Computational Design of Knit Templates

BENJAMIN JONES, YUXUAN MEI, HAISEN ZHAO, TAYLOR GOTFRID, JENNIFER MANKOFF, and ADRIANA SCHULZ, University of Washington

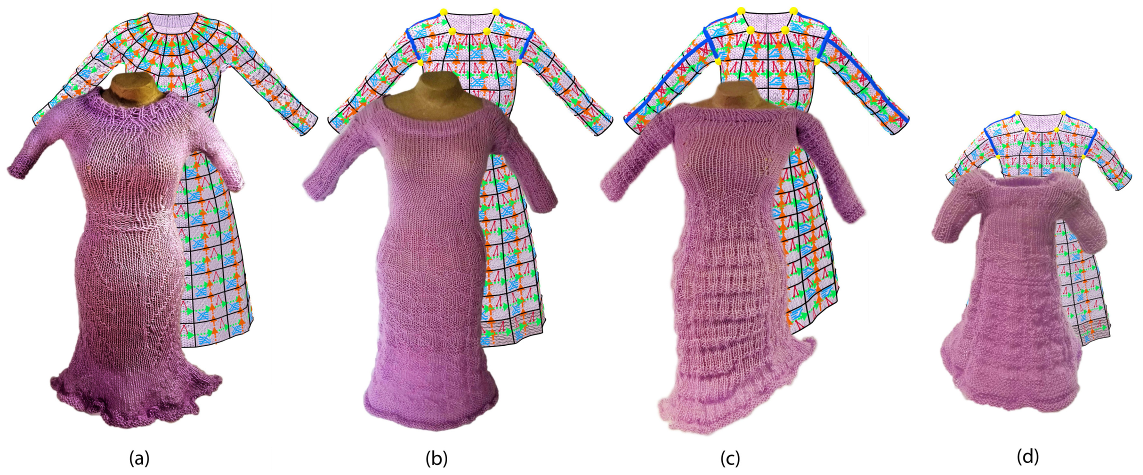


Fig. 1. Our interactive design system helps users explore key design axes for knitting to generate highly customized patterns from input shape templates; e.g., a seamless yoke dress with princess-cut apparent seams (a), and drop shoulder dresses with textures on the arms and skirt (b–d). The output of our system is a knit pattern template that lets users vary the shape while preserving the design, for example, creating a child's dress with short sleeves (d) that matches an adult dress (b), or varying skirt texture and angle, and sleeve knitting direction (c). The system guarantees that all results and variations are machine knittable.

We present an interactive design system for knitting that allows users to create template patterns that can be fabricated using an industrial knitting machine. Our interactive design tool is novel in that it allows direct control of key knitting design axes we have identified in our formative study and does so consistently across the variations of an input parametric template geometry. This is achieved with two key technical advances. First, we present an interactive meshing tool that lets users build a coarse quadrilateral mesh that adheres to their knit design guidelines. This solution ensures consistency across the parameter space for further customization over shape variations and avoids helices, promoting knittability. Second, we lift and formalize low-level machine knitting constraints to the level of this coarse quad mesh. This enables us to not only guarantee hand- and machine-knittability, but also provides automatic design assistance through auto-completion and suggestions. We show the capabilities through a set of fabricated examples that illustrate the effectiveness of our approach in creating a wide variety of objects and interactively exploring the space of design variations.

CCS Concepts: • **Computing methodologies** → *Parametric curve and surface models*; **Mesh geometry models**; • **Applied computing** → **Computer-aided manufacturing;**

Additional Key Words and Phrases: Knitting, quad-meshing

ACM Reference format:
Benjamin Jones, Yuxuan Mei, Haisen Zhao, Taylor Gotfrid, Jennifer Mankoff, and Adriana Schulz. 2021. Computational Design of Knit Templates. *ACM Trans. Graph.* 41, 2, Article 16 (December 2021), 16 pages.
https://doi.org/10.1145/3488006

Authors' address: H. Zhao, Am Campus 1/31/3, AT-3400 Klosterneuburg, Vienna, 3400, Austria; email: haisen@cs.washington.edu; T. Gotfrid, Paul G. Allen School of Computer Science & Engineering, University of Washington, Box 352355, Seattle, WA 98195-2355; email: gotfrid7@cs.washington.edu; J. Mankoff, Paul G. Allen School of Computer Science & Engineering, University of Washington, Box 352355, Seattle, WA 98195-2355; email: jmankoff@cs.washington.edu; A. Schulz, Paul G. Allen School of Computer Science & Engineering, University of Washington, Box 352355, Seattle, WA, 98195-2355; email: adriana@cs.washington.edu.

1 INTRODUCTION

Knitting is a versatile craft with rich aesthetic and functional design spaces. Its scope ranges from garments and toys to architectural structures and medical implants. The ubiquity of knit textiles in our lives is driven by programmable knitting machines. Machine knitting has the potential to become the next 3D printing: knit textiles are pervasive, customization of knit objects like clothing is valued, and machine prices have fallen within reach of maker spaces, small shops, and hobbyists.

However, makers lack design tools that provide needed control over familiar design axes, enable customization of existing designs, and encourage exploration of the design space. Consider the variety of dress shapes in Figure 1. There are several options for knitting patterns that can construct these shapes. The knitting designer must choose from patterns like these to achieve functional and aesthetic effects. Based on interviews we conducted with knitting designers, we identified seven design axes that are typically present in knitting patterns.

Shape Variability. Specific aspects (or geometry parameters) of a knit object's shape and size can vary without requiring a wholly new pattern. For example, a pattern designer might vary the length of a skirt and sleeves and the height of the waist to change an adult's dress to a child's dress (**b** vs **d**).

Composition. The basic building blocks of knit objects are sheets and tubes, which are composed to form a shape. For example, shoulder design in garments may be composed as smoothly merging or abutting tubes (**a** vs **b**).

Seaming. Related to composition, seams are used to connect building blocks (e.g., the shoulders in **b**) but also within a building block (e.g., the sleeve in **c**).

Orientation. Knitting looks different, and stretches differently, in the horizontal and vertical directions. As the sleeve of **c** shows, changing orientation can change the locations of seams. Thus, the orientation, or alignment of stitches along the surface, is an important design choice.

Surface Layout. Knitted objects are typically comprised conceptually meaningful regions. Layout includes the axes of symmetry, a line of increases or decreases, and the boundaries of a texture region, such as textures on the dress skirts, and the apparent seams in (**a** and **c**).

Curvature Shaping. Sheets and tubes are flat grids of stitches until shaping stitches are added, creating non-grid formations that add intrinsic curvature. Curvature is distributed around the bodice in (**c**) and concentrated towards the front in (**d**).

Surface Texture. Texture is achieved by varying stitches on the surface of a sheet or tube without varying curvature. Complex surface patterning of stitch variants gives knitting its aesthetic versatility, as shown in the dress skirts (**a–d**).

Customization and exploration of designs require *interactive control* of these seven design axes. There are two challenges to interactive design. First, knitting patterns must meet several discrete local and global constraints in order to be fabricable. Stitches must cover the surface with a small number of yarns while avoiding helical structures that cause cyclic dependencies in the fabrication process. Shaping stitches must be placed to capture mesh curvature, but also respect limitations on their type and relative alignment. Second, the seven axes are strongly intertwined at the stitch level, so making a decision along one axis can undo decisions made along other axes. For example, changing the *orientation* of stitches in the sleeve of the dress between (**b**) and (**c**) requires different *curvature shaping* stitches, changes the *composition* of the sleeve from a tube to a sheet, requiring a *seam* down the length of the arm, and rotates the *surface texture* by 90 degrees. Ideally, a design tool should enforce the constraints without overly limiting the designers' ability to explore, which is hampered if design decisions undo each other.

Prior work addressed several of these design axes. For example, Yuksel et al. [2012] demonstrate that a coarse quad-dominant mesh modeling of geometry enables the representation of important design axes of knitting, such as orientation and surface texture, and supports iterative modifications to these axes. However, without a design tool that can *automatically generate the quad mesh from high-level design input*, this representation cannot support iteration on axes such as composition and shape variability. Similarly, without a *strong theoretical connection between knittability constraints and the algorithm that generates a pattern from a quad mesh*, it is impossible to guarantee knittability. As a result, a design tool cannot suggest design solutions or warn users when they make changes that will break their design.

To address these limitations, we introduce two theoretically-grounded advances. First, we present a **novel meshing tool** that expresses the theoretical relationship between the singularity structure of quad meshes and the knitting design axes. This lets users of all knitting design skill levels generate coarse meshes that satisfy their design goals and avoid helical structures that lead to undesirable patterns. Furthermore, our algorithm takes as input a parametric template geometry that can vary over a specified parameter space, for example, a dress whose sleeves can vary from short to long or whose skirt can be elongated, allowing a design to be customized for a user.

Second, we introduce **formal knittability criteria** over the coarse mesh to ensure knittability without over-constraining the design space. By *knittability* in this work, we mean a valid *machine* knitting patterns in conjunction with constraints to account for physical limits of knitting machines and yarn that improve design robustness. Our validation and accompanying algorithms enable not only notifications about knittability problems, but promote interactive design across multiple design axes supported by auto-completion and automated design suggestions.

Based on these theoretical insights, we contribute a practical knitting design framework that supports

- requirements for design axes drawn from real-world knitting design experts;
- variable *template* patterns that correctly propagate design decisions as parameters of the geometry are modified;
- direct manipulation by users of multiple, interdependent design axes;
- automatic knittability checking and auto-complete assisted design;
- generation of knitting machine instructions.

2 UNDERSTANDING KNITTING DESIGN AND ITS RELATION TO QUAD MESHES

Knitting builds on a long craft, design, and artistic history [Spencer 2001]. Knitters can refer to books (e.g., Budd [2002]) or websites (e.g., Ravelry [2019]) that include a wealth of knitting patterns and design strategies. To discuss pattern design strategies, we first briefly review how knit objects are constructed (see McCann et al. [2016] and Underwood [2009] for a more thorough review). We then present the results of our study with knitting pattern designers, which drove our technical innovations. We will also explain how knitting relates to quad meshing to facilitate with understanding later sections.

Constructing Knit Objects . Knit objects consist of a grid-like fabric of interconnected stitches. A grid of stitches can be a *sheet* of fabric, or the ends can be joined to form *tubes*, called *knitting in the round* (Figure 2). Knit objects are composed of joining and cutting these elements in various ways and orientations.

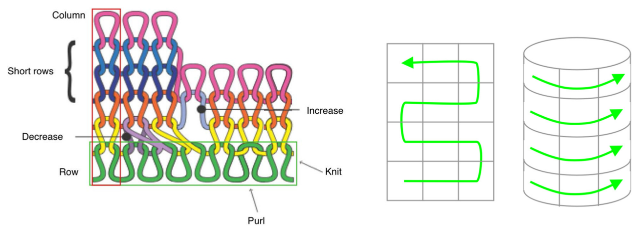


Fig. 2. Knit object construction. (left) A segment of knit fabric showing the basic stitch types and curvature shaping. (right). The two basic knitting primitives: sheets and tubes. Grid cells represent stitches, and arrows denote order of fabrication.

