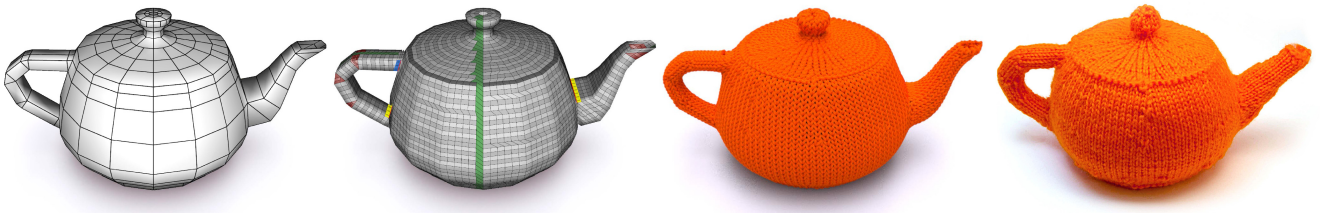
Knittable Stitch Meshes

KUI WU, HANNAH SWAN, and CEM YUKSEL, University of Utah



(a) Input mesh (b) Our Knittable Stitch mesh (c) Simulated model (d) Knitted model

Fig. 1. **Stages of our knittable garment modeling system:** (a) We begin our interactive modeling process with an input polygonal mesh that specifies the global shape of the model. (b) Using this polygonal mesh, we produce a high-resolution stitch mesh, including shift paths (green faces) that form knittable spiral structures, splitting (yellow faces) and joining (blue faces) mismatched faces that connect them without seams, and short rows (red faces). Afterward, we can either (c) generate the yarn curves from the stitch mesh and use a physically based relaxation process to produce the final yarn-level shape for rendering or (d) knit the model using the knitting instructions generated from our knittable stitch mesh.

We introduce *knittable stitch meshes* for modeling complex 3D knit structures that can be fabricated via knitting. We extend the concept of stitch mesh modeling, which provides a powerful 3D design interface for knit structures but lacks the ability to produce actually knittable models. Knittable stitch meshes ensure that the final model can be knitted. Moreover, they include novel representations for handling important shaping techniques that allow modeling more complex knit structures than prior methods. In particular, we introduce *shift paths* that connect the yarn for neighboring rows, general solutions for properly connecting pieces of knit fabric with mismatched knitting directions without introducing seams, and a new structure for representing *short rows*, a shaping technique for knitting that is crucial for creating various 3D forms, within the stitch mesh modeling framework. Our new 3D modeling interface allows for designing knittable structures with complex surface shapes and topologies, and our knittable stitch mesh structure contains all information needed for fabricating these shapes via knitting. Furthermore, we present a scheduling algorithm for providing step-by-step hand knitting instructions to a knitter, so that anyone who knows how to knit can reproduce the complex models that can be designed using our approach. We show a variety of 3D knit shapes and garment examples designed and knitted using our system.

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

Additional Key Words and Phrases: Knitting, fabrication, stitch meshes

This work was supported in part by NSF grant #1538593. Kui Wu is supported in part by University of Utah Graduate Research Fellowship.

Authors' addresses: K. Wu, H. Swan, and C. Yuksel, University of Utah, School of Computing, Salt Lake City, Utah, 84112, USA; emails: {kwu, hswan}@cs.utah.edu, cem@cemyuksel.com.

0730-0301/2019/01-ART10 $15.00
https://doi.org/10.1145/3292481

ACM Reference format:
Kui Wu, Hannah Swan, and Cem Yuksel. 2019. Knittable Stitch Meshes. *ACM Trans. Graph.* 38, 1, Article 10 (January 2019), 13 pages.
https://doi.org/10.1145/3292481

1 INTRODUCTION

In computer graphics, yarn-level simulations of cloth are known to produce compelling animations (Cirio et al. 2015; Jiang et al. 2017; Kaldor et al. 2008, 2010). This is particularly important for knitted cloth, as it often exhibits complex yarn-level deformations. The challenging problem of designing yarn-level knitted cloth models is solved by the stitch mesh modeling framework (Yuksel et al. 2012). Stitch meshes provide a full 3D modeling interface for knits with a collection of high-level and low-level modeling operations that allow designing arbitrary knitting patterns, including complex constructs like knitted cables. In fact, the stitch mesh modeling framework, with its ability to model arbitrary 3D forms, is superior to the modeling interfaces used in the textile industry.

Yet, stitch meshes do not produce knittable models (except for flat swatches). This is partially because each row of stitches on a stitch mesh forms a closed yarn piece with no endpoints, and knitting direction inconsistencies of neighboring faces are ignored, resulting in models that can be used for yarn-level simulation but cannot be produced via knitting operations. Furthermore, the original stitch mesh representation does not support short rows–a crucial 3D shaping technique used heavily with most knit cloth. This severely limits the 3D forms that can be reliably modeled (and then simulated) using stitch meshes and precludes using the stitch mesh modeling framework for fabrication purposes.

In this article, we introduce *knittable stitch meshes* that can guarantee that the designed final model is actually knittable. The technical contributions in this article include the following:

- *Shift Paths:* a novel concept that is extremely simple to implement for converting nonknittable tubular sections of a stitch mesh into knittable helix form

- *Knittable Mismatched Directions:* four different (including two novel) types of mismatched knitting directions that are needed for designing complex 3D forms and the corresponding automated modifications to the stitch mesh structure for ensuring that they are knittable
- *Short Rows:* a novel structure required for representing short rows within the stitch mesh modeling framework
- An algorithm for generating step-by-step knitting instructions from a given knittable stitch mesh

Thus, we extend the concept of stitch mesh modeling for designing more complex 3D models and for producing knittable structures that can be fabricated via knitting. As long as the input mesh can be completely labeled (using the labeling process of the stitch mesh modeling framework (Yuksel et al. 2012)) and thus can be converted to a stitch mesh, we guarantee that it can be converted to a knittable stitch mesh using our approach. To verify that our results are knittable, we also developed a graphics interface for aiding a knitter by providing step-by-step instructions. We present topologically complex models knitted by following the knitting instructions using this interface. An example model generated using our modeling framework is shown in Figure 1.

Our main goal in this article has been solving the problems of the stitch mesh modeling framework for representing realistic knit structures and actually knittable models. We do not, however, consider the physical limitations of knitting machines. Therefore, while we guarantee that the resulting models are knittable, we do not test whether the required sequence of knitting instructions can be performed by a particular knitting machine.

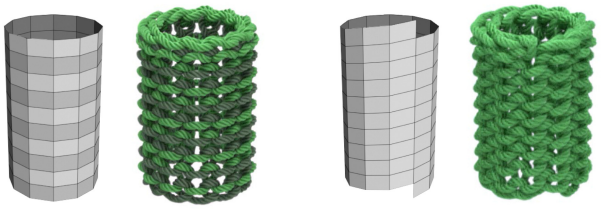
2 BACKGROUND

Cut and sewn garments need to be converted to several, separate, planar shapes, which can then be sewn together to create a more complex shape, a process that has been modeled in several ways (Carignan et al. 1992; Decaudin et al. 2006; Luo and Yuen 2005; Mori and Igarashi 2007; Robson et al. 2011; Turquin et al. 2007; Umetani et al. 2011; Volino and Magnenat-Thalmann 2005, 2012). Knit structures, however, are constructed by continuously looping yarn through existing loops, creating stitches. There are only two basic types of stitches: a *knit* stitch, made by pulling a new loop from the back of an existing loop, and a *purl* stitch, made by pulling from the front. Two or more loops can be combined together with a new loop (stitch *decrease*) or new multiple loops created through one existing loop (stitch *increase*). These techniques add 3D shape by changing the length along the rows (i.e., the *course* direction) of the material. *Short rows* (additional partial rows in the middle of other rows), meanwhile, add length along the knitting (*wale*) direction. Therefore, shaping can be added to the structure as the material is created, resulting in a single piece of material that can take various arbitrary 3D shapes.

Much of the work involving cloth in computer graphics has been aimed toward the simulation of woven cloth (Baraff and Witkin 1998; Bridson et al. 2002; Chu 2005; Goldenthal et al. 2007; Grinspun et al. 2003; Volino et al. 2009), but there has been some work devoted to modeling, simulating, and predicting knit structures (Cirio et al. 2015; Igarashi and Mitani 2010; Jiang et al. 2017; Kaldor et al. 2008, 2010; Nocent et al. 2001). Some researchers have looked to knitting machines to help define yarn loop behavior but only support a small subset of possible stitches (Duhovic and Bhattacharyya 2006; Eberhardt et al. 2000; Meißner and Eberhardt 1998). The textile community has also attempted to model knit structures using spline curves (Choi and Lo 2003, 2006; Demiroz and Dias 2000; Goktepe and Harlock 2002; Renkens and Kyosev 2011) or by simplifying a pattern using representative cells holding swatches of the knit structure (Kurbak 2009; Kurbak and Alpyildiz 2008; Kurbak and Soydan 2009). Generating yarn geometry can be similar to texture synthesis (Heeger and Bergen 1995; Kwatra et al. 2003), and some methods have attempted to use these algorithms for adding fine-level repeating geometry to high-level shapes (Cabral et al. 2009; Lai et al. 2010; Zhou et al. 2006).

