Wearable 3D Machine Knitting: Automatic Generation of Shaped Knit Sheets to Cover Real-World Objects

Kui Wu, Marco Tarini, Cem Yuksel, James McCann, Xifeng Gao

Abstract—Knitting can efficiently fabricate stretchable and durable soft surfaces. These surfaces are often designed to be worn on solid objects as covers, garments, and accessories. Given a 3D model, we consider a knit for it *wearable* if the knit not only reproduces the shape of the 3D model but also can be put on and taken off from the model without deforming the model. This "wearability" places additional constraints on surface design and fabrication, which existing machine knitting approaches do not take into account. We introduce the first practical automatic pipeline to generate knit designs that are both wearable and machine knittable. Our pipeline handles knittability and wearability with two separate modules that run in parallel. Specifically, given a 3D object and its corresponding 3D garment surface, our approach first converts the garment surface into a topological disc by introducing a set of cuts. The resulting cut surface is then fed into a physically-based unclothing simulation module to ensure the garment's wearability over the object. The unclothing simulation determines which of the previously introduced cuts could be sewn permanently without impacting wearability. Concurrently, the cut surface is converted into an anisotropic stitch mesh. Then, our novel, stochastic, any-time flat-knitting scheduler generates fabrication instructions for an industrial knitting machine. Finally, we fabricate the garment and manually assemble it into one complete covering worn by the target object. We demonstrate our method's robustness and knitting efficiency by fabricating models with various topological and geometric complexities. Further, we show that our method can be incorporated into a knitting design tool for creating knitted garments with customized patterns.

Index Terms—knitting, fabrication, stitch meshes.

1 Introduction

Machine knitting is a powerful mechanism for fabricating 3D shapes. Knitting machines can naturally produce curved surfaces directly in 3D, using various shaping techniques, i.e., short-rows and increase/decrease stitches, as shown in Fig. 3. While knitted artifacts can typically withstand relatively strong external forces without breaking, most yarn materials form soft fabrics that easily deform, even with their own weights under gravity. This makes knitting an ideal way to fabricate garments and coverings with user-defined shapes (e.g., upholstery, equipment covers).

In the commercial sphere, knitting machine programming remains a difficult process which must be undertaken by skilled engineers working in low-level languages. Recent research has made strides in automating the programming of machine-knittable 3D shapes [1], [2], [3], [4]. However, these methods do not consider the eventual assembly of the fabricated knit artifact with a target solid object. Therefore, the resulting knit artifacts with these methods may not be placed over the target real-world solid objects and, thereby, fail to fulfill their main purpose.

- *K. Wu is with Massachusetts Institute of Technology.*
- *M. Tarini is with the University of Milan*
- *C. Yuksel is with University of Utah*
- *J. McCann is with Carnegie Mellon University*
- *X. Gao is with Florida State University*
- *Corresponding Author: Xifeng Gao, E-mail: gao@cs.fsu.edu*

Manuscript received August 2, 2020; revised November 6, 2020.

To address this crucial problem, we define the term *wearability* in the context of fabricating knit artifacts. Given a target configuration of a deformable 3D surface S and rigid object O, both embedded in $\mathbb{R}^3$, we define fabricated deformable surface S' as *wearable* regarding O if it can be near-isometrically deformed to S without intersecting O.

Though most previous works focus on tubular knitting [2], [5], tubular surfaces are often *not* wearable by our definition for rigid 3D shapes (e.g., a cover for a torus or a sweater for a statue of a character in a T pose). Rather than creating tubes that must be cut to obtain wearability, our framework follows the industry-standard practice of "fully-fashioned" knitting – creating shaped sheets that are hand-assembled into the final product.

In this work, we present the first pipeline that can automatically produce a *wearable* and *machine-knittable* object S' given a 3D garment surface S and a target rigid object O. In particular, given S and O, our pipeline determines the set of stitches and knitting instructions needed to fabricate a garment, including automatically determining and placing cuts to ensure that the resulting garment can be slipped onto the target 3D object. These cuts are closed by lacing manually after the garment is placed on the object. Note that though our pipeline is designed to create coverings that conform to the target solid, it is not limited to it. Our system supports designing full coverings, partial coverings, and even loose coverings.

Our pipeline first cuts the input 3D surface into a topological disc (*cut-mesh*). It then runs two modules in parallel: the wearability test and the machine-knittability test. The

wearability test is used to determine which edges of the cut-mesh should be temporarily laced and which may be permanently sewn. Concurrently, the machine-knittability module generates an anisotropic *stitch meshes* [6], and schedules the machine knitting instructions, adding additional cuts as needed. Finally, the garment is fabricated using an industrial knitting machine, and cuts are manually assembled using sewing or lacing (as determined by the pipeline). We verify our pipeline by fabricating models with various topological and geometrical complexities that cannot be handled by state-of-the-art knitting techniques.