Stitches in the grid are formed sequentially by pulling a yarn loop through a "parent" loop in the row below it. To create a stitch, the loop it is pulled through must have been knitted already. This means that knitting inherently constrains the order in which stitches can be constructed. The central constraint of knitting is that the graph of stitch dependencies must be non-cyclic. The first row of stitches have no parent stitches, so they must be created with a special type of stitch called a *cast-on.* Similarly, the final row of stitches is closed with a stitch called a *bind-off,* which acts to stabilize the knit object. This is important because a non-bind-off stitch with no children can unravel.

The grid can be locally distorted by adding and removing loops to add curvature to the fabric. This is done along columns using special *increase* and *decrease* stitches and along rows using *"short rows"* (Figure 2). Also, by varying the direction, each loop is pulled through (knit and purl)—as well as other loop properties that create twists, holes, and overlayed loops—it is possible to create *surface texture*, such as cables, lace, and ribbing. As an example, varying the order that loops are stacked or pulled in an increase or decrease can create *leaning* increases or decreases, which appear to tilt left or right. Aligning several leaning stitches in a line can create the appearance of a seam in the fabric, as visible in Figure 11(b). Throughout the article, we will use "apparent seam" to distinguish these seam-like stylistic choices from true seams, which occur when separately knitted edges are sewn together as a post-process.

Knitting machines do not change the important axes of knit design, but they do add fabrication constraints not found in hand knitting. A V-bed knitting machine contains two beds of small needles at fixed spacing. A piece of yarn is shuttled back and forth between the beds by a carrier, and the needles are programmed to interact with the yarn (e.g., grabbing a loop), or each other (e.g., passing loops) as the yarn passes. Each needle holds one or more active loops at a time, and only these loops can be built upon. Once a machine drops a loop, it cannot pick it back up, so all stitches needed in the future must be held on needles. While loops can move between needles to create gaps for increases and overlaps for decreases, the physical dimensions of needle size and spacing limit how large a gap can be without snapping the yarn (*overstretching* the yarn), and how many loops can overlap before dropping off the small needles (*overstacking a needle*). These impose an upper bound on the number of loops each stitch can increase or decrease, which we conservatively cap at two, and make short-rows preferable for shaping (for hand knitters, increases and decreases are preferred).

Designing Knitting Patterns: A Research Survey and Study. To understand how designers construct knitting patterns, we surveyed popular keywords used with over 200,000 free knitting patterns available on Ravelry.com and conducted a contextual inquiry [Beyer and Holtzblatt 1999] with five knitters, focusing on the design process, motivations behind design decisions, and the use of patterns and other artifacts or tools. We describe the details of this study in our supplemental material and discuss here the key results.

Our *survey* of patterns showed that the most popular search keywords specified composition (129 K patterns mentioned seaming; 160 K mentioned seamless; almost all specified sheets (flat, 227 K patterns) or tubes (in the round, 181 K patterns)). Next in frequency came orientation (163 K patterns) and shaping (32 K patterns use short rows, which could underestimate the importance of shaping since almost all patterns use increases and decreases). The use of such keywords suggests that the identified design axes are of interest not only to pattern designers but also to knitters.

Regarding *study* results, participants tended to enter initial planning stages based on some inspiration (e.g., a picture) or an internal image of the final object they wanted to create. They drew this out as a sketch of the objects' composition or directly translated it onto a grid (a stitch-level representation) using (something like) perler beads or graph paper. They also determined knitting orientation, texturing, added symmetry, and created an assembly plan, as needed, at this phase of the project. More complex objects were broken down into different components to be designed individually.

Participants also discussed the challenges of modifying an existing pattern by re-sizing, coloring, texturing, and modifying design elements. Of these, resizing was by far the most common since small changes (like the specific knitter, yarn, and needles) could alter the number of stitches needed to achieve their goal. They did not have automated methods to do this. Instead, they used arithmetic, visual inspection, trying things on, or comparing theirs to a to-scale pattern. Because many knitters preferred to knit in the round, a second common change was to the composition of a pattern of sewn sheets (such as a sweater) into a tube. Thus, a knitting design tool could be of value not only to designers but also to the much larger group of knitters who simply want to make things that fit. This also demonstrates that stitch level decisions can be deferred until fabrication time while still respecting a design intent, indicating that the design is actually captured by a higher level structure.

Our survey and study provide evidence that an ideal tool for knitting pattern design should support changes in the target shape (especially size), composition, orientation, seaming, curvature shaping, texture, and surface layout features such as symmetry. However, simply supporting these design axes is insufficient. Knitters want to modify them, which is currently time-consuming and difficult. They struggle to ensure that the resulting pattern will be knittable and to preserve one design decision when modifying others.

Quad Meshing and Knit Design. Quad meshes are meshes with only quadrilateral faces, and the mesh vertices typically have four adjacent edges (or three if on boundary). Such vertices are called regular, while others are called irregular or singular. The surplus or deficit of adjacent edges is the index of a singularity (a singular vertex). A typical knit stitch, like a quad in a quad mesh, has exactly four neighboring stitches, a compelling parallel that motivates the most popular stitch-level representation: Stitch Meshes [Yuksel et al. 2012]. In addition to this low-level correspondence, we found two higher level correspondences.

First, based on our study with knitters, we see that they think and lay out knitting designs in sheets and tubes. When they combine the sheets and tubes, these structures may come together in greater or fewer than four edges, creating singularities. We found connections between several common knitting patterns and groups of singularities with particular indices, which we compiled into a set of composition rules that can be applied to a surface to control how the surface is broken down into sheets and tubes.

Second, knitting has two orthogonal directions formed by rows and columns of stitches (course and wale, respectively). In quad meshing, locally orthogonal axes are represented by a *cross field*—a pair of vector fields over a surface that are always locally orthogonal. Several methods for converting a triangle mesh to a quad mesh (remeshing) use cross fields to guide the orientation of quads. The orientation of rows and columns in knitting is an important design decision, so we employ a cross field to capture the designer's intent.

Quad meshes have rows and columns found by following neighboring quads on opposite edges. If two different quads in the same row are also in the same column—meaning there is a cycle, the row is a helix. While knitting in the round is technically knitting one helix, in our representation (and also in standard knitting patterns) this helix is not explicitly represented. Instead, it is broken into individual rows, and the overall helix is constructed only at knitting time when transitioning between these rows. This view of knitting makes it easier for us to reason about stitch construction dependencies; specifically that all of the stitches in a row must be constructed before any stitches in later rows. A quad mesh helix creates a cyclic dependency between stitches, which is not knittable.

3 RELATED WORK

Knitting design research can be segmented into three domains of inquiry: *representation*, which is typically stitch, primitive or mesh-based; *pattern knittability*, which includes both generation and verification of hand and machine knit patterns; and *interactivity*, which includes support for the seven design axes identified in our survey and study. In addition to these topics, we will discuss prior works on quad meshing for knitting, an important step in our system.

Representation of Knit Patterns. Representations fall into three categories. Stitch-based representations specify individual knitting operations and can be written as language, charts, or annotated meshes. Primitive-based ones address tubes and sheets directly, while patch-based ones extend mesh representations to multiple stitches per element.

Traditionally, knitting patterns are conveyed as stitch-based fabrication instructions, typically in a language called *knitspeak* [Hofmann et al. 2019] or visually in a chart (e.g., Briar [2019], SHIMA SEIKI [2019], and STOLL [2019]). Several systems use quad-dominant meshes rather than a chart or language to represent

Table 1. Desirable Features of a Knitting Design Tool Supported by Literature

	Variable Shape	Compositon	Orientation	Surface layout	Seaming	Curvature shaping	Surface texture	Machine Knittability	Hand Knittability	Mesh Input
I ra et al 2		☆							★	★
Yuk et . 20 2		☆	☆	☆		★	★			☆
W t a . 2			☆							★
W t a . 2			☆	☆	★	★	★		★	☆
P e a 2								★	★	☆
N ya n et .			☆			☆		★	★	★
N ya n e a .			☆	☆		★	★	★	★	★
M nn e a 2	★	★	☆			☆		★	★	
Kas 20 9	★	★	☆			☆	★	★	★	
S I S IKI/		☆	☆	☆		☆	☆			
O r Wo	★	★	★	★	★	★	★	★	★	★
	D r n ive							Desı e Feat		

Unfilled stars show partial satisfaction of the goal; either users have direct, but incomplete control, or complete but indirect control. For example, Yuksel et al. [2012] gives direct, but incomplete control over orientation since orientations may only align with coarse input mesh edges, and Narayanan et al. [2019] gives complete but indirect control over surface layout since symmetries and feature placement can be specified exactly by moving stitches one by one. If properties are assumed from the input, or are both indirect and incomplete, they are not considered controlled. An unfilled star for mesh input indicates that pre-processing outside the system is required.

objects at the stitch level [Igarashi et al. 2008; Wu et al. 2018, 2019; Yuksel et al. 2012]. In these systems, quad faces represent regular stitches, triangles the ends of short rows, and pentagons increases and decreases. Additional data are embedded to indicate orientation and to differentiate stitch types, such as knits from purls. While stitches directly correspond to the fabrication process, knittability is a primary issue here, which we discuss in the following section.

Since knitting is composed of tubes and sheets, an intuitive alternative is to specify a knit object as a composition of parametric sheet and tube primitives (e.g., Kaspar et al. [2019] and McCann et al. [2016]). This approach has the advantage of supporting parameterization of these primitives, enabling a single pattern to act as a template for customization. It is also possible to provide knittability guarantees over this representation, though matching an arbitrary input target shape is not straightforward.

If we generalize meshing to represent multiple stitches per quad, we gain many of the benefits of primitive-based approaches, while still being able to match an input target shape [Yuksel et al. 2012]. However, this approach has not been extended to include pattern generation and ensure knittability, and depends on a high-quality patch input that aligns with the desired design.

Knittability. Knitting an object requires the generation of a valid sequence of stitches. Because stitches are created by pulling loops through other stitches, such an ordering is not guaranteed to exist. Thus, stitch-based representations typically lack inherent guarantees of knittability (though commercial systems warn about potential failures). When representing the dual graph of a stitch-level mesh, dependency errors in a pattern can be found by cycle checking. Popescu et al. [2017] pioneered the graph approach with a hybrid representation, and Narayanan et al.

[2018] extended it by directly constructing a graph over an input model and identifying a sufficient set of constraints on the graph to guarantee machine knittability. However, these checks must be done for every iteration to a pattern.

Using a primitive-based representation, McCann et al. [2016] developed a provably correct transfer planning algorithm that schedules a pattern on a machine knitting machine. Kaspar et al. [2019] extended this work with more primitives, more composition options, and a more robust texturing system.

Knittability guarantees have not been demonstrated for patch-based representations. This would require imposing constraints that ensure knittability. If it were possible to guarantee knittability, a patch-based representation would be preferable to stitch-based approaches because of its generality, and to primitive-based approaches because of its flexibility in easily representing a wide variety of input shapes.