Others have attempted to construct a knitting pattern from some surface input. Igarashi et al. (2008a, 2008b) presented an interesting system that semiautomatically creates a knitting pattern from a 3D model by covering the surface with a winding strip and finding areas where increases or decreases are needed. However, it requires manual segmentation of a surface and does not include all shaping techniques, and the resulting physical structures can be different from what is expected. McCann et al. (2016) recognized the lack of tools available to instruct knitting machines to create intricate and seamless 3D surfaces. They presented a compiler that translated simple shape primitives, such as tubes and sheets, into low-level machine instructions and proposed a system of scheduling construction of their patterns on a knitting machine. Their system is a promising start for expanding the use of knitting by giving more control over knitting machines and making the design process easier. Knitting manufacturers have commercial tools for developing and constructing a pattern (Shima Seiki 2011; Stoll 2011). However, achieving a desired shape for common items that differ from their few templates can be difficult. Others focus only on flat panels of texture and color, only altering the appearance but not the shape of a pattern (Soft Byte Ltd. 1999).

Yuksel et al. (2012) developed over 30 types of different stitch mesh faces to abstract the yarn-level geometry, which allows the designing of yarn-level knit cloth models without explicitly modeling yarn curves. Each face of a stitch mesh corresponds to a single stitch including knit, purl, yarn over, increase, and decrease. Edges are labeled as course or wale edges depending on whether they are aligned with the course or wale knitting direction, respectively. In general, stitch-mesh faces do not have to be quadrilaterals, but each face must have exactly two wale edges for yarn entry and exit. A stitch-mesh face with multiple-course edges along the top or bottom of the face represents a single stitch that is connected to multiple stitches on another row—a stitch increase or decrease. Loose ends on wale edges are connected to another loose end on a nearby wale edge. Stitch-mesh faces on the first row represent cast-on stitches, and those on the last row represent bind-off stitches. Stitch meshes provide a powerful interface for designing 3D knit structures. However, while the faces ensure that the structure is topologically valid in that there will be no unraveling, they do not guarantee knittability. This is partially because faces of a stitch mesh on a tubular section of the model correspond to a yarn piece (forming the represented stitches) with no endpoints. Furthermore, any inconsistencies in knitting directions of neighboring faces, which are unavoidable except for relatively simple models,



(a) Separate Rows (b) Helix Progression

Fig. 2. An example stitch mesh with (a) separate rows that lead to separate yarn pieces per row with no endpoints and (b) helix progression that connects the rows and can be knit by a single yarn piece with endpoints at the first and the last stitches of the helix.

are practically ignored, forming tiny yarn pieces with no endpoints that cannot be produced by knitting operations. Our goal in this article is to extend the types of knit structures that can be represented using stitch meshes and make these structures knittable, so this approach could be useful for fabrication purposes as well.

More recently, Popescu et al. (2017) presented a method that automatically generates knitting patterns for nondevelopable surfaces, and Narayanan et al. (2018) and Wu et al. (2018) introduced methods for converting 3D models into knit structures. The method of Narayanan et al. (2018) relies on a user-defined flow field to produce machine-knittable models. This approach allows fabricating complex 3D shapes via machine knitting, but it lacks a design interface. The method of Wu et al. (2018) automatically generates stitch meshes from a given input shape, but the resulting models are not knittable.

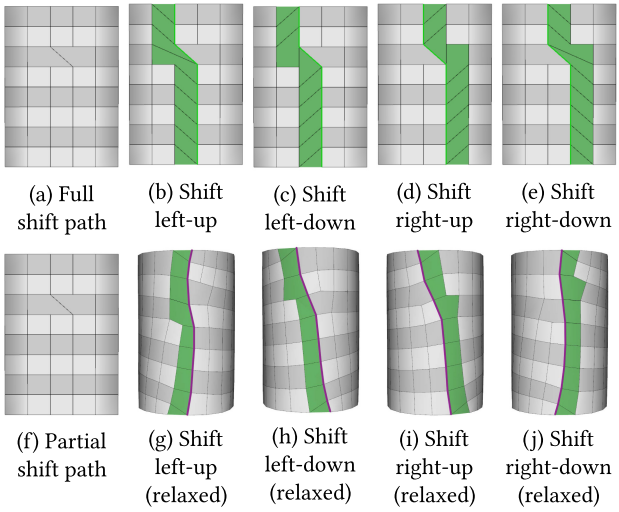
As an example application, we present a graphics interface that provides step-by-step knitting instructions to a knitter. This application is similar in spirit to prior work on assisting manual construction, such as designing assembly instructions (Agrawala et al. 2003), block assembly (Gupta et al. 2012), beadwork (Igarashi et al. 2012), and wire crafts (Iarussi et al. 2015; Miguel et al. 2016).

3 KNITTABLE STITCH MESHES

Knittable stitch meshes extend the concept of stitch meshes for designing actually knittable structures. This is achieved by removing the nonrealistic assumptions of original stitch meshes and introducing fundamental shaping concepts. In particular, our knittable stitch meshes incorporate novel concepts to the stitch mesh modeling framework in three groups: *shift paths* (Section 3.1), *knittable mismatched directions* (Section 3.2), and *short rows* (Section 3.3). These concepts are necessary for modeling general 3D knittable structures (Section 3.4) and sufficient for making sure that the final outcome is indeed knittable (Section 4).

3.1 Shift Paths

An important nonrealistic assumption of the original stitch meshes is that the yarn piece for each row is handled separately. Therefore, a row on a tubular part of a model corresponds to a closed yarn curve with no endpoints (Figure 2(a)). Instead, knit structures are formed by following a helix progression, as shown in Figure 2(b). However, this helix formation effectively converts multiple rows into a single (helix-shaped) row and introduces unnecessary



(a) Full shift path (b) Shift left-up (c) Shift left-down (d) Shift right-up (e) Shift right-down

(f) Partial shift path (g) Shift left-up (relaxed) (h) Shift left-down (relaxed) (i) Shift right-up (relaxed) (j) Shift right-down (relaxed)

Fig. 3. **Shift path and the shift operation on a tubular stitch mesh:** (a) full shift path, (b–e) four different shift operation options that can be selected by the user, (f) an alternative shift path that partially covers the tube, and (g–j) stitch meshes after mesh-based relaxation.

complexity to all stages of the modeling framework from labeling to stitch mesh generation and editing. In fact, this complexity is even avoided in real-world knitting instructions by defining each row separately. Therefore, an ideal solution must preserve the separate-row representation but form the helix structure only when needed.

We introduce the concept of *shift paths* for slightly modifying the stitch mesh structure to automatically construct a helix formation at the very end of the modeling process. A shift path merely marks a set of connected wale edges at consecutive rows (Figure 3(a)). The structure of the stitch mesh remains unaltered until the end of the modeling process, so all prior stitch mesh modeling methods can be used without modification. Each wale edge along the shift path indicates where the row begins and ends and where the last stitch should be connected to the first stitch of the next row.

It is easy to automatically pick a set of wale edges as a shift path where needed, but since it impacts the shape of the final model, we leave the choice of where to place the shift paths to the user. Once a shift path is specified by the user, the stitch mesh can be automatically converted to the helix progression via a *shift operation* that alters the course edges on one side of the shift path by sliding them along the shift path (similar to prior helix creation methods (Bommes et al. 2011)). These course edges can "shift" along the shift path either up or down, effectively determining the knitting (course) direction. Therefore, a shift path can modify the stitch mesh in four different options shown in Figure 3. All four options produce valid knittable structures and the choice merely determines the knitting order. In our implementation, one of the options is automatically selected by default, but we allow the user to pick a different option.

Selecting a single wale edge is enough to identify an entire shift path.[1] If the chosen shift path begins in the middle of an increase

[1]If the stitch mesh contains triangular faces, multiple wale edge selections might be required for specifying the desired shift path.

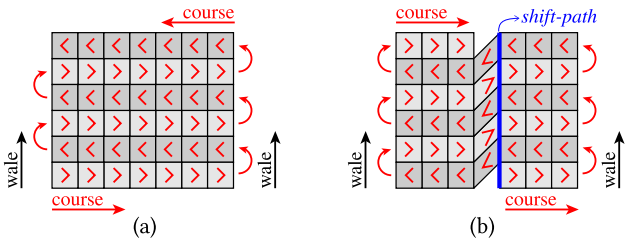


Fig. 4. **Stitch mesh rows with open ends:** (a) without a shift path and (b) with a shift path. The "<" or ">" symbol on a face indicates the course direction for the face.

