Automatic Machine Knitting of 3D Meshes

VIDYA NARAYANAN, Carnegie Mellon University
LEA ALBAUGH[1], Carnegie Mellon University
JESSICA HODGINS[1], Carnegie Mellon University
STELIAN COROS, ETH Zürich and Carnegie Mellon University
JAMES MCCANN[1], Carnegie Mellon University

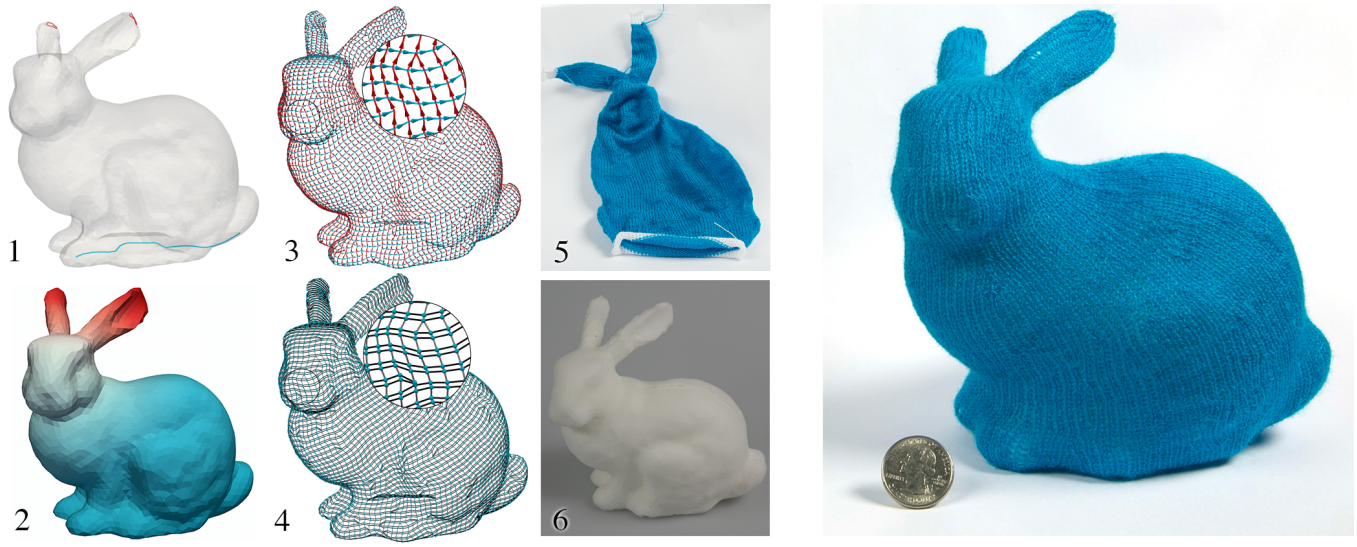


Fig. 1. Automatically knitting the Stanford bunny. Our system begins with an input mesh with user-specified starting and ending cycles (1) that are interpolated to define a knitting time function (2), remeshes the surface to create a row-column graph that represents the knit structure (3), traces the graph (4) and generates stitch instructions that are scheduled for machine knitting. The machine-knit skin (5) is a good fit for a foam replica of the input model (6).

We present the first computational approach that can transform 3D meshes, created by traditional modeling programs, directly into instructions for a computer-controlled knitting machine. Knitting machines are able to robustly and repeatably form knitted 3D surfaces from yarn, but have many constraints on what they can fabricate. Given user-defined starting and ending points on an input mesh, our system incrementally builds a helix-free, quad-dominant mesh with uniform edge lengths, runs a tracing procedure over this mesh to generate a knitting path, and schedules the knitting instructions for this path in a way that is compatible with machine constraints. We demonstrate our approach on a wide range of 3D meshes.

CCS Concepts: • **Computing methodologies** → *Graphics systems and interfaces*; • **Applied computing** → *Computer-aided manufacturing*;

Additional Key Words and Phrases: knitting, CAM, automatic knitting, 3D printing

0730-0301/2018/1-ART1 $$15.00
https://doi.org/10.1145/3186265

ACM Reference format:
Vidya Narayanan, Lea Albaugh[1], Jessica Hodgins[1], Stelian Coros, and James McCann[1]. 2018. Automatic Machine Knitting of 3D Meshes. *ACM Trans. Graph.* 1, 1, Article 1 (January 2018), 15 pages.
https://doi.org/10.1145/3186265

1 INTRODUCTION

Computer-controlled knitting machines are widely used in the textiles industry for the production of accessories, like hats and gloves, as well as full garments. The most advanced among these machines are able to shape yarn into complex, 3D items ready for use with minimal post-processing.

However, these items currently must be laboriously hand-designed using machine-knitting-specific tools that are only accessible to experts.

In this paper we present an algorithmic approach to automatically converting 3D shapes into knitting machine instructions (Figure 1). Our method allows designers to generate knit versions of digital models created with software of their choosing, without requiring a detailed understanding of how machine knitting works. In essence, this makes knitting machines as easy to use as 3D printers.

[1]Work partially done at Disney Research Pittsburgh.

Our paper is the first to demonstrate a practical scheme for automatically machine knitting a wide variety of 3D surfaces. Specifically, we contribute:

- A succinct statement of when a 3D surface can be machine knit.
- A field-guided remeshing procedure that produces a knitting machine compliant graph structure that approximates the input surface.
- A tracing algorithm that can transform such graphs into low-level knitting operations.
- A scheduling algorithm that can assign machine needle locations to these operations.

2 BACKGROUND

Commercial Software. Commercial knitting design software offers users the chance to design with a variety of templates [Shima Seiki 2011; Stoll 2011]. These templates allow for some variation within a limited design scope. Beyond these examples, knit shapes must still be designed at the stitch level, though cookbooks do exist that provide stitch-by-stitch recipes for various complex shapes [Underwood 2009]. Recently, researchers have expanded this scope by offering general tube and sheet primitives in the context of a knitting compiler [McCann et al. 2016]. However, their approach still requires a designer to place and configure these primitives by hand. In contrast, we demonstrate an automatic approach built on a more flexible low-level description.

Hand Knitting. Igarashi et al. proposed a design assistant [Igarashi et al. 2008b] and automatic design from a mesh [2008a] for hand knitting with increase-decrease shaping. This approach is excellent for hand knitting, where working on multiple components and then assembling makes patterns easier to read; however, it does not take advantage of the capabilities of knitting machines to execute more complex patterns and knit whole objects in one piece. We provide results on similar models for visual comparison to this approach in Figure 19 (compare to Igarashi et al.'s Figure 11 [2008a]). Belcastro provides a constructive proof that a 2D surface of any topology can be hand knit [Belcastro 2009]. This approach relies on hand knitting capabilities and does not focus on matching a specified geometry.

Machine Knitting. Recently, Popscu et al. described a system for building machine-knit 3D objects from quad meshes [Popscu et al. 2018]. However, many intermediate steps including patch segmentation, machine layout and scheduling, and connecting patches into the final shape remain manual in their system.

Simulation. Yarn- and stitch-level knit simulation tools, for example, [Cirio et al. 2015; Kaldor et al. 2008, 2010; Meißner and Eberhardt 1998] produce impressive visualizations of knit structures. While their goal is to generate predictive models for deformation behavior, the patterns generated may not be valid for machine knitting. Stitch meshes are built upon carefully designed input surfaces and allow interactive editing of knitting patterns for simulation [Yuksel et al. 2012]. Our goal is to generate machine knitable instructions from an arbitrary triangulated surface mesh.

Unfolding. Our system produces a close approximation of an input 3D shape with knitting, which has the advantage of not needing to be intrinsically flat. Many researchers have considered the related problem of building a 3D shape with flat sheets of fabric (or other stretchy material), e.g., [Guseinov et al. 2017; Mahdavi-Amiri et al. 2015; Skouras et al. 2014].

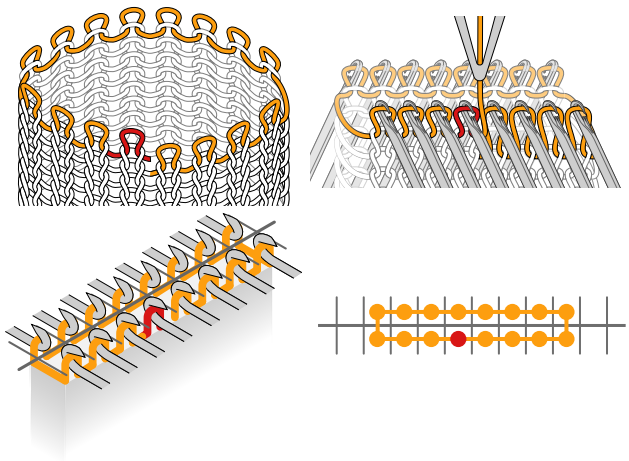


Fig. 2. The most recently knit edge of the cylinder – referred to as a cycle (**top left**) – is held flat on the two beds of the knitting machine (**top right**). A simplified view is shown for clarity (**bottom left**). We represent the cycle abstractly by a top-view of the needle bed where circles represent needles holding loops (**bottom right**). The last made stitch is shown in red.

2.1 Machine Knitting

Our method allows novice users to fabricate 3D surfaces on an industrial weft knitting machine. We begin by describing the constraints and operation of such machines.

