Printing Arbitrary Meshes with a 5DOF Wireframe Printer

Rundong Wu
Cornell University

Huaishu Peng
Cornell University

François Guimbretière
Cornell University

Steve Marschner
Cornell University

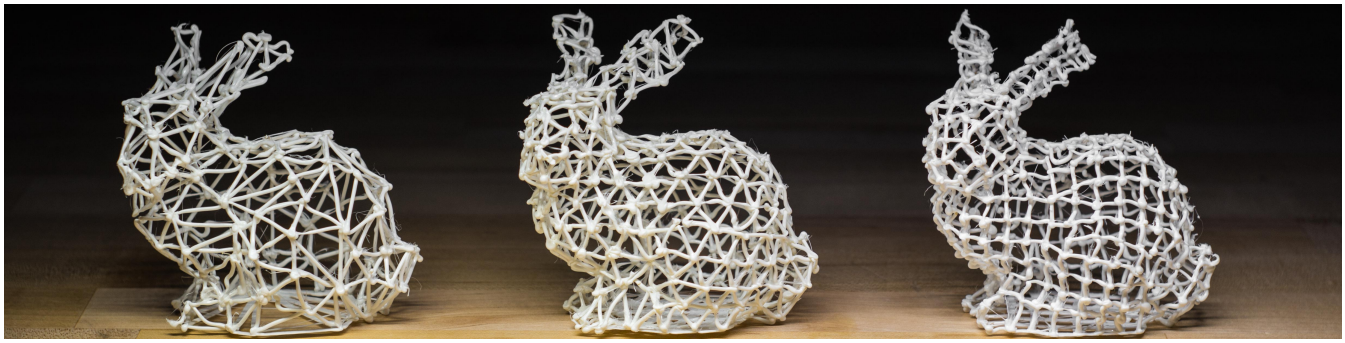

Figure 1: *Our method enables surfaces to be printed as 3D wireframes using arbitrary meshes. This enables improved shape approximation and shape depiction as compared to previous approaches.*

Abstract

Traditional 3D printers fabricate objects by depositing material to build up the model layer by layer. Instead printing only wireframes can reduce printing time and the cost of material while producing effective depictions of shape. However, wireframe printing requires the printer to undergo arbitrary 3D motions, rather than slice-wise 2D motions, which can lead to collisions with already-printed parts of the model. Previous work has either limited itself to restricted meshes that are collision free by construction, or simply dropped unreachable parts of the model, but in this paper we present a method to print arbitrary meshes on a 5DOF wireframe printer. We formalize the collision avoidance problem using a directed graph, and propose an algorithm that finds a locally minimal set of constraints on the order of edges that guarantees there will be no collisions. Then a second algorithm orders the edges so that the printing progresses smoothly. Though meshes do exist that still cannot be printed, our method prints a wide range of models that previous methods cannot, and it provides a fundamental enabling algorithm for future development of wireframe printing.

Keywords: 3D printing, WirePrint, wireframe, meshes

Concepts: •**Computing methodologies** → **Mesh models;**

1 Introduction

Affordable 3D printers make it easy for anyone to fabricate complex physical artifacts nowadays. Traditional 3D printers have 3 degrees of freedom (DOF): the printhead can move to any point in space but cannot rotate. To avoid collisions between the printhead and the print, objects generally must be printed layer by layer. This means that overhangs must be printed after building some supporting structures, and some surface features are hard to produce by only printing from straight above. Using high DOF tools eliminates these limitations and improves flexibility for fabrication [Singh 2004; Pan et al. 2014; Lee and Jee 2015].

WirePrint [Mueller et al. 2014] is an emerging technology that especially benefits from using higher DOF. To print wireframe meshes, the filament is extruded not layer by layer, but directly in 3D space, creating each edge in a single stroke. Compared with traditional 3D printed objects, 3D wire meshes are faster to print and also a good aid to surface depiction, where the inner structure of models can be seen and surface features such as curvature can be indicated by the arrangement of edges.

3DOF WirePrint generates the wireframe of a model by slicing it horizontally, in order to ensure collision-free printing. This approach badly constrains the types of meshes that can be printed. To improve flexibility, recently a 5DOF printer was introduced in the On-the-Fly Print system [Peng et al. 2016], which modifies a standard delta 3D printer by adding two rotation axes. With this machine, the edges can effectively be approached from any direction in the hemisphere, which opens up the possibility of printing arbitrary input meshes.

While tool path planning for layerwise 3DOF printing is trivial, planning for 5DOF is a fundamentally different problem: printing orientations need to be determined and collision needs to be considered so that already printed parts do not collide with the extruder. The planning problem boils down to ordering the edges of the input mesh into a sequence, and choosing a printing orientation for each edge. It is typically prone to be constrained by collisions: a naive ordering of edges often leads the printer to paint itself into a corner where certain edges cannot be approached, from any direction, without colliding with what has already been printed. On-the-Fly Print suffers from this problem: with its fixed printing order, it has to leave parts of the model unprinted to avoid collisions.

A feasible printing plan must have two properties. Every edge must be connected to what is already printed—edges cannot be printed in mid-air. And when printing an edge, the nozzle should not collide with previously printed parts. Besides these two hard requirements,

SIGGRAPH '16 Technical Paper,, July 24-28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925966

not all feasible plans are equally desirable, since printing in smooth, uninterrupted strips leads to better results. In this paper we propose an algorithm to schedule edges for 5DOF wireframe printing. Our algorithm has two major parts: generating constraints on the order of the edges to avoid collisions, and traversing the edges subject to these constraints.

There are three classes of models that cannot be printed: (a) those for which no collision-free ordering exists at all; (b) those for which the only collision-free orderings would require printing edges floating in space; and (c) those for which a solution exists but our method does not find it. Our method can prove a model to be in class (a), but cannot distinguish between (b) and (c).

We demonstrate our algorithm by printing triangular, quadrilateral, and mixed meshes created by a variety of meshing tools. We also discuss the limitations of what can be printed. Our algorithm is able to make feasible printing plans for a wide range of surfaces with complex geometry and topology that cannot be handled by previous methods. Although our algorithm is specially designed for wireframe printing, variations of the method can be applied to other scheduling problems involving collision-constrained automated assembly or high DOF additive manufacturing processes.

2 Related Work

In recent years, considerable attention in computer graphics has been drawn to research on 3D printing. Quite a few topics have been studied, such as balancing 3D printed shapes [Prévost et al. 2013], generating articulated models [Bächer et al. 2012; Calì et al. 2012] and controlling elasticity [Schumacher et al. 2015; Panetta et al. 2015; Pérez et al. 2015], but all use established printing methods to create the shapes they design. Wang et al. [2013] proposed an algorithm to generate skin-frame structures that approximate given 3D models in order to save print material. Although they have the idea of printing wireframes underneath the object's surface instead of printing the solid infill, the wire edges are printed layer-wise.

