Stitch Meshing

KUI WU, University of Utah
XIFENG GAO, New York University
ZACHARY FERGUSON, New York University
DANIELE PANOZZO, New York University
CEM YUKSEL, University of Utah

Fig. 1. *Example yarn-level models generated from input 3D surfaces using our fully automatic pipeline.*

We introduce the first fully automatic pipeline to convert arbitrary 3D shapes into knit models. Our pipeline is based on a global parametrization remeshing pipeline to produce an isotropic quad-dominant mesh aligned with a 2-RoSy field. The knitting directions over the surface are determined using a set of custom topological operations and a two-step global optimization that minimizes the number of irregularities. The resulting mesh is converted into a valid stitch mesh that represents the knit model. The yarn curves are generated from the stitch mesh and the final yarn geometry is computed using a yarn-level relaxation process. Thus, we produce topologically valid models that can be used with a yarn-level simulation. We validate our algorithm by automatically generating knit models from complex 3D shapes and processing over a hundred models with various shapes without any user input or parameter tuning. We also demonstrate applications of our approach for custom knit model generation using fabrication via 3D printing.

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

Authors' addresses: Kui Wu, University of Utah, kwu@cs.utah.edu; Xifeng Gao, New York University, gxf.xisha@gmail.com; Zachary Ferguson, New York University, zfergus@nyu.edu; Daniele Panozzo, New York University, panozzo@nyu.edu; Cem Yuksel, University of Utah, cem@cemyuksel.com.

0730-0301/2018/8-ART130 $15.00
https://doi.org/10.1145/3197517.3201360

Additional Key Words and Phrases: Yarn-level cloth, yarn-level modeling

ACM Reference Format:
Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. 2018. Stitch Meshing. *ACM Trans. Graph.* 37, 4, Article 130 (August 2018), 14 pages. https://doi.org/10.1145/3197517.3201360

1 INTRODUCTION

Knitted garments are common in our daily lives, going from socks and T-shirts to winter clothing and accessories, and are thus ubiquitous in movies and games. There are two good reasons for favoring knitting over its alternatives: knitted fabrics easily stretch and the shaping techniques used in knitting allow producing complex 3D surfaces without any seams.

However, designing knitting pattern for a given 3D surface is still an open problem. Knitting patterns are currently designed using a high level of expertise and numerous iterations of trial and error to figure out how one could knit a particular 3D shape. That is why most knitting patterns used today are merely derivations of a limited number of well-known and well-understood shapes. In computer graphics, stitch meshes [Yuksel et al. 2012] provide a powerful interface for modeling knit garments. However, they still require users to manually design the topology of the given (typically low-resolution) input mesh. This requires the user to know exactly how to knit the desired shape and prepare an input mesh accordingly. Therefore, it is extremely difficult and time consuming to design knitted models for complex and uncommon shapes, like the ones

shown in Fig. 1, each of which would require numerous design iterations by a knitting expert.

In this paper, we introduce the first automatic pipeline to deal with this challenging problem: our method takes a 3D surface as input and the desired stitch size, and automatically produces a topologically-valid yarn-level knit model. The resulting yarn-level model can be either directly used in computer graphics applications with yarn-level simulation or realized in the real world using 3D printing. The challenge we are tackling is the design of a dense network of closed, intertwined yarn curves which are not self-intersecting and hold together thanks to the interlocked curves formed by knitted stitches. The problem is inherently global, i.e. a change in one stitch can affect not only its neighborhood, but the entire shape.

We use stitch meshes [Yuksel et al. 2012] as an intermediate step in our pipeline. We begin by converting the input 3D shape into an isotropic quad-dominant mesh with approximately uniform face sizes using a two-fold rotational symmetry (2-RoSy) orientation field. This process provides a good starting point that minimizes the distortions of the final knit structure. Then, we automatically determine the knitting directions over the entire model surface using a two-step global optimization process along with custom topological operations. Finally, we subdivide the resulting mesh to generate a valid stitch mesh. After the stitch mesh is ready, we can use it to create the final yarn curves. Since the yarn-curves are topologically valid, we can use them with yarn-level cloth simulations. Also, fabricating them using 3D printing produces interlocked curves that form flexible surfaces.

We show the effectiveness of our approach by automatically producing yarn-level knit models for complex 3D shapes (Fig. 1) and its robustness by generating stitch meshes from a large number of complex 3D models. We also present yarn-level models automatically generated using our pipeline and fabricated via 3D printing.

This paper does not address the problems of fabrication via knitting and the models we generate are not guaranteed to be knittable. Also, we assume that each stitch has a roughly square shape, while in reality the height and width of a knitted stitch is often different.

2 BACKGROUND

Before we discuss the details of our method, we briefly overview of modeling fabrics and knit structures as well as the stitch mesh structure [Yuksel et al. 2012] that we use in our pipeline for representing the final yarn-level model. We also provide an overview of prior works on quad-dominant remeshing.

2.1 Cloth Modeling

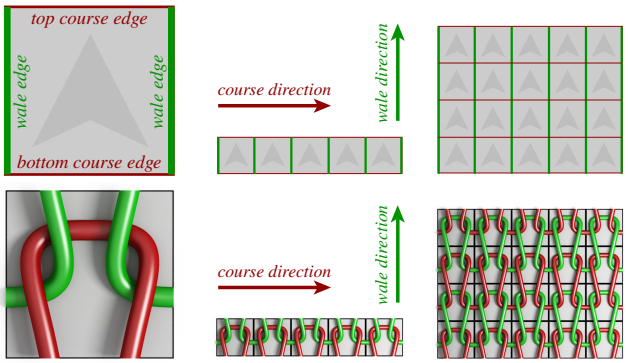
Much of the work in computer graphics involving cloth has been aimed towards simulating woven fabrics using sheet-based representations [Baraff and Witkin 1998; Breen et al. 1994; Bridson et al. 2002; Goldenthal et al. 2007; Grinspun et al. 2003; Volino et al. 2009]. The modeling approaches for sheet-based cloth mainly concentrate on fitting garment models on virtual characters [Berthouzoz et al. 2013; Carignan et al. 1992; Decaudin et al. 2006; Guan et al. 2012; Robson et al. 2011; Turquin et al. 2004; Umetani et al. 2011; Wang et al. 2003]. 3D modeling approaches have also been used in virtual garment prototyping [Luo and Yuen 2005; Volino and Magnenat-Thalmann 2005]. To produce more complex cloth models, Mori and Igarashi [2007] proposed a method for designing 3D plush toys by sketching 2D patterns. More recently, cloth capturing methods using multi-view systems [Bradley et al. 2008], single images [Daněřek et al. 2017; Zhou et al. 2013], 3D scans [Chen et al. 2015], and motion sequences [Pons-Moll et al. 2017] have been shown to successfully produce virtual garment models.

Akleman et al. [2009] introduced a method for converting arbitrary quad-meshes into plain-woven structures using graph rotation systems. While this approach is similar in spirit to our method, the knit structures we produce have entirely different constructions and requirements than plain-woven structures. 3D printing is often paired with computational fabrication techniques. Methods to design and fabricate various structures, such as flexible rod meshes [Pérez et al. 2015], ornamental curve networks [Zehnder et al. 2016], and wireframe meshes [Wu et al. 2016] have been explored in prior work. Our method fits in this trend, allowing the automatic design of 3D printable complex knit structures.

Knit structures are constructed by pulling yarn loops through other yarn loops to form stitches. Shaping techniques, such as *increases* that pull multiple yarn loops through one yarn loop or *decreases* that pull a yarn loop through multiple yarn loops, allow forming complex 3D shapes without introducing seams. Due to the properties of this construction, knit fabrics have low resistance to stretching even if the yarn itself is not stretchable.

Therefore, more complex representations than sheet-based models are favored for knits. Nocent et al. [2001] used a continuum model for simulating knitted cloth. Kaldor et al. [2008; 2010] introduced yarn-level simulation methods for animating knitted cloth models. Cirio et al. [2014] presented a reduced-order model for handling yarn interactions, which is extended to support knit cloth [Cirio et al. 2015, 2017]. Jiang et al. [2017] used the material point method for handling yarn-level interactions.

On the other hand, designing yarn-level models for knits has been a challenging problem. Meißner and Eberhardt [1998] introduced a system to simulate the knit construction process with a simplified yarn-level model. Peng et al. [2004] introduced a texture-based method to add yarn-level details to surface appearance. Igarashi et al. [2008a] introduced a semi-automatic method for generating knit models from an input 3D shape, which is manually segmented into multiple patches of disks or disks with holes. They also presented a sketch-based modeling system for designing plush toys [Igarashi et al. 2008b]. McCann et al. [2016] recently proposed a method to generate machine knitting instructions for knitting 3D models designed by their custom interface, which is limited to simple primitives, such as tubes and sheets. Yuksel et al. [2012] introduced the stitch mesh modeling framework. Stitch meshes provide a mesh-based representation of the yarn-level knit geometry and allow efficiently designing complex 3D knit models with correct yarn-level topology, such that they can be used with yarn-level simulations. However, the stitch mesh modeling framework heavily relies on the topology of the given input mesh, which must be manually constructed, such that the edges align with the knitting directions. The pipeline we introduce in this paper is based on the stitch mesh representation, so we provide a more detailed overview below.