V-bed knitting machines form objects by manipulating loops of yarn held by hook-shaped needles. The needles lie in parallel in one of two *needle beds*, which are angled toward each other in an inverted *V* shape (Figure 2, top-right). Each needle can hold a stack of loops and can be actuated to add a new loop (*tuck*), pull a new loop through the existing loops (*knit*), move the stack of loops to the opposite bed (*transfer*), and pull a new loop through a stack of loops while moving the stack of loops to the opposite bed (*split*). The principal challenge of machine knitting is that only loops of yarn currently held on needles can be manipulated by the machine, and the needles are in a fixed layout. The supplementary material includes a glossary of terms related to machine knitting used in the paper. For a more detailed mechanical introduction, we suggest the overview given by McCann et al. [2016].

For the purpose of this work, it is sufficient to think of knitting machines as devices for constructing generalized cylinders. Any in-progress cylinder is "flattened" and held on the machine's needle beds (See Figure 2). We call these flattened ends of generalized cylinders *cycles*. Cycles can be re-shaped and moved freely – translated and rotated – using the transfer-planning algorithm presented by McCann et al. [2016]. The only restrictions on this movement are that cycles cannot change their orientation, i.e., be reflected

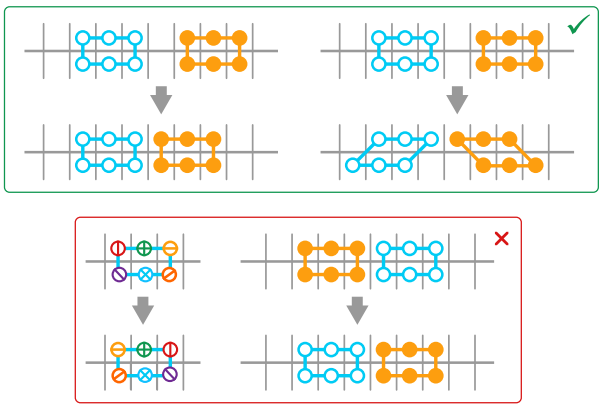


Fig. 3. Cycles may be rotated and translated on the bed of the knitting machine, but can not be reversed or pass through other cycles.

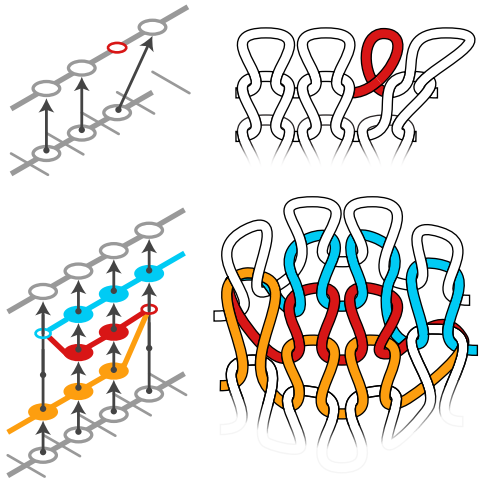


Fig. 4. Knit shaping techniques. **Top**, increase (decrease) shaping can change the local width of a fabric. **Bottom**, short-row shaping can change the local height. In each case, the left column shows construction steps while the right column shows the resulting loops. Knit stitches are shown as ⬬ and tucks as ○. Loops that are not knit through but held on the bed are shown as •. Yarn-wise connections are shown as ／ and loop-wise connections as ↑.

across the bed, and that cycles cannot cross each other on the bed (Figure 3).[1]

By adding loops of yarn to cycles in various ways, knitting machines can create generalized cylinders of different shapes. For instance, a cylinder can be widened by creating new loops, narrowed by stacking loops before knitting through them, or bent by knitting new yarn around only a portion of the circumference (Figure 4). The only constraint on these operations is that any new loop must be introduced adjacent to the last formed loop otherwise a long tail of yarn will be left to trail across the cylinder.

[1]Using an ideal yarn with no volume, cycles may be transferred to cross each other on the machine – in practice, this seldom works reliably.

It is also possible to merge multiple cycles into one by moving them to be adjacent on the needle bed, and then knitting over all of them to form a new cycle. Similarly, it is possible to split one cycle into several by knitting over only a portion with one yarn and later bringing in another yarn to knit another portion.

2.2 Knitability of arbitrary meshes

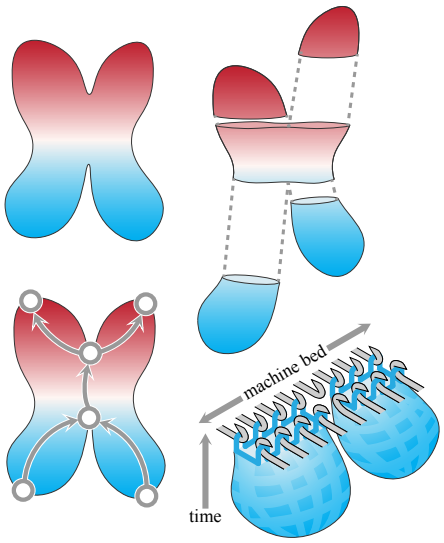


Fig. 5. An oriented 2D manifold $\mathcal{M}$ **(top left)** is knitable if there exists a monotonic time function f ( ) whose Reeb graph has an upward planar embedding **(bottom left)**. Arcs of the Reeb graph suggest a segmentation of the mesh into generalized cylinders **(top right)** that can be machine knit in one piece provided an upward planar embedding of the graph exists. The embedding ensures that cycles **(bottom right)** do not need to cross during fabrication.

Consider a continuous knitting machine that makes infinitesimal stitches; a natural question to ask is whether this machine can construct a knit model of an arbitrary 2D manifold $\mathcal{M}$.

This question is equivalent to asking if the manifold can be described as a set of generalized cylinders that can be held and shaped on the knitting machine's bed, subject to the constraints outlined above. Particularly, this immediately tells us that $\mathcal{M}$ must have at least two boundaries – for knitting to start and end – and must be oriented, because its constituent cylinders cannot change orientation while held on the machine.

Further, it must be the case that, during construction, none of the cycles cross. This property can be captured by defining a time function $f : \mathcal{M} \to [-1, 1]$ whose level sets are exactly the portions of the manifold held on the machine bed at a given step. Because only 1D curves can be held on the machine bed, f must never be constant over a 2D patch i.e., we require f to be a Morse function [Edelsbrunner and Harer 2010] and achieve local extrema only on the boundary of the manifold. By slightly perturbing the value of f, it can be guaranteed that the connected components of its level

sets consists of lines, circles or figure-8 shapes – all of which are reasonable to hold on the machine bed.

Identifying every connected component of a level set with a point produces a 1D skeleton of $\mathcal{M}$ known as its Reeb graph over f [Edelsbrunner and Harer 2010]. The arcs of this directed graph correspond to generalized cylinders, degree-1 nodes correspond to starting and ending points of cylinders, and remaining higher degree nodes represent locations where cylinders split and merge.

Now, in order to guarantee that no cylinders cross during construction, all we need to do is find an upward planar embedding of this Reeb graph (Figure 5). The planarity constraint avoids cycle crossings, while upwardness ensures that each loop's prerequisites are available when that loop is constructed.

In other words:

> An oriented 2D manifold-with-boundary $\mathcal{M}$ is knitable iff there exists a Morse function $f : \mathcal{M} \to [-1, 1]$ such that the Reeb graph of f has an upward-planar embedding.

The remainder of this paper implements this notion in a discrete setting.

3 METHOD

Our method takes as input an oriented, manifold 3D triangle mesh with two or more boundaries and a monotonic knitting time function specified by a scalar value at each vertex of the mesh. As an additional restriction, we require that the knitting time function be constant on boundaries. It produces knitting machine instructions that approximate the mesh in three main steps. First, *remeshing* creates a specially-structured knitting graph on the surface. Second, *tracing* creates knitting instructions from this graph. Finally, these instructions are *scheduled* to needle locations on the knitting machine. In this section, we describe the system in dataflow order, from input mesh to final output stitches.

3.1 Input

As a convenience, we developed a simple interface that allows users to generate suitable time functions for arbitrary meshes by specifying a sparse set of constraints along mesh edges (Figure 6). Our interface uses Laplacian interpolation to extend these constraints to a time function over the mesh (represented by the color map, -1 1, in the figures).

This tool was used to specify the time functions for all results shown in this paper.

3.2 Remeshing

Knit objects have an intrinsic row-column structure, where the rows arise from yarn-wise connections, and the columns arise from loop-wise connections. (See Figure 7).

In the remeshing phase, our method produces a directed graph to guide the row-column structure of the final knit object. This graph needs to be suitable for tracing, follow the input knitting time function, and approximate the input surface.

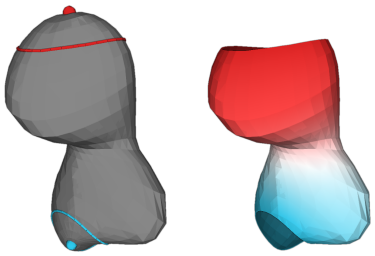


Fig. 6. Our time-field specification interface interpolates user-specified constraints to produce a smooth field.

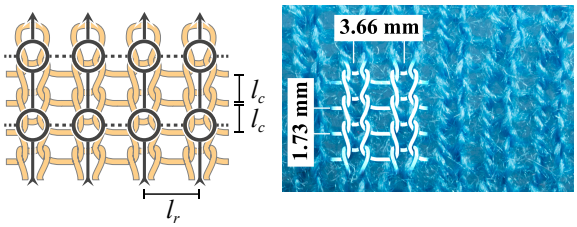


Fig. 7. Knit objects have a row-column structure. In our system, this structure is captured by the knit graph, whose nodes represent two stacked stitches (i.e., each row represents two physical *courses* or knit rows), and whose edges are constrained to be close to measured stitch dimensions (l_r, $2l_c$).

