Helix-Free Stripes for Knit Graph Design

Rahul Mitra
rahulm@bu.edu
Boston University
Boston, MA, USA

Liane Makatura
makatura@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

Emily Whiting
whiting@bu.edu
Boston University
Boston, MA, USA

Edward Chien
edchien@bu.edu
Boston University
Boston, MA, USA

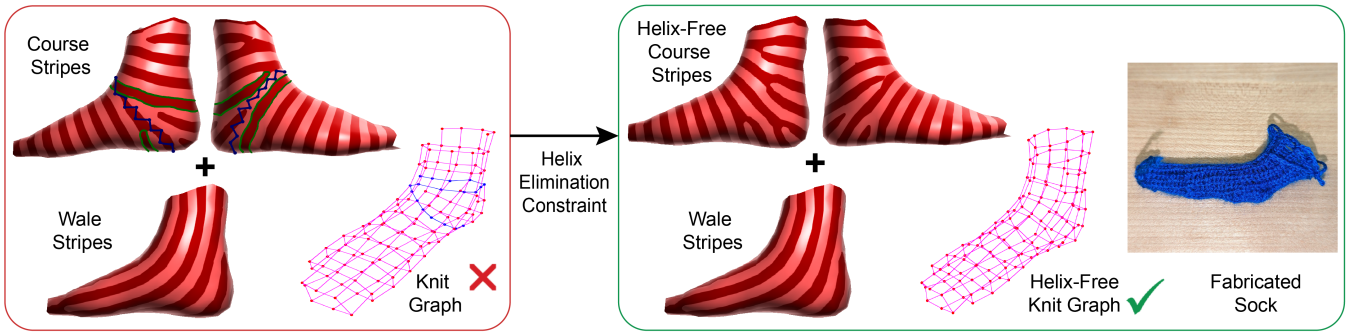


Figure 1: We generate a knit graph for a sock model by composing two orthogonal stripe patterns from mixed-integer solves. (Left) Naive boundary-aligned course stripes frequently contain helices, highlighted in green. This helix persists in the knit graph, which renders it unknittable. We introduce a helix elimination constraint along the blue cycle. (Right) After solving subject to this constraint, our course stripes are helix-free, which produces a knittable graph.

ABSTRACT

The problem of placing evenly-spaced stripes on a triangular mesh mirrors that of having evenly-spaced course rows and wale columns in a knit graph for a given geometry. This work presents strategies for producing helix-free stripe patterns and traces them to produce helix-free knit graphs suitable for machine knitting. We optimize directly for the discrete differential (1-form) of the stripe texture function, i.e., the *spinning form*, and demonstrate the knitting-specific advantages of this framework. In particular, we note how simple linear constraints allow us to place stitch irregularities, align course rows and wale columns to boundary/feature curves, and eliminate helical stripes. Two mixed-integer optimization strategies using these constraints are presented and applied to several mesh models. The results are smooth, globally-informed, helix-free stripe patterns that we trace to produce machine-knittable graphs. We further provide an explicit characterization of helical stripes and a theoretical analysis of their elimination constraints.

SIGGRAPH '23 Conference Proceedings, August 06–10, 2023, Los Angeles, CA, USA

ACM ISBN 979-8-4007-0159-7/23/08…$15.00
https://doi.org/10.1145/3588432.3591564

CCS CONCEPTS

• **Computing methodologies** → **Shape analysis**; • **Applied computing** → *Computer-aided design.*

KEYWORDS

computational knitting, stripe texturing

ACM Reference Format:
Rahul Mitra, Liane Makatura, Emily Whiting, and Edward Chien. 2023. Helix-Free Stripes for Knit Graph Design. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings (SIGGRAPH '23 Conference Proceedings), August 06–10, 2023, Los Angeles, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3588432.3591564

1 INTRODUCTION

The growing availability of computational knitting machines has spawned much recent work on algorithms and frameworks to assist users in designing stitch patterns for complex geometries. The seminal work of AutoKnit [Narayanan et al. 2018] first tackled the problem of seamless whole-garment computational knitting from arbitrary mesh input. The authors abstracted stitch patterns to the important notion of knit graphs, and various formal constraints were established to characterize machine-knittability. Key among these was the *helix-free* condition, specifying that course rows of vertices should not form spirals. While a robust method for knit graph generation was produced, its use of an iterative, local process based on geodesic distance estimates may produce knit patterns

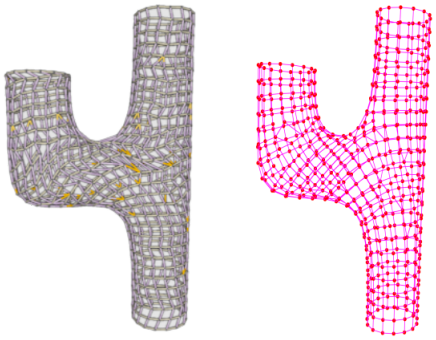


Figure 2: Level set tracing (right) produces a smoother globally-informed knit graph compared to [Narayanan et al. 2018] (left). See wale columns.

that are less smooth and globally-informed than those produced with our framework (see Fig. 2). Furthermore, it is not easy to achieve any user control over the stitch pattern in their setting.

The KnitKit pipeline of Nader et al. [2021] recognized the potential of stripe generation algorithms, as evenly-spaced stripes are analogous to evenly-spaced course rows and wale columns of a knitting pattern. They use the method of Knöppel et al. [2015], which performs a global convex solve for a complex-valued function over the surface. They then infer a discrete *spinning (1-)form* from this function, and the form specifies the stripe pattern. While this method is efficient and produces global, geometrically-informed stripe patterns, it does not guarantee that they will be helix-free and that they will align to mesh boundaries. Nader et al. [2021] use mesh modification operators from the quad meshing literature [Bommes et al. 2011] to remove helices, but these are not guaranteed to work.

In this work, we instead optimize for the spinning form directly in the space of discrete 1-forms on a given triangle mesh. This setting allows the user to easily specify placement of short row ends and increases/decreases, align course rows and wale columns to specified polylines, and to eliminate helical stripes through a variety of linear constraints. Recent work from Noma et al. [2022], considered this framework for interactive editing of a given stripe pattern, but did not consider its use for machine knitting. We utilize and generalize their constraints, identifying implications for knit structure. Amongst these constraints is a novel *helix elimination* constraint that removes identified helical stripes in a given pattern.