The slicer is the key software element in traditional 3D printing. It decomposes a 3D digital model into 2D layers and generates toolpaths and machine instructions accordingly. Several open-source and commercial slicers are available, such as Slic3r [Slic3r] and Cura [Cura]. We use a printer originally built for layer-wise Fused Deposition Modeling (FDM), but our algorithm works fundamentally differently from a slicer.

Mueller et al. [2014] introduced the idea of directly printing mesh edges in 3D space in their WirePrint system, which proposed to print wire meshes to give designers a fast preview of 3D models. They sliced the model with many horizontal planes and fabricated the model by alternately printing the planar slices and a fixed zigzag pattern between them. They used 3DOF printers, so the edges could only be printed from straight above, and their printer could not print any edge steeper than a certain threshold downward. They considered this constraint when generating the zigzags, leaving out one edge in each slice to avoid collision. Thus the limitations of 3DOF printing led them only to print horizontally sliced wire frames, a constraint which seriously affects the mesh quality.

In order to print other meshes, a 5DOF printer was introduced by [Peng et al. 2016]. Their 5DOF machine can rotate the model during printing, thereby effectively printing the edges in different directions. In their system, On-the-Fly Print, the mesh is generated based on the model's UV coordinates and the edges are ordered accordingly, alternating between vertical sticks and horizontal lines. Collision detection is done after mesh generation and edge ordering. The printing direction is determined locally by sampling the hemisphere and searching for collision-free directions. Although this approach works well for simple meshes generated from UV maps, it can get stuck when there is no feasible direction for certain edges, due to the fixed edge ordering.

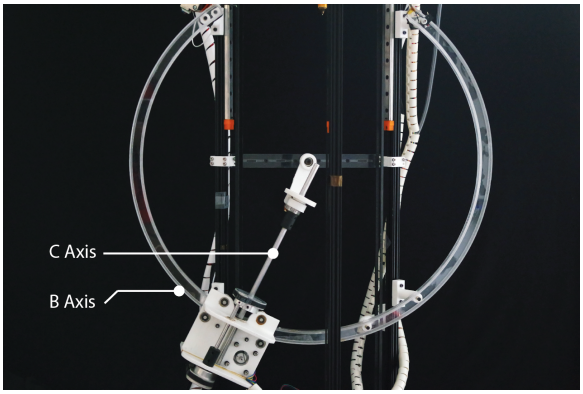

Figure 2: *5DOF Printer*

5-axis CNC machines have been widely used in the industry for manufacturing freeform surfaces [Lasemi et al. 2010]. Tool path generation and orientation identification are two main issues in freeform surface machining. Methods based on the configuration space (the space of parameters for the degrees of freedom) have been proposed for 5-axis machining. A boundary search algorithm can be used to find a set of feasible tool configurations, or a path through the configuration space, to eliminate collision and optimize the tool orientation control [Jun et al. 2003]. While these methods work for the subtractive machining process, they cannot be applied directly to find feasible printing plans for additive fabrication, in which many parts may only be reachable before other parts are printed. Slicing and tool path generation for multi-axis additive manufacturing has also been studied [Singh 2004; Pan et al. 2014; Lee and Jee 2015]. However, these works used heuristics-based approaches for scheduling and determining tool orientations, so feasible planning is not guaranteed. These systems provide excellent potential applications for our collision avoidance algorithm, which would enable them to fabricate complex geometry for which finding a collision-free plan is non-trivial.

Many methods have been proposed to compute high quality triangular or quadrangular meshes, and the motivation of this work is to fabricate such meshes, benefiting from their ability to represent shapes well with few polygons. We refer readers interested in meshing algorithms to [Botsch et al. 2006] and [Bommes et al. 2013] for surveys of triangle and quadrilateral meshing methods. Many of the quadrilateral meshing algorithms are field guided methods [Dong et al. 2005; Kälberer et al. 2007; Bommes et al. 2009], in which a cross field is generated over the surface to specify the orientations and sizes of the quadrilateral elements. Many examples in this paper are created using Instant Field-aligned Meshes [Jakob et al. 2015].

3 Overview

Our goal is to print the wireframe of any mesh for which there exists a feasible printing order; we call these *printable* meshes. On the hardware side, our 5DOF printer (Figure 2) is modified from a standard delta 3D printer using the same design as in On-the-Fly Print. In addition to the position of the extruder in 3D Cartesian space, two extra degrees of freedom are added by including two rotation axes for the platform. In the model's frame of reference, this means the edges can be printed with the extruder approaching

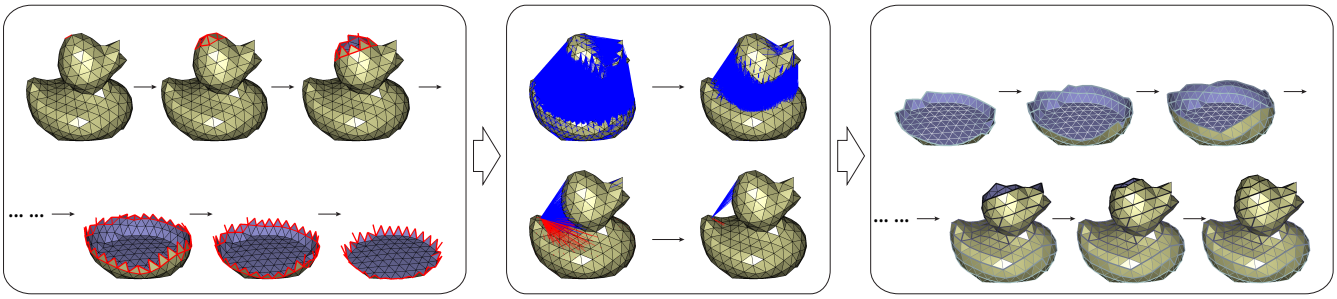

Figure 3: *This figure illustrates the 3 phases of our algorithm. Left: The peeling algorithm recursively peels a set of edges that can be printed last, which ensures that all edges have collision-free printing orientations when scheduled in the reverse order (Sec. 4.2). The thick red edges show the next level of edges to be peeled. Middle: After peeling, we remove as many ordering constraints as possible to leave flexibility for scheduling optimization (Sec. 4.3). The middle panel visualizes the constraints before (left) and after (right) removal. The top row shows constraints between all pairs of edges. The bottom row shows the constraints associated with a particular edge, where the red lines are incident collision arcs and blue lines are outgoing collision arcs. Right: The edge ordering algorithm iteratively advances the contour of the printed model, grouping edges into strips for continuous printing (Sec. 5). Thick edges indicate strip boundaries and grey scale indicates the order (from white to black).*

from any direction in the upper hemisphere; we call this direction the *printing orientation* of the edge.