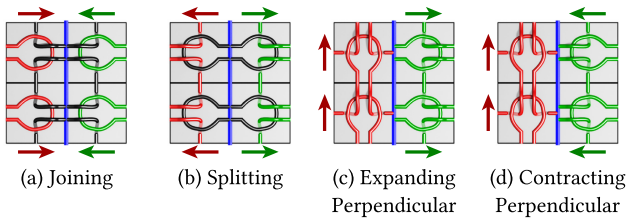


Fig. 5. **Four types of mismatched directions**, covering all possible cases that involve two stitch mesh faces with a shared edge: (a) joining, (b) splitting, (c) expanding perpendicular, and (d) contracting perpendicular. Arrows indicate the wale direction.

stitch and/or ends in the middle of a decrease stitch (as shown in Figure 3(f)), additional shift paths are needed for handling the rows that are not covered by the chosen shift path.

Shift paths can be specified automatically by repeatedly picking an arbitrary wale edge (marking an entire shift path) until all rows are covered. However, since the shift-path choice determines the knitting order and the knitting pattern along a shift path is altered (due to the shift operation), we leave this choice to the user.

With typical 2D knitting instructions, the beginning and ending of each row is implicitly defined. Shift paths are indeed a simple modification to stitch meshes, but they are essential for marking the row ends within a 3D modeling framework.

Note that shift paths are needed for tubular parts of a model. If series of rows have open ends, knitting can be done by simply moving to the next row and *turning* after reaching one end of a row (Figure 4). Shift paths can be used for such rows as well, but they are not needed to make them knittable. Thus, the original stitch meshes provide knittable structures only when all rows of a model are open and there are no tubular pieces.

3.2 Knittable Mismatched Directions

Typically, the wale and course directions on neighboring faces of a stitch mesh are aligned with each other. However, restricting that the knitting directions of *all* faces would match their neighbors would significantly limit the 3D shapes that can be modeled. Since what makes stitch mesh modeling powerful is its ability to represent complex 3D shapes, it is important to provide support for having neighboring faces with mismatched knitting directions.

We support four types of mismatched directions shown in Figure 5 that cover all possible cases that include two stitch mesh faces with a common edge. The first two types are *joining* and *splitting* mismatches (Figures 5(a) and 5(b)), where two faces on consecutive rows sharing a course edge disagree on the wale direction. We also introduce two new types of perpendicular mismatched directions (Figures 5(c) and 5(d)). These take place when two neighboring faces disagree on the type of the shared edge, which is treated as a wale edge for one and a course edge for the other. We refer to an edge that separates two faces with mismatched directions as a *mismatched edge*.

Joining Mismatched Directions. As shown in Figure 5(a), joining mismatched directions happen when the wale directions of neighboring stitch mesh faces point toward the mismatched edge. Joining mismatched directions are needed for handling certain types of singularities, such as the example in Figure 6 (a).

A common knitting technique for handling joining mismatched directions is called *three-needle bind-off*. The knitter uses two needles to hold the two knitted pieces and a third needle to knit through two stitches (one from each needle) at a time, combined with a bind-off stitch. To represent three-needle bind-off within the stitch mesh modeling framework, we extrude the mismatched edges along the inverse normal direction and form faces that are (locally) perpendicular to knit surface (Figure 6(b)). The new faces act like decrease-type stitches, each joining two loops connected to the faces on either side of the extruded edge. A bind-off stitch is used to terminate the knitting progression after joining (Figure 6(c)). This extrusion is performed automatically and it takes place at the very end of the stitch mesh modeling process, right before generating the yarn curves or knitting instructions.

Three-needle bind-off can use a separate piece of yarn for the extruded row to join two knitted pieces on either side (Figure 6(c)). Alternatively, it is possible to use the yarn of one of the two knitted pieces instead. In this case, the loop on the second row is extended, as shown in Figure 6(d). This alternative version not only avoids the extra piece of yarn but also avoids introducing an extra row of stitches used for joining the two pieces. In practice, both of these alternatives are acceptable solutions with slightly different final results, so we support both of them, leaving the choice to the user.

Splitting Mismatched Directions. Similar to joining mismatched directions, with splitting mismatched directions (Figure 5(b)), the wale directions of neighboring stitch mesh faces point away from the shared mismatched edges. Splitting mismatched directions appear in similar cases as joining mismatched directions, but with inverted wale directions (Figure 7(a)). While the actual knitting operations needed for handling splitting mismatched directions are different (i.e., three-needle bind-off cannot be used), we can handle them similarly in the context of stitch mesh modeling. As in joining mismatched directions, we extrude the mismatched edges along the inverse normal direction to introduce additional faces. However, this time we extrude two rows, as in Figure 7(b). Then, we can use an extra piece of yarn for producing cast-on stitches (Figure 7(c)). The stitches on both sides of the mismatched edges are moved to the extruded row and they are pulled through the same loops formed by the cast-on stitches. This operation effectively connects the two rows on both sides of the mismatched edges. Alternatively, one of the existing yarn pieces can be used to form the cast-on stitches (Figure 7(d)). This effectively replaces the stitches on one side with the cast-on stitches along the extruded

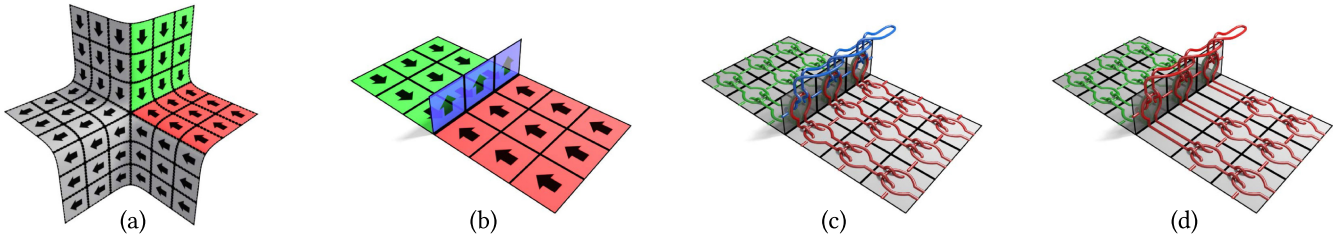


Fig. 6. **Joining mismatch directions:** (a) mismatched edges on one side of a valance 6 vertex between green and red faces, (b) the extruded row of faces (shown in blue) with arrows showing the wale direction, (c) yarn curves using an extra piece of yarn (shown in blue) along the extruded row, and (d) yarn curves using the yarn piece on one side of the mismatched edges. The arrows on the stitch mesh faces indicate the wale knitting direction.

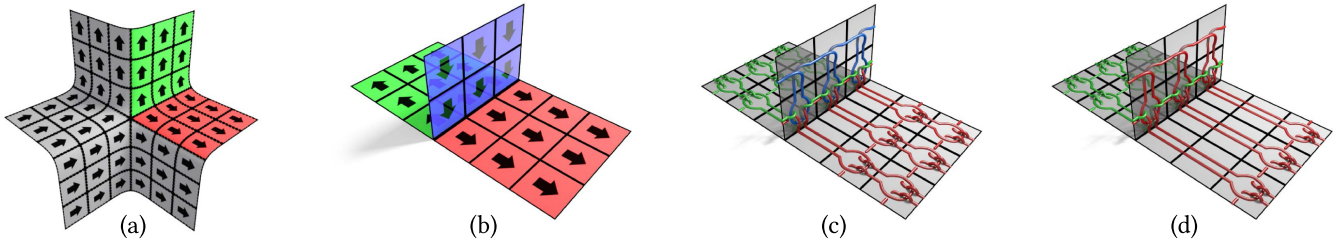


Fig. 7. **Splitting mismatch directions:** (a) mismatched edges on one side of a valance 6 vertex between green and red faces, (b) the extruded rows of faces (shown in blue) with arrows showing the wale direction, (c) yarn curves using an extra piece of yarn (shown in blue) along the extruded rows, and (d) yarn curves using the yarn piece on one side of the mismatched edges. The arrows on the stitch mesh faces indicate the wale knitting direction.

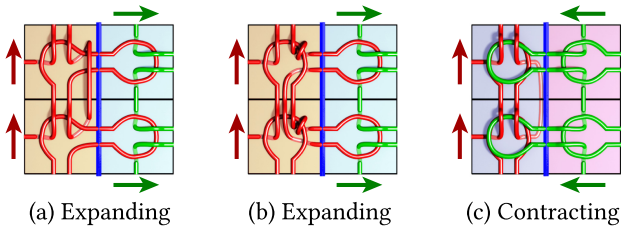


Fig. 8. Expanding and contracting perpendicular mismatched directions and the corresponding yarn curves generated from the stitch mesh. The mismatched edges are shown as blue lines. Arrows indicate the wale direction.

rows and moves the stitches on the neighboring row to the first extruded row.

