For example: (i) Highly intertwined or self-convolved wires, e.g., a densely packed 3D Hilbert curve or a dense helix with small gaps in-between cycles (compared to the sparse Helix in Figure 7). Since the bending

[1] http://www.instructables.com/id/DIWire-Bender/

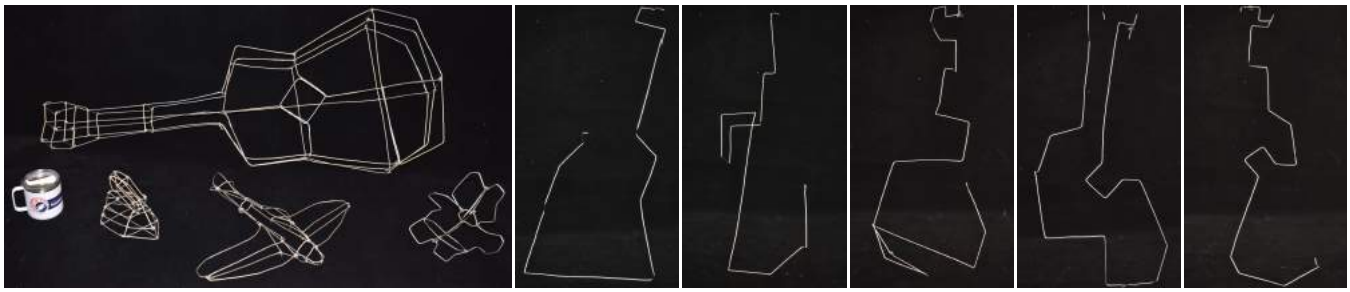


Fig. 13. Wire sculptures (left) we fabricated from our results, and the five Eulerian wires (right) constructed by our method for the Guitar model; see also Figure 1 and Figure 4 for the fabricated Iron model and its Eulerian wires.

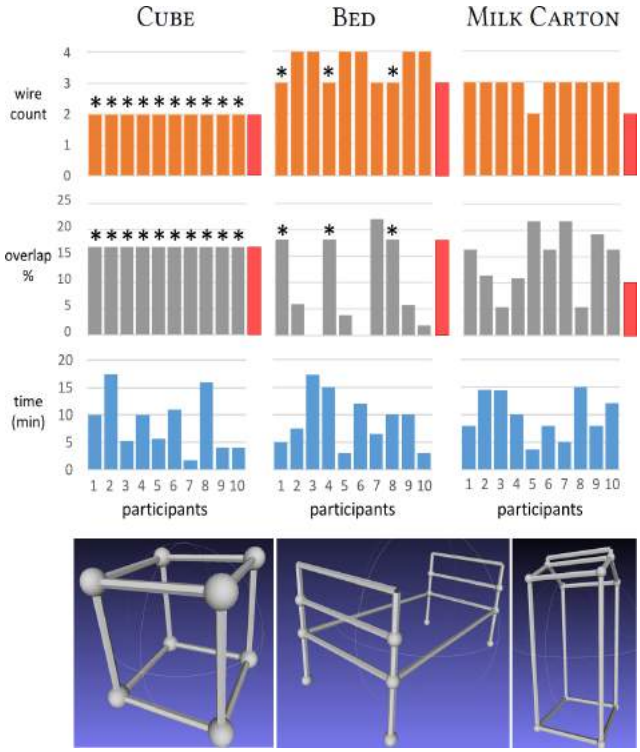


Fig. 14. Top: timing, wire count, and overlap results produced by ten participants on Cube, Bed, and Milk Carton. The red bars show results produced by our method, while the asterisks (*) mark comparable participant results. Bottom: 3D views of the three models we presented to the participants.

head can easily collide with the wire, wire fabricability would force the solution to contain many short wires. (ii) Graphs with many odd-degree vertices (e.g., the corners in the Cube); at such a vertex, we need at least one wire to start/end exactly at the vertex, or the wire would overlap next to the vertex. (iii) Models with complex structures, such as the Bunny and Armadillo (see Figure 8). While these challenging cases are generally a consequence of the problem setting and physical limitations of the machine, case (iii) is particularly challenging, since a complex sculpture involves an immense search space and requires more wires to cover the model.

8 DISCUSSION, LIMITATION, AND FUTURE WORK

We present a computational approach to realizing line and curve abstractions of 3D shapes using non-planar wires that are fabricable by a wire bending machine. The problem is complex, since the search space is immense and we have to consider wire fabricability by the machine. The method we develop combines efficient search strategies formulated as a population-based hybrid metaheuristic model, and precomputation of wire fabricability by means of subwires. We demonstrate that our method is able to efficiently find promising Eulerian wire constructions, where the wires can be physically produced by the wire bending machine and assembled into the target wire sculpture. Comparisons with other optimization alternatives including Beam Search and Simulated Annealing also clearly show the advantages of our Annealing Swarm strategy in terms of scalability and wire solution quality. Our method is general and versatile. It is applicable to wire abstractions of both man-made objects and freeform organic structures. It also offers the flexibility of trading off between optimal wire count and Eulerianity.