Interactive Knit Design Tools. Stitch-level mesh-based approaches are intuitive for representing underlying geometry, can automatically generate a knittable solution for a specific geometry, and support low-level control over specific stitches. For example, Nayaranan et al. [2019] support direct stitch mesh editing while ensuring machine knittability. However, concepts such as composition, orientation, and surface layout are not directly represented in a stitch-based mesh; rather, they are expressed through the stitches that are specified. Thus, the design tool cannot know when they are violated. Furthermore, all these modifications are lost if the original object's geometry is changed.

In contrast, primitive-based methods guarantee machine knittability and allow shape to be varied parametrically since primitives can be parameterized [Kaspar et al. 2019; McCann et al. 2016]. Such methods defer stitch-level decisions until instruction generation, which allows interactive editing of composition and other design goals, such as curvature and texture (see Table 1). However, this approach has two key disadvantages. First, it requires expertise to model a desired shape, making it particularly challenging for applications involving more complex geometry. Second, while it is possible to control composition and shape variations, editing is indirect, requiring expertise to achieve even simple variations that can depend on multiple parameters in a complex way—a classic problem in parametric **computer-aided-design (CAD)** systems [Yares 2013].

Supporting direct editing of design goals requires a representation that relates stitches to shapes. Patch-based approaches have this potential. Prior work demonstrated the power of patches to allow control over curvature shaping and surface texture, and to allow movement and changing of stitch types [Yuksel et al. 2012]. However, several key limitations remain, which our work addresses. First, Yuksel et al. [2012] use a coarse polygonal mesh as input, which requires a high level of user expertise to generate, and limits interactive control over composition and orientation. Our first key contribution shows that by developing meshes that correspond to a knitter's conceptual breakdown of a knit object, we can enable control over multiple important design axes. In particular, we prove that by controlling singularities in the mesh, knitters can intuitively and directly specify and iterate on these design axes and generate helix-free quad meshes that are necessary for knittability.

Furthermore, Yuksel et al. [2012] are not concerned with knittability, generating patterns only suitable for simulation and rendering. While rules have been developed to ensure machine knittability on stitch-level meshes, where constraints come directly from analyzing the fabrication process [Narayanan et al. 2018], it is not trivial to extend this to patches. This is the second key contribution of our work. We design a lightweight set of high-level patch constraints that do not over-constrain the design space but enable us to create and formally prove the correctness of, an algorithm for translating them into knittable patterns. This allows novel system-level contributions: constraints can be directly encoded in solvers, enabling interactive verification and completion during labeling, automatic seam placement, and geometric optimization, while respecting machine constraints and shaping preferences.

Quad Meshing. Our method works at the patch level, which is defined as a coarse quad mesh on the input surface. Quad meshing is an active research area and we refer readers to Bommes et al. [2013] for a survey. The fundamental challenge in applying existing quad-meshing techniques to patch-level knitting design is allowing users to control the composition and surface patch layout. Extensive work on quad meshing [Bommes et al. 2013] has shown that field-guided methods best enable user control. In field-guided methods, orthogonal vector "cross"-fields on the surface are optimized for a given smoothness energy and to meet user specifications (e.g., direction strokes). A quad mesh is then created by finding a parameterization whose gradients are optimally aligned with the field. For a review of concepts in field design, we refer interested readers to Section 3.2 of a state-of-the-art report [Vaxman et al. 2016].

Despite great advances in this area, directional control while avoiding helices remains challenging, particularly for coarse meshes. Solutions to directly remove helices [Bommes et al. 2011a] would change composition guidelines in unpredictable ways. Polyvector fields with curl reduction [Diamanti et al. 2015; Panozzo et al. 2014] can minimize, but fail to completely avoid, helices. Directly partitioning the mesh into quad layouts [Campen and Kobbelt 2014] could avoid helices but at the expense of a manual strategy that does not map well to how knitters make patterns.

Different quad and quad-dominant meshing techniques have been proposed for knitting. For example, Wu et al. [2018] use a field-guided method to generate a stitch-level quad-dominant mesh but cannot ensure knittability because cycles cannot be fully avoided. [Narayanan et al. 2018] ensure knittability using a harmonic scalar field meshing technique that takes as input level set constraints of a scalar function approximating knitting rows. However, this method works at the stitch level and does not allow for composition or surface layout control.

In this work, we propose a new strategy to enable design control while avoiding helices in a coarse quad mesh. This is achieved by utilizing a key insight on the relationship between common knitting compositions and quad mesh singularities. In our system, composition guidelines selected by users are directly translated to singularity constraints on the mesh, which, in turn, can be used to drive a cross-field design algorithm based on trivial

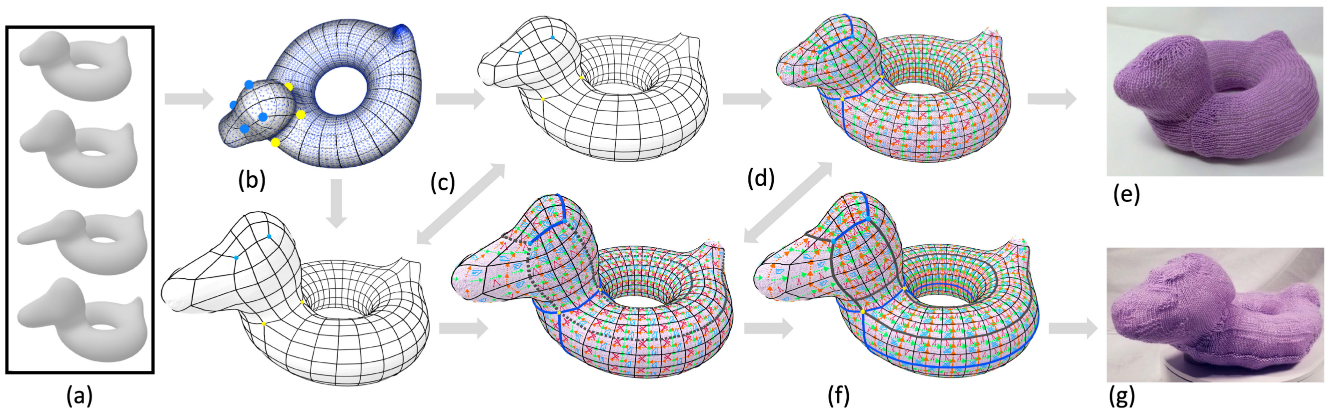


Fig. 3. Overview of our framework. (a) Triangle meshes from a parametric template (the system deals with a single mesh at a time). (b) Input triangle mesh with user annotations of composition, layout, and direction guidelines. (c) Generated quad mesh patches, which are consistent across template variations. (d) Quad mesh annotated for knitting the body tube in the round using short rows to curve the tube. Blue lines indicate seams. The same design applies to all template variations (two shown here). (e) Duck knit with short rows. (f) Quad mesh annotated with different textures and orientations; the body is knit as seamed sheets with decreases. (g) Duck knit with textures and a large head from template (f).

connections [Crane et al. 2010]. We choose to use pure quad meshes rather than quad-dominant meshes in order to exploit this singularity structure. Achieving knittability with quad-dominant meshes would require extra constraints aligning singular faces. We defer to Section 5 to introduce the details of our method.

4 SYSTEM OVERVIEW

Figure 3 illustrates our system with an example of designing a toy duck. The system takes as input one variation from a parametric template of a duck mesh (**a**); if the geometry is not already a triangle mesh, the system will tessellate it into one. A parametric template is defined by parameters that describe degrees of freedom $\mathbf{q} \in \mathcal{A}$, where $\mathcal{A}$ defines the ranges of parameter variations that map to continuous geometric deformations. For our examples, we created the input parametric templates semi-manually by first creating cuboid cages in Blender and then computing the coefficients for interpolation [Schulz et al. 2017a]. Parametric templates can also be created with a variety of geometric editing methods [Gal et al. 2009; Jacobson et al. 2011] or parametric CAD tools. Our system allows users to create knit templates by enabling consistent control of design axes across the space of geometric variations.

The user starts by directly annotating knitting *composition*, *layout*, and *orientation* guidelines for how to break the duck into patches by indicating (blue dots in (**b**)) that the top of the head should be knit as a sheet to create a flap, that the head should be a tube abutting the body (yellow dots in (**b**)), and that the layout should be symmetric across the body. These composition guidelines are selected from an illustrated menu (Figure 4) and placed by clicking a position on the input mesh. They can also draw desired stitch orientation directly on the mesh, as well as explicitly specify layout boundaries as feature lines (not needed for this example). Our system maps these knitting directives to orientation, edge, and singularity constraints to create a novel coarse quad re-meshing algorithm that jointly re-meshes the entire parameter space of the input template to create a single parametric patch layout that satisfies the guidelines for all parameter values, (two shown, (**c**)). Users can choose to enable any available symmetries for their annotations, which encourages, but does not guarantee that the resulting quad layout generation will be symmetric.

This patch layout becomes the canvas on which the user designs their template. By clicking and dragging, they specify per-quad orientations, surface textures, and curvature shaping guidelines, as well as specifying seams. Users may also enforce exact symmetries by placing "equal stitch count" constraints on specific patch edges, which will be taken into account during the stitch generation step. Our interface is backed by a patch-level knittability solver that not only validates the user's design but also assists in the design process by automatically finishing partial designs with knittable completions. In (**d**), the user has specified seaming-off the neck and knitting the inner tube with short rows, as well as a simple stockinette texture, while in (**f**), the user has rotated the orientation on the body, and our solver has assisted by finding an alternative seaming strategy that works with that orientation. At any point during design, the user can vary the template parameters to preview different customizations of their design. Finally, the user selects two variations to generate machine instructions for and fabricate (**e** and **g**).

5 INTERACTIVE SURFACE PATCH SPECIFICATION

After loading a parametric template of a triangle mesh, the user seeks to automatically generate a coarse quad mesh that adheres to the composition, orientation, and surface layout guidelines. Our key insight here is that *there exists a direct relationship between the singularity structure of cross fields and knitting composition guidelines.* By identifying this correspondence, we created a quad meshing algorithm that is both theoretically grounded and able to represent important and commonly used knit pattern design techniques. Importantly, the method ensures the resulting mesh is helix free, a fabrication requirement of knitting, by providing feedback on knitting composition requirements, as well as a helix visualization tool to help users tune the grid size parameter toward a helix-free design.