Expanding Perpendicular Mismatched Directions. It is also possible to connect two neighboring stitches that disagree on the type of the common edge between them. As shown in Figure 5(c), expanding perpendicular mismatched directions happen when the wale direction on one side of the mismatched edge is pointing away from the edge. Figure 8 shows two possible ways of handling such cases. The first one (Figure 8(a)) is done by pulling perpendicular loops though previously knitted loops, and the second one (Figure 8(b)) corresponds to increase stitches that are placed on a separate needle. Nonetheless, these two stitch types are merely two example ways of handling expanding perpendicular mismatched directions, and one could imagine other stitch types that could serve the same purpose.

Note that two stitches on consecutive rows on one side of the mismatched edges are paired with slightly different yarn-level geometry. Yet, this does not require the number of stitches along the mismatched edge to be even. When there is an odd number of stitches, one end of the yarn is simply tied, forming a knot.

Contracting Perpendicular Mismatched Directions. The last possibility, shown in Figure 5(d), appears when the wale direction on one side of a mismatched edge is toward the edge. In this case, we simply handle the stitch on the other side as a decrease type of stitch with a minor modification that effectively treats the mismatched wale edge as a course edge (Figure 8(c)). In spite of the complexity of the yarn geometry, these types of mismatched directions are relatively easy to handle during knitting by simply moving stitches from one needle to another and knitting them together to form a decrease stitch.

3.3 Short Rows

Short rows are essential for shaping knit structures. Indeed, short rows are so commonly used in knitting that even some simpler design tools for knitting support them (McCann et al. 2016), but they are not supported by the original stitch meshes. A regular knit row is formed by knitting through all of the stitches on a needle from start to end (Figure 9(a)). A short row, however, uses only a subset of the stitches on a needle. This is accomplished by stopping at a certain stitch before reaching the end of the row (Figure 9(b)) and starting to knit in the opposite (course) direction (Figure 9(c)). The stitches knit in the opposite (course) direction form a short row. After a desired number of stitches are knit, another turn changes the knitting (course) direction back to its original. After the same number of stitches are knit (Figure 9(d)), two short rows are completed. At this point, knitting can continue toward the end of the row (Figures 9(e) and 9(f)) or additional pairs of short rows can be added. Thus, short rows typically appear in pairs. Short rows cause the knit surface to bend by adding partial rows. A typical

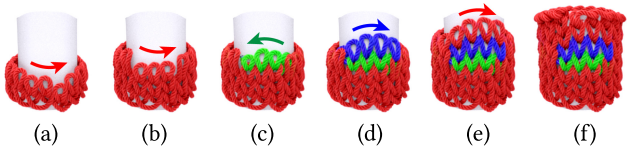


Fig. 9. **Knitting short rows:** (a) regular rows knit through all stitches on the previous row, (b) short rows begin with stopping knitting before reaching the end of a row, then (c) reversing the course direction and knitting stitches in the opposite direction; afterward the (d) knitting direction is reversed again to finish a pair of short rows, and (e, f) knitting continues in the original direction. Arrows indicate the course direction used for knitting the stitches below them.

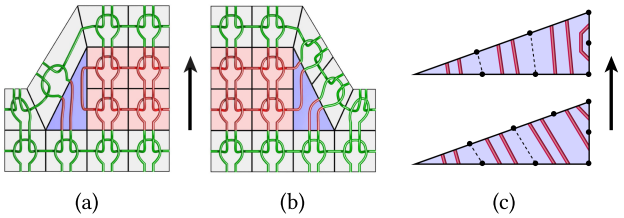


Fig. 10. **Short rows:** marked as red faces (a) beginning with a short-row face marked as the blue face and (b) ending with another short-row face. A short-row face can have four or five edges with different yarn-level connections. We also support (c) short-row faces with more than five edges that simply connect the yarn pieces of corresponding edges. Arrows indicate the wale knitting direction.

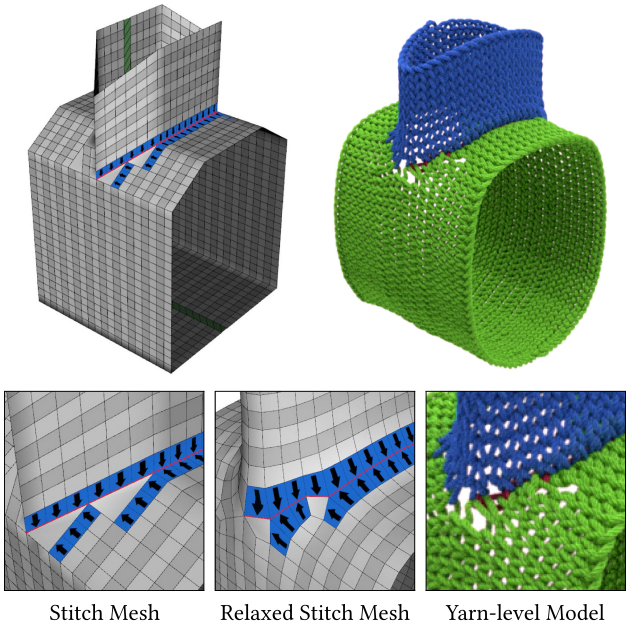


Fig. 11. An example containing short-row faces with more than five edges near joining mismatched directions, before and after mesh-based relaxation and the final yarn-level model. Arrows on the stitch mesh faces indicate the wale knitting direction.

example of short rows would be the heels of knit socks that force the straight tubular shape of a sock pattern to bend and make room for the heel.

For introducing short rows to the stitch mesh modeling framework, we introduce a new type of stitch mesh face that we call a *short-row face*. Unlike typical stitch mesh faces that have two wale edges separated by one or more course edges on either side, a short-row face places the two wale edges together on a straight line, forming a triangular shape with four (or more) edges, as shown in Figure 10(a). Thus, a short-row face makes room for a pair of short rows on one side. Therefore, it is often paired with another short-row face marking the other ends of these short rows (Figure 10(b)).

We introduce two different types of short-row faces that correspond to two different techniques for knitting short rows. The first type contains four edges, and the corresponding yarn-level model has the form in Figure 10(a). The other type of short-row face includes five edges and forms yarn-level geometries in Figure 10(b). This second type forms more stitches near the ends of the short rows, thereby avoiding large holes near short-row ends.

The fact that a short-row face spans two rows on one side causes the neighboring stitches to deform significantly. Some amount of deformation is expected near short-row ends and they are minimized after mesh-based relaxation. However, prior to mesh-based relaxation, during stitch mesh editing, the extreme deformation around short-row faces provides a poor representation of the knit model. Therefore, we allow short-row faces to have more edges than five, where the corresponding yarn pieces for the extra edges are directly connected, as shown in Figure 10(c). During mesh-based relaxation, we connect the corresponding vertices of the short-row faces with zero-length springs that effectively collapse these extra segments, as can be seen in Figure 11.

3.4 Modeling Framework

In our knittable stitch meshes, we follow the original stitch mesh modeling framework (Yuksel et al. 2012). Yet, some modifications are required to accommodate the changes needed for handling knittable stitch meshes.

These modifications begin with the labeling process, which is the first step in stitch mesh modeling that allows the user to specify the knitting directions on the input mesh faces. Labeling is typically done by starting from one open end of the model. Simply clicking on an edge along an open end automatically selects the neighboring face without a label and labels an entire row of faces. However, the input for knittable stitch meshes can contain triangles that would eventually turn into ends of short rows, so simply selecting an edge does not always provide enough information for identifying a short row. To accommodate this, we allow the user to mark triangular faces of the input mesh as short-row ends. Yet, depending on the order of labeling, short-row ends can be detected automatically, when all other faces surrounding a triangle are already labeled.

Labeling perpendicular mismatched directions also requires some minor changes to the labeling process. It is possible to automatically detect perpendicular mismatched directions when one side of the mismatched edge can be labeled without considering the mismatched directions, which is typically the case. When this is not possible, mismatched edges can be marked by the user

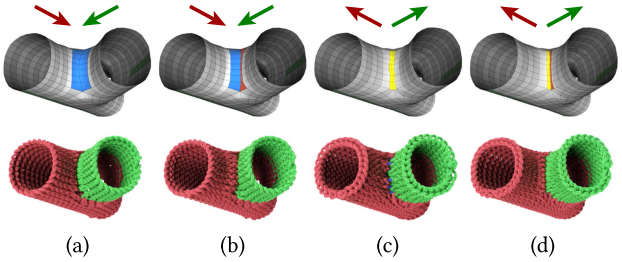


Fig. 12. **Joining and splitting mismatched directions after mesh-based relaxation and after yarn-level relaxation:** (a) joining mismatched directions using an extra (blue) yarn piece, (b) joining mismatched directions using the green yarn piece, (c) splitting mismatched directions using an extra (blue) yarn piece, and (d) splitting mismatched directions using the green yarn piece. Arrows indicate the wale knitting direction.

explicitly or the labeling process can be done one face at a time, instead of automatically labeling an entire row of faces.

