Visual Knitting Machine Programming

VIDYA NARAYANAN*, Carnegie Mellon University
KUI WU*, University of Utah
CEM YUKSEL, University of Utah
JAMES MCCANN, Carnegie Mellon University

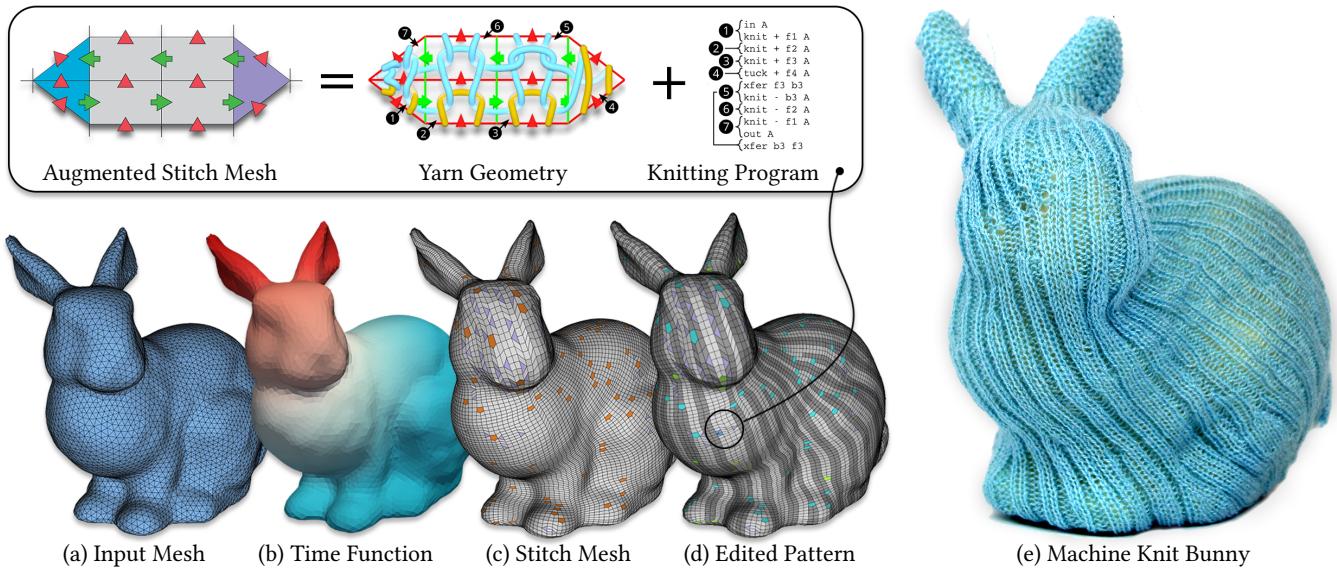


Fig. 1. ***Stages of our visual knit programming system:*** *(a) Our system begins with an input mesh; (b) generates a knitting time function; (c) remeshes the surface to create an augmented stitch mesh; (d) allows the user to interactively edit and add patterns, textures, and colorwork; and (e) generates instructions for fabrication on an industrial knitting machine. At the core of our interface is the augmented stitch mesh, which associates yarn geometry, dependency information, and a knitting program with each face.*

Industrial knitting machines are commonly used to manufacture complicated shapes from yarns; however, designing patterns for these machines requires extensive training. We present the first general visual programming interface for creating 3D objects with complex surface finishes on industrial knitting machines. At the core of our interface is a new, augmented, version of the stitch mesh data structure. The augmented stitch mesh stores low-level knitting operations per-face and encodes the dependencies between faces using directed edge labels. Our system can generate knittable augmented stitch meshes from 3D models, allows users to edit these meshes in a way that preserves their knittability, and can schedule the execution order and location of each face for production on a knitting machine. Our system is general, in that its knittability-preserving editing operations are sufficient to transform between any two machine-knittable stitch patterns with the same orientation on the same surface. We demonstrate the power and flexibility of our pipeline by using it to create and knit objects featuring a wide range of patterns and textures, including intarsia and Fair Isle colorwork; knit and purl textures; cable patterns; and laces.

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

Additional Key Words and Phrases: automatic knitting, fabrication, stitch meshes

ACM Reference Format:
Vidya Narayanan, Kui Wu, Cem Yuksel, and James McCann. 2019. Visual Knitting Machine Programming. *ACM Trans. Graph.* 38, 4, Article 63 (July 2019), 13 pages. https://doi.org/10.1145/3306346.3322995

1 INTRODUCTION

Computer-controlled knitting machines are powerful tools for computer-aided fabrication, and are widely used in the garment and accessory industries. When properly programmed, they can turn yarns into soft 3D surfaces in a wide range of shapes, textures, and colors. Knitting machines create these objects by using a small vocabulary of operations which manipulate loops on their *needle beds*, two long rows of loop storage locations. Once programmed,

*Co-first authors; equal contribution.

Authors' addresses: Vidya Narayanan, Carnegie Mellon University; Kui Wu, University of Utah; Cem Yuksel, University of Utah; James McCann, Carnegie Mellon University.

0730-0301/2019/7-ART63 $15.00
https://doi.org/10.1145/3306346.3322995

objects can be manufactured quickly and with minimal wasted yarn.

Knitting machine programming is notoriously challenging because shape, structure, texture, and color effects must all be created concurrently using a small set of low-level operations. In addition, these operations must be *scheduled* to limited needle bed locations on a knitting machine, and they are conventionally selected with limited visual and structural feedback.

We demonstrate the first general visual programming interface for 3D machine knitting, which tackles all these challenges. The core of our system is an *augmented* stitch mesh data structure, where each face contains both a visual representation and a specific set of low-level knitting operations. We couple this representation with an automatic mesh generator and a set of knittabillity-preserving editing operations to provide a system that can navigate the space of *all* tube-based 3D knitting programs. Together, these innovations result in the first general visual editor for knitting programs .

The main technical contributions presented in this paper are:

- an *augmented* stitch mesh data structure, where each face is associated with a local machine knitting program;
- a scheduling system that assigns needle bed locations and times to stitch mesh faces in order to automatically fabricate 3D knitting patterns from augmented stitch meshes;
- and an interactive visual design system to edit knitting programs directly in 3D, while preserving the machine knittability.

We provide a brief overview of prior work (Section 2) and related background on machine knitting (Section 3), then describe the details of our augmented stitch mesh structure (Section 4). Next, we describe how our system allows users to generate, edit, and machine-knit augmented stitch meshes. Particularly, we demonstrate how a machine-knittable augmented stitch mesh may be created from an input 3D object (Section 5); how augmented stitch meshes may be edited in a general and machine-knittability-preserving way (Section 6); and how the final mesh faces can be ordered (Section 7.1), and their loops can be assigned needle locations (Section 7.2) for machine fabrication. Finally, we present our results (Section 8) and discuss the limitations of our work and future directions (Section 9), before we conclude (Section 10).

2 PRIOR WORK

Before we discuss the details of our method, we provide a brief overview of prior work in the areas of fabric and knit modeling, simulation, and design.

Machine Knitting. Modeling the yarn-level structure needed for knit fabric is a complex procedure. Commercial knitting design software provides templates of standard designs with limited scope for editing [Shima Seiki 2011; Soft Byte Ltd. 1999; Stoll 2011]. More complex or non-standard patterns must be hand-designed at the stitch level, though there exist guidebooks of advanced techniques that can assist with this process [Underwood 2009]. Notably, these traditional design tools work in the construction space of the machine – requiring users to figure out the construction location and construction order of stitches at the same time as they determine stitch types and connectivity.

Meißner and Eberhardt [1998] proposed one of the earliest graphics based approaches for visualization and design of machine knitted structures. Recently, researchers have expanded this scope by offering general tube and sheet primitives in the context of a knitting compiler [McCann et al. 2016]. Since their approach still requires a designer to place and configure these primitives by hand, Narayanan et al. [2018] recently proposed an automatic approach knit a wide variety of 3D surfaces on machine. However, their method focuses on matching the topology and geometry of the input model and does not offer any options to edit these patterns for texture or colorwork. Popescu et al. described a similar system that works on topologically-disc-shaped patches [Popescu et al. 2018], which can later be connected manually. In addition to making feasible knitting patterns, researchers are also beginning to examine how to create efficient patterns [Lin et al. 2018], though this work is limited to flat knitting patterns. We introduce a system to represent and edit machine knittable structures including their shape, color and textures. We extend the scheduling algorithm proposed by [Narayanan et al. 2018] to support these details. In contrast to traditional tools, our system works in the output space of the target design – allowing designers to focus on *what* they wish to create instead of *how* they should construct it.

Hand Knitting. Relative to machine knitting, hand knitting is very flexible. Human knitters are dextrous and able to form complex stitches using loops from anywhere in the existing fabricated item. Thus, designing for human knitters is a substantially different problem than designing for machine knitting, and while approaches for the latter can be used for the former, the reverse is not true.

It is known that a 2D surface of any topology can be hand knit [Belcastro 2009]. Igarashi et al. [2008a; 2008b] presented a design assistant that semi-automatically 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.