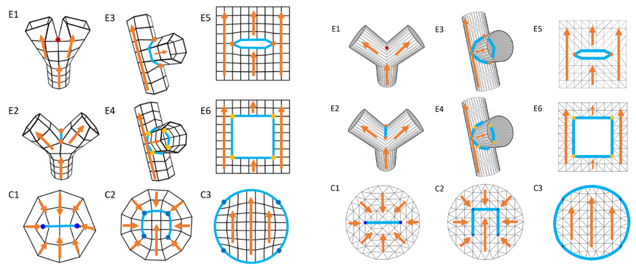


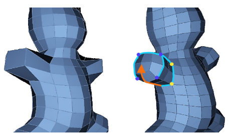
Fig. 4. Knitting composition rules shown as quad singularities on remeshed quad meshes. The corresponding triangle mesh versions are shown on the right-hand side. Knitting directions are in orange and seams in blue. Expansion rules correspond to negative singularities (top two rows): joining tubes with no change of knitting direction can be done without seams with one -1 singularity (E1) or with a seam connecting two $-\frac{1}{2}$ singularities (E2); joining tubes with direction change requires closed seams, which can be done with two $-\frac{1}{2}$ singularities (E3) or four $-\frac{1}{4}$ singularities (E4); adding a hole without changing the knitting direction can be done with two $-\frac{1}{2}$ or four $-\frac{1}{4}$ singularities (E5, E6). Contraction rules correspond to positive singularities (bottom row): knitting in the round and joining at a line seam corresponds to two $+\frac{1}{2}$ singularities (C1); knitting in the round and closing with a flap that is knitted as a sheet and seamed along its boundary corresponds to four $+\frac{1}{4}$ singularities (C2); and knitting a flat patch corresponds to four $+\frac{1}{4}$ singularities at the boundary (C3).

Singularities and Composition Guidelines. The key composition guidelines used in knitting to assemble tubes and sheets map directly to singularities in the quad mesh. This is not surprising given their mutual correlation with the shape topology. We identified nine knitting-relevant composition rules (see Figure 4), each defining a set of singularities and seams whose indices either add up to -1 or $+1$. *Expansion rules* (-1 singularity sum) compose tubes together (E1—E4) or create a hole (E5—E6). *Contraction rules* ($+1$ singularity sum) close tubes with a slit seam (C1) or flap (C2), or bound sheets (C3). Contracting to a point is omitted as a $+1$ point singularity is not possible on a quad mesh.

This approach has several advantages.

First, composition rules describe common knitting patterns; for example, E1 splits a glove into fingers, and E2 is common in the armpits of sweaters. Therefore, knitters can work in terms they already understand rather than in singularities—they select composition rules from a menu and then click on the mesh to specify where they should be placed (see pilot study in Section 8).

Second, because which composition rules to use is associated with the template's topology, we can validate a composition and provide feedback on whether more expansion or contraction rules are needed: the sum of all singularity indices must be equal to the Euler characteristic, χ: $\sum_{v \in \mathcal{V}} index(v) = \chi = 2 - 2g - b$, where $\mathcal{V}$ is the set of vertices, g the genus number, and b the number of boundary loops. Importantly, giving control over composition allows the same shape to have multiple valid compositions.



For example, in the inset figure, the arm of the teddy bear model could be created by adding curvature to the body tube (left), which is more likely to fail on a machine, or knit by doing a merge and then a flap at the hand (right), which is a more natural design for knitters to come up with. Our interface allows users to have such high-level control while ensuring that the total sum of subscribed singularities is valid.

Finally, with these composition rules, no additional seams are necessary except on surfaces of non-zero genus (e.g., a torus would need a seam to separate the first and last row), an additional benefit of our approach.

Controlled Meshing for Knitting. Based on the correspondence analyzed above, composition guidelines selected by users define singularity constraints on the mesh. Our system uses these constraints to drive the trivial connections cross-field design algorithm [Crane et al. 2010].

Designers can further provide knitting direction guidelines by drawing directly on the mesh. *Soft directional guidelines* respect the existing composition and are treated as constraints on the trivial connections solver. *Hard direction guidelines* override the singularity structure imposed by the composition; the field is completely determined through cross-field interpolations, with these direction guidelines as constraints [Ray et al. 2008]. Hard directional guidelines are typically not necessary and often ill-advised because they may cause arbitrary singularities and create cyclic dependencies (helices). To give designers full control of the directional field, our method includes this option and checks for helices [Bommes et al. 2011b] providing feedback to designers.

We further let designers sketch directly on the mesh to place feature lines for surface layout control. We also allow easy specification of smooth closed loops using the method proposed in Campen and Kobbelt [2014]. If feature lines are specified as seams, the mesh gets cut along them; this affects field optimization since there can be no smoothness constraints across seams. Otherwise, feature lines are treated as hard integer constraints in the integer grid optimization, which enforces placement of edges on the generated quad mesh.

Finally, these fields and constraints are used to create a mesh using **mixed-integer quadrangulation** (**MIQ**) [Bommes et al. 2009]. The key modification that we make to the MIQ optimization relates to templates, which we now discuss.

Parametric Template Variations. Parametric templates have been extensively used in the fabrication community to allow shape variability and customization while preserving manufacturability [Schulz et al. 2014; Shugrina et al. 2015]. Commercial systems also use templates for personalization, e.g., for 3d printing (https://www.thingiverse.com/). To generate a parametric knit template, we must define a consistent quad-mesh across the parameter space defined by template parameters $q \in \mathcal{A}$. By consistency, we mean that the user should define design axes only once, and they should propagate consistently throughout the full parameter space.

We assume that the user inputs a parametric template triangle mesh with point-wise correspondence—i.e., there is a bijective homeomorphism between $\mathcal{M}_q$ and $\mathcal{M}_q$ for all $q, q \in \mathcal{A}$, where $\mathcal{M}_q$ is a mesh representing the variation defined by q. This correspondence is directly specified when a parametric model is created by geometric deformations, and there are methods for constructing these maps for parametric CAD models [Schulz et al. 2017b]. Given a point-wise correspondence, a naive solution could define a quad-mesh for one shape and propagate the result. However, this may create quads with high distortion if variations are large. Consistent quad meshing has only recently started to be studied. Azencot et al. [2017] proposes a method for consistent cross fields between two shapes with point-wise correspondence. However, this work would not allow us to preserve composition guidelines across variations since singularities and combinatorics of the final meshes may vary.

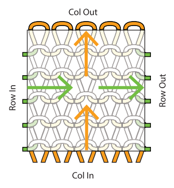
Our key insight of representing the composition axis as singularities makes consistent template generation possible by propagating the singularities with the point-wise correspondence and using them to drive the cross-field optimization on each mesh $\mathcal{M}_q$. We can then jointly solve for a parameterization using a variation on MIQ. As described by Bommes et al. [2009], MIQ takes as input a cross field, defined by two orthogonal vector fields $(\mathbf{u}_T, \mathbf{v}_T)$; it finds a parameterization onto an integer grid (u, v) by minimizing $||h\nabla u - \mathbf{u}_T|| + ||h\nabla v - \mathbf{v}_T||$ integrated over the surface, for some size parameter h and additional integer constraints derived from singularities. Since singularities are preserved in our method across template variations, we can use any value of q to define the integer constraints and minimize an energy summed over all variation of the mesh $q \in \mathcal{A}$:

$$E = \int_{\mathcal{A}} \int_{M} \left\| h\nabla u - \mathbf{u}_T^q \right\| + \left\| h\nabla v - \mathbf{v}_T^q \right\| dAdq.$$

To solve this numerically, we discretize the inner integral as a sum over triangles and the outer one by sampling values on $\mathcal{A}$. Since computation could grow significantly with the number of samples, we solve MIQ in parallel across n different configurations of the mesh and add in a linear equality constraint that the (u^q, v^q) coordinate values should be equal according to the point-wise correspondence between meshes. Because these are linear constraints, we can use them to eliminate variables that are part of the MIQ solver. This makes the system matrix for solving n samples about the same size as for one sample.

6 INTERACTIVE SURFACE PATCH ANNOTATION

Once a patch layout is designed, it is used as a grid to lay out design guidelines that directly control seaming, surface texture, and curvature shaping, and to provide additional partial control over orientation and composition. Conceptually, each patch corresponds to a quadrilateral patch of knit fabric, with each side presenting a uniform boundary (row or column) to its neighbors, as shown in the inset image. To modify the design, the designer uses five tools to set design guidelines as labels on the mesh elements, which will be used to control the final pattern generation.



The *orientation* tool allows row and column directions to be set by clicking and dragging across coarse mesh faces. The *seaming* tool allows seams to be created by clicking on a coarse mesh edge to create a seam extended to the next singularity or mesh edge. Right clicking allows non-singular vertices to be marked as stopping points for seams to allow for arbitrary seamed layouts. The *texture* brush applies knit texture labels (such as ribbing) to coarse mesh faces. The *constraints* brush can constrain the type of

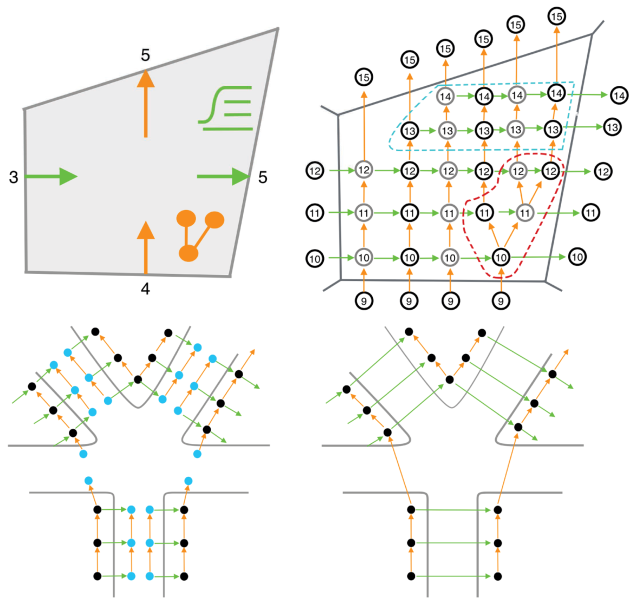


Fig. 5. Patch design and construction. (top left) Patch design seen in the user interface. Here, short rows are positioned at the top and leaning increases to the right. The gray background color indicates that the texture is a rib. (top right) Patch knit graph generated by our system. Node color indicates knit or purl. Leaning increases and short rows are highlighted to illustrate their positioning. Nodes outside the quad are patch borders used to ensure correctness and to align neighboring patches when connected. When patches are connected to form the final pattern, they are contracted away (bottom).

existence of increases, decreases, or short rows in faces, as well as add sizing constraints along coarse mesh edge paths (constraining two paths to have the same stitch count). Finally, the *eraser* tool remove previously placed design guidelines.