That is, remeshing creates a knit graph $(\mathcal{N}, \mathcal{R}, C)$:

$\mathcal{N} \equiv \{n, \ldots\}$	○	nodes
$\mathcal{R} \equiv \{(n_i, n_j), \ldots\}$	·····	directed row [yarn] edges
$C \equiv \{(n_i, n_j), \ldots\}$	↑	directed column [loop] edges

Each node of our remeshed graph represents two knit loops in the fabricated pattern, which is useful during tracing (Section 3.3).

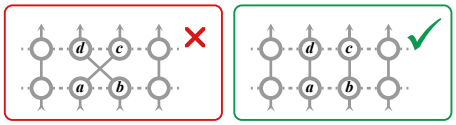
We first discuss the properties that make a directed graph knitable and then describe an iterative algorithm to construct a graph that obeys these properties.

Knit Graph Properties

Our remeshing procedure produces a knit graph with several important properties that are detailed below. To be suitable for tracing, the graph should have consistent orientation and be helix-free (Properties 1, 2). Further, to schedule on the machine, the graph must satisfy a limited node degree and allow a valid layout (Properties 3, 6). Finally, the graph needs to follow the time function and approximate the input geometry (Properties 7, 8). Properties 4 and 5 simplify the graph structure at short-rows and when cycles undergo a change in topology; although our knit graphs satisfy Property 4 and 5 by construction, they are not necessary conditions for knitability.

Property 1: Consistently Oriented. Adjacent rows should be consistently oriented. That is, for all nodes a, b, c, d:

$$(a, b) \in \mathcal{R} \wedge (a, c) \in C \wedge (b, d) \in C \implies (d, c) \notin \mathcal{R}$$

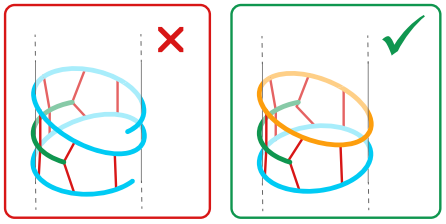


This reflects the fact that cycles on the machine bed cannot be reversed (Figure 3).

Property 2: Helix-Free. The column edges should form a partial order on the rows. That is, treating row edges as undirected, there should be no paths from the end of a column edge to its start.

$$\forall (a,b) \in C^+ : (b,a) \notin (\mathcal{R} \cup \text{reverse}(\mathcal{R}) \cup C)^+$$

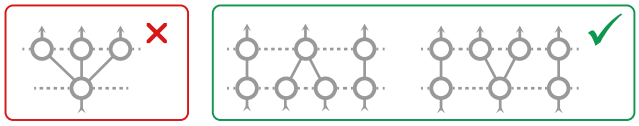
where $^+$ denotes transitive closure, i.e., paths in the graph.



Arbitrary helices break the row structure of knitting and cannot be traced in general although the final pattern generated for knitting a tube is, in fact, helical.

Property 3: Limited Node Degree. Each node n must correspond to a constructable type of stitch. That is, it should have at most two row edges (one in, one out) and at most two in and two out column edges.

$$\forall n, |\{(x,n) \in \mathcal{R}\}| \leq 1, |\{(n,x) \in \mathcal{R}\}| \leq 1,$$
$$|\{(n,x) \in C\}| \leq 2, |\{(x,n) \in C\}| \leq 2$$

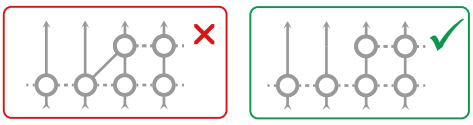


The constraint on row (yarn) edges is tight – stitches are always formed from one piece of yarn – while the constraint on column (loop) edges reflects the capabilities of the machine being used.

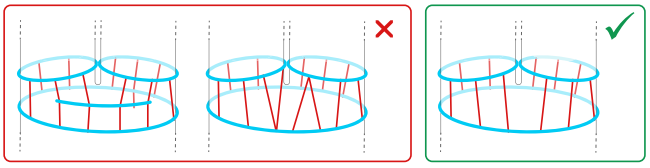
Particularly, handling multiple incoming column edges requires the machine to stack multiple loops on one needle, so is limited by the size of the needle's hook. Handling multiple outgoing column edges requires either splitting a loop, which can stress yarn, or casting-on additional stitches, which can create a small gap in the fabric. In both cases, we have chosen the conservative limit of two edges, though nothing in our pipeline intrinsically depends on these limits.

Property 4: Simple Short-rows. Each terminal end of a short-row must have a single in and out column edge.

$$\forall n, |\{(x,n) \in \mathcal{R}\}| = 0 \Rightarrow |\{(x,n) \in C\}| = 1 \wedge |\{(n,x) \in C\}| = 1$$
$$\forall n, |\{(n,x) \in \mathcal{R}\}| = 0 \Rightarrow |\{(x,n) \in C\}| = 1 \wedge |\{(n,x) \in C\}| = 1$$



Property 5: Simple splits and merges. At merges and splits, rows are complete cycle and column edges incident on its nodes are linked 1-1.



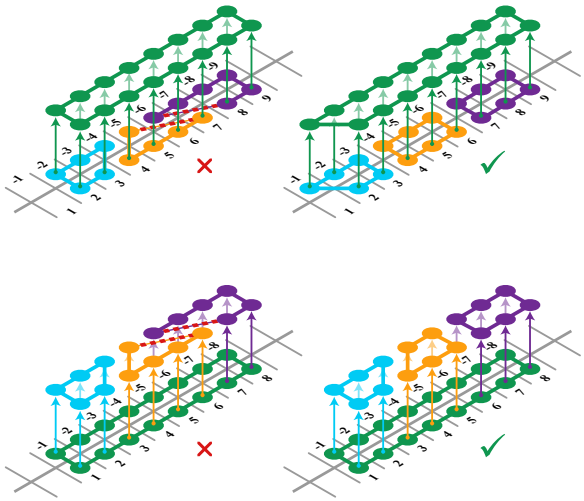
$$\forall (m,n),(m',n') \in C \text{ s.t. } (m,m') \in \mathcal{R}^+, (n,n') \notin \mathcal{R}^+ :$$
$$|\{(x,n) \in C\}| = |\{(m,x) \in C\}| = 1,$$
$$|\{(m,x) \in \mathcal{R}\}| = |\{x,m) \in \mathcal{R}\}| = 1,$$
$$|\{(n,x) \in \mathcal{R}\}| = |\{(x,n) \in \mathcal{R}\}| = 1$$

where, $\mathcal{R}^+$ indicates transitive closure i.e., paths along row edges.

This property ensures that each generalized cylinder starts and ends in a cycle (and not a short-row) and these cycles line up without requiring additional shaping operations. This property simplifies graph generation for tracing.

Properties 4 and 5 simplify linking and tracing. The constraints imposed by them additionally imply that increase and decrease shaping only occur between row-wise connected nodes.

Property 6: Feasible splits and merges. At splits and merges, nodes of both participating rows must have a feasible layout on the machine. Otherwise, splits and merges can cause yarn stress – red dashed lines show yarn stretching beyond stitch width, which cannot be laid out on the machine:



Let $\bar{C}, \bar{\mathcal{R}}, \bar{\mathcal{N}}$ be the column-edges, row-edges and nodes restricted to these rows respectively. A layout function l over $\bar{C}$, and its extension l_n over $\bar{\mathcal{N}}$, must exist such that rows in $\bar{\mathcal{R}}$ are *not stretched* for all participating cycles:

$$\exists \quad l : \bar{C} \to \mathbb{Z}, \quad l_n : \bar{\mathcal{N}} \to \mathbb{Z} \quad s.t. :$$
$$\forall (a,b),(a',b') \in \bar{C} : (a,b) \neq (a',b') \Rightarrow l(a,b) \neq l(a',b')$$
$$l_n(a) \equiv l_n(b) \equiv l(a,b)$$
$$\forall (a,b) \in \bar{\mathcal{R}} : \; | \, |l_n(a)| - |l_n(b)| \, | \leq 1$$