Knittable stitch meshes support the same low-level and high-level stitch mesh editing operations of the original stitch meshes, so no modification is needed to these operations. However, editing edges along a shift path may invalidate the shift path, which must be monitored during stitch mesh editing.

The mesh-based relaxation step also requires some minor modifications. First of all, the stretch and shear forces are computed considering the modified mesh after the shift operation. This can be done on the fly without actually modifying the topology of the mesh, so that the user can change the location of the shift path after mesh-based relaxation (followed by another mesh-based relaxation to get the final stitch mesh shape).

Second, mismatched edges along perpendicular mismatched directions use the average rest length of wale edges and course edges, since they are treated as course edges for one of their faces and wale edges for the other.

Third, for handling short-row faces that contain more than five edges, we introduce zero-length springs, so that pairs of these short-row faces collapse after mesh-based relaxation, as shown in Figure 11.

Finally, joining and splitting mismatched directions, which are handled by extruding the mismatched edges perpendicularly, must consider the impact of the extruded rows. Since these edges are extruded right before generating yarn curves, these new faces do not exist during mesh-based relaxation. Thus, we modify the rest lengths of neighboring wale edges. If a joining mismatched direction would use an extra piece of yarn, the rest lengths of the wale edges that are connected to the mismatched course edges are extended (by a factor of 1.5 in our implementation) to make room for the extra row of stitches (Figure 12(a)). If the yarn on one side would be used for the extruded faces (Figure 12(b)), the rest lengths of the wale edges on that side are reduced (by a factor of 0.5 in our implementation). In case of splitting mismatched directions with an extra piece of yarn (Figure 12(c)), the rest lengths of the wale edges on both sides of the mismatched edges are reduced (by a factor of 0.5 in our implementation). If one of the yarn pieces is used for the cast-on stitches along the extruded row (Figure 12(d)), the rest lengths of the wale edges on the second row on that side are also reduced (by a factor of 0.5 in our implementation). Note that the purpose of the mesh-based relaxation is to provide a rough approximation of the relaxed shape and the final model is produced by yarn-level relaxation after the yarn curves are generated. Therefore, the scaling factors we use for the mesh-based relaxation do not have to be precise.

4 GENERATING KNITTING INSTRUCTIONS

Knittable stitch meshes provide a novel mechanism for representing and modeling complex 3D knit structures. However, even though a knittable stitch mesh includes all information needed for precisely describing a knit structure, determining the sequence of knitting instructions needed for fabricating the represented knit structure can be highly complicated. Therefore, we describe a scheduling algorithm that converts the information in a knittable stitch mesh into step-by-step knitting instructions. This process is also crucial for verifying that the models we produce are indeed knittable.

4.1 Dependency and Knittability

Knitting begins with *casting on*, a term that refers to placing the first stitches onto a needle. Only then can a second needle be used to pull the next stitch through an existing one, an operation that transfers the original loop from the first needle onto a new loop that now exists on the second needle. Therefore, there is a strict order dependency in knitting. A knitting project is completed and removed from the needles by *binding off*. The *bind-off* stitches transfer loops on the needle to a row neighboring stitch, closing all the open loops except for one. A yarn end is pulled through this last stitch (as well as the first stitch) to prevent unraveling.

In a knittable stitch mesh, the knitting order dependency is simple: each stitch depends on a neighboring stitch on the same row in the inverse course direction and depends on all neighboring stitches on the previous row (in the inverse wale direction). Extruded stitches along joining mismatched directions depend on stitches on both sides of the mismatched edges. Stitches on one side of expanding perpendicular mismatched directions that treat the mismatched edges as course edges depend on the stitches on the other side. Similarly, stitches on one side of contracting perpendicular mismatched directions that treat the mismatched edges as wale edges depend on the stitch on the other side.

Note that the knittable stitch mesh structure is general enough to represent unknittable models as well. Yet, given an arbitrary knittable stitch mesh, it is easy to verify that the represented model is knittable. Consider the dual graph of the knittable stitch mesh, which replaces the stitch mesh faces with graph nodes and connects the neighboring nodes with directed edges that follow one of the two knitting directions (course or wale). If this dual graph has a cycle, the model is not knittable, since a cycle would mean circular dependency. If the dual graph has no cycle, we can conclude that the model is knittable.

Nonetheless, our knittable stitch mesh modeling framework produces knittable models, so this verification is not needed in practice. This is because the initial labeling step follows the dependency order. There is only one exception that can lead to an unknittable model, caused by the flexibility we provide in handling

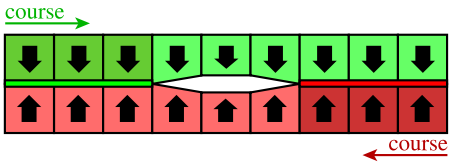


Fig. 13. **An example of the special unknittable case**, which includes two groups of joining mismatched directions. The dark faces depend on the stitches on the opposing row. One piece of yarn is used for knitting the top row and another yarn is used for knitting the bottom row. The mismatched edges are colored according to the yarn pieces that are to be used for knitting the stitches along the mismatched directions. The first group of joining mismatched directions (on the left side) uses the yarn from the top row, and the second group (on the right side) uses the yarn from the bottom part. This case leads to deadlock due to circular dependency, since the red stitches must be knit before knitting the stitches for the first group of mismatched directions and the green stitches must be knit before knitting the stitches for the second group. Slightly modifying this model by either using the same yarn for both groups of mismatched directions or using additional pieces of yarn would avoid the deadlock case.

mismatched directions. Depending on the shift paths, a model may be unknittable if it meets *both* of the two following conditions:

(1) If there are two or more *separate* groups of joining or splitting mismatched directions on the same row
(2) If extra pieces of yarn are not used for handling the mismatched directions, and one yarn is used for handling one group of mismatched directions while the *other* yarn is used for handling another group

A simple example of this special case is shown in Figure 13, and is unknittable, because the specific way that the mismatched directions are handled in this case can cause circular dependency. Yet, it is easy to avoid this special case. If there are multiple separate groups of joining or splitting mismatched directions on the same row, we can only permit using extra yarn pieces or the yarn on one side of the mismatched edges for handling *all* mismatched directions on the same row. Except for this special case, which is related to the extra flexibility we provide to the user for selecting which yarn piece should be used for handling joining/splitting mismatched directions, input meshes that can be labeled *always* produce knittable models.

4.2 Identifying Separate Yarn Pieces

Before we start generating knitting instructions, we must identify how many yarn pieces are needed for knitting the model and which yarn piece should be used for knitting which stitch. Obviously, the stitches on the same row are knit using the same yarn piece. Stitches on consecutive rows can share the same yarn piece as well, depending on the placement of the shift paths and the row types. A knittable stitch mesh can have three types of rows:

- Closed rows that form complete loops with no ends
- Open rows that begin and end on either a border (marked as a wale edge) or a perpendicular mismatched direction boundary
- Short rows that are placed between other rows

Note that one end of a pair of short rows can be on a border or a perpendicular mismatched direction boundary.

Closed rows form tubular pieces and neighboring closed rows are connected to each other via shift paths. Therefore, closed rows along the same shift path can be knit using the same yarn piece. Consecutive open rows that share wale edge borders can be knit using the same yarn piece without needing a shift path. Short rows that end on a border can be handled similarly. Short rows between closed rows are either connected to other rows via shift paths passing through them or are knit using separate yarn pieces.

We can find the number of yarn pieces needed by counting the number of separate shift paths, separate groups of open rows, short rows that are not connected to other rows, and joining/splitting mismatched directions that use extra yarn pieces. Thus, stitch mesh faces within the same group of rows that share a yarn piece are assigned the same yarn index.

Then, we determine the first stitch for each yarn piece. For a group of closed rows, the first stitch corresponds to the stitch mesh face on one side of the first edge along its shift path (the side is determined by the shift direction). The first stitch of a group of open rows that are not connected by a shift path would be a stitch on either end of the first row (the rows are ordered based on the wale direction). Pairs of short rows that are not connected to other rows can be knit starting from either end of the first row. Separate yarn pieces for joining or splitting mismatched directions can also be knit starting from either end.