Using these constraints, we also outline two optimization strategies for obtaining helix-free stripe patterns. The first assumes fixed singular triangle positions and indices and pairs them with level set or stripe alignment constraints. This results in non-helical short rows between pairs of singular triangles. The second aims for iterative automatic placement of geometrically-informed singular triangles, using helix elimination constraints to prevent helices that have arisen in previous iterations. Both strategies result in quadratic mixed-integer problems, with the former having few integer variables and being solved at interactive rates, while the latter produces reasonable approximate solutions on longer timescales.

We demonstrate these strategies on several mesh models, and the stripe patterns are used to produce machine-knittable graphs. The graphs are traced and scheduled with the AutoKnit pipeline, and .dat files and knits may be seen amongst our results. Lastly, we provide several theoretical results that are briefly mentioned below. In summary, our main contributions are:

- A novel framework for stripes-based knit graph construction, leveraging optimization in the space of discrete 1-forms. This allows for: (i) elimination of helical stripes with linear constraints, (ii) precise placement of short rows and increases/decreases, (iii) smoother globally-informed stitch patterns, and (iv) alignment of courses and rows to boundaries/feature curves.
- A theoretical analysis of helices and constraints, showing that: (i) level set constraints are equivalent on homologous curves, (ii) helix elimination constraints have practical guarantees, (iii) a rigorous notion of course helices requires wale columns, and (iv) there are extensions of many linear constraints of [Noma et al. 2022] to the interior of triangles.

2 BACKGROUND & RELATED WORK

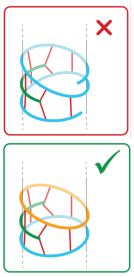
In this section, we review concepts from related work, and establish notation and background. Let us first list some more distantly-related works that have inspired us. Many of these, e.g., [Jones et al. 2022; Kaspar et al. 2019, 2021; Narayanan et al. 2019] have impressive semi-automatic workflows that allow designers to vary seam layouts, use different stitch types, and perform complex colorwork and knit texturing. Others are targeted specifically at producing hand-knittable output, e.g., [Igarashi et al. 2008; Wu et al. 2019], or output that is meant just for rendering [Wu et al. 2018; Yuksel et al. 2012]. Lastly, the work of Tricard et al. [2020] applied texture parameterization techniques to a different application: physical engineering of 3D-printed microstructures.

The majority of the knitting works above generate geometrically-informed stitch patterns by leveraging two sources of information: a combination of time function level sets and geodesic distance estimates, or quad meshing algorithms. Unlike these works, we aim to leverage algorithms for producing stripe patterns, as was done by [Nader et al. 2021], and tackle the specific problem of ensuring helix-free knit graphs. Furthermore, we restrict ourselves to the case of seamless whole-garment knitting, done "in-the-round", and with a single jersey stitch pattern. While not as comprehensive as some of the more complex workflows above, we believe that our framework and its insights could prove useful in these pipelines.

2.1 Helix-free Knit Graphs

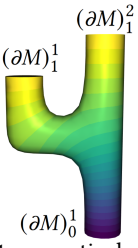
For this work, we follow AutoKnit [Narayanan et al. 2018] and use *knit graphs* to represent our knit structure. If a primer on basic knitting concepts (courses/wales, short rows, increases/decreases, knit graphs) is needed, we provide one in supplementary §1.

Given a knit graph, AutoKnit [Narayanan et al. 2018] *traces* it to produce a final knit structure and yarn path, and a scheduler compiles machine knitting instructions. For successful tracing, there are several properties that the knit graph must satisfy. We refer the reader to the paper for a list of these constraints, but describe here the most complex requirement: the helix-free property.



Within the knit graph, sets of nodes joined by course edges are *course rows*, and a graph is *helix-free* if there are no sequence of wale edges between any two nodes in the same course row. This is schematically depicted in the inset figure from [Narayanan et al. 2018] (blue row, top, is helical). This may seem to contradict the fact that knitting "in-the-round" produces a helical yarn path, but the tracing procedure of AutoKnit determines the spiraling in the final stitch structure.

For construction of the knit graph, a time function $h : M \to [0, 1]$ over the mesh is used to specify the direction of knitting and the rough alignment of course rows. This function is taken as the harmonic interpolation of boundary values set to 0 and 1 on certain boundary components. For later discussion, let us use $(\partial M)_0 = \sqcup_{i=1}^{N_0} (\partial M)_0^i$ and $(\partial M)_1 = \sqcup_{i=1}^{N_1} (\partial M)_1^i$ to label those boundaries and components on which the value is set to 0 and 1, respectively. $N = N_0 + N_1$ denotes the total number of boundary components. An example is in the inset figure. Using this time function, their construction procedure cycles through steps of geodesic distance estimation, sampling, and node linking via optimization. While effective and robust, this approach is not globally-informed and does not allow for user input in design of the knit graph.



2.2 Stripes For Knitting

As noted in the introduction, the recent KnitKit work from Nader et al. [2021] was first to leverage stripe patterning methods for stitch planning. In particular, they use the method of [Knöppel et al. 2015] to generate evenly-spaced stripe patterns guided by a vector field. An example output is shown in Figure 4. Placing course rows along the red stripes clearly suggests a sensible stitch pattern. As such, Nader et al. [2021] use two orthogonal stripe patterns to place course rows and wale columns.

2.2.1 Spinning Form. The method of Knöppel et al. [2015] optimizes for a complex function over a triangular mesh $\psi : M \to \mathbb{C}$, and uses $\arg \psi$ (the angle of ψ) as a stripe texture coordinate. E.g., for alternating red-and-pink stripes, the surface color is red when $\arg \psi \in (0, \pi)+2\pi k$, $k \in \mathbb{Z}$ and pink when $\arg \psi \in (\pi, 2\pi)+2\pi k$, $k \in \mathbb{Z}$. The objective term takes an input vector field that $\nabla(\arg \psi)$ tries to match over the mesh. We refer the reader to [Knöppel et al. 2015] for further details on the optimization.