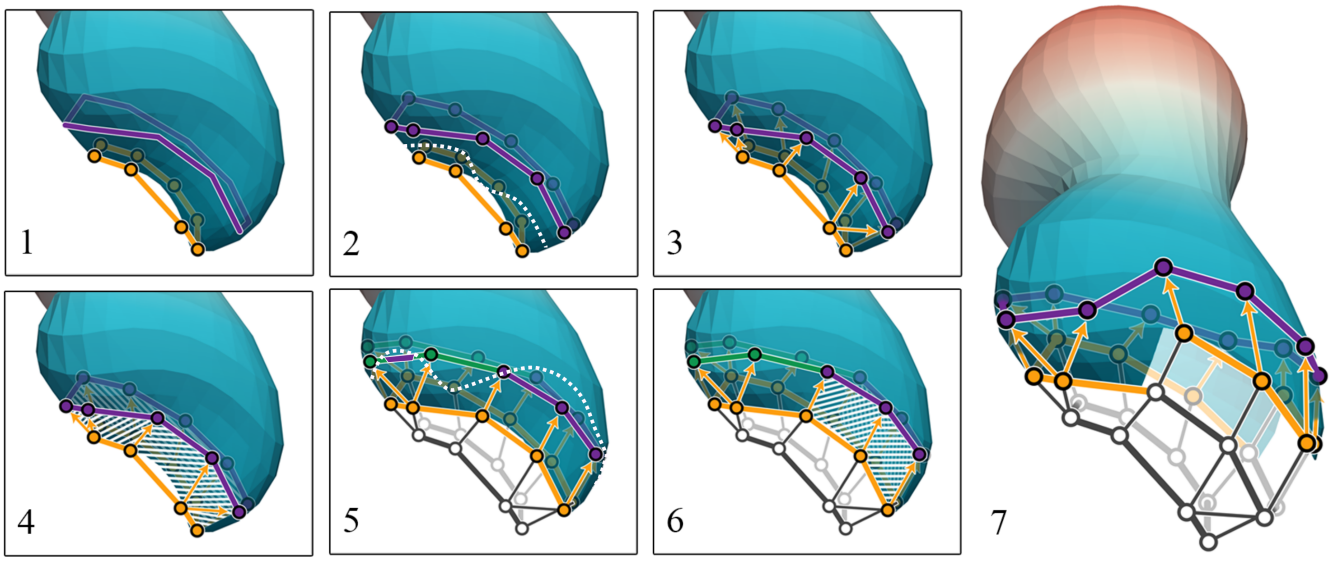


Fig. 8. During remeshing, a graph with row and column edges is constructed iteratively by: (1) finding a next cycle (purple) one row-height away from the current cycle (yellow), (2) adding nodes to the next cycle and marking nodes to keep (purple) or discard (green) based on the time function, (3) linking the nodes to the active cycle, (4) discarding the marked nodes and trimming the mesh, and (5-7) updating the active cycle and repeating. The level set of t_{active} is shown as a dotted white line. If the next cycle occurs after t_{active}, it is entirely accepted (2-3) otherwise segments that lie under the contour are accepted if longer than $2l_r$ (5-6).

By Property 5, the function l can be consistently extended as l_n from $\bar{C}$ to $\bar{\mathcal{N}}$.

Given a function l_n, a function l'_n can be constructed such that $\{l'_n(p_1)..., l'_n(p_n)\}$ is monotonic for every participating cycle $\{p_1, ..., p_n\}$ from some starting node p_1. Now, $\text{sgn}(l'_n(p))$ can be viewed as a needle bed and $|l'_n(p)|$ as a needle location. If the row edges are between adjacent needles (or across the bed), the yarn will not be stretched beyond the width of the stitch.

Property 7: Time-Aligned. The graph should respect the time field. That is, the column edges should increase in time and the row edges should remain about the same time:

$$\forall (a,b) \in C : \text{time}(a) < \text{time}(b)$$
$$\forall (a,b) \in \mathcal{R} : \text{time}(a) \approx \text{time}(b)$$

Rows can be monotonically ordered with respect to the given time function and when embedded on the mesh they follow the user defined time function i.e., row edges approximate level sets of the time function (Figure 9).

Property 8: Low Stretch. When the nodes are embedded in the surface, row and column edges should have lengths close to their corresponding knit features (Figure 7):

$$\forall (a,b) \in \mathcal{R} : \text{len}(a,b) \approx l_r$$
$$\forall (a,b) \in C : \text{len}(a,b) \approx 2l_c$$

where l_r is the measured width of a stitch, l_c is the measured height of a stitch, and len() measures distance along the surface.

Knit Graph Construction

In order to generate a graph with these properties, our algorithm proceeds by iteratively *slicing* the surface at a uniform distance from the boundary. Nodes are sampled on the slice and *linked* to previously generated rows. A portion of the linked region is then *trimmed* off the mesh, based on the time constraints. This process is repeated to generate a row-column graph over the entire mesh. An illustration is provided in Figure 8.

Our remeshing algorithm is motivated by field aligned quad meshing [Bommes et al. 2013], and quad mesh optimization [Bommes et al. 2011]. Our approach is similar to harmonic field based quad mesh generation [Dong et al. 2005], but incorporates knitability constraints during construction. Equally spaced stripe generation [Knöppel et al. 2015] can also be used to generate rows, however, in practice it can produce helices that need to be removed via post-processing.

3.2.1 Initialization. To start, all mesh boundaries that contain a local minima of the time function are added to the *active cycle* set. Nodes are sampled along these boundaries with even spacing, and adjacent nodes are connected with row edges. The number of nodes is chosen to ensure approximately l_r units between adjacent nodes, and the normals of the mesh are used to consistently orient the row edges with the surface to the left.

3.2.2 Slicing. Given a set of active cycles on the surface of the mesh, slicing generates a set of *next cycles*. Our code computes an approximate geodesic distance function on the surface of the mesh [Crane et al. 2013], starting at the active cycles, and takes the $2l_c$ level set of this function as the next cycles. The time function is linearly interpolated to all the vertices on the level set.

The next cycles so generated may not follow the time function. In order to guide the rows, our code trims the next cycles based on their

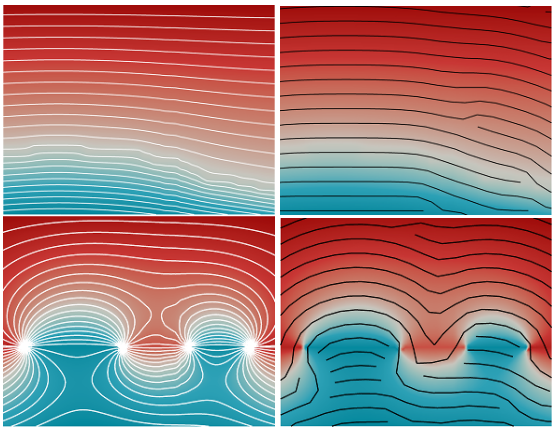


Fig. 9. The rows generated by our remeshing algorithm **(right)** align well with the contours of the time function **(left)**.

time values. Let t_{active} be the maximum time encountered along the active cycle. If the next cycle entirely appears after t_{active}, then it is accepted; otherwise, all portions of the next cycles with $t > t_{\text{active}}$ are marked for discard and the remainder ($t \leq t_{\text{active}}$) are marked for acceptance (see Figure 8, panels 5 and 6). If the next cycle is not entirely accepted, the accepted segments form "short-rows" i.e., partial, non-cyclic rows. If the length of such a segment is less than twice the stitch width $2l_r$, it is discarded, to avoid very small short-rows. If the first pass marked everything for discard, everything is re-marked for acceptance, i.e., if the entire cycle appears after t_{active} or if all segments that could have been accepted do not satisfy the minimum width $2l_r$.

Discarding segments based on time has the effect of keeping row edges approximately in line with the time function (Property 7), as demonstrated in Figure 9. Without this step, the generated patterns have no short-rows and can diverge from the time function, introducing arbitrary bind-off rows or rows that change shape rapidly (Figure 10). Of course, if a pattern without short-rows is desired,

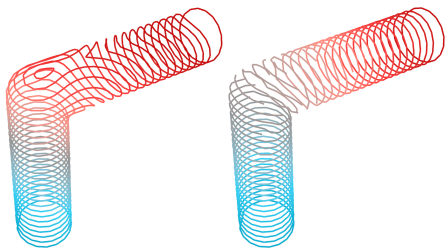


Fig. 10. Using geodesic distances only produces layouts of stitches with no short-rows **(left)**. The time function guides the knitting by introducing short-rows **(right)**.

that can still be achieved by setting the time function to the geodesic distance from the starting boundaries.

When the level set intersects a boundary, the entire boundary cycle is accepted as the next cycle. This guarantees that the next cycles are indeed cyclic and not chains, reducing the number of special cases required in our code. This can increase distortion of the shape close to the boundaries. In a production system, boundaries could be handled separately.

3.2.3 Linking. Linking is performed in two phases. First, *alignment pairs* are computed between active cycles and next cycles. The next cycles are adaptively sampled to generate nodes. Second, column edges are generated between the active and next cycle segments for each alignment pair.

Generating alignment pairs. Every node on the active cycle is assigned a target next cycle that is closest to it. Similarly, for every vertex (of the mesh) on the next cycle, a target active cycle is assigned that is closest to it. Segments of the active cycle and next cycle are paired for alignment if their targets match mutually (Figure 11). If a next cycle is not chosen by any active nodes or vice versa, the cycle is not considered for alignment in that iteration and is recomputed for subsequent iterations.

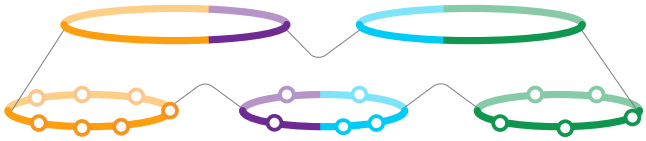


Fig. 11. Segments of the active cycle and next cycle are paired for alignment if their targets i.e., closest cycle, match mutually

Sampling. Alignment pairs allow adaptive sampling of the next cycles. The number of nodes required on the next segment is computed according to the stitch width, then clamped to maintain degree constraints (Properties 3, 4, 5). Stitch width is varied for each alignment pair to generate a valid number of next nodes.

Handling topology change. In some cases, an active cycle can link to multiple next cycles (or vice versa). We refer to this as a split (or merge). In this case, all nodes are marked accept to maintain Property 4. Further, care must be taken during linking to ensure that the final structure has a feasible layout on the knitting machine bed (Property 6). If splits and merges are strictly binary, cycles can be rotated on the bed to ensure balance during layout. In the case of ternary or higher splits and merges, our code explicitly balances the layout locally. This procedure is explained in the appendix A.