Our system assists users by validating the knittability of their choices and automatically completing partial designs as users work. Given a coarse patch mesh $M = (\mathcal{F}, \mathcal{E}, \mathcal{H}, \mathcal{V})$, with faces $\mathcal{F}$, edges $\mathcal{E}$, half-edges $\mathcal{H}$, and vertices $\mathcal{V}$, the labels are

Orientation	$D(H) : \mathcal{H} \rightarrow \{\text{Col In, Col Out, Row In, Row Out}\}$
Seaming	$S(E) : \mathcal{E} \rightarrow \{\text{True, False}\}$
Curvature Shaping	$\text{can\_shape}(F) : \mathcal{F} \rightarrow \{\text{True, False}\}$
	$\text{shaping}(F) : \mathcal{F} \rightarrow \{\text{Row In, Row Out, Both, Distributed}\}$
	$\text{can\_shortrow}(F) : \mathcal{F} \rightarrow \{\text{True, False}\}$
	$\text{shortrow\_side}(F) : \mathcal{F} \rightarrow \{\text{Col In, Col Out}\}$
Texture	$\text{tex}(F) : \mathcal{F} \rightarrow \mathbb{Z}$
Time	$T_F : \mathcal{F} \rightarrow \mathbb{Z}$

Figure 5 shows how these labels appear in our interface. Although not explicitly represented, the user can still make some compositional changes, for example, choosing between a tube and a seamed sheet in Figure 6. Curvature shaping guidelines indicate whether a type of curvature shaping is allowed in a patch, and, if so, they provide guidelines for how to place them. The location of short rows is particularly important for knittability. The surface texture parameter is an index into a database of knit-purl textures from [Kooler 2012]. Time is a proxy for the order of patch fabrication, and stitch count measures the length, in stitches, of each quad side. The time parameter is automatically set by our system.

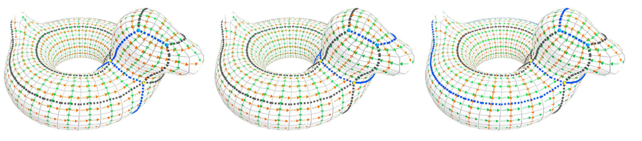


Fig. 6. Interactive seaming suggestions. (left) Initial suggestion minimizing seams (blue lines), assuming no composition or layouts are given. (middle) Suggestion after the user specified that the full head would be a separate piece seamed at the neck. (right) Suggestion after the user changed knitting direction on the body.

Although these properties must be defined for each mesh element, the user does not need to manually set all of them. Instead, any specific decisions a user does make are treated as constraints on the design space, and our system automatically completes a valid design from partial specifications by inputting these constraints plus our patch-level knittability constraints into a constraint solver, then optimizing for minimal seaming. This is especially helpful for finding seam patterns on complex shapes, as shown in Figure 6.

Patch Level Knittability. We build upon validity properties on a stitch-level mesh, defined by Narayanan et al. [2018]. While this prior work defines low-level constraints by analyzing the fabrication process, we need to guarantee fabricability using only patch level information. To accomplish this, we designed a lightweight set of constraints on the patch representation that guarantees machine knittability if patches are constructed with a small set of constraints explained in the next section.

Here, we give an overview of the constraints on patch layout and parameters. Their mathematical description and proof of sufficiency are left to supplemental material. Patches are considered neighbors only if the edge between them is not a seam, and the half-edge labels on non-seam edges must be compatible pairs: (Row In, Row Out) or (Col In, Col Out). This allows us to refer to the patch structure with respect to its *dual graph* of row and column edges (ordered from Row Out to Row In), and define row and column neighbors.

C1 - Right-Handed Patches: The row and column directions of knitting form orthogonal axes on the surface, and our first constraint ensures that patch orientations align to these axes. To enforce this, we require that orientation labels follow the order Col In, Row Out, Col Out, and Row In, when circulating a patch boundary counter-clockwise. Patches with exactly one side with each orientation in this order we call *regular*, as they are almost always the desired structure.

We additionally allow *irregular* quads to enable greater flexibility in orientation control post-quad meshing (see Figure 7), which can be particularly useful when hard constraints on orientations are used to override composition guidelines. Because faces are all quadrilateral, one of the other orientations will be missing. In knitting terms, these missing orientations are the start or end of short rows, or where a piece of fabric is knit to or from a point. We only allow the doubling of one orientation per face, except for the special cases of all Col In or all Col Out, which our system breaks into four irregular faces to use as sources and sinks.

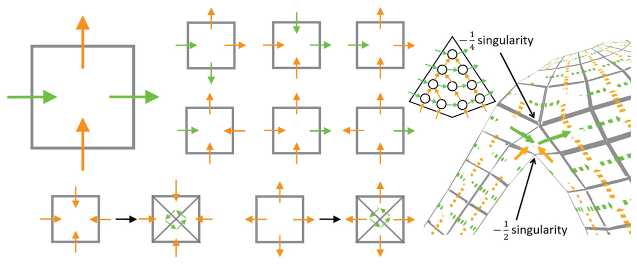


Fig. 7. Left: All valid quad faces (C1), (C3). Orange arrows are column edges; green are row edges. The large face is regular, the most common. The six on the right are allowed irregular faces, which can be interpreted as regular by adding a 0 length side of the missing direction and merging similar sides. Sources and sinks (bottom) are partitioned into four regular quads (with one zero-length side each). Right: An irregular face in use and an example tessellation.

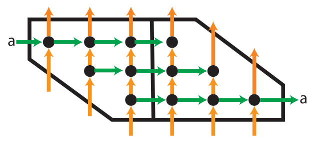


Fig. 8. A helix formed by inconsistent short rows.

C2 - Time Aligned: Time value is equal between row neighbors and strictly increases between column neighbors in the out–in direction. This ensures that there is a valid order of fabrication for the object.
C3 - Limited Row Degree: Each face has at most one Row In and Row Out side. This is necessary to avoid creating cyclic knitting dependencies when connecting patches, as it allows patches to be grouped into distinct rows.
C4 - Consistent Short Rows: Row neighbors must have the same short-row location guidelines. This is used to prevent helices from forming within a row of patches (Figure 8). They cannot be aligned with splits or merges (discussed in Section 7).

Encoding Constraints. We implement these constraints as a system of **satisfiability modulo theories** (**SMT**) equations included in our supplemental material. We use SMT because we have mixed boolean and integer constraints and because SMT solvers can validate a design before all variables are set, and will even find a complete and valid set of labels whenever possible, which we use as design suggestions. In order to encode (C1) with boolean constraints, we express orientation as a pair of boolean variables and enforce right-handedness per mesh corner by limiting which orientations can be adjacent.

Enforcing (C1) at corners allows us to make an optimization that improves both the *speed* of the solver and the *quality* of the results. As stated above, regular faces are preferable in most cases. Irregular faces are only actually necessary at singularities, such as the example in Figure 7, where their doubled or omitted directions offset extra or missing edges. We therefore limit our corner constraint to only allow non-regular adjacency at singularities by default. If the user wants an irregular face elsewhere, other mesh vertices can be designated to act like singularities.

Another optimization made for speed and quality is seam bundling and optimizing for minimal seaming. An easy solution to find is the trivial solution of seaming every edge. In most designs, using the minimal amount of seaming is desirable, but computing this across every possible edge is intractable. To mitigate this, we leverage the facts that (1) there is no structural use for a seam that does not partition the mesh, and (2) the set of mesh separatrices is a sufficient set of seams to make any shape knittable with only regular faces. Rather than consider each edge separately, we bundle paths of edges into long seams that are assigned to a single SMT variable. By default, we initially use the separatrices as bundles, but the user can click on edges to add other seam bundles for solver analysis. This makes the solver fast enough to find a minimally seamed solution by binary search on a maximum total seam length.

This heuristic is adequate for minimizing the amount of sewing that is necessary, but the solver may suggest seams on features that are not aesthetically pleasing. Because the suggestions are given *at interactive rates*, the user can interact with the model by disallowing seams in some locations and enforcing them in others. A result of such interaction is shown in the second duck image Figure 6 where the user prescribes a seam along the neck and the system in term suggests a flap on top of the head to minimize the total seams under this configuration. Finally, the third image shows what happens when the designer decides to change the knit direction on the body. The system automatically suggests a seam. All of these suggestions were provided at interactive rates.

7 KNIT PATTERN AND MACHINE INSTRUCTIONS GENERATION

Once a design is finalized, a specific template instance is chosen to be knit. Here, we describe how provably knittable machine instructions are generated for any set of template parameters, which we now consider fixed.

Sizing Optimization. The first step to creating an object is determining the shape and size of each patch. To do this, we calculate an integer stitch count for each side of each patch by minimizing the squared error between the side length of each patch in the template configuration chosen, and the length of that number of stitches as produced by the target machine. This optimization is done in the presence of several constraints to improve quality and guarantee knittability.

Symmetry is enforced along mesh symmetries chosen by the user. We also account for the user's shaping choices here: if they specify no short rows in a patch, then the number of rows in and out must be equal, and similarly for columns if forbidding increases and decreases. The ratio of width change to height and height change to width are capped to avoid needing increases or decreases of more than two stitches at a time, or overly tall short rows. The user is also allowed size lines, paths whose total length is important to get exact for sizing (such as the length of a sleeve or circumference of a cuff), which are constrained to a maximum total error.

Finally, a *feasible splits and merges* criterion is enforced. Whenever more than two tubes are joined seamlessly within a row, the center tube(s) must have an equal number of stitches on their front and back halves so that they can be flattened evenly between the front and back stitch beds. The exact formulation of the objective function and constraints are given in our supplemental material.

Pattern Generation. Pattern generation involves tessellating each patch into a composition of stitches, connecting them, and defining an order for stitch construction allowing the pattern to be scheduled on a knitting machine. Stitch-level representations of patches will have one stitch wide borders, sized according to the sizing optimization, and connected with simple 1:1 edges. These borders are used to define how patches are merged together to construct a stitch-level pattern, as shown in Figure 5.

A design goal of our system is to be extensible for future advancements, so we want our guarantees of knittability to be agnostic to how patches are generated. To this end, we define a minimal set of requirement for patches which, in conjunction with our coarse mesh properties, guarantee machine knittability:

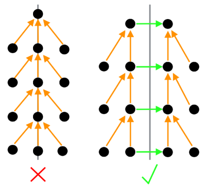
P1 - Knittable: a patch plus its border is a valid Knit graph as defined by Narayanan et al. [2018] (described below), and
P2 - Consistently Stacked: any exposed short rows (rows that connect to only one row border) are either all stacked at the top or bottom of a patch, according to its short row location, shortrow_side(F), and are exposed along the same border.

Validation. We are using knit graphs [Narayanan et al. 2018] as our formalism of knittability. These are directed graphs with row edges and column edges, where each node represents two stitches in a column. Each node also has an integer time value, similar to that of our coarse representation. Knit graphs are defined to have several properties which, if all met, ensure machine knittability. Unfortunately, validating several of these properties requires stitch level information of non-neighboring patches, which we cannot determine at the patch level. We formulate a slight variation of the Knit Graph properties that imply the original but make the problematic properties locally checkable. In particular, we remove the helix-free criterion and replace it with a stricter version of time alignment. This formulation also covers some edge cases that the prior work did not encounter, but which we must contend with. We leave a detailed discussion of the differences in our formulation and a derivation of the original properties to supplemental material. Our knit graph properties are

K1 - Consistent Handedness: Knit graph nodes are right-handed in the same sense as (C1). This ensures that the represented fabric does not twist or cross over itself on the bed.
K2 - Time Aligned: Time values are equal within a row and strictly increase up columns.
K3 - Limited Node Degree: Each node has at most one row neighbor on each side, and at most two column neighbors. The row restriction reflects the fact that a stitch has only two yarn ends, and the column restriction prevents the machine from overstacking or overstretching yarn on or between needles.
K4 - Feasible Splits and Merges: Interior tubes at splits and merges have an equal number of stitches on their front and back halves. Since splits and merges only occur along patch boundaries, this is directly enforced by sizing optimization.