To texture the mesh with stripes, the resultant ψ is used to infer a differential 1-form called the *spinning form*. These objects are represented in the setting of discrete differential geometry [Crane et al. 2013], and we provide a brief primer on discrete k-forms and their exterior derivatives d_0 and d_1 in supplementary §2, if needed.

In particular, ψ is discretized as a function over the mesh vertices $\psi : V \to \mathbb{R}$, and the spinning form is a discrete 1-form on the mesh edges $\sigma : E \to \mathbb{R}$. On an oriented edge e between vertices i and j, $\sigma_e = \sigma_{ij}$ represents the estimated change in $\arg \psi$ over that edge. Note that $\sigma_{ji} = -\sigma_{ij}$ (for σ to be well-defined).

On most faces, away from the zeros of ψ, $\arg \psi$ can be continuously extended onto the face, and evenly-spaced stripes with no

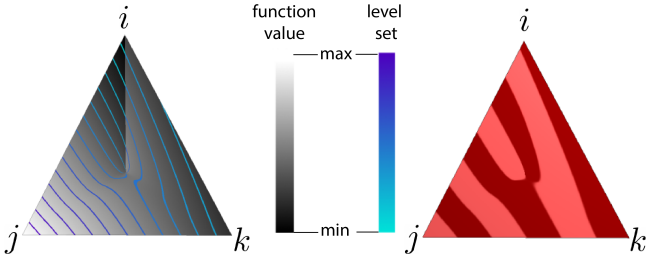


Figure 3: Left: an example of the stripe texture function on a singular triangle, with periodic level sets illustrated. Right: the resulting stripe pattern. The stripe index is -1, indicating that one stripe ends on this face.

irregularities result. On faces where ψ vanishes, $\arg \psi$ cannot be extended continuously, and an interpolant that mimics the argument function is used. The resulting texture shows that stripes are "born" or "die" on these faces, and are analogous to short row ends or increases/decreases in a stitch structure. An illustration of stripes and level sets on such a triangle is shown in Fig. 3. For completeness, we reproduce the interpolant in §3 of the supplementary.

More precisely, $(d_1\sigma)_m = 2\pi k_m$ for some $k_m \in \mathbb{Z}$ on each triangle m in the mesh. The cases above, when $\arg \psi$ does and does not extend continuously, correspond to $k_m = 0$ and $k_m \neq 0$, respectively. In the latter case, we refer to m as a *singular triangle*, and k_m as the *stripe index*. Let $K \subset F$ to denote the subset of singular triangles in the set of mesh faces. The stripe index on a triangle denotes how many stripes will be created or destroyed on that face.

Our framework may be viewed as an attempt to optimize for the spinning form directly, instead of inferring it from the optimized ψ as done in [Knöppel et al. 2015]. This optimization setting allows us closer control with simple linear constraints. As done in [Knöppel et al. 2015], we also allow for varying stripe width by replacing 2π with a user-specified *stripe period*, denoted P (stripe width is $P/2$).

2.2.2 Knittability Issues. There is a key complication in the direct use of [Knöppel et al. 2015]: the stripes produced are not helix-free, and there is no simple way to modify the underlying optimization problem to achieve this. Such a helix is highlighted in Fig. 4. Nader et al. [2021] use the mesh modification operators of [Bommes et al. 2011] to try and fix these defects, but details are not given. Moreover, in the quad-meshing setting, where these operators were initially defined, there is no guarantee on elimination of all helices.

Fig. 4 also highlights the fact that direct use of [Knöppel et al. 2015] may lead to poor stripe (and thus course row) behavior at the boundaries. In particular, stripes may not align well to the mesh boundary, or be tight to any/all boundaries.

2.3 Fast Stripe Editing

A closely related work is the recent [Noma et al. 2022]. The authors also optimize for the spinning form directly, but instead apply their method to two other fabrication methods: wireframe structures and spiral design for developable surfaces. Several constraints are constructed to control the resulting stripe pattern, including specification of indices on singular triangles, and constraints along polyline paths in the mesh. In §3.2, we generalize and discuss these

constraints in the knitting context. Primarily, we note that their approach does not reference or deal with helical stripes. We also extend many of their constraints to allow for polyline endpoints interior to triangles (§8, §9 of the supplementary).

For later discussion, we recall and augment their path notation in Fig. 5. A *polyline* is a path on a triangular mesh $\gamma : [0,1] \to M$ that is piecewise linear on each triangular face. Two examples are shown in Fig. 5. As can be seen, we additionally consider closed paths where $\gamma(0) = p_0 = \gamma(1) = p_n$ that we denote *cycles*.

Lastly, we recall Thm. 3.1 from [Noma et al. 2022], a useful index formula, in our notation and discrete setting:

Theorem 3.1 ([Noma et al. 2022]). *Given a discrete 1-form σ on an (orientable) triangular mesh M, the set of counterclockwise boundary loops $(\partial M)_0$ and $(\partial M)_1$, and the set of singular triangles K, the following holds:*

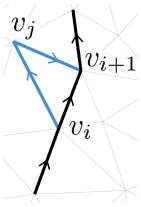
$$\sum_{m \in K} 2\pi k_m + \sum_{i=1}^{N_0} \int_{(\partial M)_0^i} \sigma + \sum_{i=1}^{N_1} \int_{(\partial M)_1^i} \sigma = 0 \tag{1}$$

The boundary integrals in the above theorem are defined as oriented sums of 1-form values:

$$\int_{(\partial M)_*^i} \sigma := \sum_{ij \in (\partial M)_*^i} \sigma_{ij}, \tag{2}$$

where the "counterclockwise" boundary orientations are the standard induced orientations (keep M to the left as you traverse with head oriented along outward normal). More generally, we define integrals of 1-forms along sequences of mesh edges as analogous oriented sums.

2.4 Homologous Paths



We briefly mention some definitions and facts from simplicial homology that will be necessary for the coming discussion. More complete expositions may be found elsewhere [Hatcher 2000] if desired. First, we note the definition of homologous curves. For simplicity, we only consider polylines whose sequence of vertices consist of mesh vertices: $p_0 = v_0, \ldots, p_n = v_n$ joined by mesh edges.