Linking nodes. Row edges are introduced between nodes on the next cycle. They are consistently oriented using surface normals. For each alignment pair, contiguous segments of nodes from the active cycle and the next cycle are linked to form column edges. As active and next cycles were extracted $2l_c$ distance apart from each other, our code now generates column edges that maintain this distance as closely as possible i.e., linking matches closest nodes by adding column edges subject to ordering (Property 1) and degree constraints (Properties 3, 4, 5).

Nodes in alignment pairs are linked to minimize cost. The cost of linking an active and next node is the squared distance between them on the surface of the mesh. The cost of linking an alignment pair is the sum of the costs of its constituent links. For each alignment pair

with nodes $\mathcal{A}$ in the active segment and nodes $\mathcal{N}$ the next segment with links $\mathcal{L}$ between them:

$$\begin{aligned} \text{cost}(\mathcal{A}, \mathcal{N}) \equiv \quad & \textstyle\sum_{(a,n)\in\mathcal{L}} \text{cost}(a, n) \\ a \in \mathcal{A}, n \in \mathcal{N} : \quad & \text{cost}(a, n) \equiv \text{len}^2(a, n) \end{aligned}$$

Our code determines optimal links using a dynamic-time-warping-like algorithm [Berndt and Clifford 1994] that considers all valid combinations of 1-1, 2-1, and 1-2 links.

3.2.4 Trimming. After generating links between the active cycles and the next cycles, the nodes on the next cycle marked for discard (along with any incident edges between them) are discarded. The remaining nodes and edges are added to the knit graph. Next, the portion of the mesh lying between the current active cycles and the *accepted* portion of the next cycles is removed. This trimming is accomplished by splitting the mesh along embedded edges of the knit graph and removing the faces bounded by row edges on the active and next cycle and column edges between them. Finally, the active cycle set is updated by reading off the nodes along the new mesh boundaries. Note that nodes on the boundary with outbound column edges are *not* considered part of the active cycles, as they have already been linked (panels 6 and 7 in Figure 8).

3.3 Tracing the knit graph

Tracing builds knitting instructions from the knit graph. Specifically, it traverses each node twice and makes two knit stitches at every node, connects nodes with yarn along row edges, connects nodes with loops along column edges, and traces the graph in the order defined by the edges.

The location of the current yarn is shown with an arrow (➔) designating its direction. The beginning ( ) and the end ( ) of a yarn are marked for clarity. Ends of short-rows are anchored by tucks ( ) where indicates *any* underlying node. In the figures below, column edges ( ) run bottom-to-top and row edges ( ) run left-to-right.

Tracing generates a list of knitting operations by traversing the knit graph using a set of local rules. These rules define an action based on the local context of the last last knitting operation performed with the current yarn. As it traverses, it will mark each node as knit once ( ) or knit twice ( ), and will query whether a node is ready ( ) or not ready ( ). Nodes are ready ( ) if all column-wise predecessors of nodes in their row have been knit twice i.e., . In the following figures, the state of the knit graph before applying the tracing rule is shown on the left for each figure, and after applying the rule is shown on the right.

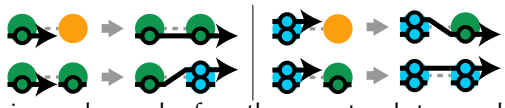
Tracing Rule 1. Start yarn:

If there is no current yarn, start a new yarn by knitting an arbitrary ready or once knit node.

Tracing Rule 2. Move to next row:

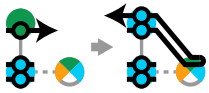
If the previous stitch made with the current yarn was at a node with a column edge to a ready node, then knit that ready node's row-wise neighbor (left). If this neighbor does not exist, tuck on the current stitch's neighbor and knit the ready node in reverse (right).

Tracing Rule 3. Continue:



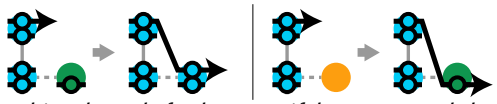
If there is no column edge from the current node to a ready stitch, and the next stitch along the current row has not been knit twice, knit it.

Tracing Rule 4. Tuck and turn:



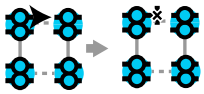
Upon reaching the end of a short row, if the current node has not been knit twice, tuck at the current node's parent's row-wise next node, then knit the current node in the opposite direction.

Tracing Rule 5. End short row:



Upon reaching the end of a short row, if the current node has already been knit twice – knit the next stitch off the end of the short row continuing in the same direction.

Tracing Rule 6. End yarn:



If the current yarn can not be extended (i.e., no row-wise or column-wise adjacent ready node exists, and all adjacent nodes have been knit twice already), end it.

Together, these local rules cause tracing to walk along every row of the graph, knitting at every stitch, and tucking at the ends of short-rows. Figure 12 shows an example that applies the tracing rules on a small segment of a tube. Cases with row edges running right-to-left, and cases with node in/out degree two are handled similarly, and are not shown. Tracing will always succeed, by construction. The rows of the graph are always singly-linked chains or cycles (Property 3) and the graph is helix-free (Property 2). Hence, it must consist of cycle-shaped rows, possibly with intermediate non-cyclic short-rows. When a short-row becomes ready, by rule 2 tracing immediately proceeds to knit it. By rules 2, 4, 5, short-rows are

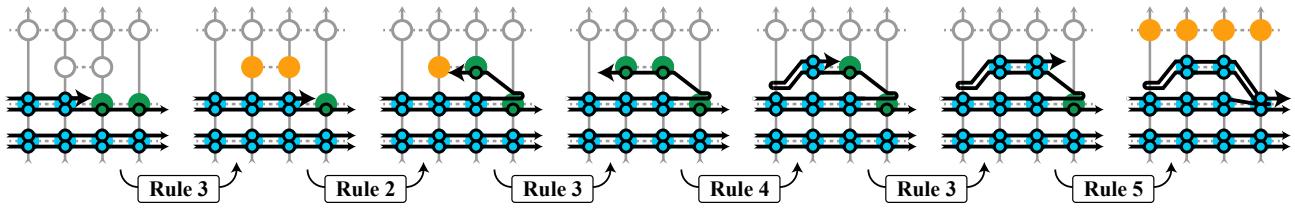


Fig. 12. Tracing a short row on a small portion of a tube, according to the tracing rules. The rule applied between two steps is shown in the box below.

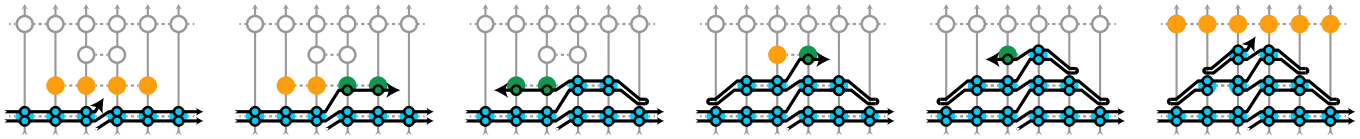


Fig. 13. Tracing of short-rows starts and ends on the same stitch and can be viewed a part of the preceding full cycle.

reversed and traced twice, ending at the same stitch as it started and using the same yarn (see Figure 13). Thus, short-rows can be knit along with the cycle that precedes it – this cycle is unique because by Property 5, no short-rows appear at a merge. Ignoring short-rows, each generalized cylinder is a stack of cycles that can be traced by single yarn. When a cycle becomes ready, the previous cycle must have been completed and the same yarn is continued over using rule 2. For a cycle that lies at a split, it is possible that one of the split next cycles becomes ready before the previous cycle is finished. In this case, tracing proceeds to knit it and a new yarn is brought in to finish the cycle and continue on each of the remaining splits. When cycles merge, the yarn from the most recently completed cycle is extended. Thus, tracing uses at most $n + (k-1)m$ yarns to trace a knit graph with n starting cycles and m k-way splits (Figure 14).

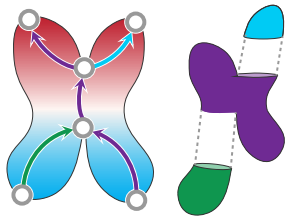


Fig. 14. Tracing generates a yarn path for each cylinder representing an arc of the Reeb graph. For each initial active cycle, a yarn is introduced (purple and green). At the merge, the existing purple yarn from the most recently completed cycle is continued. At the split, the current purple yarn continues along one of the cycles and a new blue yarn is introduced for the additional cycle.

3.4 Scheduling

After tracing is complete, our system is left with a series of knitting stitches along with adjacency information (produced and consumed loops, previous and next yarn-wise neighbors) about those stitches. In order to produce knitting machine instructions to construct these stitches, our system needs to assign each operation to a machine needle, and devise a plan to move loops between needles as required. We call this operation scheduling. In the following text, we refer to a running example, shown in Figure 15, of a genus-1 surface that was traced with two yarns.

Starting from the output of tracing (Figure 15a), our system first identifies sequences of consecutive stitches that take place on a generalized cylinder. This segmentation is accomplished by tracking the connections between loops currently held on the machine bed and performing splits/merges when a loop-wise parent of a stitch does not appear next to the yarn-wise previous stitch in the current cylinder.

At any point during the construction of one of these *segments* of stitches (Figure 15b), loops from that segment must be held on the machine bed in one of a small number of layouts (Figure 16). Specifically, for a cycle of N loops there are $5N$ (for N even) or $4N$ (for N odd) possible layouts[2]. We show layouts using a dotted outline to indicate the needles occupied by the cycle and a dotted circle to indicate the location of a designated loop in the cycle. Conveniently, the transfer planning algorithm of McCann et al. [2016] can move cycles between any two layouts; including adding or removing loops for increases and decreases.