Yuksel et al. [2012] introduced stitch meshes, a data structure for modeling knit structures for visualization and simulation. Recently, Wu et al. introduced an automatic pipeline to convert arbitrary shape into labeled quad-dominant mesh stitch meshes [2018] and extended them for hand knitting by introducing a list of hand-knittable mismatch faces [2019]. We build on the idea of the stitch mesh, adding dependency tracking and local machine knitting instructions to each face. This allows users of our system to edit both mesh topology and face programs while preserving machine knittability.

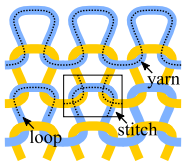
Simulation and Rendering. Complementary to fabrication efforts, yarn-level simulation methods [Kaldor et al. 2008, 2010] can produce realistic deformations of knitted structures. Recent works have focused on efficient methods to simulate yarn-level details, including combining Lagrangian and Eulerian approaches [Sueda et al. 2011], applying reduced order methods [Cirio et al. 2014, 2015, 2017], and modelling anisotropic deformations using the material point method [Jiang et al. 2017]. Leaf et al. [2018] demonstrated interactive yarn simulation for periodic pattern design using GPU computation.

Researchers have been working on photorealistic knit fabric rendering for over a decade [Groller et al. 1995; Gröller et al. 1996]. Approaches include representing the geometric complexity of knit structures with volumetric approximations [Chen et al. 2003; Xu

et al. 2001], CT scan data [Zhao et al. 2011], and procedural function [Zhao et al. 2016a]; and rendering the data with the radiative transfer framework [Jakob et al. 2010], the SGGX microflake distribution [Zhao et al. 2016b], and data-driven approaches [Aliaga et al. 2017; Khungurn et al. 2015]. Generating details on-the-fly has been used to reduce memory usage [Luan et al. 2017] and achieve interactive rates [Lopez-Moreno et al. 2015; Wu and Yuksel 2017a,b]. Readers interested in a more extensive survey may refer to [Schröder et al. 2012].

Clothing Design. In contrast to constructing fabric from the yarn-level, most clothing design work in graphics has focused on the "cut-and-sew" paradigm, where clothing is sewn together from multiple panels cut from flat fabric; with many contributions in simulation [Carignan et al. 1992; House and Breen 2000; Volino and Magnenat-Thalmann 2000; Volino et al. 2009] and interactive and intuitive interfaces [Decaudin et al. 2006; Mori and Igarashi 2007]. One of the difficulties in cut-and-sew clothing is fitting 3D models, which is done by placing specific shaping features like darts and folds [Li et al. 2018; Turquin et al. 2007; Umetani et al. 2011; Wang 2018] or by combining and adjusting patterns in a physically meaningful manner [Bartle et al. 2016]. Recently, data driven approaches have been used for parsing sewing garments into 3D draped forms as well as for exploring the multi-modal design space of body shapes, textures, and 2D patterns [Berthouzoz et al. 2013; Wang et al. 2018].

3 BACKGROUND



Knitting is a technique to produce fabric from yarn by manipulating loops of yarn to form *stitches*. In the inset figure, adjacent stitches are yarn-wise connected to their left and right neighbors and loop-wise connected to their top and bottom neighbors. This yarn-wise and loop-wise connectivity of stitches gives rise to a grid-like structure and characteristic knit texture.

Machine Knitting. Industrial knitting machines are programmable systems that manipulate loops into knit structures using hundreds of *needles* arranged in two parallel *beds*. Needles receive yarn from *yarn carriers* that run between them, and can be actuated to perform one of three basic instructions:

(1) `tuck D N CS` : add a loop to needle `N` using yarn carrier set `CS` in the specified direction `D`.
(2) `knit D N CS` : add a loop through all the existing loops on needle `N` using yarn carriers `CS` in the specified direction `D`. All previously held loops at the location are dropped.
(3) `xfer N1 N2` : move all loops on needle `N1` to target needle `N2`. Needles `N1` and `N2` must be aligned and on opposite beds.

These operations, along with others to perform machine configuration (bed alignment, stitch sizing) and yarn handling, comprise the *knitout* low-level knitting language, which we use for output in our system. Details of knitout can be found in its specification [McCann 2017], while more information about knitting machines in general can be found in [Spencer 2001].

Even though there are only three basic instructions, they can be combined to form a dazzling array of textures, colors, and shapes [Underwood 2009]. Indeed, industrial knitting machines are used to fabricate items as diverse as shoes, sweaters, car upholstery, and glass-fiber reinforcement for composites.

Valid Knitting Programs. While most arbitrary sequences of needle operations can be executed by a knitting machine, few will produce results beyond a yarn tangle. This is because operations depend on previous operations to place loops and yarn carriers in specific places in order to run smoothly. Specifically, for a knitting program to form a desired set of stitches, it must respect the yarn-wise and loop-wise dependencies of those stitches.

We formalize this notion of validity with two properties:

Property 1 (Order). *All stitches must be constructed in the yarn-wise order specified by the pattern. All loops that a stitch depends upon must be constructed before it and must be available on a needle at the time of constructing the stitch.*

Property 2 (Adjacency). *All yarn-wise adjacent stitches must be constructed on adjacent needles.*

In other words, a given sequence of stitches is machine knittable if and only if its associated knitting program can be ordered and scheduled to machine needles such that the loop(s) and yarn(s) each stitch depends on are held on adjacent needles at the time of its construction. In the next section, we introduce a data structure for knitting representation built with this important notion in mind.

4 AUGMENTED STITCH MESHES

The data structure at the core of our system is the *augmented* stitch mesh (Figure 1), a 3D mesh in which each face is labelled with both yarn topology and machine instructions, and each edge is labelled with yarn or loop dependency information.

The stitch mesh data structure, introduced by Yuksel et al. [2012], represents complex yarn topologies as compositions of a few basic face types which can be connected as long as their edge labels match–essentially, as a set of generalized Wang tiles [1961]. By selecting the appropriate vocabulary of face types, stitch meshes can be used to represent knit, knotted, and woven structures. We augment this representation in two ways in order to support the editing of machine knitting programs: first, we add directed edge labels to track dependency information; second, we associate with every face type a knitting machine program that can construct the yarn-level topology on that face.

Directed Edges. The edge labels in our augmented stitch mesh capture dependencies between faces (Figure 2, left) – *loop in* edges

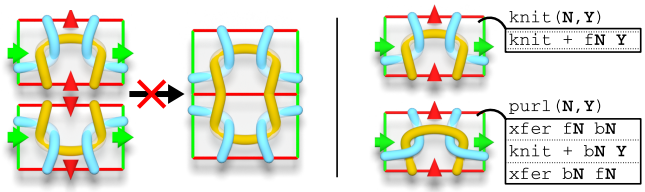


Fig. 2. *Augmented stitch mesh faces have,* left*, directed edges to prevent locally unknittable assembly; and,* right*, associated knitting programs.*

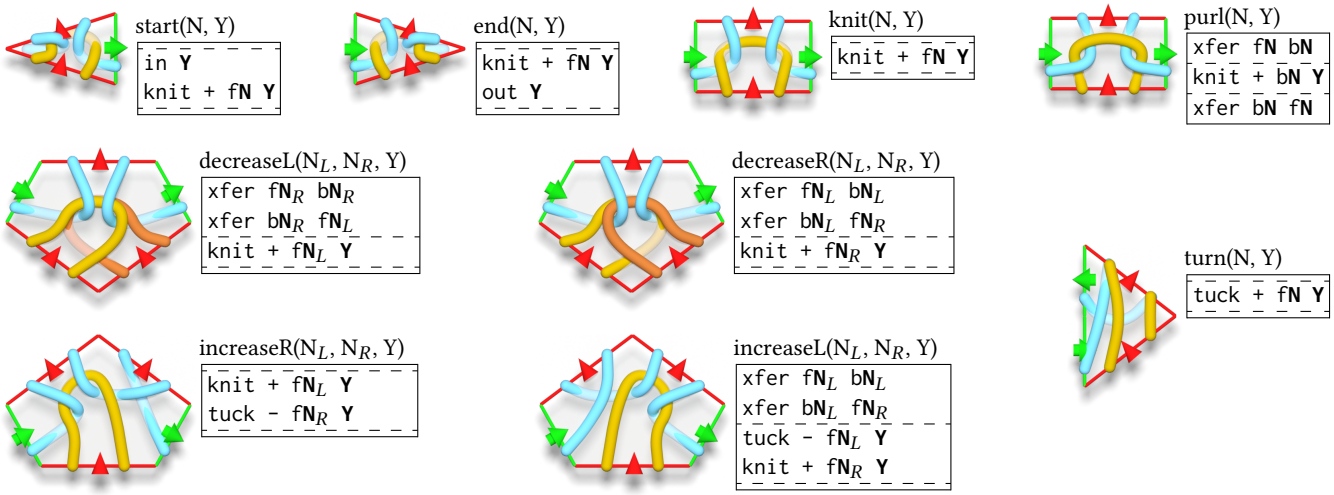


Fig. 3. *Basic face types and their associated knitting code fragments. Faces with opposite yarn direction proceed similarly. Dashed lines indicate divisions between construction passes. For increases, the input loop is always stored on needle* $\texttt{fN}_L$*. For decreases, the output loop can arrive at either needle* $\texttt{fN}_L$ *or* $\texttt{fN}_R$*. This is pseudo-code; the (JavaScript) code used in our system are available in the supplementary material.*

indicate that a loop is needed, while *loop out* edges indicate that a loop is produced; *yarn in* and *yarn out* give similar information about yarns. Any *in* edge may only connect to an *out* edge of the same type, and visa-versa. Notice that these directed edge labels induce a directed graph on the faces, which is useful when checking for dependency cycles.