Now we sketch a proof of knittability—a formal proof is found in our supplemental material. Property (K4) is an exact constraint on the sizing optimization ((C4) only allows full rows at splits and merges). Properties (K1) and (K3) depend only on the edge structure around nodes, which does not change with the contraction used to join patches, so (P1) is sufficient to guarantee them. (K2) will be true if the border nodes on adjacent patches have matching time values. While this is not true by construction, (C3)–(C4) and (P2) together allow us to re-scale time values within each patch so that they do align on borders.

When designing our framework, we deliberated between having graph nodes or graph edges on the borders between patches. The inset image illustrates why we chose edges. If nodes are chosen, then it would be possible to change the local edge structure when merging, violating (K3) (left). Using edges also allows easy specification of increases or decreases leaning into a shared edge, a common knitting effect creating an apparent seam (right).



Knit Graph and Instruction Generation. Our system uses a simple patch generation algorithm. An example patch is shown it Figure 5. We construct patch knit graphs in rows of constant time value (K2), linearly interpolating their widths. Sizing optimization ensures that no row is more than double the width of its neighbors, so we can distribute increases and decreases without violating (K3). (C1) says that the overall patch has the same orientation requirement as (K1), so we can place all internal edges with the same orientations. We place short rows in accordance with (P2) to account for differences between rows in and out. Finally, we construct a simple, one node border for each edge attached by simple edges.

8 RESULTS

We set out to design a system for creating high-level knitting templates that can be customized to enable shareability and remixing, and that enables fast and easy iteration over the seven axes of knit design. We demonstrate the effectiveness of our approach by a series of examples highlighting the capabilities of our system in quality parametric meshing, creating a wide variety of objects, and interactively exploring the space of design variations.

Quad Meshing. Other work has taken a field-based meshing approach to knitting, but ours is the first to explicitly incorporate singularity structure to control composition. The teddy bear example demonstrates the benefits of this approach. It would be natural to knit the teddy bear using tubes for each limb and one for the body and head. Achieving this composition from only user-provided direction strokes is very difficult because specific compositions need specific singularity placements, and singularities are difficult to control with only directional strokes. In Figure 9, the left two images are typical examples of a purely orientation-based meshing of the model. The inability to precisely control field singularities leads to helices wrapping around the body. These require long and unnatural seams to break the dependency cycles they induce. On the right is a structure resulting from the application of our composition rules, and the resulting knit bear.

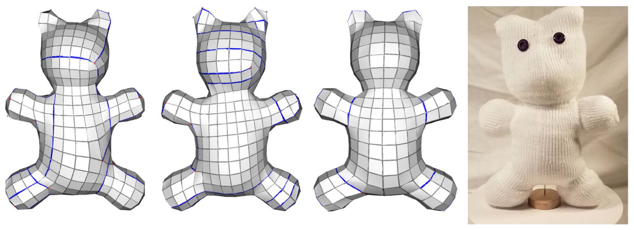


Fig. 9. (left two) Meshing results achieved by sketching directions on the surface. Both have helices that must be seamed off and would be complicated and non-intuitive to sew. (third) The meshed teddy that our system generated to match the composition rules of knitting each limb in the round and then sewing them onto the torso followed by (fourth) an image of its physical realization as a multi-part knit. The blue lines are seaming suggestions proposed by our algorithm.

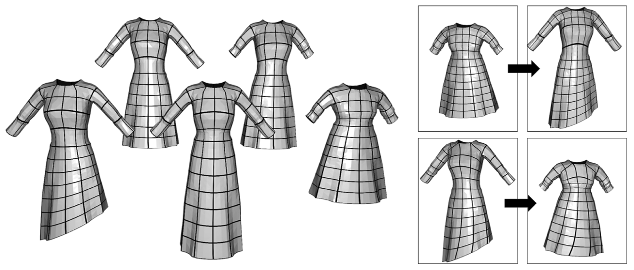


Fig. 10. Our consistent meshing is shown on the left and compared to the naive approach of running the MIQ on one mesh and transferring the resulting quadrangulation onto another using the point-wise correspondences (two examples shown on the right).

We also validate our joint parametric MIQ by comparing it to the nave strategy of solving against a single mesh variation and propagating via pointwise correspondence. In Figure 10, all dresses have the same singularity structure. The dresses on the left were jointly parameterized using our approach, while the pairs on the right were computed on one dress and transferred to the second. Compared to the joint parameterization, transferring the child's pattern to the adult dress leads to distortions in the midsection, whereas the other direction has distortions in the bust and asymmetries in the skirt.

Design Space Coverage. Next, we analyze our tool's coverage of the design space, based on the design axes that we have identified.

Surface Texture. Textures are illustrated in Figures 1 and 11. As can be seen in both, texture does not need to be uniformly applied over the entire model but instead can be applied to any region that aligns with patch borders. This provides full control over texture since surface layout features can be used to influence border placement during meshing. Interaction of texture with other design axes can be complex. For example, textures and shaping can co-exist in the same patch and are automatically handled by our pattern generation algorithm. In Figure 11(c), the diamonds at the bottom are much wider than the diamonds at the top, due to decreases necessary to change the radius of the skirt from bottom to waist.

Curvature Shaping. Shaping plays an aesthetic as well as a functional role. Figure 11 illustrates user control over how decreases are placed for different aesthetic results. Skirt (a) is fully symmetric, while skirts (b) and (c) have flat backs, with shaping only allowed on the front, and skirt (d) only has shaping at the sides with a flat front and back. Skirts (b–d) have decreases aligned and leaning toward a patch edge, creating an apparent seam.

Surface Layout. As already described, surface layout features impact the placement and transitions between textures. Figure 11 shows another fundamental, though more subtle, impact on curvature shaping—the user prescribed feature lines to control the placement of apparent seams, as can be seen when comparing skirts (b) and (c). Symmetry is another important layout feature. Once this is specified, the system will automatically ensure symmetry in the placement of increases and decreases, seams, and even singularities on compositing guidelines. For example, in Figure 3, it would be hard to place singularities symmetrically without automated support.

Seaming. Key functional aspects of seaming are specified during composition, but these interact through decisions about the surface layout of the mesh; seaming is often necessary to support orientation changes. Our system helps the user navigate this space. As was discussed in Figure 6, even if seaming guidelines from composition and surface layout are not provided, a valid and minimal seam choice will be presented to the user, who can interactively control the placement by enforcing or disallowing seams on certain areas. The system updates the seam suggestions at interactive rates, to allow easy exploration while guaranteeing knittability. Figure 12 shows the seaming layouts of our machine knit examples with seams.

Orientation. As discussed in multiple examples, orientation affects the ease of knitting, shaping choices, and seaming placement. Further, local changes in orientation can lead to non-fabricable designs if not validated globally (an important reason orientation change is not supported in [Narayanan et al. 2019]). In addition to allowing orientation control during meshing, our system allows users to easily flip the orientations locally. As can be seen in Figures 3 and 13, the system will automatically suggest seams after a direction change to ensure knittability and update shaping to conform to the mesh—e.g., use short rows instead of increases/decreases on the sleeves to match the circumference change from shoulder to wrist.

Composition. Composition guidelines allow designers to create large pattern variation from the same input mesh. For example, in Figure 14, the design on the left uses a Norwegian drop shoulder and is seamed at the arms. The design on the right is a seamless yoke sweater, which is done with merges at the armpits followed by evenly distributed decreases up to the neckline. A similar composition variation is shown on the dresses (a) and (b–d) in Figure 1. Composition guidelines are particularly useful when knitting complex shapes, as discussed in Figure 9. We further illustrate how they can be used to structure irregular shapes like the bunny (see Figure 15). By specifying how we wish to knit the ears and tail, our system discovers appropriate knitting directions to capture both the compositional structure and the complex curvature.

Variable Shape. As discussed in Section 2, resizing is an important and common aspect of knit pattern design and use. Resizing is challenging because it requires changing both the stitch counts of shaping operations, such as the number of short rows, increases, and decreases, as well as re-applying any textures to the new stitch layout. Furthermore, resizing typically requires variations on the geometry itself. A dress made to fit a child is not simply a rescaled adult dress.

Our method allows users to create a mesh that is jointly optimized over multiple parameter values of a shape, which allows users to specify knitting guidelines on a single template and have them be directly applied to different shape variations. For example, in Figure 1, the adult and child dresses are variations of the same drop shoulder pattern with identical textures but different relationships between arm length, skirt length, and torso height. Figure 16 further illustrates how our system allows designers to create customizable templates for knitting, by illustrating three fabricated variations of a hat. Both of these examples were enabled by our consistent quad-meshing method, shown in Figure 10.

Interactive Exploration. All models took about 4–13 minutes to design: the quad meshing step took about 1–6 minutes, the labeling about 1–3, and pattern generation with sizing optimization in 2–4 minutes (except the bunny, which took 15 minutes to optimize sizing). To establish the effectiveness of our interactive editing capabilities for design space iteration, we asked Narayanan to recreate some variations of the dresses in Figure 1 using [Narayanan et al. 2019]. In our system, we were able to create an initial design in 5 minutes, and create the variants (c) and (d) in 2 minutes each, most of which is spent in pattern generation. Variant (a) took 8 minutes as it required composition changes. Narayanan estimated that it would take between 15 and 40 minutes for each texture variation, depending on how carefully textures were applied, and between 45 and 60 minutes to change shaping between short rows and increases and decreases. Their system would not be able to handle direction changes or re-sizing without complete re-design. This shows how our approach and solver assisted editing enables exploration of design alternatives on the scale of minutes rather than hours.

Pilot Study. We validated the usability of our system by conducting a pilot user study with three participants having experience in knitting or garment design but not in geometry processing. In the study, we first gave a tutorial on how to use our system and then asked the participants to reproduce a textured variant the duck design shown in Figure 3(e), and also to create their own dress design using the model from Figure 1. All participants were able to determine the correct composition rules to recreate the duck within 9 minutes on average, and were also able to design a knittable dress within the half-hour provided them. As shown in Figure 17, all three dress designs have a different composition structure. While the users had no understanding of quad-meshing singularities they managed to achieve the desired structure using the intuitive composition guidelines in our tool. These dresses further illustrate the design freedom in textures, shaping, seaming, and surface layout. The details of the study can be found in the supplemental materials.