(a) stitch mesh face (b) faces on a row (c) multiple rows

Fig. 2. ***Stitch mesh representation:*** *(a) a typical stitch mesh face and the corresponding yarn-level model, (b) stitch mesh faces on a row, (c) multiple rows of stitch mesh faces representing interlocked stitches on consecutive rows.*

2.2 Stitch Meshes

The stitch mesh structure [Yuksel et al. 2012] is an abstraction of the yarn-level geometry that provides a powerful interface for modeling knit structures. Each stitch-mesh face corresponds to a stitch of the knit structure, shown in Fig. 2a. These faces are placed side-by-side along the *course* knitting direction, forming *rows* (Fig. 2b). Consecutive rows are connected along the *wale* knitting direction (Fig. 2c). Each stitch-mesh face has two *wale edges* that are aligned with the wale knitting direction. Most stitch mesh faces are quads with two wale edges and two *course edges* that are aligned with the course knitting direction and separate the wale edges. The yarn used for knitting the stitch represented by a face enters the face from one of the wale edges, forms the stitch, and then exists the face from the other wale edge (green yarn curves in Fig. 2a). The top part of a yarn loop formed by the stitch in the previous row enters and exits the face from the *bottom course edge* (green yarn curves in Fig. 2a). Similarly, the loop forming the stitch of a face exits and enters the face from the *top course edge*, connecting it to the next row.

Stitch mesh faces can have more than four edges. Faces with multiple top course edges are called *increases*, as they increase the number of stitches on the next row. Similarly, faces with multiple bottom course edges are called *decreases*. The stitch mesh structure also permits faces with no bottom course edges or no top course edges, but such faces must be placed with caution, since they do not form stable stitches and placing them side-by-side may cause the yarn-level model to unravel. That is why we entirely avoid such faces in our framework. Yet, this limitation has no practical consequence, since the yarn-level models including such faces can be represented differently. For example, a triangular face next to a quad face can also be represented by a face with five edges.

2.3 Structured Meshing

The generation of quadrilateral or quadrilateral-dominant meshes has received a lot of attention in the last two decades. We restrict our survey to the most recent works in global and local parametrization, and we refer an interested reader to [Bommes et al. 2013] for a complete survey.

Global parametrization methods [Alliez et al. 2002; Gu et al. 2002; Khodakovsky et al. 2003; Marinov and Kobbelt 2006] flatten the surface after cutting it into a topological disk, generate a regular lattice on the plane, and then lift it back to the original surface, producing a structured mesh. To control edge alignment, it is possible to solve an optimization that strive to align the parametrization gradients to a guiding field [Bommes et al. 2009; Ebke et al. 2014; Kälberer et al. 2007; Nieser et al. 2012]. Designing the guiding field is a difficult problem on its own [Crane et al. 2010; Hertzmann and Zorin 2000; Jiang et al. 2015; Knöppel et al. 2013, 2015; Lai et al. 2010; Palacios and Zhang 2007; Panozzo et al. 2014; Ray et al. 2008], and we refer an interested reader to the recent state-of-the-art report of Vaxman et al. [2016]. These methods fix the singularities of the quadrilateral mesh during the orientation field design, and they thus inevitably introduce distortion in the parametrization (since the orientation field is not integrable [Diamanti et al. 2014]), which results in quads of varying size. While this is not problematic for most remeshing applications, it is not acceptable for stitch meshes, since the size of stitch has to be uniform.

Local parametrization methods [Gao et al. 2017; Jakob et al. 2015; Ray et al. 2006; Sokolov et al. 2016] provide a radically different approach, where a perfectly isometric parametrization is computed locally for every vertex/triangle of a surface. Local inconsistencies between neighboring parametrizations, which are unavoidable since exact isometry is enforced, lead to the introduction of non-quad elements or T-junctions, producing hybrid meshes composed of a majority of isotropic quadrilateral elements. These meshes are ideal for our purposes, since they contain minimal distortions and they produce approximately uniform face sizes.

Our remeshing algorithm is heavily based on the Robust Instant Meshing (RIM) quad-dominant meshing pipeline of Gao et al. [2017]. In RIM, the orientation field is encoded as a unit vector attached to every vertex, which is unique up to an integer rotation. The position field encodes a local isometric parametrization whose gradient is aligned with the orientation field, i.e. it encodes a regular grid in the tangent space. It is called position field, since the only available degree of freedom is the origin of the grid (up to an integer translation), which is represented as a 3D point. The position field can be visualized as a new set of 3D coordinates for the vertices of the input triangle mesh, that are mapping each vertex to the position of the closest vertex of the output quadrilateral mesh. RIM extracts the final quad mesh by collapsing the edges of the input mesh, using the position field to identify which edges should be preserved as final edges of the quad-dominant mesh, which edges are diagonals, and which edges should be collapsed.

3 OVERVIEW

Our pipeline begins with an input model. Unlike the stitch mesh modeling framework, however, we do not rely on the topology of this input model. Instead we begin with remeshing the model to produce a quad-dominant mesh. Then, we perform a series of optimizations and topological operations to generate the knitting directions over the mesh. Finally, we perform a subdivision operation that produces

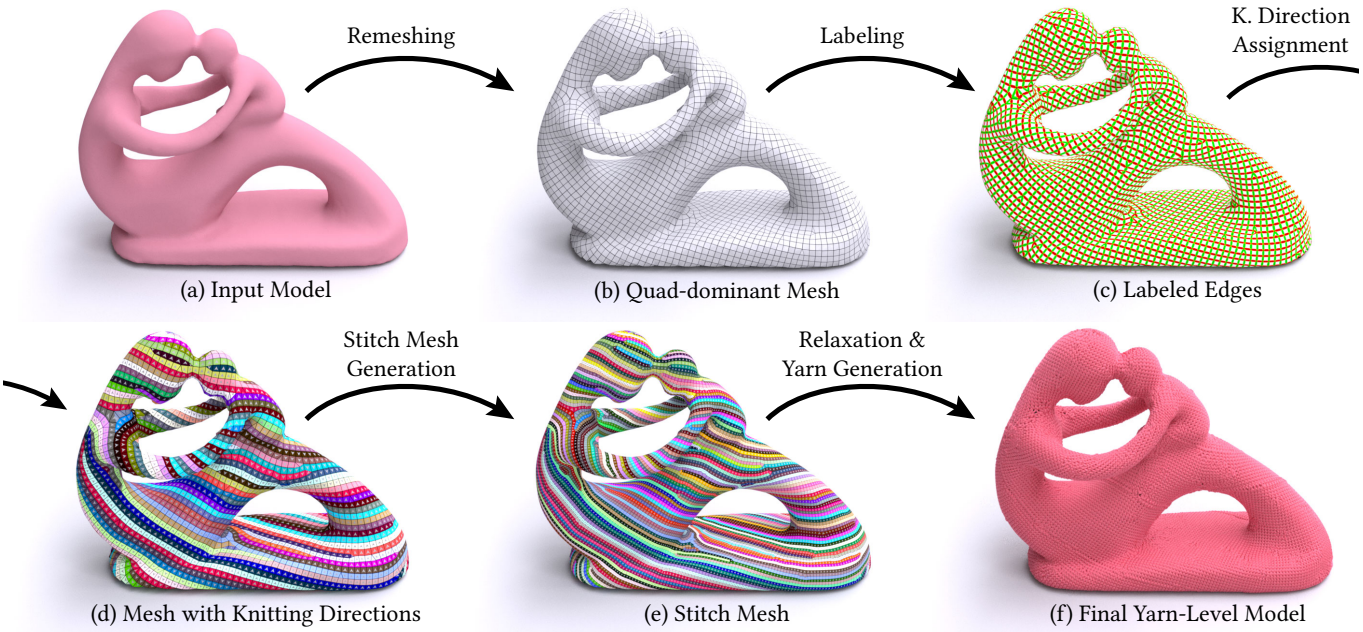


Fig. 3. ***The overview of our pipeline:*** *(a) an arbitrary input 3D model is converted into (b) an isotropic quad-dominant mesh with only quads and triangles via remeshing. Then, (c) the edges of the mesh are labeled, and (d) knitting directions over the surface are determined (arrows showing the wale knitting direction on each face). Finally, (e) a stitch mesh is generated and (f) the final yarn-level model is produced from the stitch mesh via relaxation and yarn generation operations.*

a valid stitch mesh, and the final yarn-level model can be easily created from this stitch mesh. Fig. 3 demonstrates the individual steps of our pipeline that are listed below:

- **Remeshing:** Starting with a given input model, We generate an isotropic quad-dominant mesh that only contains triangles and quads (Section 4).
- **Labeling:** We formulate a Mixed-Integer Programming (MIP) problem and perform custom topological operations to label each edge as a wale or course edge (Section 5).
- **Knitting Direction Assignment:** We determine the knitting (wale) direction based on the edge labels by solving another optimization problem (Section 6).
- **Stitch Mesh Generation:** The stitch mesh is formed via a subdivision operation that considers the knitting directions and edge labels (Section 7).
- **Relaxation and Yarn Generation:** We perform mesh-based relaxation and then generate the yarn-curves from the stitch mesh (Section 8). The final yarn-level model is produced via yarn-level relaxation.

This process allows us to produce a yarn-level knit model starting with an arbitrary 3D shape. No user interaction is required at any step. We describe each one of these steps in detail in the following sections.

4 REMESHING

The requirements for producing valid stitch meshes are different from traditional FEM applications. Stitch meshes are quad-dominant meshes that must satisfy two requirements:

(1) **Topology Requirement**: It must be possible to separate the faces into groups of rows, such that the knitting directions are aligned along edges (Fig. 2c),
(2) **Geometry Requirement**: All faces must have approximately the same size.