ALGORITHM 1: Generate Knitting Instructions

```
Q ← list of first stitches for each yarn piece
while Q is not empty do
    s ← the next ready stitch from Q
    while true do
        Generate knitting instructions for s.
        Mark s as knitted.
        s_c ← the next stitch after s along the course direction
        if s_c exists then
            s ← s_c
        else
            s_w ← the stitch after s on the next row in wale direction
            if s_w exists and yarn of s_w = yarn of s then
                s ← s_c
            else
                Terminate the yarn piece
                break
        if s is not ready then
            Enqueue s into Q.
            break
```

4.3 Step-by-Step Knitting Instructions

Algorithm 1 shows how a knittable stitch mesh can be used for generating step-by-step knitting instructions. We begin with placing the first stitch of each yarn piece in a queue. Then, we pick a stitch that is ready to be knit from the queue. A stitch is

considered *ready* after all other stitches that it depends on are knitted. Initially, only cast-on stitches can be considered ready, since they are not connected to a stitch on a previous row. Knitting can begin with any cast-on stitch in the queue.

We use the stitch type of the stitch mesh face to produce the knitting instructions. The knitting instructions for a single stitch mesh face can be as simple as a single *knit* or *purl* instruction, or it might consist of a series of instructions (as with increases and decreases) and it might include complex instructions like combining previously knit stitches from other needles.

We mark a stitch as *knitted* after it is knit and move to the next stitch along the course direction. If there is no next stitch in the course direction, we check the next stitch in the wale direction. If there is no next stitch in the wale direction or if the next stitch in the wale direction belongs to a different yarn piece, it means that we have completed knitting all stitches of the current yarn piece. After we move to the next stitch, we check if it is ready to be knit. If it is ready, we repeat the same process; otherwise, we place it into the queue.

Note that knitting can begin and continue with any stitch in the queue that is ready. Therefore, for complex models, the implementation of the queue determines which parts of the model are knit first. In our implementation, we used a stack with First-In-Last-Out order. When a stitch s is enqueued, we check its first dependent stitch t and we move the stitch in the queue that uses the same yarn piece as t to the top of the queue. This effectively prioritizes knitting one part of a model before starting to knit the other parts, attempting to reduce the number of needles needed during knitting.

5 IMPLEMENTATION AND RESULTS

We have tested our knittable stitch mesh modeling framework on two fronts: (1) generating complex yarn-level models containing short rows and various forms of mismatched directions and (2) fabricating models via assisted hand knitting.

5.1 Graphics Interface for Hand Knitting

We have developed a graphics interface that displays the step-by-step knitting instructions to a knitter, shown in Figure 14. At every step our interface instructs the user to perform a series of knitting instructions. For simplification, short sets of knitting instructions that are consecutively repeated are grouped together. In these cases, the interface presents how many times the given knitting instructions should be repeated. We have found this approach favorable to providing a single knitting instruction at a time. A 3D viewport shows the previously knitted part of the model and the part that corresponds to the current set of knitting instructions. We use the real-time fiber-level cloth rendering method of Wu and Yuksel (2017a, 2017b) to display the yarn-level model.

One difficulty with hand knitting is that it is easy to lose count. A simple solution that is used in hand knitting is placing markers between stitches on needles. If the knitter loses count, only counting the stitches back to the last marker would be sufficient to identify the current position. Therefore, our knitting interface includes markers as well. Our system instructs a user to place a marker when a shift path is reached, right before the first stitch of a new row. The user is also instructed to place markers before and after a group of mismatched directions and the beginning and ending of short rows. Our system also keeps track of these markers and provides knitting instructions using them. For example, instead of instructing the user to perform a set of instructions a certain number of times, we can instruct the user to perform the instructions until a marker is reached. We have found that this simple feature makes it much easier to follow the knitting instructions in practice. We also permit users to place markers after any groups of knitting instructions. After a marker is placed by the user, upcoming knitting instructions can use that marker.

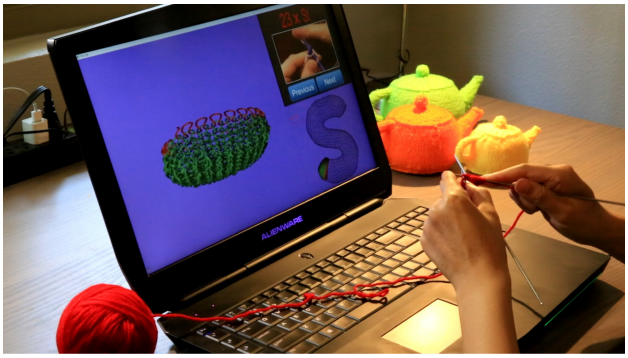


Fig. 14. **Our knitting interface**: On the top-right corner the knitting instruction code is displayed along with how many times the instruction should be repeated. Below the instruction code a short video clip shows how to perform the instruction. At the bottom-right side the entire model is displayed, along with the previously knitted part shaded in green and the stitches that correspond to the current instructions shaded in red. The main view provides a yarn-level rendering of the part of the model that is previously knit and the part that is currently being knit.

5.2 Knitted Models

An example teapot model prepared using our system is shown in Figure 1. The input mesh model (Figure 1(a)) is converted to a knittable stitch mesh (Figure 1(b)) with three shift paths; one group of joining, two groups of splitting, and six groups of perpendicular mismatched directions, and multiple short rows. The simulated model (Figure 1(c)) is produced via yarn-level relaxation after generating the yarn curves from the knittable stitch mesh. The knitted model (Figure 1(d)) is fabricated via hand knitting following the step-by-step knitting instructions provided by our knitting interface.

Figure 15 shows three different teapot models generated from the same input mesh with different levels of tessellations used for generating the knittable stitch meshes. Notice that higher levels of tessellations lead to models with more stitches that can represent more details but take longer to knit. Note that knittable stitch meshes can be designed using any modeling operations of stitch meshes (Figure 16). The simulated models (Figure 17(a)) are constrained to take the shape of the stitch mesh model. We have noticed that removing the shaping constraints and using one frame of yarn-level simulation (Figure 17(b)) produces shapes with similar features to the hand-knitted models that are stuffed with cotton.

Fig. 15. Knitted teapots with different numbers of stitches using different knittable stitch meshes generated from the same input mesh in Figure 1. They are all knitted using six separate yarn pieces and they contain 6.3K, 4.4K, and 2.6K stitches from left to right.

Fig. 16. Teapot models with different stitch mesh patterns.

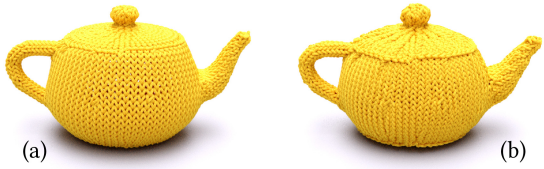


Fig. 17. The small teapot model (a) after yarn-level relaxation and (b) after one frame of simulation with gravity.

We test our approach for handling different mismatched directions using simple models in Figure 18 and Figure 19. The model in Figure 18 is knitted using two long yarn pieces along with two shorter yarn pieces used for handling the joining and splitting mismatched directions. Expanding and contracting perpendicular directions appear along the flat face of the model. The model in Figure 19 shows two different ways of handling joining and splitting mismatched directions. One of them uses extra pieces of yarn for handling all mismatched directions and the other one avoids using extra yarn pieces, so the entire model can be knit using only three yarn pieces.

For testing more challenging cases of topological connections and short rows, we have modeled and knitted some example letters shown in Figure 20. Most letters demonstrate how short rows (red faces) can be used for producing curved shapes: "R" and "A" include sharp corners that can be generated using short rows, and "R," "P," and "H" show the importance of handling perpendicular mismatched directions (yellow-green and blue-purple face pairs) for knitting complex shapes. While these letters are relatively small knitting projects, as compared to a larger model like a full-size sweater, they are excellent examples of complex shapes and topologies. In that respect, they are more complex cases than typical garment models.

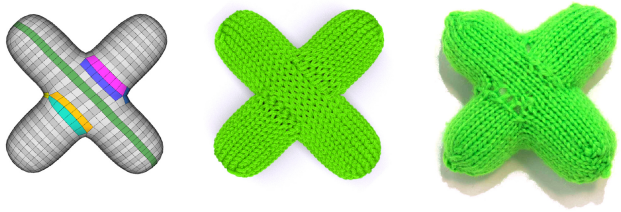


Fig. 18. **Two bars intersecting**: (left) knittable stitch mesh, (middle) simulated yarn-level model, and (right) final knitted model. The model contains 1.5K stitches and it is knitted using four yarn pieces, two of which are used for handling joining and splitting mismatched directions.