Additional Implementation Details. With the user-designed cross field as input, we use the libigl [Jacobson et al. 2019] implementation of MIQ to generate the global parameterization; users

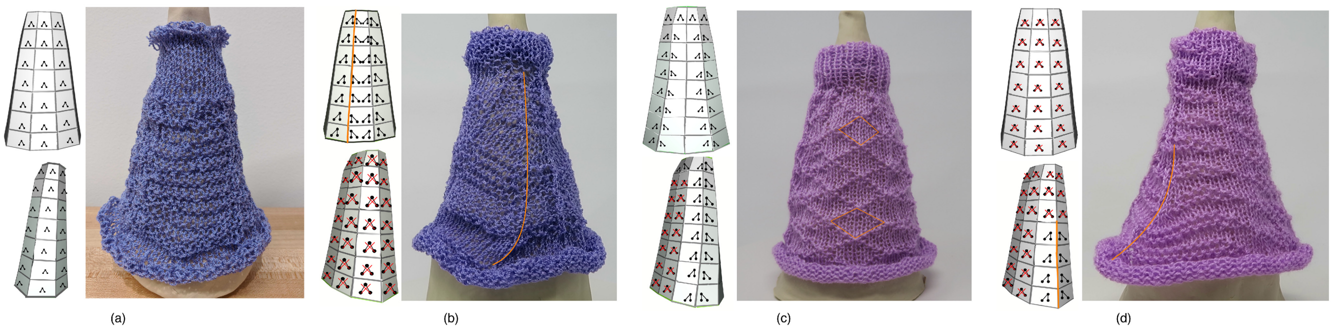


Fig. 11. Meshing, labeling, and fabricated results of four skirts generated from the same input mesh, with different coarse meshing and labeling. Orange lines on (b) and (d) highlight the apparent seams caused by concentrating shaping, and on (c) highlight the effect of shaping to narrow the texture toward the top of the skirt. Red "X"s over a shaping label indicate that no shaping is allowed in that patch, a symmetric decal indicates distributed decreases, and an angled decal indicates leaning decreases aligned to the edge the decal leans toward.

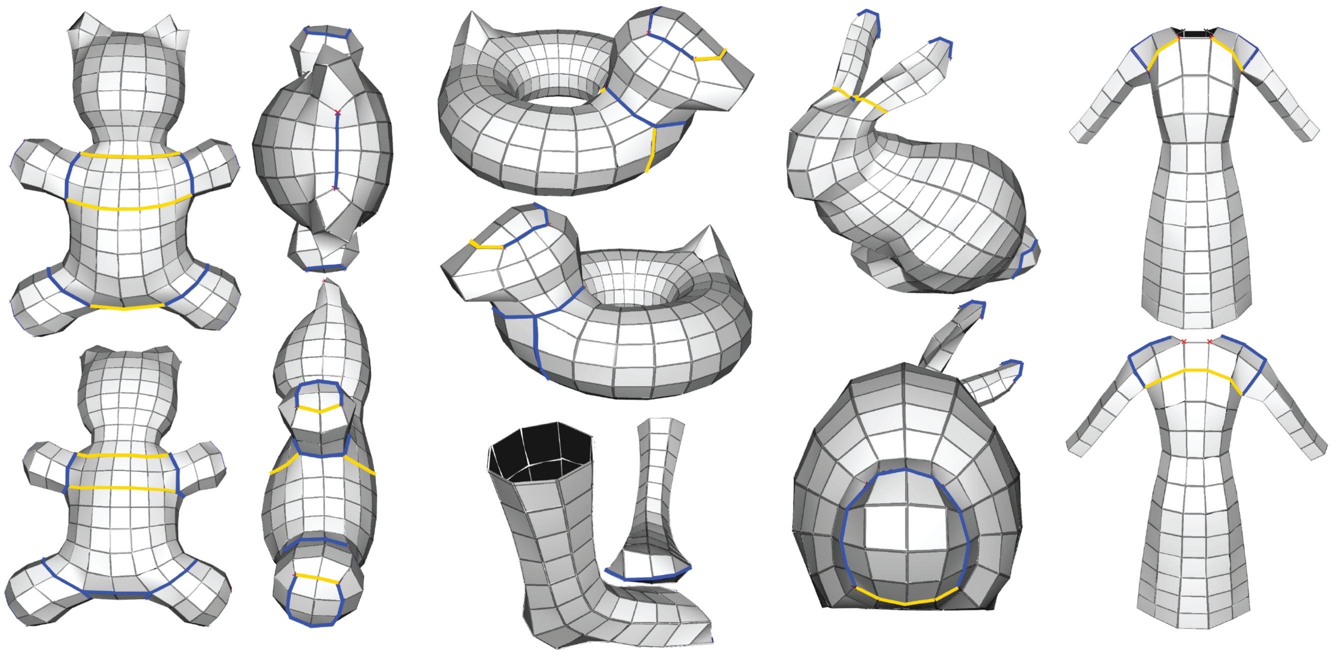


Fig. 12. Seaming layouts of all machine knit examples. Blue lines indicate seams in the original design, whereas yellow seams were added manually to account for missing functionality in our scheduler implementation. The skirts, hats, and seamless dress were omitted because they do not have any seams. The angled and child dress are ommitted because they use the same template as the drop shoulder dress (far right). The human scale sweater also uses this seaming layout.

can adjust a parameter for quad size to obtain the desired level of coarseness. To compute the cross-field given composition rule singularities, we use the implementation of the trivial connections from Directional [Vaxman et al. 2020]. LibQEx [Ebke et al. 2013] is used to extract the quad mesh. Z3 solver [de Moura and Bjørner 2008] is used for SMT equations. We implemented 20 textures from Kooler [2012] and applied different combinations to most of the models we fabricated to illustrate this capability. For machine knitting, we use the scheduler code provided by Narayanan et al. [2018; 2019] to generate instructions for the knitting machine. Hand knitting instructions were generated using custom code. All examples are knitted either by hand or by a 7-gauge SHIMA SEIKI SWG091N2 knitting machine. Models were hand-stitched together along seam lines after knitting, and the toy models were stuffed with batting.

9 LIMITATIONS AND FUTURE WORK

Our system invites several avenues for future work.

Our quad meshing pipeline has limitations, some of which are long-standing problems in meshing. Global parameterization will not be interactive if the resolution of the input mesh is fine, or if too many constraints are added. In order for our composition rules to prevent cycles, it is important that the singularities are connected by mesh edges. This is not explicitly guaranteed

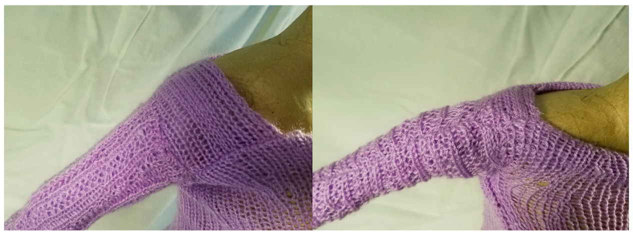


Fig. 13. Close up of the sleeves from Figure 1 dresses (b) and (c). In (b), columns are aligned down the sleeve, while in (c) they wrap around.

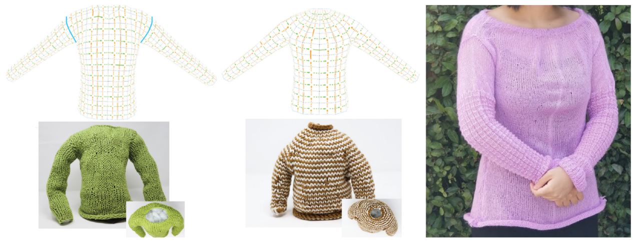


Fig. 14. Three sweater patterns from the same input model but meshed and labeled differently. The left sweater has seamed sleeves and short rows on the neck, while the one in the middle was completely knitted in the round using increases and decreases. These two sweaters were knitted by hand, showing how our method can be used for both machine- and hand-knitting. The rightmost sweater has the same composition as the leftmost, machine knit to human scale with textures added on the arms.

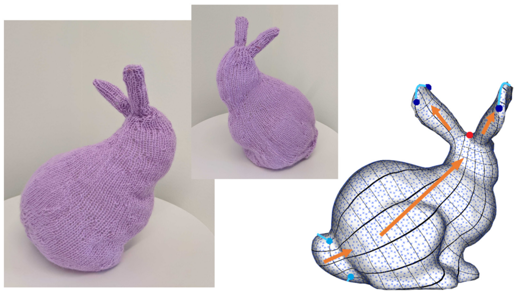


Fig. 15. The Stanford bunny illustrates composition guidelines. It is meshed by placing line seams (C1) on the tips of the ears, a split from the head to two ears (E1), and a flap on the tail (C2).

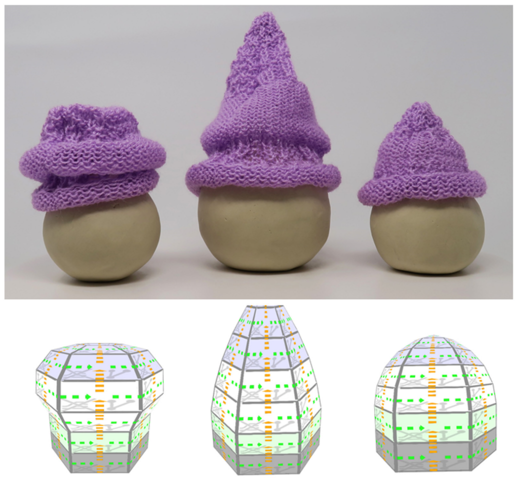


Fig. 16. Customizable knit patterns for hats created with our systems.

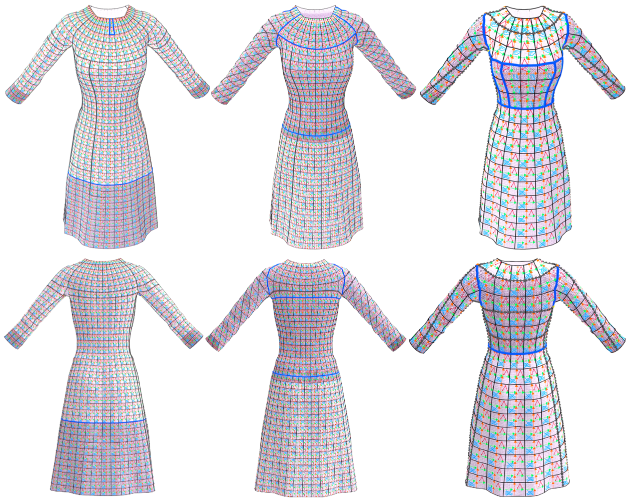
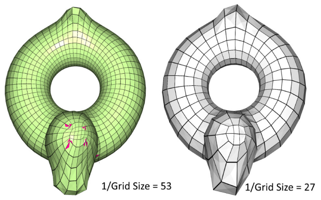


Fig. 17. Variations of the dress created by users in the pilot study (top row is front view, bottom row back view), illustrating the usability and design freedom in the system. Users were able to control composition using only prior experience on knitting or garment design but no understanding of geometry processing, quad meshing or singularities.

by our meshing algorithm and can fail, for example, if two symmetric composition rules are only slightly offset from each other. However, in most cases, our helix checking visualization and the tunable grid size parameter combined can avoid the helices. For example, when a participant of the user study created the duck design, they initially chose a grid size that led to a helix, but the researchers were able to help the participant tune the grid size a bit to create a helix-free design (shown in the inset figure: left contains a helix and the magenta triangles indicate where the helix could start; right is helix-free).