The key insight that makes our scheduling algorithm possible is the observation that at connections *between* segments, loops must obey a valid layout both for the segment that produced them and the segment that will consume them (Property 6).

Therefore, the next step of scheduling is for our system to select a layout for each connection between tubes. These connection layouts are chosen to minimize a heuristic cost defined over the segments, and so that they do not force any segments to cross during knitting (Figure 15c). In effect, our system is determining an upward planar embedding of the Reeb graph of the object, as discussed in Section 2.2.

Though there is an exponentially large search space, the number of connections between segments is small enough that our system's greedy enumeration of layouts for these connections terminates quickly – either by exhausting all options or by finding a valid assignment.

[2]In general, the winding direction of a cycle cannot be changed through transfers, so our system only considers layouts in one winding direction.

Our system selects connection layouts to minimize (lexicographically) the following tuple of per-segment costs:

$$\sum (\mathrm{u}(L_s) + \mathrm{u}(L_e), \mathrm{r}(L_s, L_e), \mathrm{s}(L_s, L_e))$$

Where the terms are defined as follows:

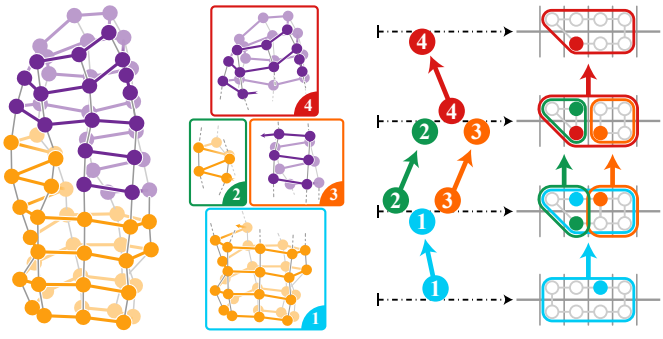
- $\mathrm{u}(L)$ (alignment): 0 if the shape is aligned on the front and back bed (only possible for even cycles) and 1 otherwise
- $r(L_s, L_e)$ (roll): the number of loops that must change beds between the beginning (L_s) and end (L_e) of the segment
- $s(L_s, L_e)$ (shift): the number of loops that must shift left or right between the beginning and end of the segment.

The position of a loop at the beginning of a segment is considered to be the position of its earliest column-wise ancestor in the same segment.

Our system optimizes connection layouts, which imply the starting and ending layouts of *connected* segments, while this cost is defined over the starting and ending layout of each *individual* segment.

Once layouts for the connections between segments have been determined, our system finds an optimal layout for every construction step in each segment, constrained by the already-assigned starting and ending layouts (Figure 15d). Here, again, optimal means that layouts should be aligned, loops should not switch beds, and loops should not shift left or right.

At this point, the layout of every cycle of loops held on the knitting machine bed during each construction step has been determined, as well as their left-to-right order on the bed. However, our system still needs to assign horizontal offsets to each cycle. To do this, our system begins by arranging the cycles as compactly as possible, given the left-to-right order determined by the directed acyclic graph enumeration step above. Starting from this configuration, our system then makes a series of optimal adjustments – where each adjustment involves adding or removing between zero and ten needles of space between two adjacent layouts on every construction step. Once no



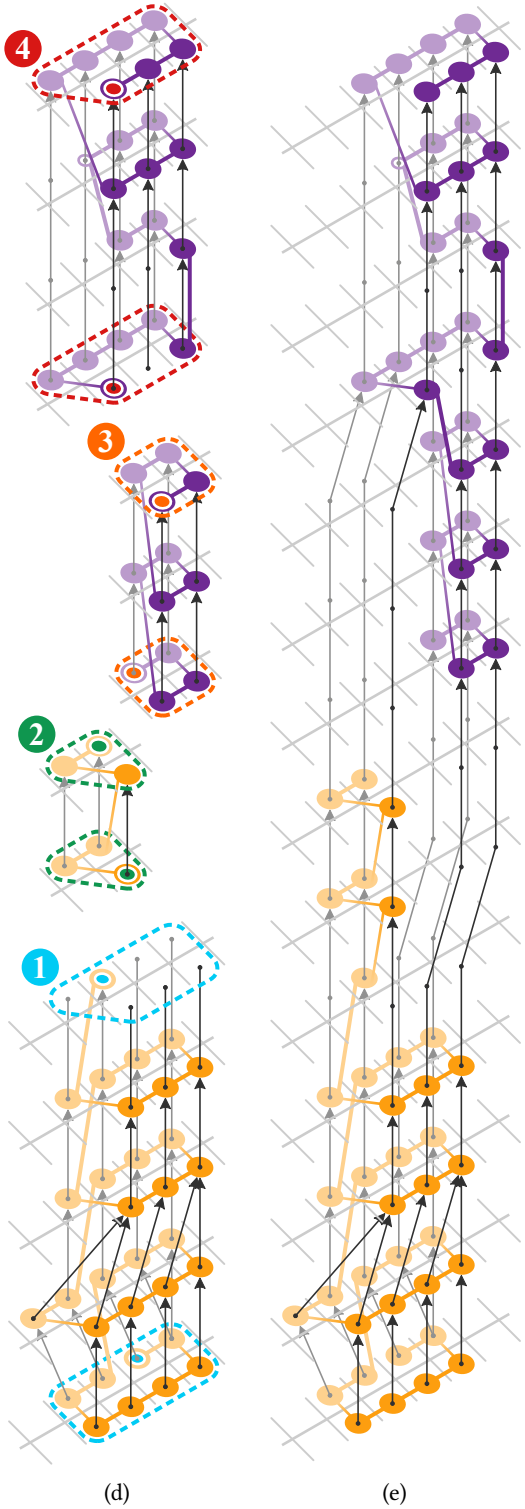


Fig. 15. Given (a) input stitches, our system (b) divides them into tube-like segments, (c) determines layouts for the connections between segments and their left-to-right ordering, (d) determines layouts for each construction step within each segment, and (e) combines the segments into a final knitting program. Knit stitches are shown as ⬬ and tucks as ⬭. Loops that are not knit through but held on the bed are shown as ▪. Yarn-wise connections are shown as ／ and loop-wise connections as ↑. Stitches that start and end segments are marked as ◉.

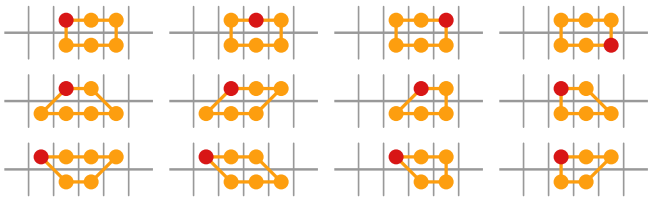


Fig. 16. All of the shapes (and some of the rolls) for cycles of five and six stitches. A stitch is colored red to distinguish cycles with different rotations.

adjustment that lowers the summed absolute distance between loop positions in subsequent steps is available, the scheduling is finished, and needles are assigned (Figure 15e).

4 RESULTS

Fig. 17. Plush toys with complex shapes fabricated using our system. The fox plushie (right) was generated in two parts and sewn – note the gap in the graph on the chest that indicates the two separate graphs.

All results in this paper were knit on a Shima Seiki SWG091N2 15-gauge V-bed knitting machine. The machine has two 91cm long beds with 15 needles per inch. We knit all our designs in half-gauge. The machine speed was set to 30% of its maximum speed because it was installed on an office floor; a production system on a concrete floor could run faster. We used Tamm Petit acrylic yarn for all designs. With this yarn, we measured the loop width (l_r) and height (l_c) to be 3.66mm and 1.73mm respectively with our machine settings, and we used these constants for all results.

Our system writes patterns in an intermediate low-level knitting language, which is further translated by backend code into the Shima Seiki KnitPaint "dat" format. However, nothing in our system

Fig. 18. Garments and accessories fabricated using our system.

Fig. 19. Plush versions of the kitten, a teddy bear, a horse, the Utah teapot and the Stanford bunny.

relies on special features of Shima Seiki machines. In theory, one should be able to build additional backend translators to support any knitting machines with two beds and transfer support.

Note that our instruction generation code includes some additional – white, in our examples – yarn whenever a cycle is created (cast on) or finished (bound off). This waste yarn allows our instruction generation to use faster cast-on and bind-off methods, stabilizes the models for stuffing, and is easy to manually remove or tuck inside in a post-processing step.

Our method is able to automatically create knit patterns for common textile objects. For plush toys (Figure 17), this functionality allows designers to freely innovate using familiar 3D modelling software. Garment designers can take advantage of this flexibility as well (Figure 18), and, in addition, can use 3D scans to make body-specific items; for example, a custom glove based on a hand scan. Of course, our system supports any manifold, orientable triangle mesh, including graphics classics (Figure 19). Additional images showing multiple views of our results are included in the supplementary material.

As our system's input is a 3D mesh, one can quickly generate a variety of results in different scales by scaling the input model (Figure 20). Such scale variations would otherwise require extensive re-tooling of the knitting pattern. One can also change the direction of knitting by altering the time function (Figure 21). The time function can be used to guide the surface pattern of the final knit

Fig. 20. Ducks of various sizes generated by uniformly scaling the input mesh. The input time function (and knit graph for the largest scale) are shown to the right.

item and trade off pattern fidelity with post-processing time – particularly, meshes often appear best if the knitting column direction is aligned with thin protrusions; though their general shape will still be captured regardless as long as the feature is larger than the stitch size.