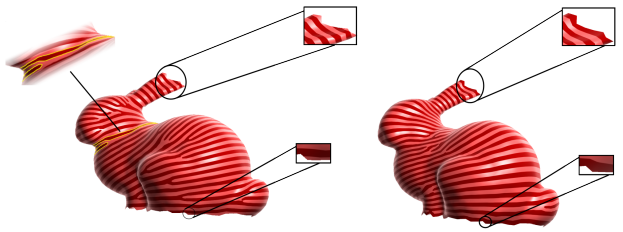


Figure 4: Potential knittability issues with direct use of [Knöppel et al. 2015], as done in [Nader et al. 2021] (left): helical stripes (in yellow, inset), poor boundary alignment (insets) and stripes not tight to boundary (insets). We can fix these with linear constraints and still obtain a reasonable global stripe pattern (right). Both patterning algorithms use the gradient of the time function as a guiding vector field.

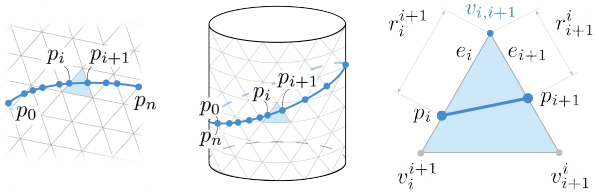


Figure 5: Polyline notation. Inspired by Fig. 1 from [Noma et al. 2022] supplementary. p_i is a point on an edge e_i that polyline γ passes through. $v_{i,i+1}$ denotes the vertex shared by e_i and e_{i+1}, while v_i^{i+1} and v_{i+1}^i denote the other vertices in the shared triangle. r_i^{i+1}/r_{i+1}^i denote the barycentric coordinates of p_i/p_{i+1} associated with v_i^{i+1}/v_{i+1}^i, respectively. s_i^{i+1}/s_{i+1}^i are sign terms that are +1 if the canonical orientation of e_i/e_{i+1} agrees with the direction from v_i^{i+1}/v_{i+1}^i to v_i^{i+1}, respectively. They are -1 otherwise.

Consider the following change to the curve: take any two successive vertices $v_i, v_{i+1 \pmod n}$, and replace them with v_i, v_j, v_{i+1} where v_j is the third vertex in the triangle that v_i and v_{i+1} are a part of. This is illustrated in the inset figure. We say that two paths in M are *homologous* if they differ by a sequence of changes above.

3 OPTIMIZATION FRAMEWORK

We consider a framework where we optimize for the spinning 1-forms directly, using a pair of them to construct a knit graph for our mesh M. One of these forms designates the course rows, σ_c, while another designates the wale columns, σ_w. In this section, we describe the objective and constraints used to find σ_c and σ_w.

3.1 Time Function Guidance Objective

As with several other works, we use a piecewise linear harmonic time function h that interpolates between 0 and 1 on user-provided boundaries to denote the desired start and end of the knit. The gradient ∇h specifies a direction that should be approximately orthogonal to course rows, and approximately parallel with wale columns. For specification of course rows, we compare σ_c to the one-form ω_c, defined on edge ij, with faces l and r to the left and right of the edge:

$$(\omega_c)_{ij} = \frac{1}{2}\left(\frac{(\nabla h)_l}{||(\nabla h)_l||} + \frac{(\nabla h)_r}{||(\nabla h)_r||}\right) \cdot \mathbf{e}_{ij} \tag{3}$$

In this expression, $(\nabla h)_l$ and $(\nabla h)_r$ denote the gradients on faces l and r, respectively. As can be seen, ω_c averages the integrals of the normalized $(\nabla h)_l$ and $(\nabla h)_r$ along e_{ij}. It reflects the desired change in the stripe texture function suggested by ∇h over each edge.

For creation of the wale columns, we compare σ_w to an analogous one-form ω_w. Its expression is the same as Eqn. (3), but with ∇h replaced by $(\nabla h)^\intercal$, the vector field obtained by rotating ∇h by 90° with respect to the outward-pointing normal.

The comparison objective is: $||W(\sigma_* - \omega_*)||^2$, the quadratic norm, with cotangent weighting (W diagonal with weights). This is the analogue of the energy from [Noma et al. 2022] for a face-based

vector field, which naturally arises from our piecewise-linear time function h.

3.2 Linear Constraints

3.2.1 Stitch Irregularity Placement. As shown in Fig. 3 & Fig. 6, stitch irregularities occur at singular triangles, with the birth and death of stripes occurring there. In the knitting context, this corresponds to the ends of short rows in the course context, and increases/decreases in the wale context. Specifying these is as simple as asking that $(d_1\sigma_*)_m = k_m$ for an integer stripe index k_m. These constraints were referred to as "winding number constraints" in [Noma et al. 2022]. Oftentimes, finer control is desired for exact stripe placement, and this is achieved in conjunction with the stripe placement constraints of §3.2.4.

3.2.2 Stripe Alignment Constraints. We may also align stripe patterns to user-specified polylines on the mesh. These can be employed for design specification of course rows or wale columns, or can be used to join singular triangles and correct a helical stripe (see Fig. 6). If used for these purposes, they may also be used in conjunction with some of the stripe placement constraints of §3.2.4.

Stripe alignment constraints are referred to as "brush" constraints in [Noma et al. 2022]. Assuming the polyline notation adopted in §2.3, these are expressed:

$$\forall i \in \{1, \dots, n-1\}, \qquad s_i^{i+1}\sigma_{e_i} r_i^{i+1} = s_{i+1}^{i}\sigma_{e_{i+1}} r_{i+1}^{i}$$

It should be noted that the triangles that the polyline extends through must be non-singular faces, as the stripe texture function is nonlinear on singular faces. This is the form of the constraint if p_0 and p_n lie on edges. In §8 of the supplementary, we extend these to polylines where the endpoints may lie on the interior of non-singular faces. If there are N^{sa} stripe alignment constraints, we denote the polylines $\{\gamma_j^{\text{sa}}\}_{j=1}^{N^{\text{sa}}}$ and their component edges $\{e_i^j\}_{i=1}^{n-1}$.