We particularly wish to emphasize that by focusing on disc-based knitting (flat knitting), we are filling a gap in the current automatic knitting design space. This is a particularly relevant gap, given that many currently-deployed knitting machines don't include the take-down systems necessary to deal with complex shaping on tubes, and knitting of shaped sheets that are hand-assembled ("fully-fashioned" knitting) remains industry-standard practice.

In summary, our key technical contributions are the following:

- an automatic cutting algorithm to turn any 3D surface into a knittable topological disc;
- an anisotropic stitch meshing technique and a set of local mesh modifications to improve knitting stability;
- a stochastic, any-time scheduler for knit sheets;
- a conservative simulation scheme to reduce lacing cuts while preserving wearability.
- a complete system that puts it all together to guarantee both machine-knittability and wearability for 3D knitting design.

2 Background

Before we discuss the details of our method, we provide a brief review of the most related prior work on fabrication, knitting, surface cutting, and undressing covers.

2.1 Fabricating Surfaces

Researchers have considered the problem of 3D surface fabrication in many areas. For instance, freeform surfaces can be discretized with planar polygons for fabrication from hard materials like glass (e.g., [7], [8], [9], [10]). These techniques extend to the soft domain, including designing custom plush toys [11], inflatable structures [12], and garments (e.g., [13], [14]). Researchers have also proposed novel techniques to improve the fabrication, e.g., woven wires [15], pre-stretched 2D cloth pieces embedded in a planar rod network [16], ribbon-like pieces of fabric attached with zippers [17], and fabric formwork [18]. Closest to our work is the existing research on covers for 3D objects [19], [20]; but while these previous approaches create covers from cut-and-sewn flat cloth, our system uses the intrinsic shaping available with machine knitting to produce surfaces with a large range of curvature.

2.2 3D Forming Knitting

Knitting is a technique to produce fabric from yarn by forming *stitches*. The juxtaposition of various stitch types (e.g., *knit*, *increase*, *decrease*, and *short-row*) allow fabrics with complex surface textures [21] and 3D shapes [22] to be formed. Overall, knitting allows the creation of curved 2D surfaces from a wide variety of yarn materials; but designing yarn-level structures for given target shapes remains complex.

It is known that a 2D surface of any topology can be hand-knit [23]. Igarashi et al. [24], [25] presented a design software that semi-automatically creates a knitting pattern from a 3D model by covering the surface with a winding strip and finding areas where increase or decrease type stitches are needed. Yuksel et al. [6] introduced stitch meshes, a data structure for modeling knitting structures. Recently, Wu et al. [26] introduced an automatic pipeline for converting arbitrary shapes into quad-dominant stitch meshes, though their results are not guaranteed to be knittable. Wu et al. [27] also extended stitch meshes to represent complex 3D hand-knittable structures.

Knitting machines add more complexity to the knitting design process. Industrial knitting machines use hook-like shaped *needles* in two rows: the *front bed* and *back bed* (Fig. 1). These needles are used for holding the yarn loops and manipulating the loops to create stitches. The needles on the two beds (front and back) are aligned with each other, allowing yarn loops to be transferred between beds. Knitting machines can also *rack* (laterally move) the beds to change which needles are aligned, which can be used for moving loops along the beds. Much of the complexity of designing for these machines comes from deciding how to *schedule* loops to needles.

Commercial knitting design software includes built-in templates to defray some of this design complexity [28], [29], [30]; however, these templates are limited to standard designs, e.g., sweaters, jackets, gloves, and skirts. More complex or non-standard patterns must be hand-designed at the stitch level, though there exist guidebooks of advanced techniques that can assist with this process [22].

Meißner and Eberhardt [31] proposed one of the earliest methods to visualize machine knittable structures during design, using particles to simulate the dynamics of yarn-level structures. Researchers have proposed alternative design tools including mid-level primitives [1], [4], direct conversion of 3D models [2], [3], and stitch-level output-domain editing [5]. As these systems become more prevalent, researchers are also beginning to examine how to create more efficient patterns by changing how loops are moved between needles [32] and inlay strong yarn as tendons [33]. Much of this work has been enabled by the generic low-level knitting assembly language, knitout [34], which we also use in the present work. The system proposed by [3] is the closest to the method introduced in this paper. While their

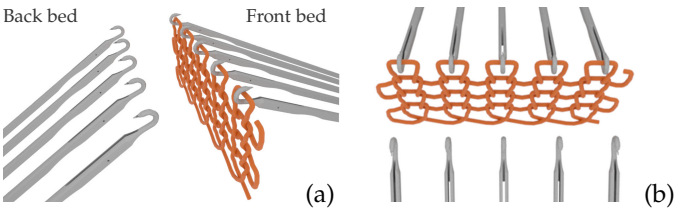


Fig. 1. **Machine needles and beds**: (a) side view and (b) top view.