Face Programs. Each face in our augmented stitch mesh data structure represents a fragment of a knitting program that operates on the yarns and loops provided by its *in* edges in order to produce the yarns and loops indicated by its *out* edges (Figure 2, right). That is, the edge labels provide a type signature – input and output loop and yarn counts – for a knitting program fragment.

The basic face types provided by our system, along with pseudo-code for their knitting program fragments, are shown in Figure 3. Our system makes it easy to extend this list, since the editing operations it supplies depend on face edge labels, not on the referenced program.

Knittability. For an augmented stitch mesh to be machine-knittable, both the ordering property (Property 1) and the adjacency property (Property 2) must hold. The ordering property is easy to check – it holds locally by construction, and can be checked globally with a topological sort (Section 7.1). The adjacency property is somewhat more subtle, in that it depends on the existence of a valid schedule among the (exponentially-large) set of possible needle allocations. Fortunately, previous work demonstrated that the existence of a valid schedule is a property of the mesh topology along with an easy-to-check feasibility condition at splits and merges within the mesh [Narayanan et al. 2018]. The former can be checked once on mesh import and the latter can be preserved through UI design.

Therefore, we focus our efforts on maintaining the ordering property.

5 STITCH MESH GENERATION

Our pipeline begins by creating a machine-knittable augmented stitch mesh from a given oriented manifold 3D surface, as illustrated in Figure 4.

As a first step, the mesh is segmented into tubular regions using either a user-specified time function (as per [Narayanan et al. 2018]) as an input to the system or by computing the Fiedler vector of the mesh Laplacian [Zhang et al. 2010]. The user may also edit boundaries for better alignment (as in [Igarashi et al. 2008a]). Then, boundaries are discretized based on stitch width such that segments align one-to-one. Once the starting and ending boundary counts are computed, each segment is quad meshed based on the stitch dimensions [Dong et al. 2005].

To maintain boundary lengths while limiting the change in stitch counts between rows for reliable fabrication, stitch counts are optimized by relaxing integer constraints on the counts and rounding

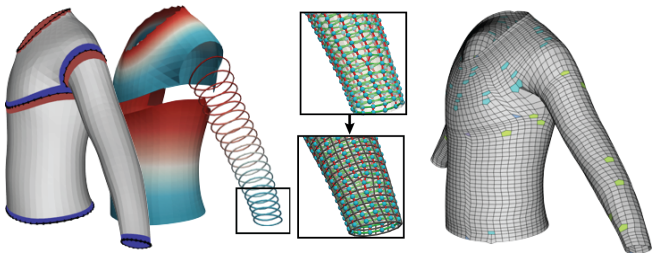
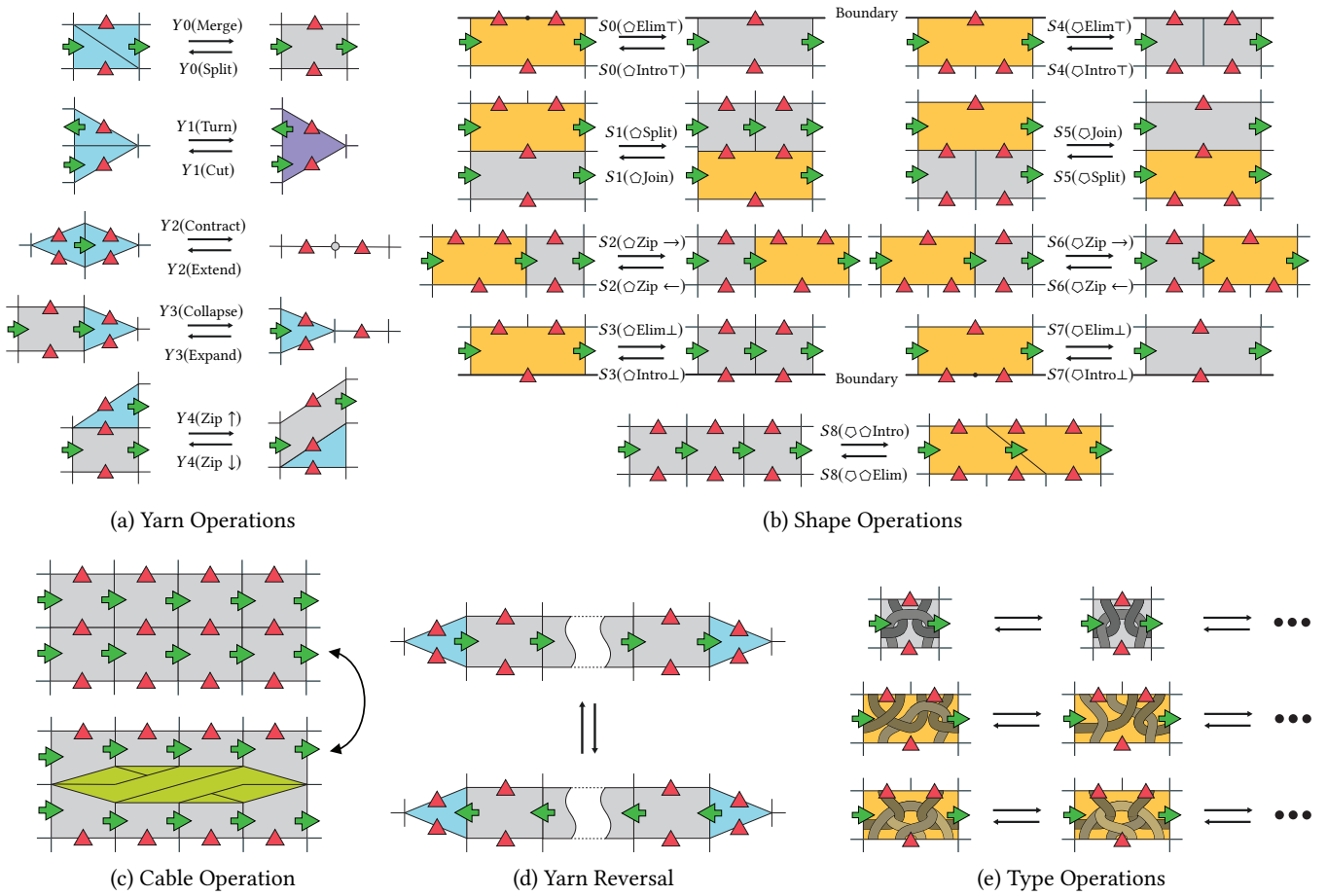


Fig. 4. *Stitch mesh generation: The input mesh is segmented into tubular regions and remeshed to identify stitch connectivity. Its dual is extracted and refined to generate an initial stitch mesh.*



(a) Yarn Operations (b) Shape Operations

(c) Cable Operation (d) Yarn Reversal (e) Type Operations

Fig. 5. ***Editing Operations:*** *blue indicates yarn-end faces, grey indicates regular faces, orange indicates pentagons, purple indicates short-row faces, and green indicates cable faces. Green and red arrows indicate yarn direction and loop direction respectively.*

the results:

$$s_f = \underset{x}{\operatorname{argmin}} \sum_i^n (x_i - s_i)^2$$

$$\text{subject to: } \frac{2}{3}x_{i-1} \le x_i \le \frac{3}{2}x_{i-1}$$

$$x_1 = s_1, x_n = s_n$$

where s_f is the vector of final stitch counts and s_i is the vector of initial counts.

The stitch mesh is extracted from the dual graph of this structure and faces with higher length distortion are refined or merged. This refinement can generate *short-rows* – a chain of faces that form a partial loop. The mesh is sub-divided along its rows to ensure an even number of short-rows followed by local edits to convert it into a single helical sequence of faces [Wu et al. 2019]. This ensures that to begin with, each tubular region can be constructed from a single yarn and yarn-wise label consistency is maintained. Loop-wise label consistency is maintained by following the time function for remeshing. This initial mesh is machine knittable as long as the input model topology is feasible as proposed by [Narayanan et al. 2018] and requires at most one yarn per tubular region.

From this starting mesh, the user can edit the mesh to refine topology or change stitch types as described next.