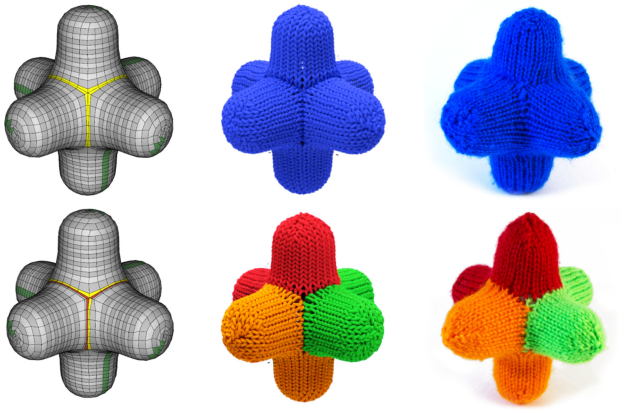


Fig. 19. **Three bars intersecting:** (left) knittable stitch meshes, (middle) simulated yarn-level models, and (right) final knitted models, showing joining and splitting mismatched directions handled (top row) using four extra yarns pieces, two for cast-on stitches along splitting mismatched directions and two for binding of stitches along joining mismatched directions in addition to the three yarn pieces needed for knitting the rest of the model, and (bottom row) using one of the yarn pieces near the mismatched edges, requiring only three yarn pieces for knitting the entire model. Both models contain 2.1K stitches.

More traditional examples involving sweaters with different sizes are shown in Figure 21. These sweater models are simpler, since they do not include any mismatched directions or short rows, but they include three shift paths that correspond to three yarn pieces used for knitting each sweater model. They were modeled

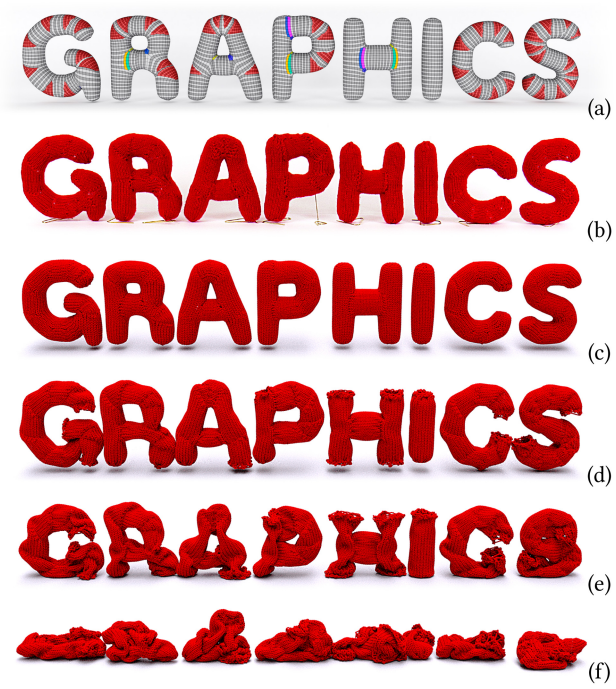


Fig. 20. **Example letters:** (a) knittable stitch mesh models, (b) knitted models, (c) simulated models, (d–f) yarn-level simulation results with gravity after removing the cast-on and bind-off stitches. Each letter model contains between 2.5K and 2.9K stitches, except for "I," which has only 1K stitches. "G" is knitted using three yarn pieces; "R," "A," and "P" need four yarn pieces; "H" contains six separate yarn pieces for handling the joining and splitting mismatched directions; and "I," "C," and "I" are knitted using a single piece of yarn.

using our framework and knitted using the step-by-step instructions provided by our knitting interface. The size differences of the final models are due to the yarn types and needle sizes used for knitting them.

5.3 Performance

By far the slowest part of our pipeline is hand knitting. It can take hours or days to knit a model, depending on the experience of the knitter and the number of stitches needed to complete the model. The second-slowest component is yarn-level relaxation that is used for generating the simulated models, which can also take hours, depending on the complexity of the model (see the performance results of Yuksel et al. (2012) for yarn-level relaxation times of different models). The rest of the operations we use can be handled interactively, except for the mesh-based relaxation, which takes several seconds to about a minute.

6 LIMITATIONS AND FUTURE DIRECTIONS

Knittable stitch meshes provide a powerful structure for representing and designing knittable models. While they are general and able to incorporate a wide variety of knitting patterns and complex shapes, they cannot represent *everything* that can be knit. For example, multilayer patterns that are used for knitting multicolor designs cannot be represented using our formulation. With hand

Fig. 21. Sweaters designed using our system and knitted by following the step-by-step instructions we generate using different yarn types and needle sizes. All sweaters use the same knittable stitch mesh model with 5.7K stitches and are knitted using three separate yarn pieces.

knitting, it is possible to pull a loop through any previously knitted loop, not just the loops on a previous row, but such operations cannot be represented with stitch meshes. Consequently, we cannot represent models that are knit as separate pieces and then sewn together. The knittable stitch mesh representation is limited to the stitch types that can be abstracted using stitch mesh faces.

Our modeling framework relies on the topology of the input mesh. This allows precisely designing the desired knit structure but requires the user to have some understanding of the knitting process. Thus, our framework cannot convert any polygonal mesh into a knittable stitch mesh. In particular, modeling the input meshes for unusual models, such as the teapot model in Figure 1, may require many iterations to figure out how the model can be knit. However, once the user determines how to knit a model and prepares the input mesh, the knittable stitch mesh modeling framework allows for quickly designing the final model. Afterward, the knitting instructions generated from the final model can be used by any knitter to fabricate the model. Therefore, combining our framework with an automated stitch mesh generation process for a given arbitrary 3D model would be an important direction for future research (Wu et al. 2018).

The stitch mesh modeling framework allows defining knitting instructions on a given 3D input model shape, but it does not guarantee that the final rest shape of the knit model would align with the shape of the input model. This is because the final shape of a model depends on the knitting instructions, and the knittable

stitch mesh modeling framework provides enough flexibility to design knitting instructions that would contradict the shape of the input model. In fact, it is a challenging problem to determine the set of knitting instructions that would produce a desired shape. Yet, a simple analysis of the excessive stretching or compression of stitch mesh faces after the mesh-based relaxation step often provides a good indication of how well the knitting instructions agree with the input model shape. Therefore, the majority of the modeling iterations can be done at the stitch mesh modeling phase, without having to fabricate each iteration via knitting.

Yet, the same knitting instructions performed by different knitters often produce somewhat different results. This is because the final shape of a knit structure depends on not only the properties of the yarn and the needles used but also the forces applied during the knitting process. Therefore, accurately predicting the final shape of a model that would be produced by a particular knitter is a challenging problem that our approach does not address.

Using machine knitting, as opposed to hand knitting, allows more precisely controlling the forces used during the fabrication process. While contemporary knitting machines are extremely powerful and highly customizable, they also impose additional constraints on the types of knit structures that they can produce. Even though it is theoretically possible to generate machine knitting instructions from a knittable stitch mesh, since we do not consider the limitations of knitting machines, it is reasonable to expect that some knittable stitch meshes may not be knit by existing knitting machines. A knitting machine can perform knitting operations only on the yarn loops that reside on its needles, so all yarn loops that will be used for knitting stitches later must be kept on some needles. Therefore, the scheduling of knitting operations and the placement of these yarn loops on the needles must consider the physical constraints of the machine and the model, as well as the tension that will be applied on the yarn to prevent breaking the yarn. In particular, operations like three-needle bind-off can be challenging to perform using a knitting machine, depending on the local complexity of the stitch mesh model. Therefore, incorporating the limitations of knitting machines into the knittable stitch mesh modeling framework would be an interesting direction for future research.

7 CONCLUSION

We have presented knittable stitch meshes that extend the powerful concept of stitch meshes into models that can be fabricated via knitting. By introducing shift paths and properly handling mismatched knitting directions, we can convert any stitch mesh into a knittable structure. We have also introduced novel representations for handling shaping techniques that allow designing knit structures with unprecedented complexity. Finally, we have presented an algorithm that generates step-by-step instructions from a given knittable stitch mesh. We have shown a variety of example models with complex topologies to demonstrate the effectiveness of our approach.

ACKNOWLEDGMENTS

We thank Jonathan M. Kaldor for the yarn-level simulation code, Manuel Vargas for the volume generation code, and Yan Wang for knitting the models. All yarn-level models are rendered using MITSUBA.

REFERENCES

Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. 2003. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH'03)* 22, 3 (July 2003), 828–837.

Ergun Akleman, Jianer Chen, Qing Xing, and Jonathan L. Gross. 2009. Cyclic plain-weaving on polygonal mesh surfaces with graph rotation systems. *ACM Transactions on Graphics* 28, 3 (2009), 78.

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. *ACM Transactions on Graphics* (1998), 43–54.

David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global structure optimization of quadrilateral meshes. *Computer Graphics Forum* 30, 2 (2011), 375–384.

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics* 21, 3 (2002), 594–603.

Marcio Cabral, Sylvain Lefebvre, Carsten Dachsbacher, and George Drettakis. 2009. Structure-preserving reshape for textured architectural scenes. *Computer Graphics Forum* 28 (2009), 469–480.

Michel Carignan, Ying Yang, Nadia Magnenat Thalmann, and Daniel Thalmann. 1992. Dressing animated synthetic actors with complex deformable clothes. *ACM Transactions on Graphics* (1992), 99–104.