Limitations. Our current solution does not consider weight bearing, stability, or other structural properties of the physical wire sculpture produced. If the wire sculpture is not only displayed as an art piece but also put to use, then functionality is also a concern, which we do not yet account for. In addition, our current search for candidate wires is not symmetry-aware. That is, we cannot ensure that all the final wires produced are symmetric to one another, even if the input abstract has global symmetry. Such symmetries may be desirable from an aesthetics or functionality perspective.

Another aesthetic issue related to wire construction is the placement of *connectors* to stabilize the wire assembled. Minimizing the number of connectors is certainly desirable. In our current problem setting, a connector is needed at any graph vertex whose valence is greater than two, since two separate wires would go through that vertex. Hence reducing the need for other types of connectors is equivalent to minimizing wire count. Furthermore, we may need additional connectors to reinforce the stability of long overlaps. While our optimization scheme does not directly address this concern, the proposed solution explicitly avoids long overlaps. Finally, reachability may be a general concern for adding connectors or tying wires, as it may be hard to reach the interior of a sculpture. This issue is

not problematic for our current inputs, since they are all surface abstractions of 3D shapes.

How to properly utilize force and friction when assembling the wire sculpture to ensure stability without tying wires is also an interesting problem. Miguel et al. [2016] addressed it for 2D wires. However, like WrapIt [Iarussi et al. 2015], we opted to use thin tying wires to join adjacent wire pieces together. This is a simple and economical solution that helps ensure the stability at the connections while maintaining an aesthetic appeal of the "wire-on-wire" assembly.

In terms of wire assembly, one advantage of having only few pieces of long wires is that quite likely, each wire covers a large portion of the target shape so that it is more identifiable and also more visually distinguishable from the other wires. This makes the assembly process more straightforward than the case where one must put many short (hence less distinguishable) wires together. Still, we do not compute an optimal assembly order, nor do we account for possible unresolvable collisions between the wires during assembly.

Future work. Addressing the computational and physical limitations of our current approach already suggests several avenues for future work. Incorporating symmetry, structural, and functional constraints, as well as the utilization of frictional contacts (thus removing the need for tying wires) into the problem formulation will strengthen the solution. Combining these constraints with the use of industrial grade materials and wire benders would further enable our approach to create functional wire models such as drone frames and household utensils. Computing an assembly order, possibly one leading to interlocking, is another intriguing problem to pursue.

In addition, there are interesting variations of our current wire construction problem to explore. For example, one may forego the desire to have a low wire count. Instead, the goal may be to produce a final set of fabricable wires that are suitable as pieces of a difficult assembly puzzle. Another variation is a true "Eulerian version" of the problem. Instead of taking a given wire abstraction as input and seeking a cover, we take in a 3D surface model without any wireframe specifications. The goal is to compute a *single* and fabricable wire, which provides the best abstraction of the input model.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their insightful comments. We also thank Matthew Cao for his help assembling the wire bending machine and Yang Tian for his help on the user study. This work is supported in part by grants from NSERC Canada (611370) and Adobe gift funds for Hao Zhang, the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK 14203416 and 14201717) for Chi-Wing Fu. Wallace Lira is supported by SFU's C.D. Nelson Entrance Scholarship.

REFERENCES

Nader Azizi and Saeed Zolfaghari. 2004. Adaptive Temperature Control for Simulated Annealing: A Comparative Study. *Comput. Oper. Res.* 31, 14 (Dec. 2004), 2439–2451. https://doi.org/10.1016/S0305-0548(03)00197-7

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

Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. 2009. iWIRES: An Analyze-and-edit Approach to Shape Manipulation. *ACM Trans. on Graph. (SIGGRAPH)* 28, 3, Article 33 (July 2009), 10 pages. https://doi.org/10.1145/1531326.1531339

Akash Garg, Andrew O. Sageman-Furnas, Bailin Deng, Yonghao Yue, Eitan Grinspun, Mark Pauly, and Max Wardetzky. 2014. Wire Mesh Design. *ACM Trans. on Graph. (SIGGRAPH)* 33, 4, Article 66 (July 2014), 12 pages. https://doi.org/10.1145/2601097.2601106