6 STITCH MESH EDITING

Our system provides a suite of editing operations which allow users to navigate the space of knittable augmented stitch meshes (Figure 5). These operations all involve replacing some portion of an augmented stitch mesh while maintaining compatible edge labels. Edits of this sort can still introduce global dependency cycles (i.e. violate Property 1), and we will discuss how our system prevents this at the end of this section. During editing, vertex positions are updated using projective dynamics [Bouaziz et al. 2014]. From the *ShapeOp* library [Deuss et al. 2015], edge-strain constraints are used to ensure faces retain the approximately correct size and bending and plane constraints are used to maintain the 3D shape.

Our system supports two classes of editing operations: mesh editing and data editing. Mesh editing operations change the mesh structure (face counts or connections) and include *yarn operations*

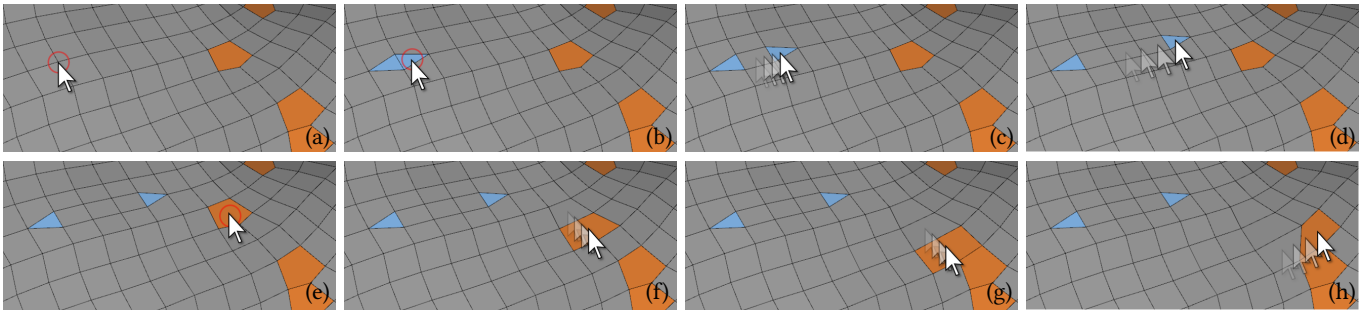


Fig. 6. *Demo of applying Y2 in (a) and Y3 in (b - d) to create a short-row as a Yarn "Zipper" and aligning the pentagon using the Shape "Zipper" by S5 in (e - g) and S6 in (h).*

that deal with yarn start/end faces, *shape operations* that modify pentagons (for increasing or decreasing loops), and *cable operations* that add/remove cable faces (for reordering loops after construction). Data editing operations do not change the mesh structure, and include *type operations* that change face type and *yarn reversal* that reverses yarn direction along a row. Although the stitch mesh structure might be modified by mesh editing, the genus of the input surface is not modified by any of the editing operations.

Yarn Operations. We call operations that involve single-yarn-edge triangles yarn operations (Figure 5a). Merging two yarn-start/end triangle faces over a *loop* edge will either form a regular quad, $Y0$(Merge), or short-row face to turn the yarn based on the types of remaining four edges, $Y1$(Turn). On the other hand, one row can be broken into two rows by their reverse operations. Removing or adding pair of yarn-start/end triangles can be used to add or remove a row, $Y2$(Contract/Extend). Yarn-start/end triangles are allowed to move along the loop-wise direction as well as along the yarn-wise direction ($Y3$ and $Y4$).

Shape Operations. Shape operations allow the user to move pentagons along the loop-wise and yarn-wise direction as illustrated in Figure 5b ($S1$, $S2$, $S5$, and $S6$). Note that operations $S0$, $S3$, $S4$, and $S7$ can remove/add a vertex without causing problems because they do so at the boundary of the mesh.

Cable Operations. Transposing loops after knitting them create interesting patterns called *cables*. Our system supports insertion and removal (Figure 5c) of cable faces of any length between two rows of regular stitches. These faces do not have yarn edges, so they cannot construct new loops, only rearrange them.

Type Operations. These simple editing operations change face types, for example, swapping a "knit" face for a "purl" face. Our system will use the corresponding face program to generate machine code during instruction generation.

Yarn Reversal. In addition to these face modifications, our system also includes an operation for reversing yarn direction by changing the face types and internal edge labels of an entire row (Figure 5d). While not strictly necessary, this operation is much more convenient than removing and re-inserting a yarn stitch-by-stitch to change its direction.

These edits work together to enable natural dragging-based edits, where faces are moved across the mesh, locally altering topology. For example, Figure 6, users can "zipper" in and out partial rows of yarn by moving yarn start or end faces, and do the same with columns of loops by moving increase or decrease faces. This gives users access to both "short-row" and "increase-decrease" shaping techniques in a very intuitive way. Our dragging-based tools are similar to singularity editing [Peng and Wonka 2013], but also preserve the machine knittability.

Preserving Machine Knittability. Though our local edits will never introduce locally conflicting edge labels, they are not always legal to apply because they can potentially introduce a dependency cycle as shown in in Figure 7. When editing, our interface checks the legality of each operation by attempting a topological sort on the dependency graph induced by the edge labels; if a directed cycle is found between a face and itself, then the ordering property has been violated and the edit is not permitted. This ensures that the ordering property (Property 1) is preserved. Also, as discussed earlier, none of our edit operations modify the topology of the input model and therefore the adjacency property (Property 2) is never violated.

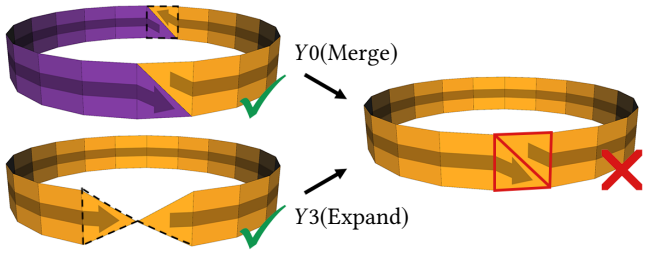


Fig. 7. *Examples of edits (shown with dashed edges) that our interface would prevent because the resulting mesh contains a cyclic dependency between faces. Arrows show yarnwise dependencies, and loopwise dependencies (not shown) point from bottom to top. Yarn-end and yarn-start faces (highlighted in red) form an unknittable structure by introducing a cyclic dependency.*

Generality. We refer to an editing operation that passes the global ordering check, and thus can be executed, as *valid*. Importantly, there is always a sequence of valid editing operations that can be used to transform one machine-knittable augmented stitch mesh into another (of the same input topology).

Indeed, we can prove a restricted version of this statement, though we first need a few lemmata:

Lemma 6.1. *Shrinking a row using operations Y2(Contract) and Y3(Collapse) is always valid.*

Proof. Removing nodes and edges from a directed graph will not create a cycle. □

Lemma 6.2. *If the edit f is valid on machine-knittable mesh A, its inverse operation f^{-1} is valid on $f(A)$.*

Proof. $f^{-1}(f(A)) = A$ is machine-knittable by hypothesis, so it passes the ordering check and f^{-1} is valid on $f(A)$. □

Lemma 6.3. *Splitting faces by S1(⬠Split), S3(⬠Elim⊥), S4(⬡Elim⊤), and S5(⬡Split) are always valid.*

Proof. Duplicating an edge in a directed graph will never create a cycle. □

Lemma 6.4. *Any pentagon face can be removed from the stitch mesh while maintaining machine-knittability.*

Proof. As shown in Figure 8, by repeatedly applying operation S5(⬡Split), the decreasing face can be moved to the top boundary and the stitch mesh remains valid (Lemma 6.2). If the pentagon is trapped by a yarn-end face, the yarn-end can be moved (Lemma 6.1), the pentagon can be moved to the boundary. Then, operation S4(⬡Elim⊤) can be used to remove a decrease pentagon. Similarly, repeatedly applying operation S1(⬠Split) and S3(⬠Elim⊥) can be used for eliminating increase pentagons. □

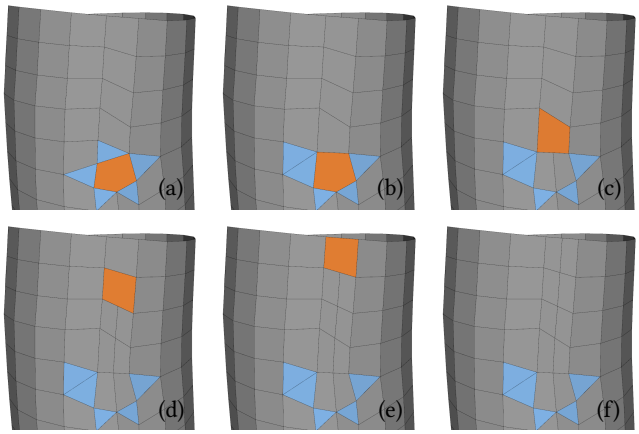


Fig. 8. ***Removing a pentagon while preserving knittability:*** *(a - b), move the yarn-end face above the pentagon; (c - e), move the pentagon to the boundary; (f), remove the pentagon.*

Theorem 6.5. *The editing operations supplied by our interface are sufficient to connect the space of all machine-knittable augmented stitch mesh tubes.*

Proof. First, we show that any valid stitch mesh tube can be turned into a trivial pattern. Any pentagon face can be edited out by Lemma 6.4 without breaking validity. Operation Y1(Cut) can be used to remove any yarn-turn triangles. Finally, all yarn-start triangles can be moved closer to their yarn-end by repeatedly applying operation Y2(Contract) and Y3(Collapse) (Lemma 6.1). The result is a stitch mesh consisting only of a ring of edges and no faces.