To print a wire mesh on this machine, we need to sequence its edges and provide five coordinates for each vertex of an edge: three Cartesian coordinates of the extruder plus two rotation angles of the platform. Thus on the software side, we need to order the edges and determine the rotation angles of the platform for each edge, and a feasible printing plan must obey two types of constraints: support and collision.

Support constraints require that every edge must be connected to already printed parts. Collision constraints require that the nozzle does not collide with printed edges. Either type is readily solved on its own by polynomial-time algorithms, leading to a choice of two strategies: enforce the support constraints first, then fix the solution to avoid collisions; or plan around collisions first, then choose a support-compatible ordering. In our experience, algorithms considering support in isolation often paint themselves into corners where collision-free orientations don't exist, and heuristics don't help a lot to solve this. Since support is easier to satisfy, we choose to enforce collision constraints first.

This leads to the three phases of our method, illustrated in Figure 3. We first use an algorithm that peels layers from the model, starting with the edges to be printed last, constructing a set of constraints on the ordering which ensure that collision-free printing orientations will exist for every edge. These constraints are encoded as *collision arcs* in a directed graph in which each node represents a mesh edge, and each collision arc represents a constraint indicating that one mesh edge must be printed before another.

Second, we reduce that set of constraints by locally removing redundant collision arcs to provide more ordering flexibility.

Finally, we order the edges, building the model back up by adding strips one at a time, ensuring that every edge is attached to already printed ones while observing the constraints found in the first phase. In each iteration, we advance the contour separating printed edges from unprinted edges by one step, first printing the edges between the old and new contours, then printing the new contour. In order to keep the contours smooth, we define a cost on each edge, and we observe that minimizing the cost of the new contour corresponds to finding a minimum cut on a dual graph to a portion of the surface.

Symbol	Meaning
G	Full collision graph
G'	Subgraph
V	Set of nodes in collision graph
E	Set of arcs in collision graph
E'	Subset of arcs
D	A discrete set of all available printing orientations
$R_c(e)$	Orientations associated with arc e
$S_c^i(v)$	Orientations causing self-collision when printing node v from the i-th vertex
$D_o^i(v)$	Open orientations of node v
$D_c^i(v)$	Closed orientations of node v
S^i	i-th peeling layer
S	$S = S^0 \bigcup S^1 \bigcup ... \bigcup S^n$
R	Remaining set of nodes after peeling

Table 1: *A table of symbols used in Sec. 4*

We explain the algorithms in detail in Sections 4 and 5.

4 Avoiding Collisions

An algorithm that makes only local decisions in scheduling the edges often gets into a situation where the next edge cannot be printed in any orientation without the nozzle colliding with already-printed parts of the model (Figure 4). Further, achieving good results in practice requires batching edges into contiguous strips, and greedy strip-building methods run into unprintable edges even more often. In this section we propose an algorithm that solves this problem by constructing a partial order on the set of edges such that any compatible total order will always have a feasible printing orientation for every edge. By using as few constraints as possible, we leave flexibility for our edge ordering algorithm (Section 5) to construct good strips while avoiding collisions.

4.1 Problem Formalization

When printing an edge A in a particular orientation, there is a collision if and only if some previously printed edge B overlaps the volume occupied by the printer mechanism. A collision can be

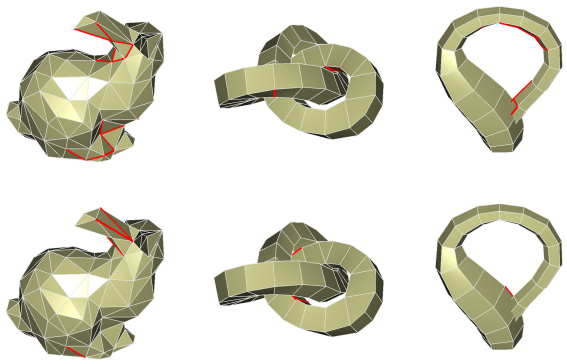

Figure 4: *Naive traversal methods often lead to unprintable edges. In the top row the edges are ordered by breadth first search. The red edges have no collision-free orientations so they cannot be printed. Ordering the edges in ascending order of height (bottom row) does a better job, but still results in some unprintable edges.*

avoided either by printing A before B or by printing A in a different orientation where B does not interfere. Our formulation centers on a *collision graph* that encodes these relationships between printing order, printing orientation, and collision.

In the full collision graph $G(V, E)$, nodes correspond to mesh edges and directed arcs correspond to collision constraints: $(v_a, v_b) \in E$ means that printing v_a after v_b causes a collision in at least one orientation. To avoid ambiguity, we will write "edges" for edges of the model to be printed (nodes in G), and "arcs" for edges in G. And we will write "vertices" for vertices of the mesh and "nodes" for vertices in G. For example, in Figure 5, G is the full collision graph of a tetrahedron. The six nodes in the graph correspond to the six edges of the tetrahedron.

Because printability depends on orientation, each arc and node of G has an associated set of orientations, selected from a finite sampling $D \subset S^2$. Given an arc in the graph, the map $R_c : E \mapsto (2^D \setminus \{\emptyset\})$ records the set of colliding orientations for the pair of nodes on this arc: $\omega \in R_c((v_a, v_b))$ means that v_a cannot be printed in orientation ω after v_b has been printed; if $\omega' \notin R_c((v_a, v_b))$ then it is safe to print v_a after v_b in orientation ω'. For example, in Figure 5, there are three printing orientations. The red sectors of the circle on an edge indicate the set of orientations that cause collision. For example, edge b cannot be printed in orientations 2 after edge e is printed, so there is an edge from b to e and orientation 2 is colored red on this edge. In practice, we uniformly sample 305 directions on the upper hemisphere for D, and detect collision for every pair of edges by checking whether the nozzle geometry intersects one edge when moving along the other.

It is also possible for an edge to collide with itself while printing from some orientations; to encode this there are also two maps $S_c^1, S_c^2 : V \mapsto 2^D$ that record the infeasible orientations due to self-collision: $\omega \in S_c^1(v)$ (resp. $S_c^2(v)$) means it is not safe to print v in orientation ω when starting from the first (resp. second) endpoint.

With the constraints of the problem encoded in G, R_c, and S_c^i, the collision avoidance problem can be formalized in terms of choosing a subgraph $G'(V, E' \subset E)$ of the full collision graph, which represents the particular subset of collision constraints we intend to follow: $(v_a, v_b) \in E'$ means that we will print v_a before v_b. G' needs to have two properties to be a solution: first, it must be acyclic, because a cyclic graph will have conflicting constraints, in which two mesh edges must each be printed before the other; and second, there must be some feasible printing orientation for every edge. These two properties are in tension: more edges in the subgraph put more constraints on the ordering, so we have more guaranteed feasible orientations, but the subgraph is also more likely to be cyclic.