Yijiang Huang, Juyong Zhang, Xin Hu, Guoxian Song, Zhongyuan Liu, Lei Yu, and Ligang Liu. 2016. FrameFab: Robotic Fabrication of Frame Shapes. *ACM Trans. on Graph. (SIGGRAPH Asia)* 35, 6, Article 224 (Nov. 2016), 11 pages. https://doi.org/10.1145/2980179.2982401

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

Jon Kleinberg and Eva Tardos. 2005. *Algorithm Design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Lingjie Liu, Duygu Ceylan, Cheng Lin, Wenping Wang, and Niloy J. Mitra. 2017a. Image-based Reconstruction of Wire Art. *ACM Trans. on Graph. (SIGGRAPH Asia)* 36, 4, Article 63 (July 2017), 11 pages. https://doi.org/10.1145/3072959.3073682

Ligang Liu, Charlie Wang, Ariel Shamir, and Emily Whiting. 2014. 3D Printing Oriented Design: Geometry and Optimization. (2014). https://doi.org/10.1145/2659467.2675050 SIGGRAPH Asia Course.

Min Liu, Yunbo Zhang, Jing Bai, Yuanzhi Cao, Jeffrey M. Alperovich, and Karthik Ramani. 2017b. WireFab: Mix-Dimensional Modeling and Fabrication for 3D Mesh Models. In *Proc. CHI Conf. on Human Factors in Computing Sys.* 965–976.

Eder Miguel, Mathias Lepoutre, and Bernd Bickel. 2016. Computational Design of Stable Planar-rod Structures. *ACM Trans. on Graph. (SIGGRAPH)* 35, 4, Article 86 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925978

Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, François Guimbretière, and Patrick Baudisch. 2014. WirePrint: 3D Printed Previews for Fast Prototyping. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology.* 273–280.

Hao Pan, Yang Liu, Alla Sheffer, Nicholas Vining, Chang-Jian Li, and Wenping Wang. 2015. Flow Aligned Surfacing of Curve Networks. *ACM Trans. on Graph. (SIGGRAPH)* 34, 4, Article 127 (July 2015), 10 pages. https://doi.org/10.1145/2766990

Jesús Pérez, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, José A. Canabal, Robert Sumner, and Miguel A. Otaduy. 2015. Design and Fabrication of Flexible Rod Meshes. *ACM Trans. on Graph. (SIGGRAPH)* 34, 4, Article 138 (July 2015), 12 pages. https://doi.org/10.1145/2766998

Ariel Shamir, Bernd Bickel, and Wojciech Matusik. 2016. Computational Tools for 3D Printing. (2016). https://doi.org/10.1145/2897826.2927367 SIGGRAPH Course.

Cesar Torres, Wilmot Li, and Eric Paulos. 2016. ProxyPrint: Supporting Crafting Practice Through Physical Computational Proxies. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS '16).* ACM, New York, NY, USA, 158–169. https://doi.org/10.1145/2901790.2901828

Nobuyuki Umetani, Bernd Bickel, and Wojciech Matusik. 2015. Computational Tools for 3D Printing. (2015). https://doi.org/10.1145/2776880.2792718 SIGGRAPH Course.

Wikipedia. 2018a. Beam search — Wikipedia, The Free Encyclopedia. (2018). https://en.wikipedia.org/wiki/Beam_search [Online; accessed 29-May-2018].

Wikipedia. 2018b. Wire sculpture — Wikipedia, The Free Encyclopedia. (2018). https://en.wikipedia.org/wiki/Wire_sculpture [Online; accessed 3-January-2018].

Rundong Wu, Huaishu Peng, François Guimbretière, and Steve Marschner. 2016. Printing Arbitrary Meshes with a 5DOF Wireframe Printer. *ACM Trans. on Graph. (SIGGRAPH)* 35, 4, Article 101 (July 2016), 9 pages. https://doi.org/10.1145/2897824.2925966

Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. 2014. True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization. *ACM Trans. on Graph. (SIGGRAPH)* 33, 4, Article 131 (July 2014), 13 pages. https://doi.org/10.1145/2601097.2601128

Ya-Ting Yue, Xiaolong Zhang, Yongliang Yang, Gang Ren, Yi-King Choi, and Wenping Wang. 2017. WireDraw: 3D Wire Sculpturing Guided with Mixed Reality. In *Proc. CHI Conf. on Human Factors in Computing Sys.* 3693–3704.

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

Yu-Dong Zhang, Shuihua Wang, and Genlin Ji. 2015. A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. 2015 (01 2015), 1–38.

Henrik Zimmer, Florent Lafarge, Pierre Alliez, and Leif Kobbelt. 2014. Zometool shape approximation. *Graphical Models* 76, 5 (2014), 390–401.