These constraints may be imposed on cycles within the mesh, if specification of entire course rows are desired. To obtain boundary alignment of the stripe pattern (see Fig. 4), we impose this on mesh boundaries. This simplifies to asking that $\sigma_c = 0$ for edges in ∂M. This alignment is important to ensure knittable output at the boundaries, especially when they are not very smooth, i.e., have high geodesic curvature. This is succinctly expressed as $\sigma_c\big|_{\partial M} = 0$.

3.2.3 Level set constraints. Instead of specifying stripe alignment exactly, one may also ask that the path integral of σ_* be equal to 0 along a specified polyline γ. This ensures that level sets (and thus stripes) do not cross γ without ultimately returning to their side of origin. This is a less restrictive way to join singular triangles and eliminate helices, as can be seen in Fig. 6.

These are referred to as "rotation order" constraints in [Noma et al. 2022], and were expressed on open paths as:

$$\int_\gamma \sigma = \sum_{i=0}^{n-1} -s_i^{i+1}\sigma_{e_i} r_i^{i+1} + s_{i+1}^{i}\sigma_{e_{i+1}} r_{i+1}^{i} = C$$

for some $C \in \mathbb{R}$. For σ_c, we only use $C = 0$, while for σ_w nontrivial values are useful, as noted at the end of this section. As with stripe alignment constraints, the triangles the polyline passes through must be non-singular. Furthermore, we argue in §5 of the supplementary that we can consider, without loss of generality, only level set constraints over polylines that are along mesh edges. In this same section, we establish:

Lemma 1. *Level set constraints are equivalent when applied to homologous curves on $M' = M \setminus K$.*

Intuitively, this follows as $(d_1\sigma)_m$ is the integral sum around the boundary of triangle m, and if $(d_1\sigma)_m = 0$, then it should not matter which "side" of triangle m that γ passes on. If there are N^{ls} level set constraints, we denote the polylines $\{\gamma_j^{\text{ls}}\}_{j=1}^{N^{\text{ls}}}$.

Helix elimination constraints. Unlike [Noma et al. 2022], we also consider level set constraints on closed cycles, referring to them as *helix elimination* constraints. We characterize helices via the knit graphs that result from tracing the course and wale stripe patterns (described in supplementary §3). Any helix in the knit graph arises from a helical stripe, following a level set that is birthed at one singular triangle and ends at another. We argue the following two remarks in supplementary §6, §7.

Remark 1. *Helical stripes or level sets may not be defined without reference to wale stripes, and their intersections with them.*

Remark 2. *A helical stripe starting at singular triangle a and ending at singular triangle b, can be effectively eliminated with a*

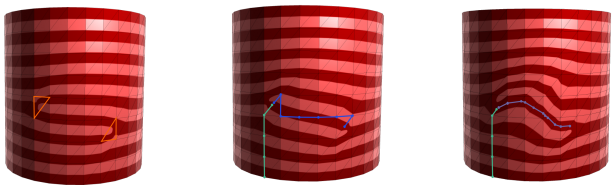


Figure 6: Various constraints demonstrated. Left: Stitch irregularity placements at the red triangles with stripe index $+1, -1$ induce a helical stripe. Middle: A stripe placement constraint (green) and a level set constraint (blue) match up the singular triangles and eliminate the imposed helix. Right: A stripe alignment constraint (green) and a level set constraint (blue) eliminate the helix. Note that the level set constraint is more flexible, and does not force the stripes to strictly conform to it.

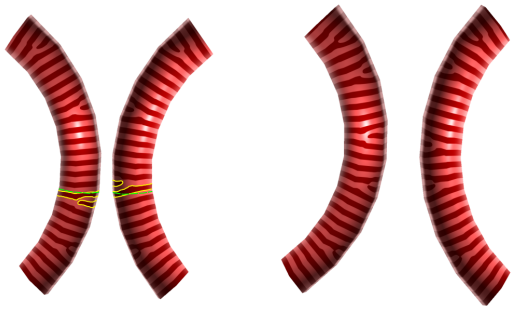


Figure 7: The helical stripe highlighted in yellow is fixed by a helix elimination constraint imposed along the bright green cycle.

helix elimination constraint along a cycle that separates a and b locally.

Practically speaking, we use cycles that roughly follow level sets of the time function. An example of one in action is shown in Fig. 7. If there are N^{he} helix elimination constraints, we denote the corresponding cycles $\{\gamma_j^{\text{he}}\}_{j=1}^{N^{\text{he}}}$.

Wale constraints. Lastly, we do specify nontrivial ($C \neq 0$) level set constraints when optimizing for σ_w. On ∂M, we ask that $\int_{(\partial M)_*^i} \sigma_w = Pk_*^i$ for some $k_*^i \in \mathbb{Z}$ for each boundary component, except one. The path integral over the unconstrained boundary will be an integer multiple of P by Thm. 3.1. This guarantees that the wale stripes "close up" along boundary components (see Fig. 8).

3.2.4 Stripe placement constraints. All previous constraints may be combined with stripe placement constraints which force stripes to pass through certain points on the mesh. These can be used to more precisely place short row ends and increases/decreases, especially when the mesh is relatively coarse in comparison to the stripe period P.

This is done via path integral constraints along a polyline γ that reaches from one of our boundaries to the point in question. From there, we specify the following integral constraints:

$$\int_\gamma \sigma = Pk + \frac{P}{4}, \qquad \text{for some } k \in Z$$

The additional increment of $P/4$ is used, as the stripe is centered on the $P/4$ level set of the texture function (by our convention). Note that the particular path chosen does not matter, as we are merely trying to specify the stripe function (mod P). This integral will take a special form for points inside of triangles, depending on whether the triangle is singular or not, presented in supplementary §8, §9. If there are N^{sp} such constraints, we denote the corresponding polylines $\{\gamma_j^{\text{sp}}\}_{j=1}^{N^{\text{sp}}}$, and their integer variables k_j^{sp}.

Lastly, we note that a version of such constraints is used to achieve stripes tight to all boundaries. This requires a constraint that adds $P/2$ as the additional increment, and is imposed on $N-1$ paths $\{\gamma_j^{\text{bs}}\}_{j=1}^{N^{\text{bs}}}$, joining one (arbitrary) root boundary component to another. This may be seen in Fig. 8.