We say that an orientation ω is *open* in G' for a node v if the constraints in E' guarantee that v can be safely printed in orientation ω; otherwise we say ω is *closed* in G' for v. It follows from the meaning of R_c and S_c^i that in $G'(V, E')$, the set of open orientations of a node v is

$$D_o^i(v) = \Big(\bigcap_{e \in E_v \setminus E'_v} \bar{R}_c(e) \Big) \bigcap \bar{S}_c^i(v), \quad i = 1, 2 \qquad (1)$$

where E_v and E'_v are the outgoing arcs of v in G and G' respectively; $\bar{R}_c(e)$ and $\bar{S}_c^i(v)$ are the complements of $R_c(e)$ and $S_c^i(v)$ in D respectively. That is, an orientation of a node v is open if and only if (a) it does not cause self-collision and (b) the subgraph requires all nodes that would prevent printing v in this orientation to be printed after v. For example, in G'_3 of Figure 5, orientations 2 and 3 of a are open. Although printing a in these orientations would collide with e and f, the arcs present in G'_3 require e and f to be printed after a, so printing a in these orientations is safe. Orientation 1 of a is closed, because printing a in this orientation collides with d, and d may be printed before a since arc (a, d) is absent in this subgraph.

As with S_c^1 and S_c^2, D_o^1 is the set of orientations that are open for printing v starting from its first endpoint, and D_o^2 is for starting from its second endpoint. And the set of *closed orientations* is $D_c^i(v) = D \setminus D_o^i(v)$.

A node v is called open if $D_o^1(v) \cup D_o^2(v) \neq \emptyset$. Finally, the goal can be stated briefly: find an acyclic subgraph $G'(V, E')$ for which all nodes are open. We call such a graph a *feasible acyclic subgraph*. In Figure 5, G'_2, G'_3 and G'_4 are all feasible acyclic subgraphs. In the full graph G, all orientations of all nodes are open, but it's not a solution because it's cyclic and we cannot find an order that obeys all the constraints. In the empty graph G'_1, there are no constraints so we can print the nodes in any order, but it's not a solution either, because nodes a, b and c are closed and not guaranteed to be collision-free.

4.2 Resolving the Collision Graph

Now we describe our algorithm for finding a feasible acyclic subgraph of G by removing and adding back arcs.

The first phase of our algorithm is inspired by the observation that, for any printable mesh, there must be edges that can be printed after all the others have been printed. We can print these edges last so they do not interfere with others. And if we put them at the end of the sequence, some of the other edges can be printed just before them. We can do this recursively until all the edges are considered. The procedure is like peeling an onion and we call it the peeling algorithm (Algorithm 1).

In each iteration of the peeling, we can either peel all nodes that are newly opened, or just peel a subset of them. In practice we found that peeling all newly opened nodes often puts too many constraints on the order of edges, such that edges have to be printed without support, see Figure 6. To address this we want to avoid deciding to print edges far from the printing platform before those near the platform, so we arrange them to prefer retaining collision arcs that point away from the platform as measured by traversal distance. To achieve this, before the peeling we label each node with the length

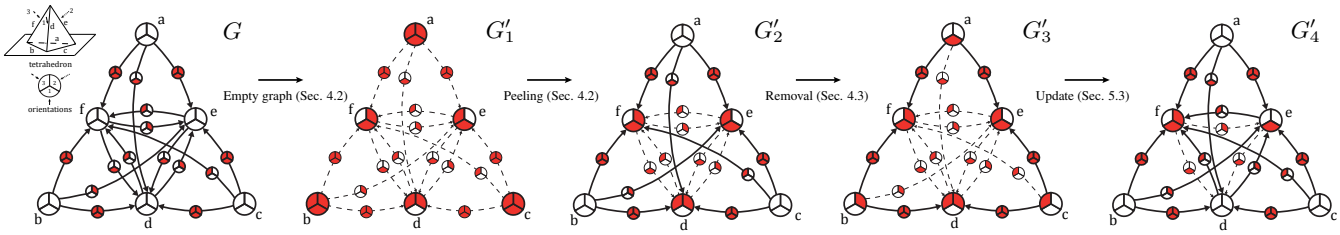

Figure 5: *This figure illustrates the states of the collision graph for a tetrahedron, in various phases of our algorithm. Each node (large circles) and arc (curves) is annotated with a set of orientations; each sector of the circle represents a printing orientation. For the nodes, red means closed and white means open. For the arcs, red sectors are orientations that cause collisions. Dashed curves are removed arcs. G: full collision graph (Sec. 4.1). G'_1: empty collision subgraph (Sec. 4.2). G'_2: result of peeling (Sec. 4.2). G'_3: result of constraint removal (Sec. 4.3). G'_4: updated state after printing order is determined (Sec. 5.3).*

Algorithm 1

Label each node with the length of its shortest path from platform.
Remove all arcs in the graph.
Let S^0 be the set of nodes that are open and have maximum label.
$i \leftarrow 0$.
while (not all nodes are open) && ($S^i \neq \emptyset$) **do**
  Add back all arcs incident to nodes in S^i, except for arcs from $S^j, j \leq i$.
  $S^{i+1} \leftarrow$ the set of nodes that are open after adding these arcs and have the maximum label.
  $i \leftarrow i + 1$.
end while
$n \leftarrow i$.
$R \leftarrow$ the set of nodes that are not opened by the process above.
if $R == \emptyset$ **then**
  We can print in the order of $S^n, S^{n-1}, ..., S^0$.
  Nodes in the same subset S^k can be printed in arbitrary order.
else
  We cannot resolve this collision graph.
end if

of the shortest path between the platform and itself. Then at each step, we only peel the nodes that have the maximum label among the newly opened ones. This makes the algorithm tend to follow the reverse traversal order and add a small set of constraints each time, which reduces the chance of having to print unsupported edges.

A simple example of the peeling process is shown in Figure 5. We start with the full collision graph G and remove all arcs to get the empty graph G'_1. In G'_1, nodes d, e and f are still open, so $S^0 = \{d, e, f\}$. Then we add edges incident to the nodes in S^0 and get graph G'_2. In G'_2, nodes a, b and c are also opened, so $S^1 = \{a, b, c\}$ and $R = \emptyset$. The algorithm has found a solution.