The first requirement ensures the existence of a valid set of knitting patterns for the faces, while the second models the physical constraint that the stitch size is constant and thus the stitch mesh faces need to have a homogeneous size.

Stitch meshes contain two primary directions (course and wale) that are roughly perpendicular to each other over the entire mesh. Therefore, a 2-RoSy field, which produces a 2-colorable mesh [Lei et al. 2017], provides a suitable topological construction for generating stitch meshes, where one of the primary directions is later aligned with the field. A more typical 4-RoSy field, on the other hand, leads to directional misalignments that require additional topological operations to resolve them. However, 2-RoSy fields are rarely used in other applications, since their singularities induce large geometric distortions: a low-order 2-RoSy field singularity can be approximated by 2 quads, which necessarily have flat angles, and introduce large distortions in the neighboring regions. Fortunately, this is not a problem for stitch meshes, since stitches near singularities naturally deform, making modern field-guided, quad-dominant pipelines ideal for our purpose.

We extend the RIM [Gao et al. 2017] quad-dominant meshing pipeline to produce meshes that satisfy (in a soft sense) the requirements of stitch meshes. Our method for orientation and position field generation is identical to RIM, with the exception of using a

2-RoSy symmetry instead of a 4-RoSy symmetry, which is a trivial modification. RIM automatically adds T-junctions and triangles to ensure a uniform mesh element size, which satisfies our geometry requirement. The mesh extraction part is modified to restrict the non-quad elements to be either pentagons (using T-junctions) or triangles. We implemented this as a postprocessing step, which is applied after the extraction procedure of RIM, using:

(1) For each triangle, we pick the edge whose opposite angle is closer to 90 degrees (excluding the edges corresponding to sharp features, identified by a negative dot product between the normal of the two incident faces) and mark it as a diagonal, encouraging the extraction algorithm to merge it with the neighboring elements, if possible.
(2) We split each polygon with more than 5 sides by adding the edge that is most aligned with the orientation field. The splitting is done recursively until all subpolygons have less than 5 sides.

These operations are interleaved with the extraction algorithm in RIM, until no changes to the final mesh are made in one iteration. To complete the pipeline, each pentagon (T-junction) is split into 3 triangles, connecting the T-junction with the two vertices on the opposite side. As a result, at the end of our remeshing step we get a quad-dominant mesh that contains a relatively small number of triangles.

5 LABELING

Labeling helps us determine the knitting directions over the mesh surface. Similar to stitch mesh modeling, our goal is to label each edge as a wale edge or a course edge.

Our labeling process begins with representing each edge as two *half-edges*, each belonging to one of the two faces sharing the edge. Obviously, border edges that are used by a single face would only have a single half-edge. We label each half-edge, following certain rules that will allow us to define valid knitting directions over the surface (Section 5.1). This process involves solving an optimization problem that would minimize the number of edges with conflicting half-edge labels. Thus, we find a valid half-edge labeling that maximizes the number of edges with consistent half-edge labels. Then, we assign the edge labels by resolving the half-edge labeling conflicts using simple topological modifications (Section 5.2). Finally, we perform post-processing operations to ensure that we have desirable final edge labels and mesh topology (Section 5.3).

5.1 Labeling Half-Edges

Our half-edge labeling must follow certain rules, so that the resulting labels define valid knitting directions over the surface. Thus, we can only permit a limited number of configurations for labeling.

Each quad face must have two wale edges and two course edges. Also, wale edges must be separated by course edges. Therefore, the only acceptable combination of labeling for quad faces is the one shown in Fig. 4a.

Our final stitch meshes do not contain triangles, but we do have triangles at this intermediate step. When labeling triangles, we cannot permit all edges of a triangle to be labeled as course edges, because this would prevent building stitches within the triangle,

(a) (b) (c)

Fig. 4. ***Valid half-edge configurations*** *for quad and triangle faces. Course half-edges are colored as red and wale half-edges are colored as green.*

effectively turning the triangle into a hole. Similarly, we cannot permit labeling all edges of a triangle as wale edges either, since this would also prevent building stitches within the triangle. Therefore, the only two labeling alternatives we can permit for triangles include either one wale edge or one course edge, as shown in Fig. 4b-c.

Following these labeling rules for quads and triangles shown in Fig. 4 as hard constraints, we label each half-edge in a way that would minimize the number of edges with inconsistent half-edge labels. We achieve this by representing the half-edge labeling problem as a mixed integer programming problem.

Let $\ell_0^{e_i}$ and $\ell_1^{e_i}$ represent the labels of the two half-edges for the edge e_i with index i. We assign them integer values 0 or 1 to indicate labels wale or course, respectively. These labels can also be accessed using face indices, such that $\ell_0^{f_j}$, $\ell_1^{f_j}$, $\ell_2^{f_j}$, and $\ell_3^{f_j}$ are the four half-edge labels of a quad face f_j with index j. Thus, if e_i is the first edge of f_j and f_j is the first face of e_i, we can write $\ell_0^{e_i} = \ell_0^{f_j}$. Using this notation, our optimization problem can be written as

$$\textbf{minimize} \quad \sum_{i=0}^{n-1} (\ell_0^{e_i} - \ell_1^{e_i})^2$$

$$\textbf{subject to}$$

$$\text{for each quad face } f_j, \quad \ell_0^{f_j} = \ell_2^{f_j}, \quad \ell_1^{f_j} = \ell_3^{f_j}, \quad \ell_0^{f_j} \neq \ell_1^{f_j}$$

$$\ell_k^{fj} \in \{0,1\}, \quad k = 0,1,2,3$$

$$\text{and for each triangle face } f_j, \quad 1 \leq \ell_0^{f_j} + \ell_1^{f_j} + \ell_2^{f_j} \leq 2$$

$$\ell_k^{f_j} \in \{0,1\}, \quad k = 0,1,2$$

where n is the number of non-border edges. Note that since $\ell_0^{f_j}$ and $\ell_1^{f_j}$ can only be 0 or 1, $\ell_0^{f_j} \neq \ell_1^{f_j}$ is modeled as $\ell_0^{f_j} + \ell_1^{f_j} = 1$. The constraints ensure that quad and triangle faces use one of the valid half-edge configurations. We solve this optimization problem using branch-and-bound that returns a solution with the minimum number of edges that contain conflicting half-edge labels.

In most cases the resulting labeling would contain edges with inconsistent half-edge labels, such that $\ell_0^{e_i} \neq \ell_1^{e_i}$ for some edges e_i. In fact, around certain types of singularities, we are guaranteed to have inconsistent half-edge labels. In particular, vertices with odd valance that are surrounded by quad faces, such as the example in Fig. 5, would have at least one edge with inconsistent half-edge labels. Therefore, before we solve the optimization problem, we triangulate the faces surrounding singularities containing vertices with odd valence. This provides additional flexibility in assigning half-edge labels around such singularities and makes it possible to

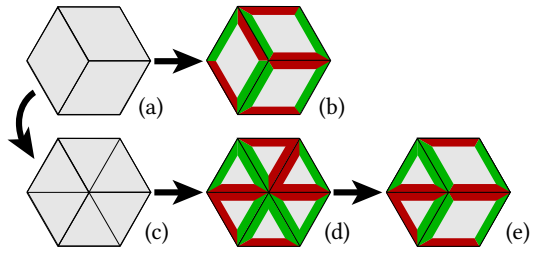


Fig. 5. ***Triangulation near singularities:*** *(a) singularities containing vertices with odd valance lead to (b) inconsistently labeled half-edges; therefore, (c) we first triangulate the quads near such singularities to provide more flexibility during half-edge labeling, and then (d) such triangles can be merged at the end of the labeling process.*

label the half-edges around them consistently. At the end of the labeling process, we can recover some of these triangulated quads via our post-processing operations (Section 5.3).

5.2 Labeling Edges

After we label the half-edges, we can label all edges with consistent half-edge labels. Edges with inconsistent half-edge label, however, require topological modifications to the mesh. There are three alternatives that are handled differently: an edge with inconsistent half-edge labels might be between two quads, a quad and a triangle, or two triangles.

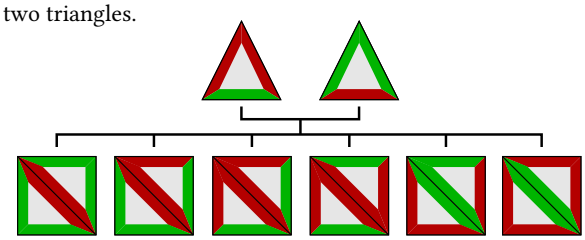


Fig. 6. ***Triangulation of quad faces:*** *(top) the two valid configurations for labeling half-edges of triangles can be used for representing (bottom) all possible configurations for labeling half-edges of quads.*

If an edge with inconsistent half-edge labels is between two quads, we label the edge as a course edge, then split the quad with the wale half-edge label into two triangles. The alternative of labeling the edge as a wale edge and splitting the other quad is also an acceptable solution, but this would split the row on one side of the edge (since neighboring quad faces sharing wale edges form rows), so we prefer the other alternative. A quad can be split into two triangles in two different ways along either one of its diagonals. They produce similar results, so we randomly pick one diagonal. Once we split a quad into two triangles, any possible half-edge labeling configuration can be represented by combinations of the two triangle labeling configurations we permit, as shown in Fig. 6. Therefore, while assigning the half-edge labels for these two new triangles, we make sure that they do not contain other edges with inconsistent half-edge labels. Thus, we simply use the half-edge labels on the other sides of their edges. An example of this operation is shown in Fig. 7, where one of the quad faces sharing an edge

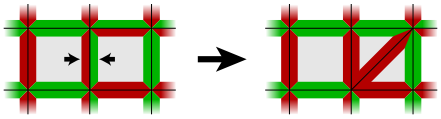


Fig. 7. ***Splitting quad faces:*** *(left) if an edge with inconsistent half-edge labels is between two quad faces, (right) the face with the wale half-edge label is split into two triangles.*

with inconsistent half-edge labels is split into two triangles and the half-edge labels of the new triangles are assigned such that the triangles do not contain edges with inconsistent labels.