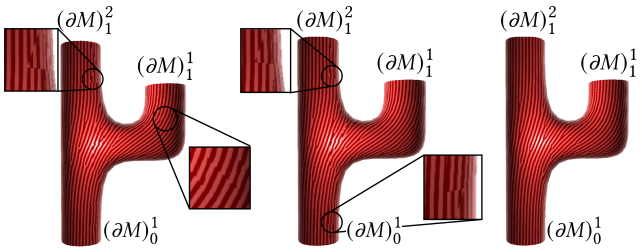


Figure 8: Effect of wale level set constraints. Left: no constraints allows stripe discontinuities near all boundaries; center: $\int_{(\partial M)_1^1} \sigma_w = Pk$, $k \in Z$ allows stripe discontinuities near $(\partial M)_0^1$ and $(\partial M)_1^2$. Right: constraints imposed on two boundaries ensures no stripe discontinuities.

4 KNIT GRAPH CREATION

We describe two strategies for using the above objective and constraints to optimize for course and wale stripe patterns. Course pattern optimizations are described first, followed by the necessary modifications for wale pattern generation.

4.1 Direct Short Row End Pairing (S1)

In this strategy, we assume input of singular triangle placements and stripe indices, which may be obtained from applying [Knöppel et al. 2015], a heuristic solution to S2 (§4.2), user input, or a combination of these. These singular triangles and their stripe indices are then matched with stripe alignment or level set constraints, to ensure that short rows will be connected by non-helical courses.

As a general guide, the singular triangles should be matched based on distance on a Reeb graph, computed with respect to the time function h. In future work, we envision automating this process, but currently leave it to the user to specify this approximately, or deviate from it for design purposes.

The matching constraints usually guarantee that there are no remaining helices in the knit graph. In some instances, stripe placement constraints must be used to tweak the stripe pattern and ensure a lack of helices when tracing the pattern. We note that when the triangles are fine relative to the stripe period, these problems are unlikely to occur. We show the optimization for σ_c.

$$\min_{\sigma_c, \mathbf{k}^{\text{bs}}, \mathbf{k}^{\text{sp}}} \quad ||W(\sigma_c - \omega_c)||^2 \tag{4a}$$

$$\text{subject to} \quad \sigma_c\big|_{\partial M} = 0, \tag{4b}$$

$$d_1\sigma_c = P\mathbf{k} \tag{4c}$$

$$\int_{\gamma_j^{\text{ls}}} \sigma_c = 0, \qquad 1 \le j \le N^{\text{ls}} \tag{4d}$$

$$\int_{\gamma_j^{\text{bs}}} \sigma_c = Pk_j^{\text{bs}} + \frac{P}{2}, \qquad 1 \le j \le N-1 \tag{4e}$$

$$\int_{\gamma_j^{\text{sp}}} \sigma_c = Pk_j^{\text{sp}} + \frac{P}{4}, \qquad 1 \le j \le N^{\text{sp}} \tag{4f}$$

$$s_i^{i+1}\sigma_{e_i^j} r_i^{i+1} = s_{i+1}^i \sigma_{e_{i+1}^j} r_{i+1}^i, 1 \le j \le N^{\text{sa}},\ 1 \le i \le n-1 \tag{4g}$$

Here, $\sigma_c \in \mathbb{R}^{|E|}$, $\mathbf{k}^{bs} \in \mathbb{Z}^{N-1}$, $\mathbf{k}^{sp} \in \mathbb{Z}^{N^{sp}}$ are the optimization variables and stripe index constraints are incorporated with $\mathbf{k} \in \mathbb{Z}^{|F|}$. The optimization may be solved quickly, as it is a quadratic mixed-integer problem with few integer variables. More precisely, the number of integer variables is bounded above by the number of boundaries and the stripe placement constraints, both typically small, resulting in quick runtimes (Table 1).

To optimize for σ_w and the resulting wales, we replace ω_c with ω_w and may adjust or remove many of the above constraints. In particular, we replace (4b) with

$$\int_{(\partial M)_*^i} \sigma_w = Pk_*^i,\ 1 \le i \le N-1, \tag{5}$$

as noted in §3.2.3. Beyond this, there are only stripe index and possible stripe alignments constraints to consider, so (4e), (4f) and (4g) are removed.

4.2 Mixed-Integer Irregularity Placement (S2)

In this strategy, we aim to produce stitch irregularities in geometrically-informed positions with a global mixed-integer optimization. While the aim is similar to [Knöppel et al. 2015], our constraints ensure stripe patterns suitable for knit graph creation. The optimization problem for σ_c is given by,

$$\min_{\sigma_c, \mathbf{k}, \mathbf{k}^{\text{bs}}} \quad ||W(\sigma_c - \omega_c)||^2 \tag{6a}$$

$$\text{subject to} \quad \sigma_c\big|_{\partial M} = 0, \tag{6b}$$

$$d_1\sigma_c = P\mathbf{k} \tag{6c}$$

$$\int_{\gamma_j^{\text{he}}} \sigma_c = 0, \qquad 1 \le j \le N^{\text{he}} \tag{6d}$$

$$\int_{\gamma_j^{\text{bs}}} \sigma_c = Pk_j^{\text{bs}} + \frac{P}{2}, \qquad 1 \le j \le N-1 \tag{6e}$$

$$s_i^{i+1}\sigma_{e_i^j} r_i^{i+1} = s_{i+1}^i \sigma_{e_{i+1}^j} r_{i+1}^i, 1 \le j \le N^{\text{sa}},\ 1 \le i \le n-1 \tag{6f}$$

The stripe indices, $\mathbf{k}$, are now integer variables, as opposed to being specified. Application of this strategy requires multiple iterations of the above optimization. Between iterations, helical stripes are identified and removed with helix elimination constraints applied along cycles that roughly follow the isocontours of h. As the number of integer variables are on the order of $|F|$, solving to optimality is prohibitively expensive. However, good approximate solutions may be found relatively quickly with mixed-integer optimization packages, e.g., Gurobi [Gurobi Optimization, LLC 2022].