It is straightforward to prove by contradiction that the algorithm is complete, meaning that it will find a feasible acyclic subgraph if one exists. Suppose that the algorithm fails: there is a feasible acyclic subgraph, but $R \neq \emptyset$ and the algorithm concludes no solution exists. In the feasible acyclic subgraph, at least one node in R has no outgoing arcs in $R \times R$, otherwise there are cycles in the graph. Therefore, this node can be opened by adding only arcs in $R \times S$, where $S = S^0 \bigcup S^1 \bigcup ... \bigcup S^n$. Since we added all arcs in $R \times S$ in the algorithm, we should have opened this node in R, which is a contradiction. Therefore, considering collision constraints only, the peeling algorithm will always find a solution if one exists. And when the algorithm cannot find a solution, there is no feasible order

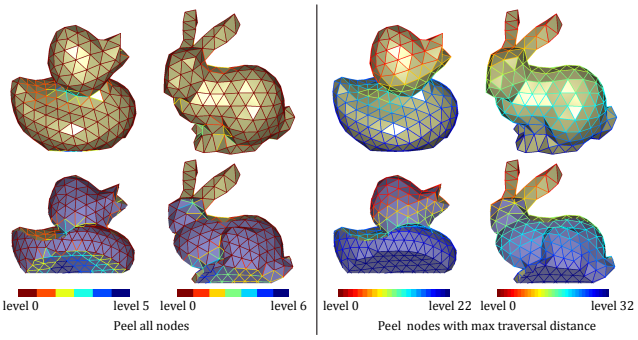

Figure 6: *This figure shows the results of the peeling algorithm. In the bottom row, half of the models are cut away to show in edges in the back. The color map indicates the levels of nodes during the peeling (S^i). Note that after peeling all constraints point from higher to lower numbered S^i. The left two columns shows the results of peeling all newly opened nodes, which results in fewer levels but a lot of unsupported edges (e.g. the blue edges on the back of the duck). The right two columns show the results of peeling only the maximum traversal distance nodes in the newly opened set, which results in more levels and the constraints are more consistent with traversal order.*

to avoid collision.

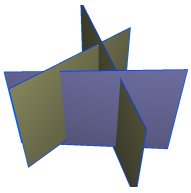

Figure 7: *The three edges on the top are cyclically dependent in the collision graph, and cannot be printed in any order.*

This proof also gives us some insight into the problem. An infeasible set of nodes is one in which every node in the graph is closed initially, *i.e.* $S^0 = \emptyset$. That is, not a single edge can be printed in any orientation, if we consider interference with all the other edges. Our proof shows that this is the only case where no collision free order exists, if we don't consider the support constraint, to be discussed in Section 5. This usually happens when there are edges close to each other that form a cyclically dependent configuration; see the example in Figure 7. This situation can in principle be resolved by splitting the edges into smaller pieces and printing them separately, so that the cyclic dependency is broken, though we have not encountered this problem in practice or implemented this

fix.

4.3 Removing Redundant Constraints

The peeling algorithm gives us a partial order following which we can sort the edges into a collision-free sequence. However, the peeling algorithm's goal is simply to find *any* feasible acyclic subgraph, and in the process it adds many redundant collision arcs that can cause unnecessary difficulty in ordering. We can actually remove many of these constraints: as long as we don't close any node in the graph, we still have a feasible acyclic subgraph. So in order to have more freedom in the ordering phase, we remove the redundant arcs after the peeling algorithm to find a set of constraints that is as small as possible.

In the removal process, we go through every node on the graph. For each node, we iterate through its outgoing arcs, and check whether removing the arc will close that node. If not, we remove that arc and close associated orientations. Then we check the next arc. The order in which the arcs are considered affects the result because it affects which constraints are left, and an inappropriate remaining set of constraints causes failure to find a feasible solution (Figure 11). We apply the heuristic of removing arcs first that most conflict with a simple ordering heuristic: we check and remove the arcs in an ascending order of the height (distance from the plane of the platform) of the arc's ending node.

In Figure 5, after peeling, we remove three arcs and get graph G_3'. Removing these arcs closes one orientation for each of a, b and c, but creates more flexibility in ordering (e.g. e can be printed before b). The remaining arcs cannot be removed because removing them would close nodes.

5 Scheduling Mesh Edges

In this section we explain how we schedule the edges for printing. In addition to the support and collision constraints, we observed that edge sequences with certain favorable properties tend to make the printing process robust and to improve print quality: (1) the edges should connect back to printed parts as frequently as possible; (2) long, straight moves that connect several previously printed parts are favorable.

The slicing based approach of WirePrint and UV map based approach of On-the-Fly Print are good examples of ordering methods with these properties. We generalize their ideas to handle arbitrary input meshes.

For description convenience we assume that the mesh is a manifold, though the idea can also be applied to non-manifold meshes except for the refinement in section 5.2. Our edge ordering algorithm starts with the platform edges, denoted by E_r. Then we repeatedly add edges to the printing sequence until the whole mesh is covered. In each iteration, we find the printed edges of the faces that are not completed, which we call the old contour C_{old}, and we denote the vertices on C_{old} by V_0. The set of vertices that are separated from V_0 by one unprinted edge is V_1. The set of unprinted edges that connect two vertices in V_0 is E_0; and those connecting two vertices in V_1 are E_2, which forms the new contour C_{new}. The set of unprinted edges between V_0 and V_1 is E_1. After determining these sets (Figure 8 top), we first print the unconstrained (Section 5.1) edges $E_1^* \subseteq E_1$ and then print the unconstrained edges $E_2^* \subseteq E_2$. (Algorithm 2)

Assuming that the input mesh is a single connected component, this algorithm can traverse the whole mesh and terminate unless the collision constraints are not compatible with traversal. When there are unprinted edges, E_0 and E_1 cannot be both empty. So as long as E_0^* and E_1^* are not both empty, the algorithm will always find new edges to print until all edges are printed.

Algorithm 2

Print the platform edges E_r.
while not all edges are printed **do**
  Find the vertex sets V_0 and V_1
  Find the unprinted edge sets E_0, E_1 and E_2
  Print unconstrained edges $E_0^* \subseteq E_0$
  Print unconstrained edges $E_1^* \subseteq E_1$
  Print unconstrained edges $E_2^* \subseteq E_2$
end while

5.1 Integrating Collision Constraints

Now we explain how to consider collision constraints in the ordering algorithm to make sure that the ordering does not conflict with them. Let $\tilde{E}$ denote the set of unprinted edges. When we attempt to print an edge $e \in E_0$, we need to check whether there is a collision arc path from $\tilde{E} \setminus E_0$ to e. If there is, then e should not be printed for now and has to wait for the next iteration, otherwise $e \in E_0^*$. Similarly, we need to check for a collision arc path from $\tilde{E} \setminus (E_0^* \bigcup E_1)$ for E_1 edges and check for a path from $\tilde{E} \setminus (E_0^* \bigcup E_1^* \bigcup E_2)$ for E_2. Collision arcs between edges in the same-subscript E set do not affect what can be printed in the current iteration, because the same-subscript edges can be ordered according to these constraints. An extra supporting constraint needs to be considered for E_2 edges. Since we only print a subset of E_1, some edges in E_2 may be unsupported, which should be excluded from E_2^*.