Finally, these edges can be collapsed to a ring with just two edges by applying Y2(Extend) and Y3(Expand) to fill the ring with a single of quads, followed by S4(⬠Intro⊥) and S0(⬠Elim⊤) to reduce the number of quads to one, followed by Y2(Contract) to remove the yarn.

Let $f_1, f_2, \cdots, f_n$ be the sequence of n operations to turn a stitch mesh F into the trivial pattern. Let $g_1, g_2, \cdots, g_m$ be the sequence of m operations to turn a stitch mesh G into the trivial pattern. By Lemma 6.3, $g_1^{-1} \circ g_2^{-1} \circ \cdots \circ g_m^{-1}$ is valid on the trivial pattern, so $g_1^{-1} \circ \cdots \circ g_m^{-1} \circ f_n \circ \cdots f_1$ is valid on F and results in G. □

We believe that a similar, more general, proof can be conducted for non-tubelike meshes by using a variant of the same construction, but some care must be taken to keep the "trivial configuration" compatible with the underlying topology (since no edits allow, e.g., the creation of a figure-8 of empty edges).

7 INSTRUCTION GENERATION

In order to convert an augmented stitch mesh into a list of machine instructions for knitting, our system must determine the order in which the faces should be created and assign machine locations to all loops produced and consumed by faces.

7.1 Face Ordering

Given that our interface preserves the ordering property (Property 1), our system will always be able to find *some* dependency-obeying order of the faces. Our system selects an ordering on the faces using a topological sort with a modified queue that always returns the next face along the currently-being-traced yarn or – failing that – the least-recently-used yarn among possible ready faces (Algorithm 1). New yarn-start faces are only knit if no other ready faces exist.

We chose this particular ordering heuristic because it avoids switching yarns and favors finishing in-action yarns before starting new yarns, while still knitting every in-action yarn reasonably frequently. Switching yarns can slow down the knitting machine, having too many active yarns can lead to time wasted by the machine moving carriers out of the way, and loops held for a long time on the machine bed can occasionally be worn out and broken.

Results of our tracing algorithm on several tricky cases are shown in Figure 9. Note that, in our system, users are also allowed to manually add dependencies between faces in order to, e.g., steer the heuristic away from non-optimal orderings (Figure 9b), or to prevent two logical yarns they intend to assign to the same physical yarn from being active at the same time. Once the faces have been

ALGORITHM 1: Face Ordering

```
Q ← all boundary yarn-in faces;
while Q is not empty do
    s ← the next stitch from Q;
    Set s as knitted;
    for all top and outgoing edge e do
        Mark e as ready;
        s_n ← neighboring stitch over e;
        if s_n is ready then
            Enqueue s_n into Q;
        end
    end
end
if at least one edge is not ready then
    Return untraceable.    // Invalid edit: the directed graph has
                           // at least one cyclic dependency.
end
```

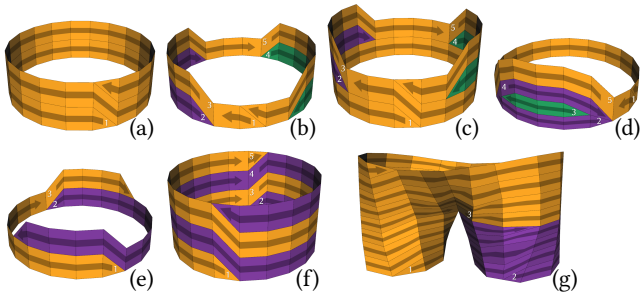


Fig. 9. ***Examples of face ordering:*** *(a) a simple spiral; (b) a case where the heuristic's arbitrary choice of starting face leads to a non-optimal ordering (if green and purple had been started first, fewer yarn changes would be needed); (c) short-rows embedded in a spiral require yarn carrier switches; (d) nested short rows; (e) yarns that depend on each-other; (f) two helicies that depend on each other, requiring multiple yarn carrier switches; and (g), two tubes merging.*

ordered, our interface allows the user to map each logical yarn to a physical yarn ID that is passed on along with the face sequences to the scheduling system.

7.2 Scheduling

Once the knitting order has been determined by tracing, our system must *schedule* (assign) storage locations for loops. Our scheduler is based on the open-source implementation released by Narayanan et al. [2018], which we have modified to support general faces instead of a fixed menu of stitch types.

Our scheduler turns each face into one more or *fragments* – low-level items that, together, produce and consume the same number of loops as a face – and breaks the sequence of fragments into logical *passes*. These fragments serve as a placeholder for the faces, and allow our system to decouple scheduling for storage locations from the operations performed on those locations. In addition to the knitting program, the configuration data associated with each face type indicates if the face requires its own pass and if it has any custom pass-level programs associated with it. Passes are constructed based on the following conditions:

- All fragments in a pass have the same direction (clockwise or counter-clockwise).
- A pass does not have more than a limited (4) number of "increase" or "decrease" shaping operations that changes its width.
- All fragments in a pass must have the same *basic type* – faces with different basic types force a pass break.

A pass a is said to depend on a pass b if pass a uses the loops produced by pass b (i.e., it reads from storage locations last written to by pass b). Each pass may be dependent on zero or more passes. If a pass depends on exactly one previous pass, it is referred to as a *regular* pass and the rest are *critical*.

Once passes are constructed, scheduling proceeds through the pipeline of [Narayanan et al. 2018]'s scheduler: An upward planar embedding is identified by enumerating all embeddings of the critical passes. Based on the shape of these critical passes, intermediate passes are filled in by assigning a shape that minimizes transfer operations. Finally, needles are assigned to all loops using the computed shapes, and instructions can be generated.

During instruction generation, where [Narayanan et al. 2018]'s scheduler only ran "knit" operations, our scheduler needs to respect the programs stored with each knitting face. To execute a pass, our scheduler emits instructions as follows: First, any pre-pass programs associated with the basic stitch type of the fragments are invoked with the storage locations of the previous and current passes. Next, transfer operations (planned by the method of [McCann et al. 2016]) are executed to align locations. Finally, each face included in the pass is generated by calling the program associated with the face on the storage locations associated with that face's fragments.

Pre-pass programs can be used for custom transfer planning and for reordering existing loops in a user defined manner and are invoked by passing the locations computed by the transfer planner for *all* the stitches active on the bed and for the stitches participating in the pass with the function signature:

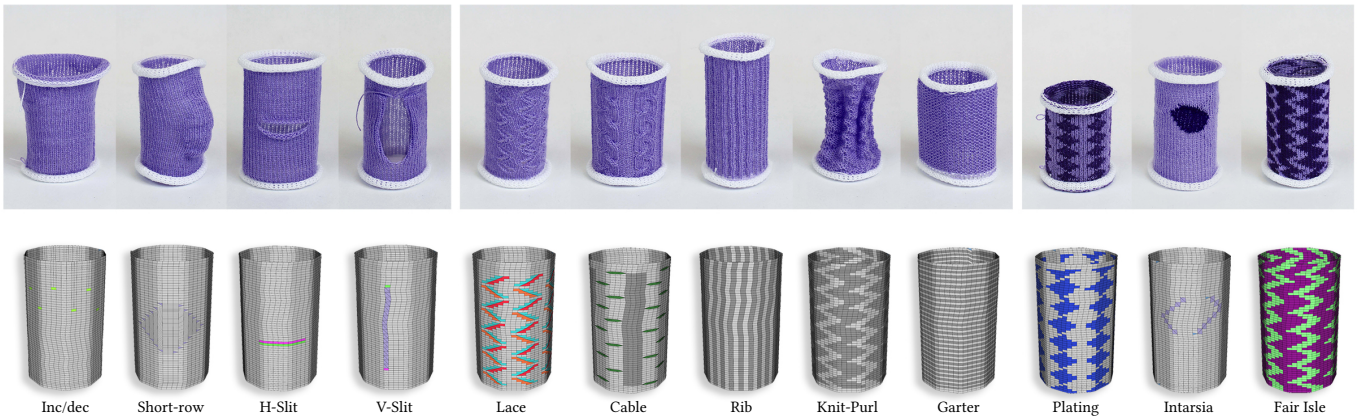
```
function pre_pass(from, to, pass_from, pass_to)
```

The face program associated with the stitch mesh face is invoked by passing the yarn direction determined during tracing, the needle allocations determined by the scheduler, and the ID(s) of the current yarn(s). Instead of invoking a single function, the face programs are divided into a preamble, main execution and a postamble that share the same signature:

```
function face_*(dirs, bns, carrier)
```

These three functions allow for coarse instruction re-ordering, which we will discuss further after introducing an non-trivial face program:

Purl Faces. A purl (i.e., a back knit) constructs the stitch on the *opposite* bed. This simple alteration generates a backward facing loop which appears structurally different from a forward facing loop. A combination of front and back stitches can give rise to a wide variety of patterns (Figure 10, columns 7-9). The face program for a purl first transfers the loop held in the storage location to the



(a) Shaping (b) Texturing (c) Colorwork

Fig. 10. ***A sampler of knitting techniques available in our system:*** *(a), increase/decrease shaping, short-row shaping, a horizontal slit using bind-off and cast-on, and a vertical slit using C-knitting; (b), lace with ordered increases and decreases, cables, rib (alternating columns of knits and purls), a complex knit-purl pattern, and garter (alternating rows of knits and purls); and (c), colorwork with the plating, intarsia, and Fair Isle methods.*

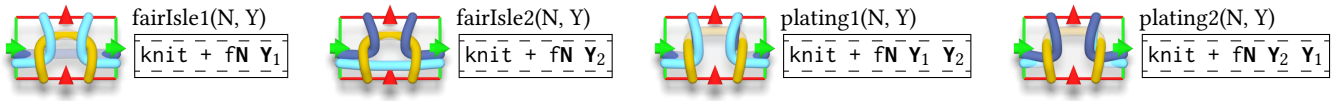


Fig. 11. *Additional face types created for Fair Isle and plating colorwork. We use* $\mathbf{Y}_1$ *and* $\mathbf{Y}_2$ *to denote the first and second elements of the yarn array,* **Y***, respectively.*

holding position on the other bed, knits on this new storage location and transfers the loop back. No pass level programs are necessary.

```
function purl_pre(dirs, bns, carrier){
  xfer(bns[0], opposite(bns[0]));
}
function purl_main(dirs, bns, carrier){
  knit(dirs[0], bns[0], carrier);
}
function purl_post(dirs, bns, carrier) {
  xfer(opp(bns[0]), bns[0]);
}
```

On a single system machine like the one used to fabricate our examples, knit and transfer instructions must be performed in separate carriage movements; this means that a row of N purl stitches – each of which requires an `xfer`, `knit`, `xfer` instruction chain – takes $3N$ carriage movements to fabricate if all the instructions are in the "main" function, but only 3 carriage movements if they are distributed across "pre", "main", and "post" functions.

8 RESULTS

We have used our pipeline to create a wide range of objects, which we discuss in this section. All results were knit on a Shima Seiki SWG091N2 industrial knitting machine with 15 needles per inch; the SWG*N2 series are basic v-bed machines typically used for hat, glove, and accessory production. All examples were knit from a 2-ply acrylic yarn (Tamm Petit). With this yarn and machine, a knit stitch is 2.88mm wide and 1.75mm tall; so we used this to set the face sizes in our interface. Knitout code for all our results is included in the supplementary material.

Fig. 12. *A sock pattern can be easily edited by adding color and ribs.*

Basic techniques. Thanks to the generality of the augmented stitch mesh structure, our system can be used to program a wide variety of common knitting techniques. A sampler of the techniques our system supports is presented in Figure 10. Of course, further techniques can be added by defining more face types.

Colorwork. Our system supports three common styles of knitting with multiple colors. *Intarsia* colorwork involves knitting different sections of the object with yarns of different colors. This is easy to achieve with our basic stitch types, but is relatively inflexible. In *Fair Isle* colorwork, yarns of different colors run along the inside of the pattern, and knit stitches are made out of whichever color the

Fig. 13. *A knit skyline decorating a cylinder by picking plating face types from the image (top right)*

Fig. 14. *Inside-out view of colorwork patterns:* left*, Intarsia patterns are created by modifying yarn paths;* middle*, plating uses two colors for every stitch creating a double sided pattern;* right*, Fair Isle patterns have a distinctive look on the reverse side due to "floating" yarns not used in stitches.*

Fig. 15. *A hand-warmer designed and edited in our system.*

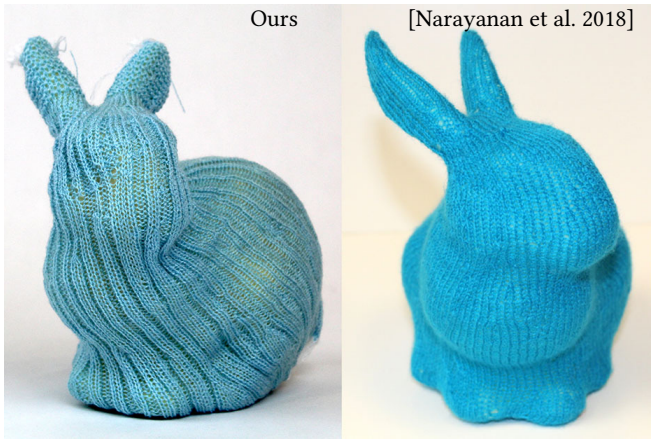


Fig. 16. *The Stanford bunny with ribs on the body and garter pattern on the ears. Textures created by knits and purls generates a distinct look in comparison to the image from the supplementary material in [Narayanan et al. 2018] shown on the right.*

designer wants to be shown. We implemented this in our system using faces that pass multiple yarns along their yarn edges but knit with only one of them (Figure 11, left). Figure 12 shows the addition of ribs and Fair Isle colorwork in a basic sock pattern. Manually editing the face types of the initial mesh took around 60 minutes for this pattern. Finally, in *plating* colorwork, two yarns are used in every stitch, with one running slightly in front of the other in order to make it appear on the front of the stitch. This has the advantage of making a reversible pattern, with the downside of some loss of contrast. This, again, was simple to implement in our system with special plating face types (Figure 11, right). The two-sided decorative item showing a skyline in Figure 13 was created using plating and a texture map to guide face selection. Figure 14 shows the inside of the basic colorwork samples to highlight their differences.

Adding detail. Our system can be used to add fine-scale detail to automatically-generated patterns. Functional and decorative knit and purl textures as well as cable patterns can be easily created by switching face types. We created a hand warmer, Figure 15, from a tube by introducing a vertical slit using modified (tuckless) short-row turn faces – a technique known as *C knitting* – and decorating it with ribs and cable faces. Starting from the tube, these edits took less than 15 minutes to finish. The finished product has a significantly more interesting surface than could be produced by previous work on high-level design for knitting machines (e.g., Figure 14 in [McCann et al. 2016]).

We also used our system to add surface texture to a classic computer graphics model (Figures 1, 16), producing a much more interesting surface appearance than was possible in previous systems. Editing the texture involved editing face types throughout the pattern and took around 60-90 minutes. Our system incorporates basic block-level pattern specification tools for tiling patterns. In general, introducing more mid-level patterning tools would further reduce editing time.

Switching yarn types can be used to not only introduce color but also to vary yarn properties. A popular choice is using conductive yarns for creating soft sensors or embedding functional electronics [Ou et al. 2019; Perner-Wilson et al. 2011]. In Figure 17, our system was used to easily add conductive yarn "rails" to carry current to LEDs in a cap.

Shaping adjustment. Professionally-designed knitting programs often feature carefully aligned increases and decreases, while current automatic generation approaches place them somewhat sporadically

Fig. 17. *Conductive yarn (cyan faces) can be integrated into the body of the object for incorporating electronics.*

– this produces higher shape fidelity, but can appear messy. Our system's editing tools makes it easy to reposition shaping features, as can be seen in the sweater example in Figure 18.

Improving robustness. Narayanan et al. [2018] reported that in their fully automatic stitch generation system, patterns could occasionally fail to properly knit owing to suboptimal schedules or extreme shaping. In our system, such meshes can be edited to avoid these situations. For example, the sweater pattern for the bear had a large number of concentrated decreases that caused the resulting pattern to fail. Our editing tool allows those decreases to be staggered across rows to produce a similar but robust pattern (Figure 19). These edits were performed in under 15 minutes.

Complex Results. Beginning from the same input mesh of a sweater, variations can be quickly created by editing the pattern using our system (Figure 20). By introducing paired increases and decreases with specific stacking orders, lace can be created. By stamping a simple pattern over the body of the sweater, *Fair Isle* style colorwork and details can be added. By directly modifying yarn path using the edit operations *Intarsia* style colorwork can be created. Starting from the initial mesh, each pattern required around 20 minutes of manual editing for generating the variations.

By using a combination of our automatic remeshing tool, editing tools, and scheduling system, novel three dimensionally shaped seamless knit objects can be designed and fabricated in an intuitive way (Figure 21).

9 LIMITATIONS AND FUTURE DIRECTIONS

In this paper, we have introduced a flexible and useful visual editor for machine knitting programs. However, there remain areas for improvement in our system.

In our system, remeshing runs before faces types are selected, so it must assume all faces are the same, generic, size. In practice, not only can face programs modify the machine's stitch size settings, but faces such as cables or combinations of adjacent knits and purls can introduce desirable but pronounced local deformation [Knittel et al. 2015]. This means that editing can change the shape of the object in ways that the initial remeshing did not account for. In the future, this could be addressed by including some coarse local stitch sizing information when doing the initial remeshing, or by adding real time simulation into the design pipeline to provide clear feedback on deformations. Indeed, incorporating predictive yarn simulation and rendering would greatly improve the overall user experience.