In solving for σ_w for wale columns, we again replace ω_c with ω_w, (6b) with (5), and we drop (6e), (6f). Furthermore, for wale columns, there is no need to solve the mixed-integer problem multiple times, as there are no helical stripes to be concerned with.

The above strategies are currently manually applied, but automation ideas are being pursued as part of future work.

4.3 Level Set Tracing

With course and wale stripe patterns determined via σ_c and σ_w, we trace level sets to find knit graph vertices and edges. We first integrate σ_c and σ_w along a spanning tree of mesh edges and store the values (mod P) at the vertices, denoted α_{mod} and β_{mod}, respectively. Locally, these integrated values define two functions, α and β over each triangle. Intersections of level sets give us the knit graph vertices. Knit graph edges are constructed per triangle face by considering the relative values of α and β at each knit graph vertex. For knit graph edges going between neighboring triangles, we create "virtual" knit graph vertices on triangle borders that are later merged and eliminated in a distance-based manner. A detailed description of stripe tracing is presented in supplementary §3.

5 RESULTS

We applied the two approaches above to a variety of mesh models that we illustrate throughout the text and in Fig. 9. Further commentary is provided in the caption. The resulting knit graphs were fed to AutoKnit's [Narayanan et al. 2018] tracer and scheduler and the knit models are shown, if they successfully completed. All models were knit on a 7-gauge Shima Seiki SWG091N2 V-bed knitting machine. Functionally speaking, we consider our knit graph to be machine-knittable if AutoKnit successfully produces a .dat file. Downstream issues that arise from the use of individual machines and their settings are not in the purview of this work. The .dat files for all models are included in supplementary materials, as well as raw photos of the successful knits.

Table 1: Run time statistics.

Model	Strategy	#V	#F	# Int. Vars.	MIP Solve Time, s
Curved Cylinder (Fig. 7)	S2	54	96	97	< 5
Hemisphere (Fig. 9(e))	S2	224	384	385	40
Sock (Fig. 1)	S2	279	538	98	21
Sock (Fig. 9(d))	S1	279	538	2	< 1
Cylinder (Fig. 6, middle)	S1	270	580	3	< 1
Cactus (Fig. 9(c))	S2	391	736	738	35
Bunny (Fig. 4, right)	S2	2669	5228	5230	287

We also present some timing data for the optimization problems of §4. The vanilla Gurobi [Gurobi Optimization, LLC 2022] solver was used, on a a 2.3GHz Intel Core i5 Macbook Pro with 8GB of RAM. As can be seen, S1 results in interactive runtimes, due to the low number of integer variables, while S2 requires use of approximate heuristic solutions. For S2, optimizations are stopped at the times listed, determined via user calibration.

Lastly, we present in the supplementary §4 some histograms of edge length error for knit graph edges. As in [Narayanan et al. 2018], most edge lengths are within 10% of the target length, as defined by the stripe period P.

6 CONCLUSION

We have demonstrated the utility of a spinning-form-based framework for generating course and wale stripe patterns. It provides smooth knit graphs (Fig. 2), and application of simple linear constraints avoids the pitfalls of direct use of [Knöppel et al. 2015] (Fig. 4). In particular, we demonstrate the use of helix elimination constraints for eliminating helical stripes, giving us machine-knittable knit graphs upon tracing, as demonstrated throughout the paper and in Fig. 9. A thorough discussion of this and other constraints and their knitting implications is present in §3.2, with accompanying theoretical analysis in the supplementary. Lastly, we believe that the insights presented here will be useful in other frameworks, suggesting the use of stripes-based methods to determine positioning of stitch irregularities, instead of the more commonplace distance computations and quad meshing.

Limitations/Future Work. Currently, our strategies are manually implemented, and automating them would be natural to pursue. On a related note, we do not include any modification of the guiding vector field for stripe pattern optimization. This could be done via

modification of the time function h or of ∇h and could be done with any number of vector field design methods. When used in conjunction with our constraints, this could improve their efficacy and thus user control.

Lastly, S1 and S2 represent tradeoffs that we'd like to improve. S1 is fast, but requires input on singularity placement, while S2 requires a lengthy approximate solve time. Much work may still be done to achieve a method that is both quick and produces globally-informed singularities. Strategies that come to mind immediately include using [Knöppel et al. 2015] to set branching priorities in a branch-and-bound algorithm, or using multi-resolution frameworks to process meshes in a coarse-to-fine manner.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation (NSF) under Grant No. 2047342 and the National Science Foundation Graduate Research Fellowship (NSF GRF) under Grant No. 2141064. We would like to thank James McCann, Kui Wu, and Alexandre Kaspar for permission to use figures.

REFERENCES

David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum* 30, 2 (2011), 375–384.

Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. 2013. Digital Geometry Processing with Discrete Exterior Calculus. In *ACM SIGGRAPH 2013 courses* (Anaheim, California) *(SIGGRAPH '13)*. ACM, New York, NY, USA, 126 pages.

Gurobi Optimization, LLC. 2022. Gurobi Optimizer Reference Manual. https://www.gurobi.com

Allen Hatcher. 2000. *Algebraic topology*. Cambridge Univ. Press, Cambridge.

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

Benjamin Jones, Yuxuan Mei, Haisen Zhao, Taylor Gotfrid, Jennifer Mankoff, and Adriana Schulz. 2022. Computational Design of Knit Templates. *ACM Transactions on Graphics* 41, 2 (April 2022), 1–16.

Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. 2019. Knitting Skeletons: Computer-Aided Design Tool for Shaping and Patterning of Knitted Garments. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*.

Alexandre Kaspar, Kui Wu, Yiyue Luo, Liane Makatura, and Wojciech Matusik. 2021. Knit Sketching: from Cut & Sew Patterns to Machine-Knit Garments. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 40, 4 (2021).

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe patterns on surfaces. *ACM Transactions on Graphics* 34, 4 (July 2015), 39:1–39:11.

Georges Nader, Quek-Yu Han, Chia-Pei Zhi, Oliver Weeger, and Sai-Kit Yeung. 2021. KnitKit. *ACM Transactions on Graphics (TOG)* (July 2021).

Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James Mccann. 2018. Automatic Machine Knitting of 3D Meshes. *ACM Transactions on Graphics* 37, 3 (Aug. 2018), 35:1–35:15.

Vidya Narayanan, Kui Wu, Cem Yuksel, and James McCann. 2019. Visual knitting machine programming. *ACM Transactions on Graphics* 38, 4 (July 2019), 63:1–63:13.

Yuta Noma, Nobuyuki Umetani, and Yoshihiro Kawahara. 2022. Fast Editing of Singularities in Field-Aligned Stripe Patterns. In *SIGGRAPH Asia 2022 Conference Papers* (Daegu, Republic of Korea) *(SA '22)*. Association for Computing Machinery, New York, NY, USA, Article 37, 8 pages.

Thibault Tricard, Vincent Tavernier, Cédric Zanni, Jonàs Martínez, Pierre-Alexandre Hugron, Fabrice Neyret, and Sylvain Lefebvre. 2020. Freely orientable microstructures for designing deformable 3D prints. *ACM Trans. Graph.* 39, 6 (2020), 211–1.

Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. 2018. Stitch meshing. *ACM Transactions on Graphics* 37, 4 (Aug. 2018), 1–14.

Kui Wu, Hannah Swan, and Cem Yuksel. 2019. Knittable Stitch Meshes. *ACM Transactions on Graphics* 38, 1 (Feb. 2019), 1–13.

Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2012. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Transactions on Graphics (TOG)* (July 2012).

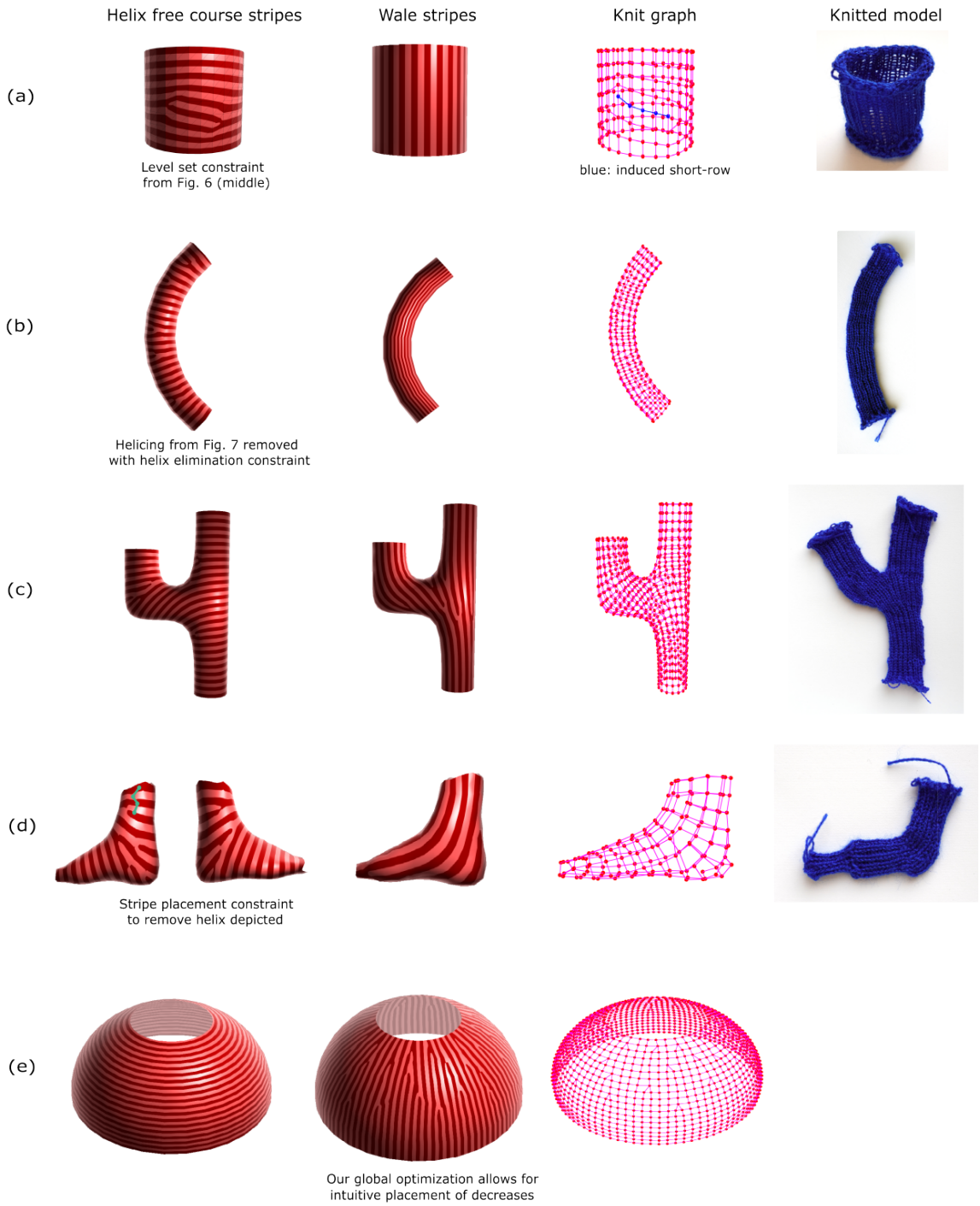


Figure 9: Results. (a) A forced helix (by specifying offset singular triangles) is fixed with both a level set constraint and a stripe alignment constraint (see Fig. 6). Both lead to a knit graph with identical connectivity. (b) A forced helix is fixed with a helix elimination constraint (see Fig. 7). (c) An application of S2 produces a helix-free knit graph for this surface of non-cylinder topology. (d) A helical stripe is fixed with a stripe placement constraint on the sock model. This complements the use of S2 on the same model in Fig. 1. (e) An application of S2 produces a reasonable distribution of decreases on this hemispherical model. Although, we generate a knit graph that is knittable according to [Narayanan et al. 2018], the knitting process failed on two machines. [Kaspar et al. 2019] discusses potential reasons for failure (e.g. yarn tension). For our purposes, "knittability" is defined relative to our ability to pass the knit graph constraints of [Narayanan et al. 2018]. The .dat file is presented in the supplementary material.