It is possible for the ordering algorithm to fail due to collision constraints. But this only happens if the collision constraints are incompatible with *any* traversal order of the mesh. To see this, note that the ordering algorithm will only get stuck when every edge in $E_0 \bigcup E_1$ has collision paths from $\tilde{E} \setminus (E_0 \bigcup E_1)$. In this case the algorithm will proceed until it reaches a state where every unprinted edge adjacent to the printed part has incident collision paths from the unprinted edges that are not adjacent to the printed part (Figure 11). This defines a cut through the mesh that can never be crossed by any traversal order, since every edge on the cut can only be printed after the algorithm has crossed the cut. Therefore, if the ordering algorithm gets stuck, it means that the collision constraints force us to print some unsupported edges, and there is no feasible solution to the ordering problem. This does not imply there is no solution to the full scheduling problem: the collision phase chooses some feasible acyclic subgraph, but it is possible that there exists a different such subgraph that would be traversal-compatible.

5.2 Smooth Contours

For better printing quality, we expect the contours to be smooth and not to fold too much, so we refine algorithm 2 to take special care when it advances the printed part. We take the edges in $E_1 \bigcup E_2$ as the candidate set and find a subset to form a smooth new contour. With a smooth scalar field, height for example, we let the cost of an edge be the difference of field values at its two vertices, by minimizing which we can find a contour that follows the isopleth of the field. We observed that the minimum-cost contour on the mesh corresponds to the minimum cut on the dual graph (Figure 8 bottom). We find the minimum cut on the dual graph and use the corresponding edges as C_{new}. The set of vertices on C_{new} and not on C_{old} is V_1'. And the set of edges between V_0 and V_1' is E_1'. In each iteration we only print the unconstrained edges in E_1', and then those in C_{new}. A special case is when the adjacent edges of a vertex in V_1

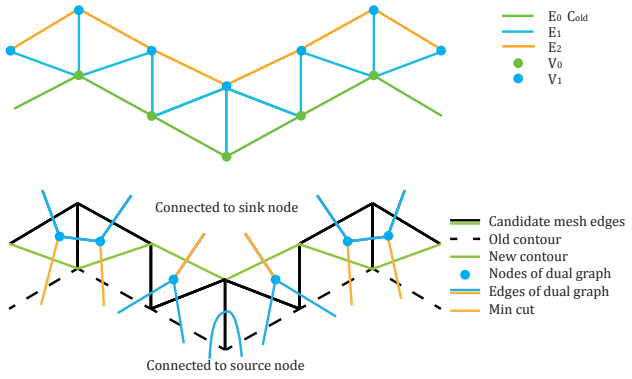

Figure 8: *Top: Sets of edges and vertices in Sec. 5. Bottom: Min cut on the dual graph corresponds to minimum cost contour on the mesh. To force the contour to advance, we don't include the edges on the old contour in the dual graph.*

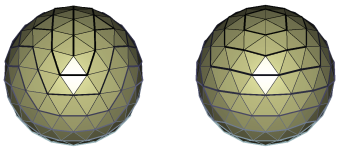

Figure 9: *The contours (thicker edges) on an icosphere found by the ordering algorithm without (left) and with (right) the min cut strategy.*

are all in E_1 (e.g. the top of the right sphere in Figure 9). In this case $E_2 = \emptyset$ and there is no new contour. When we detect such cases, we directly print the unconstrained edges in E_1.

It is possible that the refined algorithm gets stuck when the all E_1' edges have incident collision arcs from $E_1 \setminus E_1'$. When this happens we fall back to the strategy of Algorithm 2.

Figure 9 shows a comparison of the contours on an icosphere found by our ordering algorithm. With the min cut strategy, the contours are flattened by taking small advances at certain places.

5.3 Grouping Edges and Selecting Orientations

After figuring out the set of edges in E_0^*, E_1^* and E_2^*, we try to group the edges in each E_i^* into continuous strips. A greedy algorithm is used to find as long as possible paths in each set. We start from an arbitrary edge, add one of its unprinted adjacent edges in E_i^* and do this repeatedly until no edge can be added. We use the heuristic to prefer adding an edge that has unprinted adjacent edges in order to find long paths. Then we repeat to find other strips until all edges in E_i^* are printed. Collision constraints should be considered here: if an edge has incoming collision arcs from unprinted edges when we consider adding it, it should not be added for now.

Once we have ordered the edges into strips, we update the printable orientations of every edge based on the actual printing sequence. For example in Figure 5, we decide to print the edges in alphabet order. Only three arcs in the graph violate this order so they are removed, and the rest are preserved in G_4'. Given this subgraph, all orientations of nodes $a - d$ are open, and orientation 3 of e is also open since f is actually printed after e.

Then we need to assign each edge a printing orientation from among the feasible possibilities. To do this, we build a graph with a node for every pair (v, ω) where v is a edge in the strip and ω is a feasible orientation for that edge. Every pair of nodes belonging to two edges adjacent in the printing sequence is connected by an edge in the graph, labeled with a cost equal to the angle between the two orientations. We find a shortest path in this graph joining the first mesh edge to the last, which selects a sequence of orientations that has the minimum total rotation.

6 Results

We printed a variety of meshes that previous wireframe printing methods cannot, and show the results in Figure 10. The Klein bottle is meshed following its parameterization; the coarsest bunny and the fertility meshes were simplified from dense meshes using an edge-collapse method; and the other meshes were all created using Instant Field-aligned Meshes [Jakob et al. 2015].

We printed 3 bunnies with various meshing styles. In the simplified bunny, several edges on the ears are badly constrained by the neighbors (Figure 4). Our algorithm finds a collision-free plan for this mesh. Another two bunnies are printed with regular triangular and quadrilateral meshing, to show that our method enables printing nice meshes for better surface depiction.

A quadrilateral fandisk mesh is printed and compared against the result of a WirePrint-style mesh generated by the Cura slicer software (Figure 12). The features of the shape are much better preserved in the quad mesh and WirePrint has a hard time approximating the flat surface on the top.

The Klein bottle, trefoil knot and fertility model show our method's ability to find collision-free printing plans for intersecting surfaces, complex shape, and complex topology.

The duck is 8 cm tall and takes about 55 minutes to print, with much of the time spent pausing for the filament to cool. The printing can be considerably accelerated by using better cooling system, but improving the hardware is beyond the scope of this paper.

The time to compute the plans is much shorter, and is dominated by collision sampling time, which depends on the number of sample points along each edge. We generally take 3 samples per edge, which takes up to 5 minutes on a modest laptop computer. The rest of the algorithm completes in less than 20 seconds.