Our implementation of patch generation is very simple and does not capture the full richness of possible surface textures such as cables, lace, or colorwork. We also do not take into account the physical properties of surface texture on the patches themselves. However, our system is designed to be built upon using our patch formalization in order to support these capabilities, so recent work focusing on surface textures such as Hofmann et al. [2019], Leaf et al. [2018], or Karmon et al. [2018] could be used to generate patches within our framework.

To generate machine instructions, we used the open source scheduler implementation from Narayanan et al. [2019], which does not support some of the more complex composition rules that we do such as flaps to close tubes, even though the scheduler as described in that article does. Reimplementing that functionality was beyond the scope of this work, so we worked around the limitation by introducing extra seams to partition our designs into shapes that the scheduler will accept. We manually added these seams using our existing seaming tool by simply selecting and clicking one edge along each sheet-tube boundary (a transition between a sheet a tube is always bounded by singular vertices, and so can be seamed without adding), but this could be easily automated by labeling each coarse row as tube-like or sheet-like (is it a cycle), and adding a seam between any rows that alternate from sheets to tubes. Fully automatic machine instruction generation would require re-implementing the missing functionality. The yellow seams in Figure 12 illustrate where we added these seams.

Finally, it would be interesting to incorporate physical simulation into the design loop. In addition to the internal forces explored by works like [Kaldor et al. 2008], [Leaf et al. 2018], and [Karmon et al. 2018], the form of a knit object is strongly influenced by the physical context it will be used in, such as stuffing or draping over a person. For instance, our duck example was knit with short rows to achieve the torus body, but could have been knit as a simple straight tube and relied on stuffing for the shaping. Simulation of both internal and external forces could help designers visualize the final result of their design decisions before fabricating. Simulation of the machine knitting process will also be important to address the problem of machine tuning. The definition of machine knittability we use does not guarantee that the program generated will not fail on a real machine due to the interaction of machine tuning parameters (yarn tension and stitch size) and material properties (yarn thickness, friction, etc.) To the best of our knowledge, no existing work tackles this aspect of automatic knitting machine programming.

10 CONCLUSION

Our system makes the design of machine-and hand-knittable objects accessible to a lot more people. First, it lets users easily and quickly explore interrelated design axes while guaranteeing knittability and pattern production. Furthermore, because the system takes a parametric 3D model as input, it generates template patterns customizable by users unfamiliar with intricacies of knitting. As a result, machine knitting, like 3D models, can become customizable, modifiable, and universally accessible.

REFERENCES

Omri Azencot, Etienne Corman, Mirela Ben-Chen, and Maks Ovsjanikov. 2017. Consistent functional cross field design for mesh quadrangulation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

Hugh Beyer and Karen Holtzblatt. 1999. Contextual design. *Interactions* 6, 1 (1999), 32–42.

David Bommes, Timm Lempfer, and Leif Kobbelt. 2011a. Global structure optimization of quadrilateral meshes. In *Proceedings of the Computer Graphics Forum*, Vol. 30. Wiley Online Library, 375–384.

David Bommes, Timm Lempfer, and Leif Kobbelt. 2011b. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum* 30, 2 (April 2011), 375–384. DOI: https://doi.org/10.1111/j.1467-8659.2011.01868.x

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013. Quad-Mesh Generation and Processing: A Survey: Quad-Mesh Generation and Processing. *Computer Graphics Forum* 32, 6 (Sept. 2013), 51–76. DOI: https://doi.org/10.1111/cgf.12014

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer quadrangulation. *ACM Transactions on Graphics* 28, 3 (July 2009), 1. DOI: https://doi.org/10.1145/1531326.1531383

J. C. Briar. 2019. Stitch Maps. Retrieved 2021 from https://stitch-maps.com/.

Ann Budd. 2002. *The Knitter's Handy Book of Patterns: Basic Designs in Multiple Sizes & Gauges*. Interweave Press, Loveland, CO.

Marcel Campen and Leif Kobbelt. 2014. Dual strip weaving: Interactive design of quad layouts using elastica strips. *ACM Transactions on Graphics* 33, 6 (Nov. 2014), 1–10. DOI: https://doi.org/10.1145/2661229.2661236

Keenan Crane, Mathieu Desbrun, and Peter Schröder. 2010. Trivial Connections on Discrete Surfaces. *Computer Graphics Forum* 29, 5 (Sept. 2010), 1525–1533. DOI: https://doi.org/10.1111/j.1467-8659.2010.01761.x

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and Jakob Rehof (Eds.), Springer Berlin Heidelberg, Berlin, 337–340.

Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Integrable polyvector fields. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–12.

Hans-Christian Ebke, David Bommes, Marcel Campen, and Leif Kobbelt. 2013. QEx: Robust quad mesh extraction. *ACM Transactions on Graphics* 32, 6 (Nov. 2013), 1–10. DOI: https://doi.org/10.1145/2508363.2508372

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

Megan Hofmann, Lea Albaugh, Ticha Sethapakadi, Jessica Hodgins, Scott E. Hudson, James McCann, and Jennifer Mankoff. 2019. KnitPicking Textures: Programming and Modifying Complex Knitted Textures for Machine and Hand Knitting. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology - UIST'19*. ACM Press, New Orleans, LA, 5–16. DOI: https://doi.org/10.1145/3332165.3347886

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008. Knitting a 3D Model. *Computer Graphics Forum* 27, 7 (Oct. 2008), 1737–1743. DOI: https://doi.org/10.1111/j.1467-8659.2008.01318.x

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics* 30, 4 (2011), 78.

Alec Jacobson, Daniele Panozzo, et al. 2019. libigl: A simple C++ geometry processing library. Retrieved from https://libigl.github.io/.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2008. Simulating knitted cloth at the yarn level. *ACM Transactions on Graphics* 27, 3 (Aug. 2008), 1. DOI: https://doi.org/10.1145/1360612.1360664

Ayelet Karmon, Yoav Sterman, Tom Shaked, Eyal Sheffer, and Shoval Nir. 2018. KNITIT: A computational tool for design, simulation, and fabrication of multiple structured knits. In *Proceedings of the 2nd ACM Symposium on Computational Fabrication* (Cambridge, Massachusetts). Association for Computing Machinery, New York, NY, Article 4, 10 pages. DOI: https://doi.org/10.1145/3213512.3213516

Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. 2019. Knitting Skeletons: A Computer-Aided Design Tool for Shaping and Patterning of Knitted Garments. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology - UIST'19*. ACM Press, New Orleans, LA, 53–65. DOI: https://doi.org/10.1145/3332165.3347879

Donna Kooler. 2012. *Donna Kooler's encyclopedia of knitting* (updated and rev ed.). Leisure Arts, Little Rock, Ark. OCLC: ocn767630287.

Jonathan Leaf, Rundong Wu, Eston Schweickart, Doug L. James, and Steve Marschner. 2018. Interactive design of periodic yarn-level cloth patterns. *ACM Transactions on Graphics* 37, 6 (Dec. 2018), 1–15. DOI: https://doi.org/10.1145/3272127.3275105

James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jen Mankoff, and Jessica Hodgins. 2016. A compiler for 3D machine knitting. *ACM Transactions on Graphics* 35, 4 (July 2016), 1–11. DOI: https://doi.org/10.1145/2897824.2925940

Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James Mccann. 2018. Automatic Machine Knitting of 3D Meshes. *ACM Transactions on Graphics* 37, 3 (Aug. 2018), 1–15. DOI: https://doi.org/10.1145/3186265

Vidya Narayanan, Kui Wu, Cem Yuksel, and James McCann. 2019. Visual knitting machine programming. *ACM Transactions on Graphics* 38, 4 (July 2019), 1–13. DOI: https://doi.org/10.1145/3306346.3322995

Daniele Panozzo, Enrico Puppo, Marco Tarini, and Olga Sorkine-Hornung. 2014. Frame fields: Anisotropic and non-orthogonal cross fields additional material. In *Proceedings of the ACM TRANSACTIONS ON GRAPHICS (PROCEEDINGS OF ACM SIGGRAPH*.

Mariana Popescu, Matthias Rippmann, Tom Van Mele, and Philippe Block. 2017. Automated generation of knit patterns for non-developable surfaces. In *Proceedings of the Humanizing Digital Reality - Proceedings of the Design Modelling Symposium 2017*, K. De Rycke et al. (Ed.), Springer, Paris, 271–284. DOI: https://doi.org/10.1007/978-981-10-6611-5_24

Ravelry. 2019. Ravelry. Retrieved 2021 from https://ravelry.com/.

Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. 2008. N-symmetry direction field design. *ACM Transactions on Graphics* 27, 2 (April 2008), 1–13. DOI: https://doi.org/10.1145/1356682.1356683

Adriana Schulz, Ariel Shamir, Ilya Baran, David I. W. Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. 2017a. Retrieval on parametric shape collections. *ACM Transactions on Graphics* 36, 1, Article 11 (Jan. 2017), 14 pages. DOI: https://doi.org/10.1145/2983618

Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. 2014. Design and fabrication by example. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–11.

Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. 2017b. Interactive design space exploration and optimization for CAD models. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–14.

SHIMA SEIKI. 2019. SDS-ONE APEX series. Shima Seiki. Retrieved 2021 from https://www.shimaseiki.com/product/design/.

Maria Shugrina, Ariel Shamir, and Wojciech Matusik. 2015. Fab forms: Customizable objects for fabrication with validity and geometry caching. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–12.

David J. Spencer. 2001. *Knitting Technology: A Comprehensive Handbook and Practical Guide*. Technomic, Lancaster, Pennsylvania. 386 pages.

STOLL. 2019. M1PLUS®. Stoll. Retrieved 2021 from https://www.stoll.com/en/software/m1plusr/.

Jenny Underwood. 2009. *The Design of 3D Shape Knitted Preforms*. Ph.D. Dissertation. RMIT University.

Amir Vaxman et al. 2020. Directional: A library for Directional Field Synthesis, Design, and Processing. DOI: https://doi.org/10.5281/zenodo.3338174

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* 35, 2 (May 2016), 545–572. DOI: https://doi.org/10.1111/cgf.12864

Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. 2018. Stitch meshing. *ACM Transactions on Graphics* 37, 4 (July 2018), 1–14. DOI: https://doi.org/10.1145/3197517.3201360

Kui Wu, Hannah Swan, and Cem Yuksel. 2019. Knittable Stitch Meshes. *ACM Transactions on Graphics* 38, 1 (Jan. 2019), 1–13. DOI: https://doi.org/10.1145/3292481

Evan Yares. 2013. The failed promise of parametric CAD part 1: From the beginning. Retrieved from https://www.3dcadworld.com/the-failed-promise-of-parametric-cad/. (Accessed on 09/06/2019).

Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2012. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Transactions on Graphics* 31, 4 (July 2012), 1–12. DOI: https://doi.org/10.1145/2185520.2185533

Received June 2020; revised September 2021; accepted September 2021