If an edge with inconsistent half-edge labels is between a quad and a triangles, we split the quad face. Again, we can use either one of the diagonals for splitting the quad face. Similarly, we make sure that the two new triangles do not contain other edges with inconsistent half-edge labels.

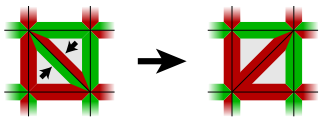


Fig. 8. ***Rotating edges between triangle pairs:*** *(left) if an edge with inconsistent half-edge labels is between two triangles, (right) the edge is rotated.*

If an edge with inconsistent half-edge labels is between two triangles, we rotate the edge, as shown in Fig. 8, and we label the rotated edge as a course edge. Note that labeling the rotated edge as a course edge is guaranteed to form two valid triangle configurations on either side of the edge. This is because the shared edge between two triangles can have inconsistent half-edge labels only when the other half-edges of one triangle are labeled as course and other half-edges of the other triangle are labeled as wale. Otherwise, the optimization process for assigning the half-edge labels would have resolved the inconsistency in half-edge labeling.

Note that none of these topological operations lead to new inconsistencies in half-edge labeling. Therefore, all edges can be labeled in a single pass without the need for multiple iterations.

5.3 Post-Processing

Our post-processing operations involve pairs of neighboring triangles. If the edge labels of a pair of neighboring triangles are such that merging them into a quad by removing the common edge between them would lead to a quad with an acceptable labeling configuration (as in Fig. 4a), we merge the two triangles into a quad. An example of this operation is shown in Fig. 9. This operation reduces the number

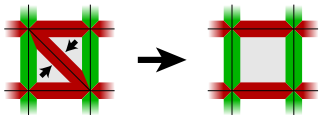


Fig. 9. ***Merging triangles:*** *(left) if removing an edge between two triangles would lead to a quad with valid labeling, (right) we merge the two triangles.*

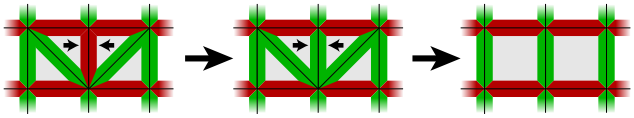


Fig. 10. ***Merging triangles after flipping the label of a course edge:*** *(left) two pairs of triangles separated by a course edge are labeled such that (middle) flipping the label of the course edge allows (right) merging the triangles into quad faces.*

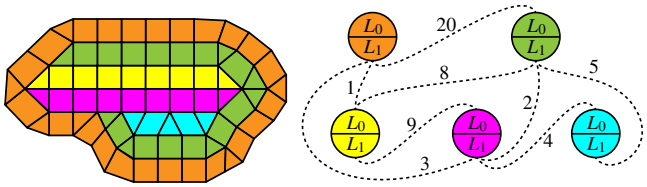


Fig. 11. ***An example meta-graph:*** *(left) mesh with separate rows colored differently, and (right) its meta-graph.*

of triangles and unnecessary complexity in the final knit structure. Such pairs of triangles commonly appear around singularities containing a vertex with an odd valence, since we triangulate the quad faces around such singularities before labeling the half-edges. Thus, this operation can recover some of the quad faces around those singularities. Yet, such pairs of triangle can appear on other parts of the model as well. In particular, the triangulation process used for labeling edges can also produce such triangle pairs, which are converted to quads in this step.

In some cases flipping the label of a course edge between two triangles can allow merging these triangles with other neighboring triangles, as shown in Fig. 10. Therefore, we scan course edges between pairs of triangles with at least one of them connected to another triangle and check if flipping the edge label would allow merging the nearby triangle into quads. If so, we flip the edge label and merge the triangles.

Finally, we consider pairs of neighboring triangles sharing an edge labeled as a wale edge. If both triangles of such an edge have other edges labeled as wale edges and that merging them would not lead to a quad face with a valid configuration, we flip the label of the shaded edge to a course edge. The reason for this operation becomes more clear after discussing the subdivision operation that generates the final stitch mesh (Section 7). This is because if the common edge label for this particular pair of triangles is kept as a wale edge, the resulting stitch mesh would contain triangular stitch-mesh faces that cannot always be safely eliminated, which may result in unstable stitches that would unravel during yarn-level simulation.

6 KNITTING DIRECTION ASSIGNMENT

After labeling the edges, we must determine the knitting directions over the model surface. On each face the course and wale knitting directions are aligned with the course and wale edges, respectively. We can arbitrarily pick either one of the two possible course directions (i.e. left-to-right or right-to-left), since a stitch can be formed using either direction. The choice for the wale directions, however, is not arbitrary, since it determines which course edges of a face are the bottom course edges and which ones are the top course edges.

We would like the wale direction to be uniform over the entire model. This means that if an edge is treated as a bottom course edge for one face, the other face sharing the edge should treat it as a top course edge. This aligns the wale knitting directions for the two faces. However, we cannot enforce this as a hard constraint, because some shapes would require having mismatched wale directions in certain places, depending on how the knit structure form the surface. Therefore, we perform another optimization that provides a solution with the minimum number of course edges that are along mismatched wale directions.

Note that in our labeling each quad face is assigned exactly two wale edge and each triangle can have one or two wale edges. Therefore, a group of edges connected with wale edges form a string of faces that we call a *row*. Each row can either form a closed loop or it can begin and end with two triangle faces, each with a single wale edge. Each face belongs to a single row and neighboring rows are separated by course edges.

One hard constraint for this optimization is that the wale directions of two neighboring faces sharing a wale edge must be aligned. Otherwise, the resulting wale directions would not form a valid stitch mesh. This means that the wale direction along each row must be consistent. Therefore, instead of formulating the optimization problem for determining the wale directions per face, we can reduce the dimensionality of the problem by formulating it per row of faces. We achieve this by building a meta-graph of the mesh, such that each row of the mesh corresponds to a node of the meta-graph. An example meta-graph generated from a mesh is shown in Fig. 11. Two nodes of the meta-graph are connected to each other via undirected weighted edges, if the rows that correspond to these nodes have common course edges. The number of common course edges determine the weight of the edge. Each node of the meta-graph contains two halves: one half corresponds to the group of course edges on one side of the row and the other half corresponds to the group of course edges on the other side. Thus, the edges between nodes connect one half of a node to one half of another node.

We formulate a similar mixed integer programming problem on the meta-graph. The two halves of each meta-graph node are labeled as either *top* or *bottom*, indicating that the course edges corresponding to those halves are either top course edges or bottom course edges. Let $L_0^{M_r}$ and $L_1^{M_r}$ represent the labels of the two halves of a meta-graph node M_r with index r. We assign them integer values 0 and 1 to indicate top or bottom labels, respectively. The same indices can also be accessed using the edges of the meta-graph, such that $L_0^{E_s}$ and $L_1^{E_s}$ are the labels of the two meta-graph node halves that are connected by the meta-graph edge E_s with index s. Using this notation, we can write the optimization problem that minimizes the number of course edges with mismatched wale directions as

$$\textbf{minimize} \quad \sum_{s=0}^{N-1} W_s \left(1 - (L_0^{E_s} - L_1^{E_s})^2\right)$$

$$\textbf{subject to} \quad \text{For meta-graph node } M_r,\ L_0^{M_r} + L_1^{M_r} = 1$$

$$L_k^{M_r} \in \{0, 1\}, \quad k = 0, 1,$$

where N is the number of meta-graph edges and W_s is the weight of the edge E_s (i.e. the number of course edges between the two rows). The constraint $L_0^{M_r} + L_1^{M_r} = 1$ ensures that the two halves of the node M_r are assigned different labels. We solve this problem using branch-and-bound. Since the meta-graph contains a relatively small number of nodes (as compared to the number of faces), this optimization can be solved efficiently.

7 STITCH MESH GENERATION

The resulting mesh after assigning the knitting directions can be directly used as a stitch mesh. However, it contains triangle faces, which are undesirable. In particular, each triangle face with a single wale edge, marking the beginning or ending of a row, would lead to a knot in the yarn-level model. To avoid this, we perform a subdivision operation, similar to Catmull-Clark subdivision [Yuksel et al. 2012], which converts each quad face into four quads and each triangle face into three quads.