Ka-Fai Choi and Tien-Yu Lo. 2003. An energy model of plain knitted fabric. *Textile Research Journal* 73, 8 (2003), 739–748.

Ka Fai Choi and Tin Yee Lo. 2006. The shape and dimensions of plain knitted fabric: A fabric mechanical model. *Textile Research Journal* 76, 10 (2006), 777–786.

Lillian Chu. 2005. *A Framework for Extracting Cloth Descriptors from the Underlying Yarn Structure.* University of California, Berkeley.

Gabriel Cirio, Jorge Lopez-Moreno, and Miguel A. Otaduy. 2015. Efficient simulation of knitted cloth using persistent contacts. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'15).* ACM, New York, NY, 55–61.

Philippe Decaudin, Dan Julius, Jamie Wither, Laurence Boissieux, Alla Sheffer, and Marie-Paule Cani. 2006. Virtual garments: A fully geometric approach for clothing design. *Computer Graphics Forum* 25 (2006), 625–634.

A. Demiroz and T. Dias. 2000. A study of the graphical representation of plain-knitted structures part I: Stitch model for the graphical representation of plain-knitted structures. *Journal of the Textile Institute* 91, 4 (2000), 463–480.

M. Duhovic and D. Bhattacharyya. 2006. Simulating the deformation mechanisms of knitted fabric composites. *Composites Part A: Applied Science and Manufacturing* 37, 11 (2006), 1897–1915.

Bernhard Eberhardt, Michael Meissner, and Wolfgang Strasser. 2000. Knit fabrics. In *Cloth Modeling and Animation.* A. K. Peters, 123–144.

O. Goktepe and S. C. Harlock. 2002. Three-dimensional computer modeling of warp knitted structures. *Textile Research Journal* 72, 3 (2002), 266–272.

Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. 2007. Efficient simulation of inextensible cloth. *ACM Transactions on Graphics* 26, 3 (2007), 49.

Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. 2003. Discrete shells. In *Proceedings of SCA.* 62–67.

Ankit Gupta, Dieter Fox, Brian Curless, and Michael Cohen. 2012. DuploTrack: A real-time system for authoring and guiding Duplo Block Assembly. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST'12).* 389–402.

David J. Heeger and James R. Bergen. 1995. Pyramid-based texture analysis/synthesis. In *Proceedings of ACM SIGGGRAPH.* ACM, 229–238.

Emmanuel Iarussi, Wilmot Li, and Adrien Bousseau. 2015. WrapIt: Computer-assisted crafting of wire wrapped jewelry. *ACM Transactions on Graphics* 34, 6 (Oct. 2015), Article 221, 8 pages.

Takeo Igarashi and Jun Mitani. 2010. Apparent layer operations for the manipulation of deformable objects. *ACM Trans. Graph.* 29, 4, Article 110 (July 2010), 7 pages.

Yuki Igarashi, Takeo Igarashi, and Jun Mitani. 2012. Beady: Interactive beadwork design and construction. *ACM Transactions on Graphics* 31, 4 (July 2012), Article 49, 9 pages.

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008b. Knitting a 3D model. *Computer Graphics Forum* 27, 7 (2008), 1737–1743.

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008a. Knitty: 3D modeling of knitted animals with a production assistant interface. In *Eurographics 2008 - Short Papers*, Katerina Mania and Eric Reinhard (Eds.). Eurographics Association.

Chenfanfu Jiang, Theodore Gast, and Joseph Teran. 2017. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Transactions on Graphics* 36, 4 (July 2017), Article 152, 14 pages.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2008. Simulating knitted cloth at the yarn level. *ACM Trans. Graph.* 27, 3, Article 65 (Aug. 2008), 9 pages.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2010. Efficient yarn-based cloth with adaptive contact linearization. *ACM Trans. Graph.* 29, 4, Article 105 (July 2010), 10 pages.

Arif Kurbak. 2009. Geometrical models for balanced rib knitted fabrics part I: Conventionally knitted 1× 1 rib fabrics. *Textile Research Journal* 79, 5 (2009), 418–435.

Arif Kurbak and Tuba Alpyildiz. 2008. A geometrical model for the double Lacoste knits. *Textile Research Journal* 78, 3 (2008), 232–247.

Arif Kurbak and Ali Serkan Soydan. 2009. Geometrical models for balanced rib knitted fabrics part III: 2× 2, 3× 3, 4× 4, and 5× 5 rib fabrics. *Textile Research Journal* 79, 7 (2009), 618–625.

Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graph.* 22, 3 (July 2003), 277–286.

Yu-Kun Lai, Miao Jin, Xuexiang Xie, Ying He, Jonathan Palacios, Eugene Zhang, Shi-Min Hu, and Xianfeng Gu. 2010. Metric-driven rosy field design and remeshing. *IEEE Transactions on Visualization and Computer Graphics* 16, 1 (2010), 95–108.

Yong-Jin Liu, Dong-Liang Zhang, and Matthew Ming-Fai Yuen. 2010. A survey on CAD methods in 3D garment design. *Computers in Industry* 61, 6 (2010), 576–593.

Ze Gang Luo and Matthew Ming-Fai Yuen. 2005. Reactive 2D/3D garment pattern design modification. *Computer-Aided Design* 37, 6 (2005), 623–630.

James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. 2016. A compiler for 3D machine knitting. *ACM Transactions on Graphics* 35, 4 (July 2016), Article 49, 11 pages.

Michael Meißner and Bernd Eberhardt. 1998. The art of knitted fabrics, realistic & physically based modelling of knitted patterns. *Computer Graphics Forum* 17 (1998), 355–362.

Eder Miguel, Mathias Lepoutre, and Bernd Bickel. 2016. Computational design of stable planar-rod structures. *ACM Transactions on Graphics* 35, 4 (July 2016), Article 86, 11 pages.

Yuki Mori and Takeo Igarashi. 2007. Plushie: An interactive design system for plush toys. *ACM Trans. Graph.* 26, 3, Article 45 (July 2007).

Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and Jim McCann. 2018. Automatic machine knitting of 3D meshes. *ACM Trans. Graph.* 37, 3, Article 35 (Aug. 2018), 15 pages.

Olivier Nocent, Jean-Michel Nourrit, and Yannick Remion. 2001. Towards mechanical level of detail for knitwear simulation. In *WSCG*. 252–259.

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

Wilfried Renkens and Yordan Kyosev. 2011. Geometry modelling of warp knitted fabrics with 3D form. *Textile Research Journal* 81, 4 (2011), 437–443.

Cody Robson, Ron Maharik, Alla Sheffer, and Nathan Carr. 2011. Context-aware garment modeling from sketches. *Computers & Graphics* 35, 3 (2011), 604–613.

Shima Seiki. 2011. Sds-one apex3. Retrieved from http://www.shimaseiki.com/product/design/sdsone_apex/flat/

Soft Byte Ltd. 1999. Designaknit. Retrieved from http://softbyte.co.uk/designaknit.htm.

Stoll. 2011. M1plus pattern software. Retrieved from http://www.stoll.com/stoll_software_solutions_en_4/patternsoftware_m1plus/3_1.

Emmanuel Turquin, Jamie Wither, Laurence Boissieux, Marie-Paule Cani, and John F. Hughes. 2007. A sketch-based interface for clothing virtual characters. *IEEE Computer Graphics and Applications* 27, 1 (2007), 72–81.

Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. 2011. Sensitive couture for interactive garment modeling and editing. *ACM Transactions on Graphics* 30, 4 (2011), 90.

Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik. 2013. OpenFab: A programmable pipeline for multi-material fabrication. *ACM Transactions on Graphics* 32, 4 (2013), 136.

Pascal Volino and Nadia Magnenat-Thalmann. 2005. Accurate garment prototyping and simulation. *Computer-Aided Design and Applications* 2, 5 (2005), 645–654.

Pascal Volino and Nadia Magnenat-Thalmann. 2012. Virtual clothing: Theory and practice. Springer Science & Business Media.

Pascal Volino, Nadia Magnenat-Thalmann, and Francois Faure. 2009. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Trans. Graph.* 28, 4, Article 105 (Sept. 2009), 16 pages.

Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. 2018. Stitch meshing. *ACM Trans. Graph.* 37, 4, Article 130 (July 2018), 14 pages.

Kui Wu and Cem Yuksel. 2017a. Real-time cloth rendering with fiber-level detail. *IEEE Transactions on Visualization and Computer Graphics* 99 (2017), 12.

Kui Wu and Cem Yuksel. 2017b. Real-time fiber-level cloth rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D'17)*. ACM, 8.

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, 3 (2012), Article 37, 12 pages.

Kun Zhou, Xin Huang, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. 2006. Mesh quilting for geometric texture synthesis. *ACM Transactions on Graphics* 25, 3 (July 2006), 690–697.

Received May 2018; revised October 2018; accepted November 2018