By design, none of our editing operations can introduce a surface topology change. Introducing this class of edits with knittability guarantees would require quickly verifying that the mesh retains an upward planar embedding, which is hard [Garg and Tamassia 2001]; however, incremental re-verification approaches may make this tractible for interactive editing.

An implementation limitation that is worth pointing out involves *balancing* faces participating in splits and merges so that a valid layout exists (Property 6 in [Narayanan et al. 2018]). Although easy to detect, our system does not implicitly maintain balance and relies on the user to ensure that shaping edits on the splitting/merging cycles are symmetric. Similarly, our current implementation allows mapping logical yarns to two (or more) physical yarns to support plating and Fair Isle colorwork; however, it does not allow faces that merge or split logical yarns. Supporting such faces should not require any major algorithmic changes.

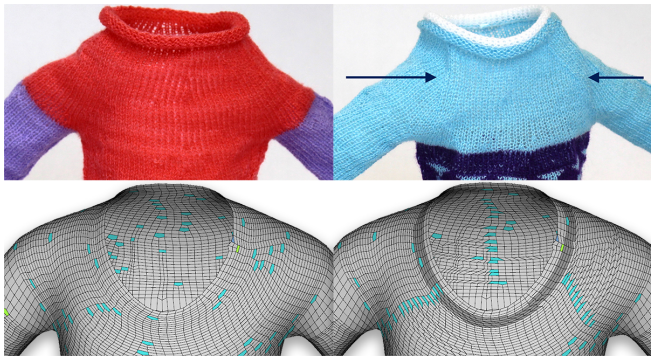


Fig. 18. *By aligning shaping edits using our system (right), a more traditional appearance can be created.*

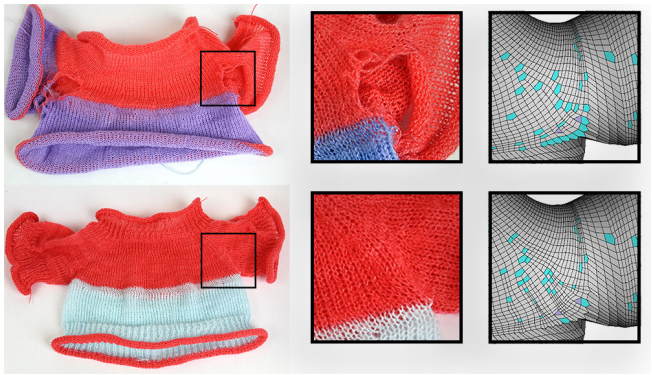


Fig. 19. *Clustered decreases in one row (top) were distributed to improve the robustness of the sweater pattern (bottom). Notice the holes under the sleeves in the top example.*

Fig. 20. *Multiple variations – including lace (center) and colorwork (intarsia stripes left, Fair Isle right) – on the same base pattern can be easily introduced.*

Our system remains agnostic to machine specifications such as choice of yarn carrier and yarn carrier position. However, when multiple yarn carriers are active, their exact position and relative motion can potentially introduce yarn tangling. A possible extension to our system's instruction generation phase would detect these machine configurations and trigger special-case logic to avoid them, e.g., by adding extra dependencies between stitches or providing scheduling hints.

Our scheduler can only perform resource allocation for *tubular* surfaces (and flat surfaces that can be viewed as a part of a tube); in part, this is because it depends on having a specific arrangement of free needles to allow it to re-arrange held loops. Therefore, it cannot handle techniques that require specific global layouts of stitches (e.g., fiber inlay), or that occupy needles in a way that prevent flexible racking adjustments (e.g., complex full-gauge flat-knitting patterns). We believe that, in the future, additional constraints can be introduced into the scheduling process to handle these cases, though more user intervention may be required to produce valid schedules.

Our system makes use of an automatic transfer planning algorithm that can perform substantially worse than hand-designed solutions for the same loop movements [Lin et al. 2018]. This is acceptable for prototyping, but may be a problem in manufacturing. Better instruction generation through improved planning and scheduling algorithms – essentially, the work of building better compilers for machine knitting – remains an interesting open problem.

Finally, our system shows that the augmented stitch mesh is useful for representing and editing machine knittable structures. However, it is a research prototype. Performing usability testing and refinement, especially around mid-level editing tools, is an interesting avenue for future research.

10 CONCLUSION

In this paper, we demonstrated a new visual programming tool for computer-controlled knitting machines. The core of our approach is an *augmented* stitch mesh data structure which associates each face in a mesh with machine knitting instructions and maintains dependency information between the faces. Our system allows users to automatically create a machine-knittable augmented stitch mesh from a 3D model, edit this augmented stitch mesh while preserving machine-knittability, and transform the mesh into a knitting program for machine fabrication.

Knit programmers are still doing the same thing they were eighty years ago. Though the media has changed – from cam cylinders to card chains to paper tape to floppy disks to flash drives and ftp servers – programmers still explicitly tell the machine what to do with each needle on each carriage pass. In contrast, our system allows a user to manipulate output structures, describing exactly what the machine should make. This shift – from the *how* of fabrication to the *what* of the finished product – is empowering and delightful.

This is the knitting design tool we have always wanted, and we are exceptionally pleased to have finally created it.

ACKNOWLEDGMENTS

We thank Lea Albaugh and Justin Macey for their support on this project. The bunny model is provided by the Stanford Computer Graphics Laboratory. The sock model (product:1354042) by Flindigo is provided by Turbosquid. Kui Wu is supported in part by University of Utah Graduate Research Fellowship.

REFERENCES

Carlos Aliaga, Carlos Castillo, Diego Gutierrez, Miguel A. Otaduy, Jorge Lopez-Moreno, and Adrian Jarabo. 2017. An Appearance Model for Textile Fibers. *Computer Graphics Forum* 36, 4 (2017), 35–45.

Aric Bartle, Alla Sheffer, Vladimir G. Kim, Danny M. Kaufman, Nicholas Vining, and Floraine Berthouzoz. 2016. Physics-driven Pattern Adjustment for Direct 3D Garment Editing. *ACM Trans. Graph.* 35, 4 (Jul 2016), 1âĂŞ11.

Sarah-Marie Belcastro. 2009. Every Topological Surface Can Be Knit: A Proof. *Journal of Mathematics and the Arts* 3, 2 (2009), 67–83.

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

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* 33, 4, Article 154 (July 2014), 11 pages.

Fig. 21. *Stuffed animals with accessories designed in our system. The bear (left) is wearing a custom beanie with slits for ears and a matching sweater. The cat (right) is wearing a cap with knit and purl patterns and a sweater with cable work and ribbed sleeves.*

Michel Carignan, Ying Yang, Nadia Magnenat Thalmann, and Daniel Thalmann. 1992. Dressing Animated Synthetic Actors With Complex Deformable Clothes. *ACM SIGGRAPH'92* (1992), 99–104.

Yanyun Chen, S. Lin, Hua Zhong, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. 2003. Realistic rendering and animation of knitwear. *IEEE Transactions on Visualization and Computer Graphics* 9, 1 (Jan 2003), 43–55.

Gabriel Cirio, Jorge Lopez-Moreno, David Miraut, and Miguel A. Otaduy. 2014. Yarn-level Simulation of Woven Cloth. *ACM Trans. Graph.* 33, 6, Article 207 (Nov. 2014), 11 pages.

Gabriel Cirio, Jorge Lopez-Moreno, and Miguel A. Otaduy. 2015. Efficient Simulation of Knitted Cloth Using Persistent Contacts. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '15)*. ACM, 55–61.

G. Cirio, J. Lopez-Moreno, and M. A. Otaduy. 2017. Yarn-Level Cloth Simulation with Sliding Persistent Contacts. *IEEE Transactions on Visualization and Computer Graphics* 23, 2 (Feb 2017), 1152–1162.

Phillipe Decaudin, Dan Julius, Jamie Wither, Laurence Boissieux, Alla Sheffer, and Marie-Paule Cani. 2006. Virtual Garments: A Fully Geometric Approach for Clothing Design. *CG Forum (Eurographics)* 25, 3 (2006), 625–634.

Mario Deuss, Anders Holden Deleuran, Sofien Bouaziz, Bailin Deng, Daniel Piker, and Mark Pauly. 2015. *ShapeOp—A Robust and Extensible Geometric Modelling Paradigm*. Springer International Publishing, Cham, 505–515.

Shen Dong, Scott Kircher, and Michael Garland. 2005. Harmonic Functions for Quadrilateral Remeshing of Arbitrary Manifolds. *Computer Aided Geometric Design* 22, 5 (2005), 392–423.

Ashim Garg and Roberto Tamassia. 2001. On the Computational Complexity of Upward and Rectilinear Planarity Testing. *SIAM J. Comput.* 31, 2 (2001), 601–625.

E. Groller, R. T. Rau, and W. Strasser. 1995. Modeling and visualization of knitwear. *IEEE Transactions on Visualization and Computer Graphics* 1, 4 (Dec 1995), 302–310.