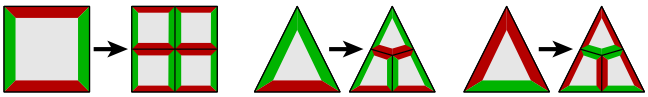


Fig. 12. ***Subdivision rules*** *for (a) quad faces, (b) triangle faces with two wale edges, and (c) triangle faces with two course edges.*

There are three cases to consider for labeling the new edges generated by the subdivision operation, as shown in Fig. 12. Quad faces form four regular quads. Triangle faces, however, form two regular quads and one special quad with a different labeling configuration, where wale edges are not separated by course edges. We handle these quad faces with different labeling configuration differently.

Triangles with two course edges form a special quad with one bottom course edge and one top course edge. Such quads mark the beginnings and endings of stitch mesh rows. Therefore, they are handled differently than other stitch mesh faces when generating the yarn curves, as explained in Section 8.

Triangles with two wale edges, however, form a special quad with either two bottom course edges or two top course edges. Such quads do not correspond to a valid stitch; therefore, we eliminate them. We begin with triangulating these quads by splitting them with a diagonal wale edge that forms two triangles, each with a single course edge. Finally, we merge these two triangles with the quad faces on either side, forming pentagons that represent either increase or decrease type stitches. Note that our post-processing after labeling edges (Section 5.3) ensures that there is always a quad face next to these triangles, since we do not permit having two triangles with two wale edges side-by-side, sharing a wale edge.

Fig. 13 shows an example row that is subdivided into a stitch mesh. Special quad faces appear on either ends of the row as well as the top center of the row, which are handled differently. Note that after the subdivision operation, all rows of the resulting stitch mesh form closed loops with no end points.

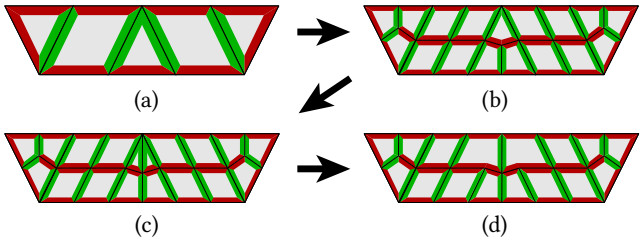


Fig. 13. ***Stitch mesh generation:*** *(a) the faces on each row are (b) subdivided into quad faces; (c) the face at the center with two bottom course edges is triangulated; and finally (d) the triangles are merged with the neighboring quad faces.*

8 RELAXATION AND YARN GENERATION

Before we generate the yarn curves from the stitch mesh, we perform mesh-based relaxation [Yuksel et al. 2012] . While the initial remeshing step provides a good starting point that results in faces with approximately the same size over the entire model, due to the topological operations we perform during labeling and stitch mesh generation, an optional mesh-based relaxation step can provide some minor improvement in unifying the edge lengths and minimizing the deformation of quad faces. Fig. 14 demonstrates zoom in view of "fertility" model before and after mesh-based relaxation.

Fig. 14. ***Mesh-based relaxation:*** *(a) stitch mesh before mesh-based relaxation, and (b) stitch mesh after mesh-based relaxation.*

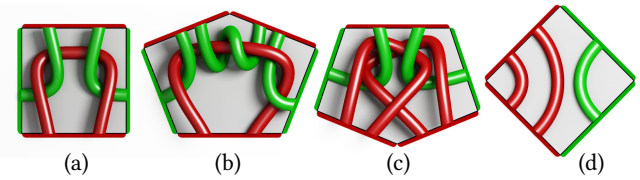


Fig. 15. ***Stitch types:*** *(a) regular quads, (b) increases, (c) decreases, and (d) special quads.*

The stitch mesh models we generate have four different types of faces: (1) regular quad faces, (2) pentagon faces representing increases, (3) pentagon faces representing decreases, and (4) special quad faces marking row ends. For regular quad faces, we generate yarn curves of a knit stitch (k), shown in Fig. 15a. Pentagon faces representing increases use a knit followed by a purl that are pulled through the same loop (kp), as shown in Fig. 15b. Pentagon faces representing decreases use a knit stitch that is pulled through two loops ($d_{12}k$), as shown in Fig. 15c. Finally, special quads that mark

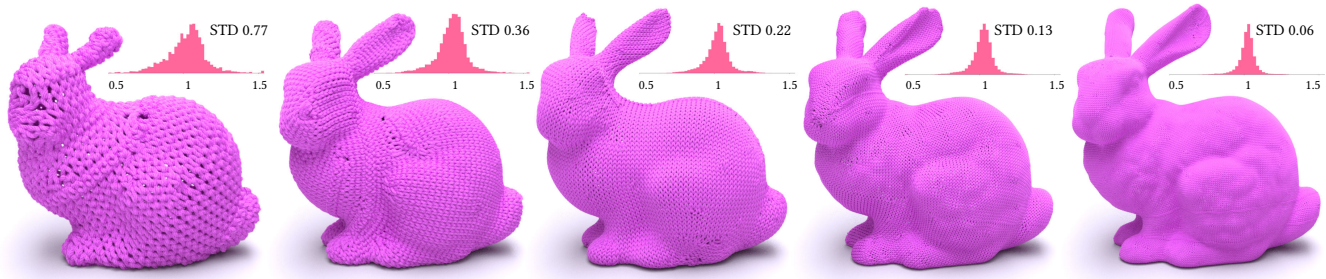


Fig. 16. *Yarn-level knit structures generated from the "bunny" model with three different resolutions: 1.3K, 4K, 7K, 16K, and 48K stitches.*

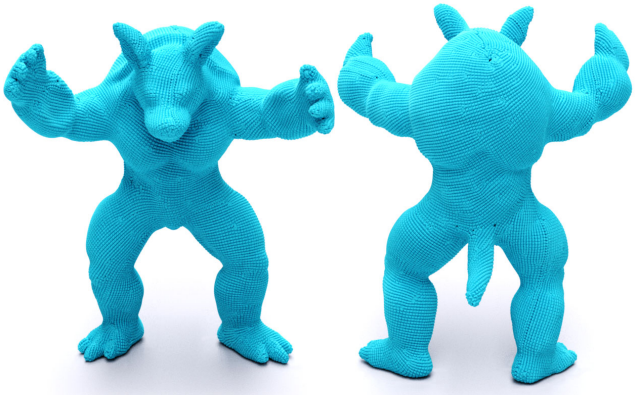


Fig. 17. *Yarn-level knit structure for the Armadillo model with high-curvature areas around the ears, the tail, the fingers, and the toes.*

the end of the rows simply connect the yarn of the course edges together and the wale edges together, as in Fig. 15d. Note that the stitch mesh representation allows replacing these stitches with other stitch types, if desired. However, this would require manual stitch mesh editing, so we simply use the corresponding stitch type in Fig. 15 for all faces.

9 RESULTS

Fig. 1 shows complex 3D models that are automatically converted to yarn-level knit structures using our pipeline. Notice that the resulting yarn-level models have uniform stitch sizes over the model surfaces. Our pipeline supports high genus surfaces, as demonstrated in Fig. 3f. We can also handle models with high-curvature areas, such as the example in Fig. 17.

The surface details preserved in the final yarn-level model depends on the resolution of the generated stitch mesh. Fig. 16 shows the "bunny" model with five different resolutions. While all five results are valid stitch meshes with uniform stitch sizes, only the one with the highest resolution captures the small-scale details of the input surface. Notice that representing small-scale surface details also introduce additional singularities that are needed for shaping the knitted model. On the other hand, using low-resolution remeshing loses surface details during the remeshing step.

Yet, if the input model does not have small-scale features, a low-resolution model generated using our framework can produce acceptable results. An example low-resolution model with a reasonably complex shape is shown Fig. 18.

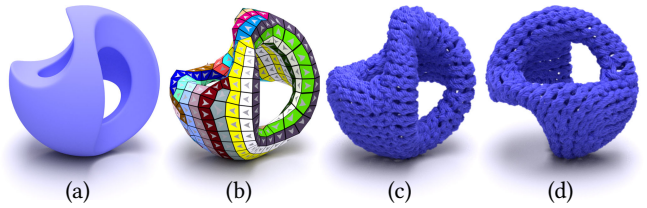


Fig. 18. *Low-resolution stitch mesh and the yarn-level model generated using the "sculpt" model: (a) the input shape, (b) the stitch mesh, and (c-d) two views of yarn-level knit model.*

Even CAD models with sharp features can be processed by out pipeline (Fig. 19): the sharp features of the input model are partially smoothed due to the relatively low resolution of the yarn-level model, but the overall shape is preserved. Notice that when using an intrinsic orientation field (Fig. 19a), the knitting directions are not aligned with the surface details [Huang and Ju 2016; Jakob et al. 2015]. Using an extrinsic orientation field instead (Fig. 19b) makes the knitting directions of the final model follow the surface details better, but it also introduces additional singularities to align the orientation field with the model features. Therefore, the yarn-level models in Figures 1, 3, 17, and 16 are generated using an intrinsic orientation field.