Failure cases: An example where our algorithm fails to find a traversal is a cube with a blind hole in the top face (Figure 11). After the peeling and redundant arc removal, there are still collision arcs pointing from edges in the hole to edges on the top. The traversal algorithm gets stuck on the top because of these constraints. The root problem is that the path-length-based ordering heuristic used to select which collision arcs to keep is inappropriate for this example. If we rotate model by 90 degrees so that the hole is on the side, the algorithm finds a feasible plan.

7 Conclusions and Future Work

In this paper we propose an algorithm to generate printing plans for 5DOF wireframe printers given arbitrary input meshes. Our collision avoidance algorithm finds a locally minimal set of constraints on the order of edges, by following which the mesh can be printed without collision. The algorithm constrains the ordering as little as possible to leave freedom to order the edges in the next phase. Our edge ordering algorithm is designed to generate a sequence of edges that will make for smooth progress when printing. A number of results have shown that our algorithm enables us to print wire meshes that are infeasible with previous methods.

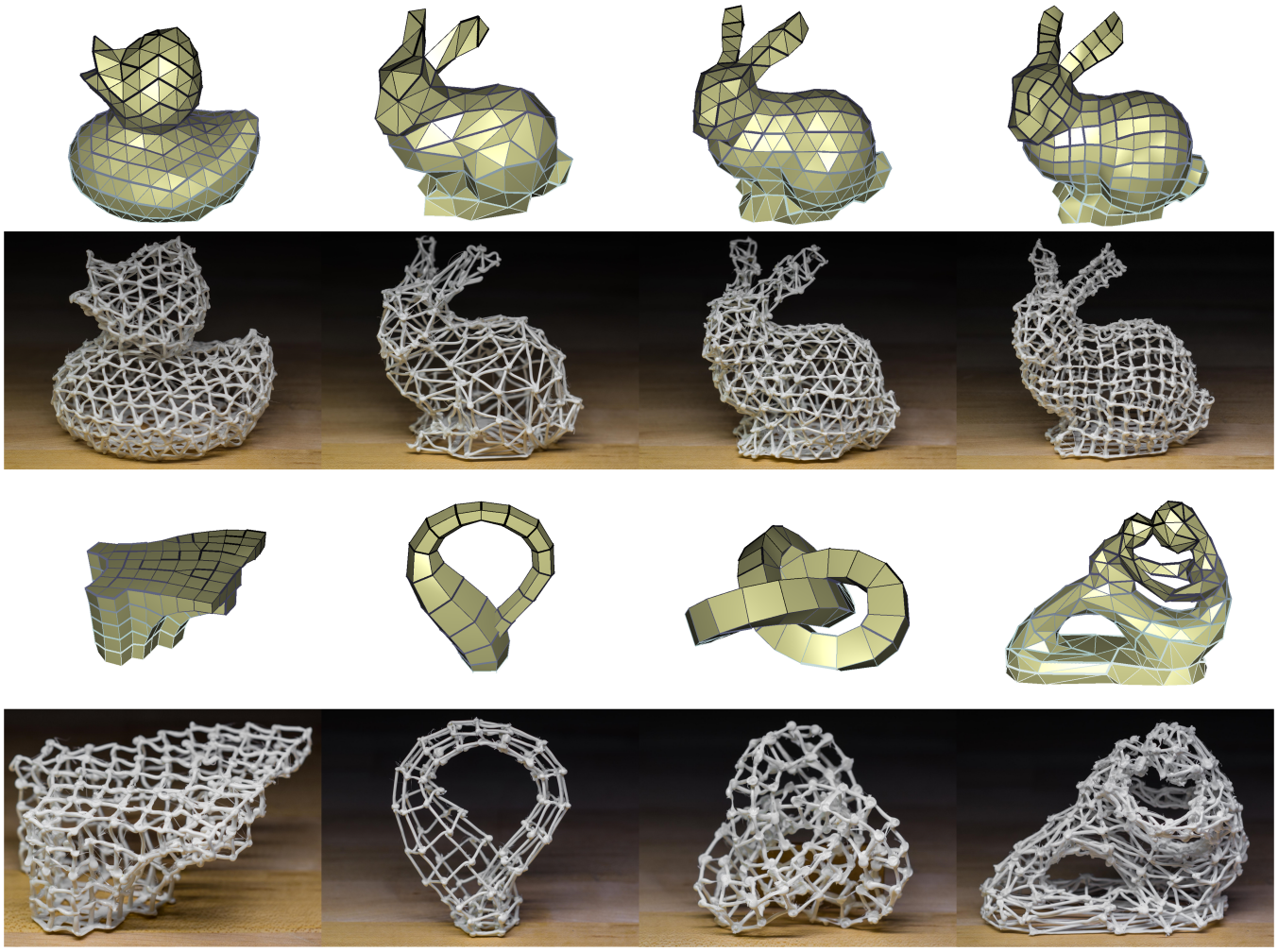

Figure 10: *Printing results. In each pair of rows, the top shows the order of edges, in which the contours are indicated by thicker lines and order is indicated by gray scale (from white to black). The bottom shows the meshes printed by the 5DOF printer.*

Our method is the first to allow wireframe printing of meshes that are not specially designed for wireframe printing—a fundamental enabling technology needed for further progress in wireframe printing. The ability to print nearly any mesh opens many possibilities in moving FDM printing beyond the strictly layer-wise approach, including designing wire meshes to have particular mechanical properties or using wire meshes as scaffolds to print solid surfaces faster and with less material for a given strength. Moreover, with a different traversal approach, our scheme can readily be adapted to non-surface-like lattice structures.

Limitations and future work: One significant limitation is that our algorithm can fail to find a plan for some printable meshes (Figure 11), when the first phase selects collision constraints that are incompatible with *any* traversal order. How to find traversal-compatible constraints remains an open question. Besides, the bottleneck of our algorithm's computation cost is at sampling collisions for every pair of edges, which is quadratic, so efficiency is an issue for dense meshes. Although the computational time is much less than the printing time, one might want to accelerate it in order to support applications at interactive rate. There is much potential in speeding up the collision sampling using some hierarchical structures.

Our collision algorithm finds a small set of constraints on the order of edges, which leaves a lot of flexibility in the ordering. Any algorithm that traverses the edges can fit in our scheme. Currently we use height to guide the ordering, which might not be the best choice for all meshes. One future topic is to explore better ordering methods for different models, such as trying other scalar fields on the mesh as guidance.