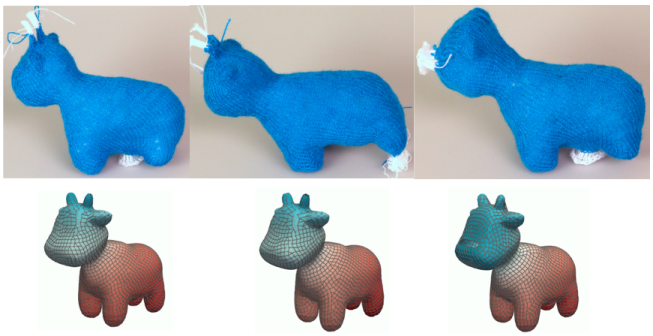


Fig. 21. The same model knit with different time fields. Starting at the horns and ending at the belly results in all the feet being short-rowed (**left**). Ending at the rear feet instead results in short-rowed front feet and increase-decrease shaped rear feet (**middle**). Starting at the face and ending at the belly results in the horns and feet being created with short-row shaping (**right**).

5 DISCUSSION

Accuracy. Knit objects are inherently stretchy, and stuffing does not provide uniform pressure. Therefore, the shape of our system's outputs are not exactly the shape specified in the input mesh (see teapot lid in Figure 19).

We tested our system's ability to reproduce overall model shape by knitting models of bent cylinders (Figure 22). The knit cylinders, once stuffed, assume angles remarkably close to their source models.

To test the accuracy of our system's output without the deformation induced by stuffing, we cast a soft foam Stanford bunny model in a mold generated from a 3D print of the same input mesh we provided to our system. The stuffed bunny is matched well by the knit skin as seen in Figure 1, though the covering is loose in some of the concave areas, as there is no force to hold it into these areas. If the goal were to upholster a Stanford bunny with knit fabric, spray adhesive could overcome this limitation, while if the goal were to provide a dust-cover for a bunny, an overall looser skin might be more appropriate.

In general, while we think that the current system's outputs are remarkably good, we suspect that future work using a closed-loop design procedure is likely to be able to produce higher-fidelity results. Such a procedure may include optimizing the input mesh to better match the desired output shape by accounting for deformations, as suggested for modeling balloons [Skouras et al. 2012] or integrating yarn- and stitch-level knit simulation tools with our system [Cirio et al. 2015; Kaldor et al. 2008, 2010; Meißner and Eberhardt 1998]. We also see an opportunity to use stitch-level editing tools, such as those developed by Yuksel et al. [2012], as a way to allow advanced users to perform detailed editing of our algorithm's output.

The geometric accuracy of our results is limited by the size of the stitches used to knit them. This size, in turn, depends on the gauge of the machine and is typically in the order of millimeters. Features smaller than the stitch size cannot be represented.

Knitting machines can change row lengths using shaping operations at a limited rate (Property 3). This limits the amount by which the radius of a generalized cylinder can be varied along its rows thus affects the approximation of the mesh by the knit graph.

However, under refinement of stitch size, the accuracy of our remeshing improves. To see why, consider a cycle of a cone at radius r with $n = \frac{2\pi r}{l_r}$ stitches on its circumference. Due to property 3, the number of stitches in the next cycle at a distance l_c apart, is at most $2n$ and its radius can be at most $2r$. If the size of the stitch width and height is reduced by a factor of 2, the number of stitches in a cycle of radius r is now $2n$ allowing a wider radius of $4r$ at a distance l_c on the surface – in the limit, as the stitch size reduces, any surface feature can be accurately represented (Figure 23).

Comparison to Previous Work. Our Figure 19 shows results on the models depicted in Igarashi et al.'s Figure 11 [2008a]. The machine-constructed versions are more detailed and require much less human effort to produce.

It is interesting to observe that our knit graph structure is the dual of Yuksel et al.'s stitch-mesh structure [Yuksel et al. 2012], constrained to be machine-knitable for "all knit" (i.e., "stockinette") patterns.

Timings. Fabrication is the largest component of result production time. That said, we believe that our system's computation time could be optimized significantly; the code is research-level, and

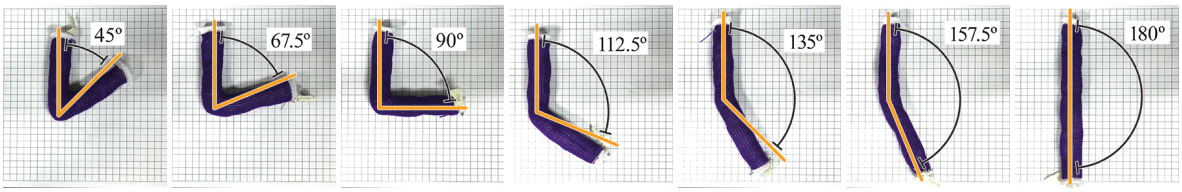


Fig. 22. Knitted models of bent tubes assume reasonable angles when stuffed. The overlay shows the bend angle of the input model.

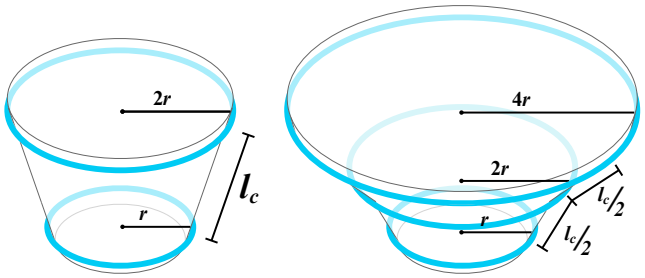


Fig. 23. Decreasing the stitch size uniformly increases the number of stitches in each cycle and thus increases the maximum width achievable by the next cycle.

Model	Remesh	Trace & Schedule	Fabricate
Ram (Figure 17)	20 min	1 min	100 min
Hand (Figure 18)	38 min	0.5 min	90 min
Teddy (Figure 19)	37 min	3.5 min	120 min
Bunny (Figure 19)	11 min	1.5 min	90 min

Table 1. Time (in minutes) for the model that took longest to process in each group on a 2.7GHz Intel Core i5 Macbook Pro with 16GB RAM. The knitting machine was set to 30% of its maximum speed.

includes unnecessary validity checks and recomputations of quantities that could be maintained incrementally. For each result group, timings for the model that took the longest time to process and fabricate has been listed in Table 1.

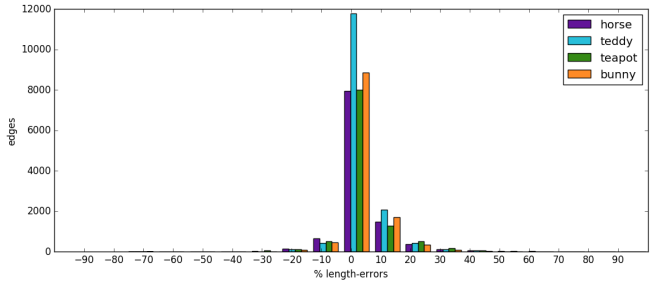


Fig. 24. Most edges have less than 10% length error introduced due to fabrication constraints and discretization.

Stretch. Our remeshing step seeks to produce a low-stretch knit graph (Property 8). In general, it succeeds, with the majority of edges within 10% of their target length (Figure 24).

While much of this error is due to constraints imposed by the machine design and yarn properties, some of the length errors also stems from the incremental nature of our remeshing algorithm. As our implementation accepts the ending cycles entirely once the boundary is reached, the links generated at the boundary also contribute to higher distortion. Including short-rows at the bind-off could reduce some error. Distortion in remeshing affects the appearance of the knitted object but does not affect knitability, which is defined by the topology of the knit graph and maintenance of properties for tracing and scheduling (Property 1-3, 6-7).

The histogram is not symmetric because remeshing always computes cycles at $2l_c$ geodesic distance from the active cycle. Thus, column edges, by construction, only contribute to positively signed errors. It would be interesting to consider an adaptive remeshing procedure that distributes column errors or a global mesh refinement pass for reducing edge length errors. Alternatively, one might seek to use a yarn that can withstand more strain, or a needle that can accommodate more loops, relaxing the constraints.

Impossible meshes. Examples that cannot be fabricated by our system (Figure 25) fall into two categories: unschedulable meshes, and meshes with sheets. Unschedulable meshes have no possible upward planar embedding. In the future, it might make sense to extend our system to make it easy to add additional cuts to these meshes to make them knitable (at the cost of additional post-processing with sewing).

Meshes with sheets are theoretically knitable, but include boundaries that aren't starting or ending points for knitting. For now, one can modify the mesh to cap these boundaries and then post-process by cutting holes in the model or mark the boundaries as starts or ends; in the future, it would be interesting to add support for sheets to our system. Our initial attempts in this direction suggest that the additional freedom in scheduling may make the problem intractable.

Robustness. Though our system guarantees that yarn constraints are satisfied locally, it ignores more global constraints, such as: yarn bridges from merges that occurred several rows prior, the pile-up of fabric caused by knitting the middle tube of a 3-way split before the right or left tube, or the wear on yarn caused by extensive layout changes. In some cases, this can result in yarn breaks while knitting.