Our pipeline can also be used with custom orientation fields, to provide additional control over the final knitting directions. Fig. 20 shows the stitch meshes and the final yarn-level model generated from the same input shape using both the intrinsic orientation field and a custom orientation field, generated with a small set of user-defined strokes. The stroke compete with the field smoothness,

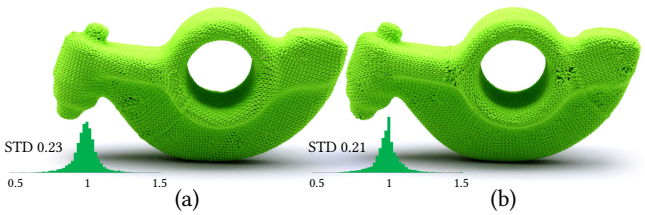
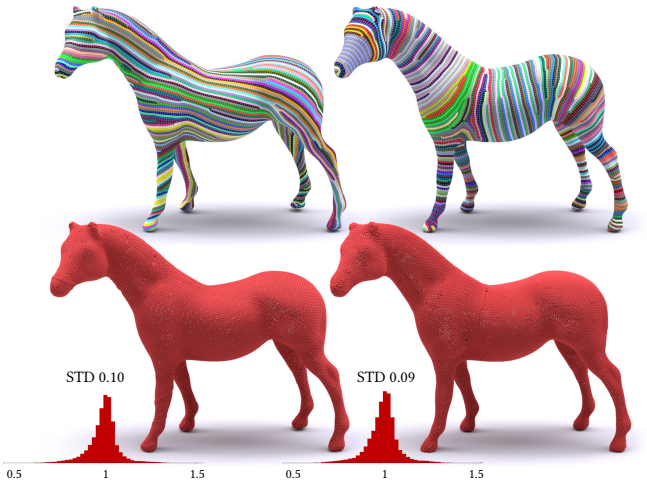


Fig. 19. *Yarn-level "rocker arm" model generated using (a) an intrinsic orientation field and (b) an extrinsic orientation field.*



(a) intrinsic orientation field (b) custom orientation field

Fig. 20. *Stitch meshes and yarn-level knit models generated using (a) the default orientation field, and (b) user-defined orientation field with orientation constraints interactively drawn on the model surface.*

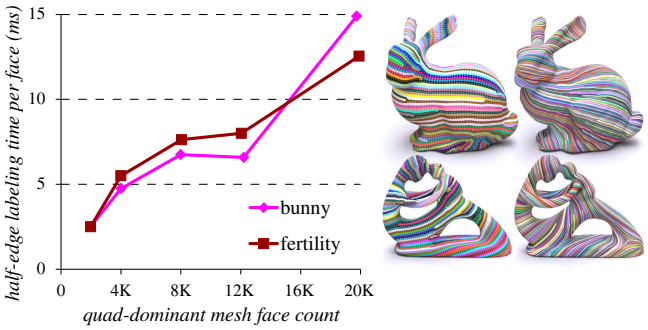


Fig. 21. *Half-edge labeling time per face for different quad-dominant mesh resolutions. The final stitch meshes for the lowest and highest resolution examples in the graph are shown on the right.*

leading to a small increase in the number of irregularities in the knitting pattern.

Edge Lengths. We present the distribution of the stitch mesh edge lengths after mesh-based relaxation as histograms in Fig. 16, 19, and 20. The standard deviation (STD) of edge lengths mainly depends on the mesh resolution after remeshing, rather than the type of orientation orientation filed used. The variations on edge lengths are mostly introduced due to singularities and stitch size variations around singularities. Note that such variations on stitch sizes near singularities also appear in real-world knitting.

Performance. The performance results of our pipeline for generating various yarn-level models that are presented in this paper are shown in Table 1. Notice that most of the steps in our pipeline can be computed within several seconds to a few minutes, depending on the size and complexity of the input model and the resolution of the output model. However, after we generate the yarn curves, the yarn-level relaxation step that produces the final yarn curve shapes can take hours. Aside from the relaxation operations, the most expensive component of our pipeline is the optimization we use for labeling the half-edges. Fig. 21 shows half-edge labeling time per face for different quad-dominant mesh resolutions, indicating that computation time per face increases with mesh resolution and it depends on the topological complexity of the mesh.

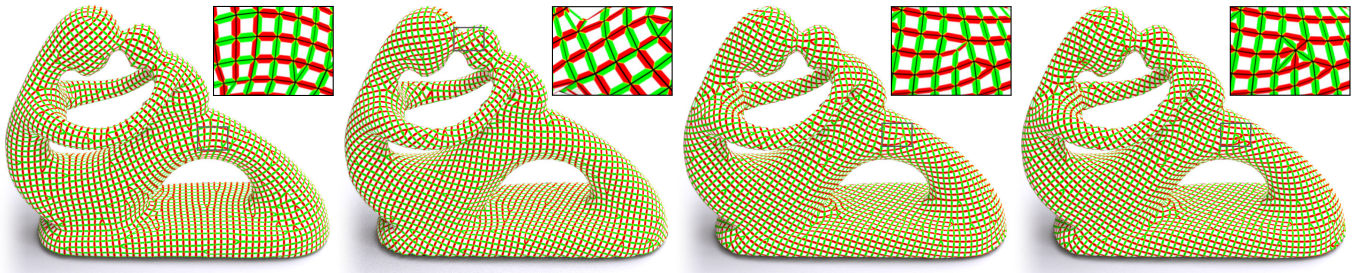
The two-step optimization for labeling and direction assignment (Sections 5 and 6) is the key to the efficiency of our algorithm. We experimented with an integrated optimization tackling jointly both problems, and observed that the larger solution space of the combined optimization dramatically increases the computational requirements. On the 16K "bunny" model (Fig. 16), the combined optimization finished the 16GB of available memory after 40 minutes of computation and started thrashing. In comparison, our two-step

Table 1. ***The computation performance*** *measurements for the steps of our pipeline.*

		# Input Faces	# Mesh Faces	# Stitch Faces	Remesh	Labeling	K. Direction Assignment	Stitch Mesh Gen.	Mesh-based Relaxation	Yarn Gen.	Yarn-level Relaxation
Rocker Arm	(Fig.19a)	62K	2,018	7,880	2 s	8 s	99 ms	593 ms	12 s	18 ms	2 hr
Rocker Arm	(Fig.19b)	62K	2,037	7,790	2 s	4 s	127 ms	583 ms	9 s	22 ms	2 hr
Chinese Lion	(Fig.1)	100K	3,495	13,606	4 s	19 s	198 ms	1,049 ms	18 s	39 ms	2 hr*
Kitten	(Fig.1)	100K	3,690	14,460	4 s	16 s	124 ms	1,083 ms	16 s	37 ms	3 hr
Dragon	(Fig.1)	104K	4,218	16,458	4 s	26 s	370 ms	1,234 ms	53 s	35 ms	4 hr*
Horse	(Fig.20a)	134K	4,640	18,172	6 s	17 s	159 ms	1,297 ms	25 s	55 ms	2 hr
Horse	(Fig.20b)	134K	4,655	18,160	6 s	18 s	306 ms	1,311 ms	45 s	52 ms	2 hr
Elephant	(Fig.1)	299K	4,791	18,686	13 s	26 s	237 ms	1,421 ms	28 s	51 ms	2 hr
Fertility	(Fig.3)	167K	4,979	19,490	8 s	32 s	192 ms	1,495 ms	46 s	54 ms	1 hr*
Armadillo	(Fig.17)	280K	6,591	25,734	13 s	58 s	567 ms	1,963 ms	88 s	77 ms	2 hr*
Bunny (1.3K)	(Fig.16)	111K	353	1392	4 s	2 s	45 ms	119 ms	6 s	4 ms	<1 hr
Bunny (4K)	(Fig.16)	111K	1059	4124	4 s	2 s	66 ms	315 ms	8 s	12 ms	<1 hr
Bunny (7K)	(Fig.16)	111K	1,821	7,090	4 s	2 s	131 ms	550 ms	10 s	19 ms	1 hr*
Bunny (16K)	(Fig.16)	111K	4,003	15,704	5 s	16 s	147 ms	1101 ms	12 s	45 ms	2 hr
Bunny (48K)	(Fig.16)	406K	12,195	48,096	37 s	84 s	399 ms	3526 ms	130 s	159 ms	3 hr*

The computation times are generated using a computer with Intel Core i7 6700HQ CPU @ 2.60 GHz with 16 GB RAM.
** Yarn-level relaxation timings are generated using a computer with Intel Core i7 3930K CPU @ 3.20 GHz with 32 GB RAM.*

Fig. 22. *Stitch meshes generated by our fully automatic pipeline using an extrinsic orientation field.*



(a) [Gao et al. 2017] 115 inconsistencies | (b) 4-rosy 130 inconsistencies | (c) 2-rosy 52 inconsistencies | (d) 2-rosy with triangulation 22 inconsistencies

Fig. 23. ***Comparison of different methods for orientation field generation:*** *and the number of inconsistent edge labels they produce after half-edge labeling, showing (a) [Gao et al. 2017], (b) our method with a 4-RoSy field, (c) our method with a 2-RoSy field, and (d) our method with a 2-RoSy and triangulated singularities. Note that different methods create inconsistencies on the different parts of the model surface as highlighted, but the 2-RoSy field leads to fewer inconsistencies, especially when combined with triangulated singularities.*

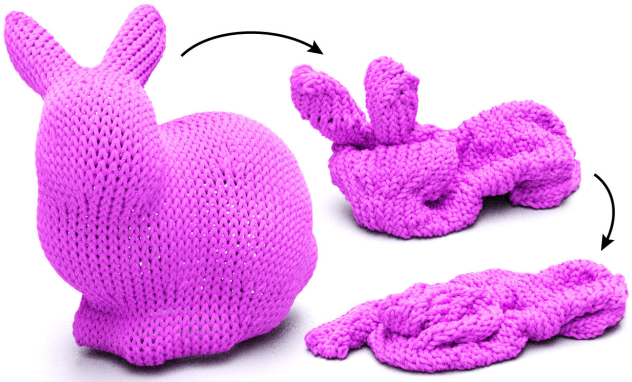


Fig. 24. *Example frames from a yarn-level simulation of a bunny model deforming under gravity.*

Fig. 25. *An octopus wearing a knitted sweater: (left) simulated model and (right) fabricated via 3D printing.*

solution takes 24 seconds for half-edge labeling and less than a second for knitting direction assignment. While it is possible that the combined optimization could produce results with fewer mismatched knitting directions, our two-step optimization provides superior computational performance and lower memory usage.