Eduard Gröller, René T Rau, and Wolfgang Straßer. 1996. Modeling Textiles as Three Dimensional Textures. In *Rendering TechniquesâĂŹ 96*. Springer, 205–214.

Donald House and David Breen. 2000. *Cloth modeling and animation*. AK Peters/CRC Press.

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

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008b. Knitty: 3D Modeling of Knitted Animals with a Production Assistant Interface. In *Eurographics*.

Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. 2010. A Radiative Transfer Framework for Rendering Materials with Anisotropic Structure. *ACM Trans. Graph.* 29, 4, Article 53 (2010), 13 pages.

Chenfanfu Jiang, Theodore Gast, and Joseph Teran. 2017. Anisotropic Elastoplasticity for Cloth, Knit and Hair Frictional Contact. *ACM Trans. Graph.* 36, 4, Article 152 (July 2017), 14 pages.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2008. Simulating Knitted Cloth at the Yarn Level. *ACM Trans. Graph. (SIGGRAPH'08)* 27, 3 (2008), 65.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2010. Efficient Yarn-based Cloth with Adaptive Contact Linearization. *ACM Trans. Graph. (SIGGRAPH'10)* 29, 4 (2010), 105.

Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. 2015. Matching Real Fabrics with Micro-Appearance Models. *ACM Trans. Graph.* 35, 1, Article 1 (2015), 26 pages.

Chelsea Knittel, Diana Nicholas, Reva Street, Caroline Schauer, and Genevieve Dion. 2015. Self-Folding Textiles through Manipulation of Knit Stitch Architecture. *Fibers* 3, 4 (Dec 2015), 575âĂŞ587.

Jonathan Leaf, Rundong Wu, Eston Schweickart, Doug L. James, and Steve Marschner. 2018. Interactive Design of Yarn-Level Cloth Patterns. *ACM Trans. Graph. (Proceedings of SIGGRAPH Asia 2018)* 37, 6 (11 2018).

Minchen Li, Alla Sheffer, Eitan Grinspun, and Nicholas Vining. 2018. FoldSketch: Enriching Garments with Physically Reproducible Folds. *ACM Trans. Graph.* 37, 4 (2018).

Jenny Lin, Vidya Narayanan, and James McCann. 2018. Efficient Transfer Planning for Flat Knitting. In *Proceedings of the 2Nd ACM Symposium on Computational Fabrication (SCF '18)*. ACM, New York, NY, USA, Article 1, 7 pages.

Jorge Lopez-Moreno, David Miraut, Gabriel Cirio, and Miguel A. Otaduy. 2015. Sparse GPU Voxelization of Yarn-Level Cloth. *Computer Graphics Forum* (2015), 1–13.

Fujun Luan, Shuang Zhao, and Kavita Bala. 2017. Fiber-Level On-the-Fly Procedural Textiles. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 123–135.

James McCann. 2017. The "Knitout" (.k) File Format. [Online]. Available from: https://textiles-lab.github.io/knitout/knitout.html.

James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. 2016. A Compiler for 3D Machine Knitting. *ACM Trans. Graph.* 35, 4, Article 49 (July 2016), 49:1–49:11 pages.

Michael Meißner and Bernd Eberhardt. 1998. The art of knitted fabrics, realistic & physically based modelling of knitted patterns. In *Computer Graphics Forum*, Vol. 17. Wiley Online Library, 355–362.

Yuki Mori and Takeo Igarashi. 2007. Plushie: An Interactive Design System for Plush Toys. *ACM Trans. Graph. (SIGGRAPH'07)* 26, 3 (2007), 45.

Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James McCann. 2018. Automatic Machine Knitting of 3D Meshes. *ACM Trans. Graph.* 37, 3, Article 35 (Aug. 2018), 15 pages.

Jifei Ou, Daniel Oran, Don Derek Haddad, Joseph Paradiso, and Hiroshi Ishii. 2019. SensorKnit: Architecting Textile Sensors with Machine Knitting. *3D Printing and Additive Manufacturing* 6, 1 (2019), 1–11.

Chi-Han Peng and Peter Wonka. 2013. Connectivity Editing for Quad-dominant Meshes. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing (SGP '13)*. Eurographics Association, 43–52.

Hannah Perner-Wilson, Leah Buechley, and Mika Satomi. 2011. Handcrafting Textile Interfaces from a Kit-of-no-parts. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '11)*. ACM, 61–68.

Mariana Popescu, Matthias Rippmann, Tom Van Mele, and Philippe Block. 2018. Automated Generation of Knit Patterns for Non-developable Surfaces. In *Humanizing Digital Reality*, De Rycke K. et al. (Ed.). Springer, Singapore.

Kai Schröder, Shuang Zhao, and Arno Zinke. 2012. Recent Advances in Physically-based Appearance Modeling of Cloth. In *SIGGRAPH Asia 2012 Courses (SA '12)*. ACM, New York, NY, USA, Article 12, 52 pages.

Shima Seiki. 2011. SDS-ONE Apex3. [Online]. Available from: http://www.shimaseiki.com/product/design/sdsone_apex/flat/.

Soft Byte Ltd. 1999. Designaknit. [Online]. Available from: https://www.softbyte.co.uk/designaknit.htm.

David J Spencer. 2001. *Knitting technology: a comprehensive handbook and practical guide*. Vol. 16. CRC press.

Stoll. 2011. M1Plus pattern software. [Online]. Available from: http://www.stoll.com/stoll_software_solutions_en_4/pattern_software_m1plus/3_1.

Shinjiro Sueda, Garrett L. Jones, David I. W. Levin, and Dinesh K. Pai. 2011. Large-scale Dynamic Simulation of Highly Constrained Strands. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 39, 10 pages.

Emmanuel Turquin, Jamie Wither, Laurence Boissieux, Marie-Paule Cani, and John Hughes. 2007. A Sketch-based Interface for Clothing Virtual Characters. *IEEE Comp. Graph. and Applications* 27, 1 (2007), 72–81.

Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. 2011. Sensitive Couture for Interactive Garment Editing and Modeling. *ACM Trans. Graph. (SIGGRAPH'11)* 30, 4 (2011), 90.

Jenny Underwood. 2009. *The design of 3D shape knitted preforms*. Ph.D. Dissertation. Fashion and Textiles, RMIT University.

Pascal Volino and Nadia Magnenat-Thalmann. 2000. *Virtual Clothing: Theory and Practice*. Springer.

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 (2009), 105.

Hao Wang. 1961. Proving Theorems by Pattern Recognition II. *Bell System Technical Journal* 40 (1961), 1–42.

Huamin Wang. 2018. Rule-free Sewing Pattern Adjustment with Precision and Efficiency. *ACM Trans. Graph.* 37, 4, Article 53 (July 2018), 13 pages.

Tuanfeng Y. Wang, Duygu Ceylan, Jovan Popović, and Niloy J. Mitra. 2018. Learning a Shared Shape Space for Multimodal Garment Design. *ACM Trans. Graph.* 37, 6, Article 203, 13 pages.

Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. 2018. Stitch Meshing. *ACM Trans. Graph. (Proceedings of SIGGRAPH 2018)* 37, 4, Article 130 (jul 2018), 14 pages.

Kui Wu, Hannah Swan, and Cem Yuksel. 2019. Knittable Stitch Meshes. *ACM Trans. Graph.* 38, 1, Article 10 (Jan. 2019), 13 pages.

Kui Wu and Cem Yuksel. 2017a. Real-time Cloth Rendering with Fiber-level Detail. *IEEE Transactions on Visualization and Computer Graphics* PP, 99 (2017), 1–1.

Kui Wu and Cem Yuksel. 2017b. Real-time Fiber-level Cloth Rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D 2017)*. ACM, New York, NY, USA, 8.

Ying-Qing Xu, Yanyun Chen, Stephen Lin, Hua Zhong, Enhua Wu, Baining Guo, and Heung-Yeung Shum. 2001. Photorealistic Rendering of Knitwear Using the Lumislice. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 391–398.

Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2012. Stitch Meshes for Modeling Knitted Clothing with Yarn-level Detail. *ACM Trans. Graph. (Proceedings of SIGGRAPH 2012)* 31, 3, Article 37 (2012), 12 pages.

Hao Zhang, Oliver Van Kaick, and Ramsay Dyer. 2010. Spectral Mesh Processing. In *Computer graphics forum*, Vol. 29. Wiley Online Library, 1865–1894.

Shuang Zhao, Wenzel Jakob, Steve Marschner, and Kavita Bala. 2011. Building Volumetric Appearance Models of Fabric Using Micro CT Imaging. *ACM Trans. Graph.* 30, 4, Article 44 (2011), 10 pages.

Shuang Zhao, Fujun Luan, and Kavita Bala. 2016a. Fitting Procedural Yarn Models for Realistic Cloth Rendering. *ACM Trans. Graph.* 35, 4, Article 51 (2016), 11 pages.

Shuang Zhao, Lifan Wu, Frédo Durand, and Ravi Ramamoorthi. 2016b. Downsampling Scattering Parameters for Rendering Anisotropic Media. *ACM Trans. Graph.* 35, 6 (2016).