Improving the hardware is another direction for future work. Currently our printing is limited by the printer prototype. We used air cooling for the results, which results in a lot of sagging and a lot of time spent waiting for the edges to solidify. Atomized spray cooling can speed up printing and reduce sagging. The printer uses a remote extruder with a flexible tube to feed filament to the head, which affords relatively inaccurate control over starting and stopping extrusion. This results in blobs at vertices when starting extrusion and strings caused when stopping. These artifacts can be reduced by improving the extruder and readjusting the extrusion control parameters. With better hardware, we could also print longer, straighter beams and larger objects, which would improve the visual quality of our results.

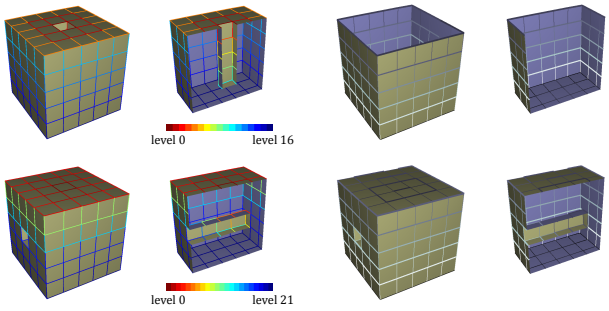

Figure 11: *A cube with a hole in the top. The left two columns show the levels of edges, and the right two show the printing order. Top row: the ordering algorithm gets stuck because the collision constraints require the edges on the top face to be printed after the edges in the hole, but the edges in the hole cannot be printed without printing at least part of the top first. Bottom row: a feasible plan can be found if the model is rotated by 90 degrees.*

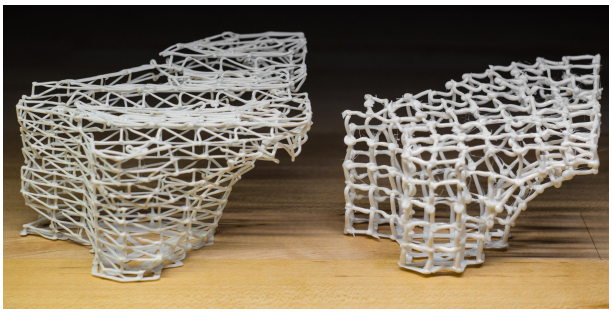

Figure 12: *Left: a WirePrint-style mesh of the fandisk model generated by Cura. The mesh quality suffers due to slicing: shape features are not preserved, and the top surface is poorly approximated. Right: a quadrilateral mesh printed using our approach, which depicts the fandisk shape better.*

Acknowledgements

This work was made possible by funding from the National Science Foundation under grants IIS-1422106, IIS-1513967 and IIS-1011919, and by generous support from Autodesk Corporation. We also thank John (Spike) Hughes for his insightful comments and suggestions.

References

BÄCHER, M., BICKEL, B., JAMES, D. L., AND PFISTER, H. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. (Proc. SIGGRAPH) 31*, 4.

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3.

BOMMES, D., LVY, B., PIETRONI, N., PUPPO, E., SILVA, C., TARINI, M., AND ZORIN, D. 2013. QuadMesh Generation and Processing: A Survey. *Computer Graphics Forum*.

BOTSCH, M., PAULY, M., ROSSL, C., BISCHOFF, S., AND KOBBELT, L. 2006. Geometric modeling based on triangle meshes. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06.

CALÌ, J., CALIAN, D. A., AMATI, C., KLEINBERGER, R., STEED, A., KAUTZ, J., AND WEYRICH, T. 2012. 3D-printing of non-assembly, articulated models. *ACM Trans. Graph. 31*, 6.

CURA. Cura. https://ultimaker.com/en/products/cura-software. Accessed: 2015-12.

DONG, S., KIRCHER, S., AND GARLAND, M. 2005. Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer-Aided Geometric Design 22*.

JAKOB, W., TARINI, M., PANOZZO, D., AND SORKINE-HORNUNG, O. 2015. Instant field-aligned meshes. *ACM Trans. Graph. (Proc. of SIGGRAPH Asia) 34*, 6.

JUN, C.-S., CHA, K., AND LEE, Y.-S. 2003. Optimizing tool orientations for 5-axis machining by configuration-space search method. *Computer-Aided Design 35*, 6.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. Quadcover-surface parameterization using branched coverings. In *Computer Graphics Forum*, vol. 26.

LASEMI, A., XUE, D., AND GU, P. 2010. Recent development in cnc machining of freeform surfaces: A state-of-the-art review. *Computer-Aided Design 42*, 7.

LEE, K., AND JEE, H. 2015. Slicing algorithms for multi-axis 3-d metal printing of overhangs. *Journal of Mechanical Science and Technology 29*, 12, 5139–5144.

MUELLER, S., IM, S., GUREVICH, S., TEIBRICH, A., PFISTERER, L., GUIMBRETIÈRE, F., AND BAUDISCH, P. 2014. Wireprint: 3D printed previews for fast prototyping. In *Proc. UIST*.

PAN, Y., ZHOU, C., CHEN, Y., AND PARTANEN, J. 2014. Multi-tool and multi-axis computer numerically controlled accumulation for fabricating conformal features on curved sufaces. *Journal of Manufacturing Science and Engineering*.

PANETTA, J., ZHOU, Q., MALOMO, L., PIETRONI, N., CIGNONI, P., AND ZORIN, D. 2015. Elastic textures for additive fabrication. *ACM Trans. Graph. (Proc. SIGGRAPH) 34*, 4.

PENG, H., WU, R., MARSCHNER, S., AND GUIMBRETIÈRE, F. 2016. On-the-fly print: Incremental printing while modeling. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

PÉREZ, J., THOMASZEWSKI, B., COROS, S., BICKEL, B., CANABAL, J. A., SUMNER, R., AND OTADUY, M. A. 2015. Design and fabrication of flexible rod meshes. *ACM Trans. Graph. (TOG) 34*, 4.

PRÉVOST, R., WHITING, E., LEFEBVRE, S., AND SORKINE-HORNUNG, O. 2013. Make It Stand: Balancing shapes for 3D fabrication. *ACM Trans. Graph. (Proc. SIGGRAPH) 32*, 4.

SCHUMACHER, C., BICKEL, B., RYS, J., MARSCHNER, S., DARAIO, C., AND GROSS, M. 2015. Microstructures to control elasticity in 3D printing. *ACM Trans. Graph. 34*, 4.

SINGH, P. 2004. A framework for reverse engineering using feature-based geometry reconstruction and multi-directional layered manufacturing.

SLIC3R. Slic3r. http://www.slic3r.org/. Accessed: 2015-12.

WANG, W., WANG, T. Y., YANG, Z., LIU, L., TONG, X., TONG, W., DENG, J., CHEN, F., AND LIU, X. 2013. Cost-effective printing of 3D objects with skin-frame structures. *ACM Trans. Graph. (Proc. SIGGRAPH Aisa) 32*, 5.