Robustness. We demonstrate the robustness of our pipeline by automatically processing a collection of 104 models (Fig. 22). The set includes models with high genus, sharp features, and thin parts, and our algorithm generated a valid stitch mesh model with roughly uniform face sizes for all of them models.

Remeshing and Labeling. We compare different methods for generating quad-dominant meshes in Fig. 23, measuring the quality using the number of inconsistencies produced after labeling half-edges, which correspond to irregularities in the stitch-mesh. Directly using RIM [Gao et al. 2017] or our modified method with a 4-RoSy field, we get a large number of inconsistent edge labels. Switching to a 2-RoSy field greatly improves the quality, but still struggles due to the topology of the mesh near some singularities. Triangulating the neighborhood of singularities before labeling the half-edges, enlarges the solution space and allows our optimization to substantially reduce the number inconsistent edge labels. We used the parallel Gurobi solver [Gurobi Optimization 2016] (with 8 threads) to solve the MIP problems for labeling and direction alignment.

Simulation. We produce valid stitch meshes and, therefore, yarn-level models with topologically correct knitted structures. All our models can be directly used for yarn-level simulation. Fig. 24 shows example frames from an animation of a bunny model deforming under gravity computed using a yarn-level simulation. Notice that all stitches remain intact during the simulation.

Fabrication via 3D printing. We show a 3D printed yarn models in Fig. 25: the sweater is made of black nylon, and it has been printed in nylon using Fused Deposition Modeling and a water-soluble supporting material (Polyvinyl Alcohol). A clip documenting the fabrication procedure using the "Ultimaker 3" [Ultimaker 2018] printer is attached in the additional material. Since nylon is a stiff material, the sweater is only mildly flexible and it does not collapse under its weight. This fabrication method is affordable, enabling the production of interesting decoration and lightweight physical realizations of 3D shapes using commodity 3D printers.

We also prototyped a design pipeline for tailored gloves, combining this fabrication method with a 3D scanning pipeline (David 3D Scanner [DAVID 2018]), shown in Fig. 26. First, the user's hand is scanned; then, a desired part is manually selected and enlarged; finally, the model is automatically transformed into a yarn model

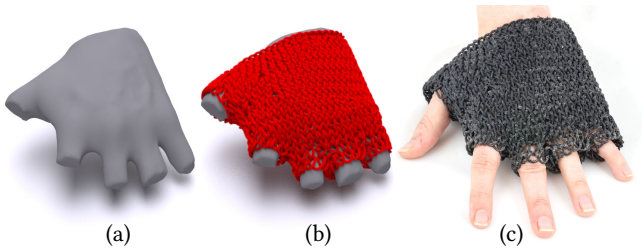


Fig. 26. *Our method used for designing a custom fitted glove: (a) the hand model acquired using a structured light 3D scanner, (b) the simulated model, and (c) 3D printed glove.*

by our method and printed, leading to wearable nylon glove. The comfort and flexibility of the final model depends on the material used for printing.

Finally, we present a knit bunny model printed using selective laser sintering (Fig. 27). The fabricated model is flexible and robust, as shown in the supplementary video.

10 CONCLUSION AND FUTURE DIRECTIONS

We have introduced a fully automatic method for converting arbitrary 3D shapes into knit structures, starting with quad-dominant mesh generation, followed by a two-step optimization process and topological operations that generate a valid stitch mesh. We have demonstrated the effectiveness of our approach with complex knit models generated using our pipeline and the robustness of our method by processing a large number of different 3D models. The yarn-level models we produce are guaranteed to have valid knit topologies and they are ready to be used with yarn-level simulations. To our knowledge, this is the first fully automatic method that can produce yarn-level knit model for arbitrary 3D shapes.

One important limitation of our approach is that fine-scale details of the input surface may not be properly represented in the final knit model, unless a high-enough resolution stitch mesh is generated. Since we rely on stitch meshes, we share the limitations of the stitch mesh representation. In particular, we cannot produce multi-layer knit structures that are used for colored knitting patterns.

Even though we generate valid yarn-level models, they are not ready to be fabricated via typical knitting operations. Extending our approach to automatically produce knittable structures would be an interesting direction for future work. In addition, Generating machine knitting instructions for fabricating such models using industrial knitting machines could greatly expand the potential applications of our approach [McCann et al. 2016], but this may require additional considerations to incorporate possible limitations of knitting machines into our optimizations.

Our current remeshing method can generate isotropic quad-dominant meshes by assuming each quad-shaped stitch face would ideally be square. However, in reality the ratio between the width and height of stitches can vary depending on the yarn type, the needle size, and the details of the knitting operations. It would be an interesting future direction to investigate variations of our pipeline that generate rectangular stitch mesh faces with a user-specified aspect ratio.

Fig. 27. *A knitted "bunny" model generated with our pipeline and printed using selective laser sintering.*

Though our method allows custom orientation fields to be used to provide additional control over the final knitting direction, automatically generating an optimal orientation field for minimizing singularities or providing a better representation of the input shape would be an interesting direction for future work. Our implementation does not support non-orientable surfaces, such as a Mobius strip, but our approach can be easily extended to handle them.

ACKNOWLEDGMENTS

We thank nTopology for fabricating the bunny model, Nghia Truong for his help with rendering the results, Jonathan M. Kaldor for the yarn-level simulation code, Manuel Vargas for the volume generation code. Mesh models are courtesy of the Aim@Shape Repository, the Stanford 3D Scanning Repository, and the dataset of [Myles et al. 2014]. All results are rendered using MITSUBA. This work was supported in part by NSF grant #1538593, NSF CAREER award IIS-1652515, and a gift from Adobe. This work was also supported in part by the NYU IT High Performance Computing resources, services, and staff expertise.

REFERENCES

Ergun Akleman, Jianer Chen, Qing Xing, and Jonathan L. Gross. 2009. Cyclic Plain-weaving on Polygonal Mesh Surfaces with Graph Rotation Systems. *ACM Trans. Graph.* 28, 3, Article 78 (July 2009), 8 pages.

Pierre Alliez, Mark Meyer, and Mathieu Desbrun. 2002. Interactive Geometry Remeshing. *ACM Trans. Graph.* 21, 3 (2002).

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, New York, NY, USA, 43–54.

Floraine Berthouzoz, Akash Garg, Danny M. Kaufman, Eitan Grinspun, and Maneesh Agrawala. 2013. Parsing Sewing Patterns into 3D Garments. *ACM Trans. Graph.* 32, 4, Article 85 (July 2013), 12 pages.

D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. 2013. Quad-Mesh Generation and Processing: A Survey. *Comp. Graphics Forum* 32 (2013).

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer Quadrangulation. *ACM Trans. Graph.* 28, 3 (2009).

Derek Bradley, Tiberiu Popa, Alla Sheffer, Wolfgang Heidrich, and Tamy Boubekeur. 2008. Markerless Garment Capture. *ACM Trans. Graph.* 27, 3, Article 99 (Aug. 2008).

David E. Breen, Donald H. House, and Michael J. Wozny. 1994. Predicting the Drape of Woven Cloth Using Interacting Particles. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. 365–372.

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. *ACM Trans. Graph.* 21, 3 (July