For example, in Figure 17, our scheduler chose an order that knit the torso and lower body of the fox before knitting its arms, causing the yarn joints to the arms to break. In this case, we manually

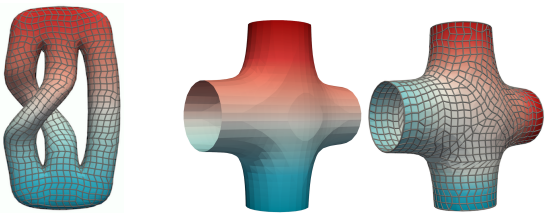


Fig. 25. **Left**, meshes that do not have an upward planar drawing of the knit graph on the bed cannot be knit scheduled without additional seams. **Middle**, meshes with sheet-like regions are not handled by our system owing to their additional scheduling degrees of freedom. **Right**, Marking the boundaries on the sides as a starting or ending cycle would make such a mesh knitable in our system.

added a seam to the torso to the knit graph, forcing the result to be knit in two pieces, and then manually stitched these together (with contrasting orange yarn). It is worth emphasizing that this is the only example for which such intervention was required.

Knitting time functions. In all our results, the knitting time function was generated using our interface. Automatically computing a suitable function given the input mesh would be an interesting addition. Local extrema of the mesh curvature can be used as starting and ending points to effectively capture mesh features (Figure 21). Geodesic functions from boundary cycles can provide short-row-free time functions, but are hard to control (Figure 10). Lower order eigenvectors of the mesh Laplacian, like the Fiedler vector, have been used to provide a *natural* ordering of vertices in the mesh graph [Lévy and Zhang 2010] and may act as suitable time functions.

6 CONCLUSION

Knitting machines are as robust and repeatable as 3D printers, but – until now – they have not enjoyed the same popularity. We believe this deficiency stems from the lack of easy-to-use software and consumer-level hardware. The hardware landscape is already starting to change: "Kniterate," a consumer-level industrial-style knitting machine, was recently funded on Kickstarter [2017]. This paper provides an important addition to the software landscape. By automatically producing machine knitting instructions from 3D models, our system makes these machines as easy to control as 3D printers. We see this as a crucial step in moving industrial-style machine knitting from an industrial technique to a widely-available fabrication technology.

ACKNOWLEDGMENTS

We are grateful to various designers for their freely available 3D models. The bunny model is provided by the Stanford Computer Graphics Laboratory. The horse model is from the Georgia Tech Large Geometric Models Archive. The kitten model is provided courtesy of Frank_terHaar by the AIM@SHAPE-VISIONAIR Shape repository. The shoe last model (thing: 10307) by Prattotyper, the scanned hand (thing:148909) by versonova and the witch hat (supplementary, thing: 2603419) by MrsCreepyPumpkinHead is provided by Thingiverse. The glove model (product: 223158) by 3DUA and dress models by UAG (product: 727699) and ociany (product:675666) are provided by Turbosquid. We thank Keenan Crane for the cow and duck models; Kyna McIntosh and Michelle Ma for the plushie models.

REFERENCES

Sarah-Marie Belcastro. 2009. Every topological surface can be knit: a proof. *Journal of Mathematics and the Arts* 3, 2 (2009), 67–83.
Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series.. In *KDD workshop*, Vol. 10. Seattle, WA, 359–370.
David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global structure optimization of quadrilateral meshes. In *Computer Graphics Forum*, Vol. 30. 375–384.
David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013. Quad-Mesh Generation and Processing: A Survey. In *Computer Graphics Forum*, Vol. 32. 51–76.
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*. 55–61.
Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2013. Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow. *ACM Trans. Graph.* 32 (2013). Issue 5.
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.
Herbert Edelsbrunner and John Harer. 2010. *Computational topology: an introduction*. American Mathematical Soc.
Ruslan Guseinov, Eder Miguel, and Bernd Bickel. 2017. CurveUps: Shaping Objects from Flat Plates with Tension-actuated Curvature. *ACM Trans. Graph.* 36, 4, Article 64 (July 2017), 64:1–64:12 pages.
Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008a. Knitting a 3D Model. In *Computer Graphics Forum*.
Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008b. Knitty: 3D modeling of knitted animals with a production assistant interface. In *Eurographics 2008 Annex to the Conference Proceedings*.
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), 65:1–65:9 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), 105:1–105:10 pages.
Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe Patterns on Surfaces. *ACM Trans. Graph.* 34 (2015). Issue 4.
Bruno Lévy and Hao Richard Zhang. 2010. Spectral mesh processing. In *ACM SIGGRAPH 2010 Courses*. ACM, 8.
Ali Mahdavi-Amiri, Philip Whittingham, and Faramarz Samavati. 2015. Cover-it: an interactive system for covering 3d prints. In *Proceedings of the 41st Graphics Interface Conference*. Canadian Information Processing Society, 73–80.
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. 355–362.
Mariana Popscu, 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.
Rubio,Gerard and Saxena,Triambak and Catling,Tom. 2017. Kniterate. [Online]. Available from: https://www.kniterate.com. (2017).
Shima Seiki. 2011. SDS-ONE Apex3. [Online]. Available from: http://www.shimaseiki.com/product/design/sdsone_apex/flat/. (2011).
Mélina Skouras, Bernhard Thomaszewski, Bernd Bickel, and Markus Gross. 2012. Computational design of rubber balloons. In *Computer Graphics Forum*, Vol. 31. 835–844.
Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. 2014. Designing Inflatable Structures. *ACM Trans. Graph.* 33, 4, Article 63 (July 2014), 63:1–63:10 pages.
Stoll. 2011. M1Plus pattern software. [Online]. Available from: http://www.stoll.com/stoll_software_solutions_en_4/pattern_software_m1plus/3_1. (2011).
Jenny Underwood. 2009. *The design of 3D shape knitted preforms*. Ph.D. Dissertation. Fashion and Textiles, RMIT University.
Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2012. Stitch Meshes for Modeling Knitted Clothing with Yarn-level Detail. *ACM Trans. Graph.* 31, 4, Article 37 (July 2012), 37:1–37:12 pages.

A BALANCED LINKING

An active cycle can split into multiple next cycles (or vice-versa). In these cases, to satisfy Property 5, we need to ensure the next cycle is sampled such that 1-1 linking is possible. Further, linking must be such that the nodes of the active and next cycles have a valid layouts on the knitting machine bed (Property 6).

Our code locally balances the layout for cycles undergoing a split or a merge. Nodes on the active cycle and vertices on the next cycle can be colored based on the alignment pair they belong to. If a feasible layout exists, the coloring on any cycle has the following structure: there exist at most two segments (i.e., contiguous nodes with the same color) with a unique color – these terminal segments can lie at the farthest ends when the cycle is placed on the needle bed. All intermediate segments form pairs – from two segments of the same color that should lie across each other on the needle bed.

Splits. When a cycle splits into multiple cycles, our code re-assigns target vertices for nodes in the active cycle such that the intermediate segment pairs on the active cycle have an equal number of nodes. Beginning from the central pair, any segment that has extra nodes is relabeled by propagating adjacent target labels inwards. Now, for each alignment pair, next segments can be sampled such that they have an identical number of nodes to the active segment (Figure 26).

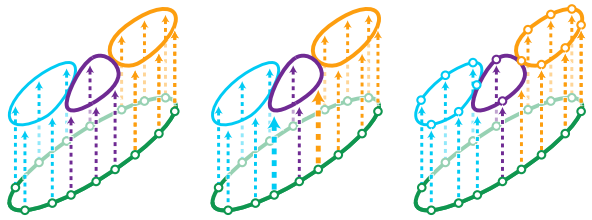


Fig. 26. In the initial alignment pairs (**left**, dotted lines), the purple segments has unequally distributed nodes on its active cycle segments. To make them even, the nodes are relabeled (see bold arrows) by propagating adjacent labels (**center**). Nodes are placed on next cycles for each alignment pair such that 1-1 alignment is possible (**right**).

Merges. When a next cycle merges from multiple active cycles, internal segment pairs on the merged cycle are sampled such that the active cycle nodes in its alignment pair are equally distributed between the segments on the next cycle. (Figure 27) In the case of multiple internal segments with an odd number of nodes, we maintain parity by alternating the side along which the higher number of nodes are placed on the merged cycle.

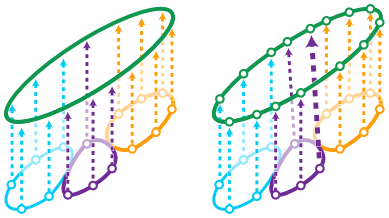


Fig. 27. In the initial alignment (**left**, dotted lines), nodes on the active cycle for the purple pair are not equally distributed across the two segment. For a valid layout, these are equally distributed (see bold purple arrow) when the next cycles are sampled for 1-1 linking (**right**).

This balancing procedure fixes the layout locally. However, Property 6 depends on the layout of all splitting and merging rows at a given instance, i.e., between consecutive slices. A situation when our heuristic fails and a global layout is needed is when multiple splits and merges are coupled (Figure 28). We have never encountered this situation occur in practice.

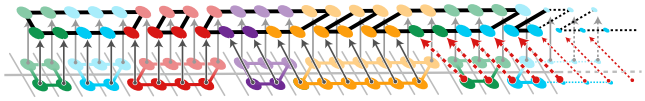


Fig. 28. The cycles that merge are each independently balanced and the trapezoid cycles that introduce an odd number of stitches may all align to the same side of the bed. Because these merges are coupled by splits, they cannot be rotated during layout. By repeating such a construction , a layout can be forced that makes the front and back bed stitches unbalanced.

Our system could be modified to work in these rare situations by forcing all cycles to have an even number of stitches, with the downside of a slight loss of dimensional accuracy. If splits and merges are strictly binary, cycles can be rotated on the bed to ensure balance during layout. Therefore, another approach to satisfy layout conditions is to turn a k-way splits (or merges) into $k-1$ binary splits (or merges) at the cost of some additional distortion. This operation can be viewed as a perturbation of the input knitting time function.