2002), 594–603.
Michel Carignan, Ying Yang, Nadia Magnenat Thalmann, and Daniel Thalmann. 1992. Dressing Animated Synthetic Actors with Complex Deformable Clothes. *SIGGRAPH Comput. Graph.* 26, 2 (July 1992), 99–104.
Xiaowu Chen, Bin Zhou, Feixiang Lu, Lin Wang, Lang Bi, and Ping Tan. 2015. Garment Modeling with a Depth Camera. *ACM Trans. Graph.* 34, 6, Article 203 (Oct. 2015).
Gabriel Cirio, Jorge Lopez-Moreno, David Miraut, and Miguel A. Otaduy. 2014. Yarn-level Simulation of Woven Cloth. *ACM Trans. Graph.* 33, 6, Article 207 (Nov. 2014), 207:1–207:11 pages.
Gabriel Cirio, Jorge Lopez-Moreno, and Miguel A. Otaduy. 2015. Efficient Simulation of Knitted Cloth Using Persistent Contacts. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '15)*. 55–61.
G. Cirio, J. Lopez-Moreno, and M. A. Otaduy. 2017. Yarn-Level Cloth Simulation with Sliding Persistent Contacts. *IEEE Transactions on Visualization and Computer Graphics* 23, 2 (Feb 2017), 1152–1162.
Keenan Crane, Mathieu Desbrun, and Peter Schröder. 2010. Trivial Connections on Discrete Surfaces. *Computer Graphics Forum* 29, 5 (2010).
R. Daněřek, E. Dibra, C. Öztireli, R. Ziegler, and M. Gross. 2017. DeepGarment : 3D Garment Shape Estimation from a Single Image. *Computer Graphics Forum* 36, 2 (2017), 269–280.
DAVID. 2018. DAVID 3D Scanner. http://www.david-3d.com/.
Philippe Decaudin, Dan Julius, Jamie Wither, Laurence Boissieux, Alla Sheffer, and Marie-Paule Cani. 2006. Virtual Garments: A Fully Geometric Approach for Clothing Design. *Computer Graphics Forum* (2006).
Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. 2014. Designing N-PolyVector Fields with Complex Polynomials. *Computer Graphics Forum* 33, 5 (2014).
Hans-Christian Ebke, Marcel Campen, David Bommes, and Leif Kobbelt. 2014. Level-of-detail Quad Meshing. *ACM Trans. Graph.* 33, 6, Article 184 (2014).
Xifeng Gao, Wenzel Jakob, Marco Tarini, and Daniele Panozzo. 2017. Robust Hex-dominant Mesh Generation Using Field-guided Polyhedral Agglomeration. *ACM Trans. Graph.* 36, 4, Article 114 (July 2017), 13 pages.
Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. 2007. Efficient Simulation of Inextensible Cloth. *ACM Trans. Graph.* 26, 3, Article 49 (July 2007).
Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. 2003. Discrete Shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '03)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 62–67.
Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. 2002. Geometry Images. *ACM Trans. Graph.* 21, 3 (2002).
Peng Guan, Loretta Reiss, David A. Hirshberg, Alexander Weiss, and Michael J. Black. 2012. DRAPE: DRessing Any PErson. *ACM Trans. Graph.* 31, 4, Article 35 (July 2012), 10 pages.
Inc. Gurobi Optimization. 2016. Gurobi Optimizer Reference Manual. http://www.gurobi.com
Aaron Hertzmann and Denis Zorin. 2000. Illustrating Smooth Surfaces. In *Proceedings of ACM SIGGRAPH*. ACM Press/Addison-Wesley Publishing Co., 517–526.
Zhiyang Huang and Tao Ju. 2016. Extrinsically Smooth Direction Fields. *Comput. Graph.* 58, C (Aug. 2016), 109–117.
Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008a. Knitting a 3D Model. *Computer Graphics Forum* 27, 7 (2008), 1737–1743.
Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008b. Knitty: 3D Modeling of Knitted Animals with a Production Assistant Interface. In *Eurographics 2008 - Short Papers*, Katerina Mania and Eric Reinhard (Eds.). The Eurographics Association.
Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-aligned Meshes. *ACM Trans. Graph.* 34, 6, Article 189 (Oct. 2015), 15 pages.
Chenfanfu Jiang, Theodore Gast, and Joseph Teran. 2017. Anisotropic Elastoplasticity for Cloth, Knit and Hair Frictional Contact. *ACM Trans. Graph.* 36, 4, Article 152 (July 2017), 14 pages.
Tengfei Jiang, Xianzhong Fang, Jin Huang, Hujun Bao, Yiying Tong, and Mathieu Desbrun. 2015. Frame Field Generation through Metric Customization. *ACM Trans. Graph.* 34, 4 (2015).
Felix Kälberer, Matthias Nieser, and Konrad Polthier. 2007. QuadCover – Surface Parameterization using Branched Coverings. *Computer Graphics Forum* 26, 3 (2007).
Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2008. Simulating Knitted Cloth at the Yarn Level. *ACM Trans. Graph.* 27, 3, Article 65 (Aug. 2008), 9 pages.
Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2010. Efficient Yarn-based Cloth with Adaptive Contact Linearization. *ACM Trans. Graph.* 29, 4, Article 105 (July 2010), 10 pages.
Andrei Khodakovsky, Nathan Litke, and Peter Schröder. 2003. Globally Smooth Parameterizations with Low Distortion. *ACM Trans. Graph.* 22, 3 (2003).
Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2013. Globally optimal direction fields. *ACM Trans. Graph.* (2013).
Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe Patterns on Surfaces. *ACM Trans. Graph.* 34, 4, Article 39 (July 2015), 11 pages.
Yu-Kun Lai, Miao Jin, Xuexiang Xie, Ying He, J. Palacios, E. Zhang, Shi-Min Hu, and Xianfeng Gu. 2010. Metric-Driven RoSy Field Design and Remeshing. *IEEE TVCG* 16, 1 (2010).
Na Lei, Xiaopeng Zheng, Hang Si, Zhongxuan Luo, and Xianfeng Gu. 2017. Generalized Regular Quadrilateral Mesh Generation based on Surface Foliation. *Procedia Engineering* 203 (2017), 336 – 348.
Ze Gang Luo and M. M. F. Yuen. 2005. Reactive 2D/3D Garment Pattern Design Modification. *Comput. Aided Des.* 37, 6 (May 2005), 623–630.
Martin Marinov and Leif Kobbelt. 2006. A Robust Two-Step Procedure for Quad-Dominant Remeshing. *Computer Graphics Forum* (2006).
James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. 2016. A Compiler for 3D Machine Knitting. *ACM Trans. Graph.* 35, 4, Article 49 (July 2016), 11 pages.
Michael Meißner and Bernd Eberhardt. 1998. The art of knitted fabrics, realistic & physically based modelling of knitted patterns. In *Computer Graphics Forum*, Vol. 17. 355–362.
Yuki Mori and Takeo Igarashi. 2007. Plushie: An Interactive Design System for Plush Toys. *ACM Trans. Graph.* 26, 3, Article 45 (July 2007).
Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust Field-aligned Global Parametrization. *ACM Trans. Graph.* 33, 4, Article 135 (2014).
M. Nieser, J. Palacios, K. Polthier, and E. Zhang. 2012. Hexagonal Global Parameterization of Arbitrary Surfaces. *IEEE Transactions on Visualization and Computer Graphics* 18, 6 (2012).
Olivier Nocent, Jean-Michel Nourrit, and Yannick Remion. 2001. Towards mechanical level of detail for knitwear simulation. In *WSCG*. 252–259.
Jonathan Palacios and Eugene Zhang. 2007. Rotational symmetry field design on surfaces. *ACM Trans. Graph.* 26, 3, Article 55 (2007).
Daniele Panozzo, Enrico Puppo, Marco Tarini, and Olga Sorkine-Hornung. 2014. Frame Fields: Anisotropic and Non-Orthogonal Cross Fields. *ACM Trans. Graph.* 33, 4 (2014).
Jianbo Peng, Daniel Kristjansson, and Denis Zorin. 2004. Interactive Modeling of Topologically Complex Geometric Detail. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 635–643.
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. Graph.* 34, 4, Article 138 (July 2015), 12 pages.
Gerard Pons-Moll, Sergi Pujades, Sonny Hu, and Michael J. Black. 2017. ClothCap: Seamless 4D Clothing Capture and Retargeting. *ACM Trans. Graph.* 36, 4, Article 73 (July 2017), 15 pages.
Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez. 2006. Periodic global parameterization. *ACM Trans. Graph.* (2006).
Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. 2008. N-symmetry Direction Field Design. *ACM Trans. Graph.* 27, 2 (2008).
Cody Robson, Ron Maharik, Alla Sheffer, and Nathan Carr. 2011. Context-aware Garment Modeling from Sketches. *Comput. Graph.* 35, 3 (June 2011), 604–613.
Dmitry Sokolov, Nicolas Ray, Lionel Untereiner, and Bruno Lévy. 2016. Hexahedral-Dominant Meshing. *ACM Trans. Graph.* 35, 5, Article 157 (June 2016), 23 pages.
Emmanuel Turquin, Marie-Paule Cani, and John F. Hughes. 2004. Sketching Garments for Virtual Characters. In *Proceedings of the First Eurographics Conference on Sketch-Based Interfaces and Modeling (SBM'04)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 175–182.
Ultimaker. 2018. Ultimaker 3. https://ultimaker.com/en/products/ultimaker-3.
Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. 2011. Sensitive Couture for Interactive Garment Modeling and Editing. *ACM Trans. Graph.* 30, 4, Article 90 (July 2011), 12 pages.
Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. 2016. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum* (2016).
Pascal Volino and Nadia Magnenat-Thalmann. 2005. Accurate Garment Prototyping and Simulation. *Computer-Aided Design and Applications* 2, 5 (2005), 645–654.
Pascal Volino, Nadia Magnenat-Thalmann, and Francois Faure. 2009. A Simple Approach to Nonlinear Tensile Stiffness for Accurate Cloth Simulation. *ACM Trans. Graph.* 28, 4, Article 105 (Sept. 2009), 16 pages.
Charlie C.L. Wang, Yu Wang, and Matthew M.F. Yuen. 2003. Feature based 3D garment design through 2D sketches. *Computer-Aided Design* 35, 7 (2003), 659 – 672.
Rundong Wu, Huaishu Peng, François Guimbretière, and Steve Marschner. 2016. Printing Arbitrary Meshes with a 5DOF Wireframe Printer. *ACM Trans. Graph.* 35, 4, Article 101 (July 2016), 9 pages.
Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2012. Stitch Meshes for Modeling Knitted Clothing with Yarn-level Detail. *ACM Trans. Graph.* 31, 3, Article 37 (2012), 12 pages.
Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. 2016. Designing Structurally-sound Ornamental Curve Networks. *ACM Trans. Graph.* 35, 4, Article 99 (July 2016), 10 pages.
Bin Zhou, Xiaowu Chen, Qiang Fu, Kan Guo, and Ping Tan. 2013. Garment Modeling from a Single Image. *Computer Graphics Forum* 32, 7 (2013), 85–91.