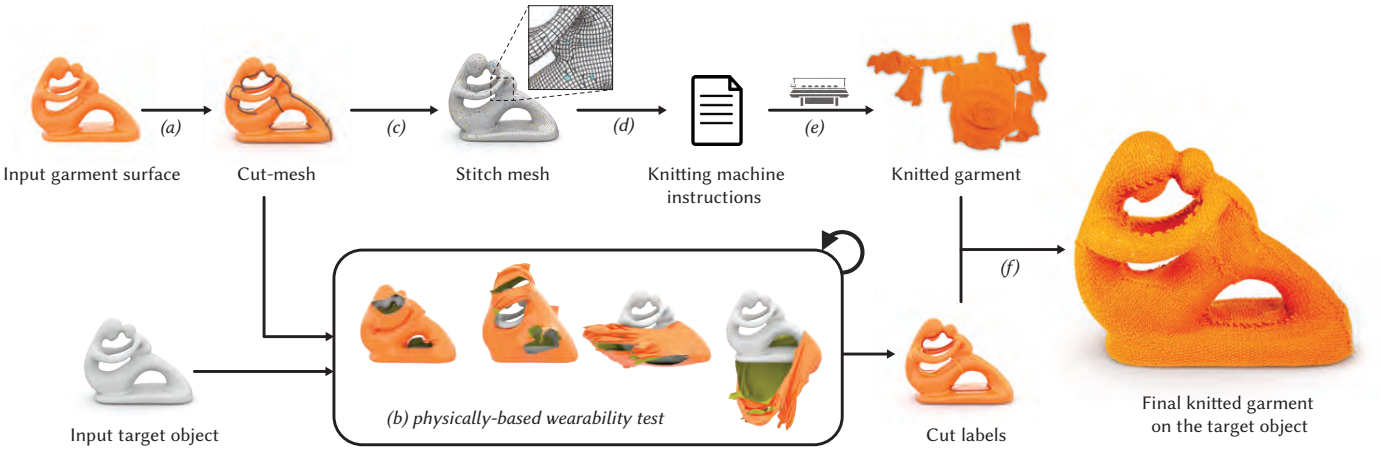


Fig. 2. **Pipeline overview:** our pipeline converts an input garment surface into a machine-knittable output wearable by an input target object. (a) the garment surface is cut into a disc; (b) a physically-based simulation is used to verify the wearability of the *cut-mesh* with given cut label assignments; (c) an anisotropic stitch mesh is generated from the initial cut-mesh; (d) knitting instructions are generated and (e) sent to an industrial knitting machine for fabrication; and (f) the knitted fabric is manually sewn, the target object is dressed, and the garment is laced.

method requires placing directional singularities manually to split the input surface into a number of topologically-disc-shaped patches, our system achieves singularity-placement automatically and generates a single topological disc. Directional and positional singularities in stitch meshes correspond to unknittable mismatch and irregular faces (short-row and increase/decrease). Please refer to [35] for more details about singularities. In order to design any knitted item, an underlying representation is required. For this, our pipeline uses the augmented stitch meshes structure, as shown in Fig. 3, embedded with machine knitting instructions. More details can be found in [5], [6].

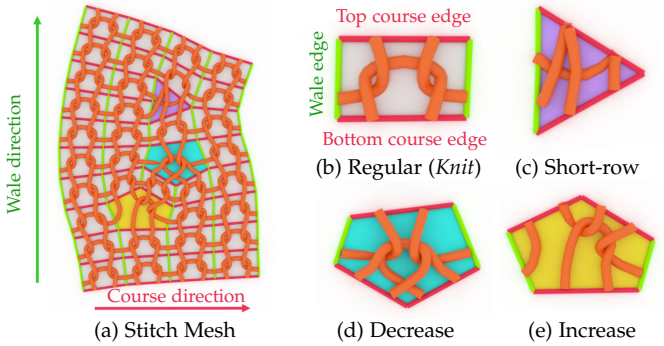


Fig. 3. **Stitch mesh representation:** (a) an example of stitch mesh patch with corresponding yarn geometry; (b) regular quad stitch mesh with the yarn geometry of *Knit* type; (c) short-row face, (d) decrease, and (e) increase.

2.3 Surface Cutting

In geometry processing, cuts are used during parameterization to minimizes mapping distortions between a cut surface and a planar domain with either a fixed [36] or freely moving [37], [38] boundary. Gu et al. [39] proposed a simple strategy to cut a manifold surface into a topological disk, which provides the first step for parameterization. More recent works provide topological and geometrical guarantees, such as locally injective mapping and intersection-freedom [40], [41]. While many parameterization works assume a fixed cut and start their optimization from the Tutte embedding [42], [43]; cut modification has been shown to lower distortion in UV mapping [44] and surface flattening [45]. Li et al. [46] recently introduced a joint optimization method for minimizing both seam lengths and distortion, though the resulting irregular seams are not ideal for machine knitting. Surface cuts have also been employed to connect directional singularities in order to generate pure quad meshes [47], [48]. For this purpose, the distortion of the cut is often ignored. While the above is by no means a comprehensive literature review on surface cutting, to the best of our knowledge, our cut algorithm is the first one that is designed for easing physical constraints from knitting machines and the wearability of the garment.

3 System Overview

Our system's input is a 3D surface model for the garment and another 3D model representing the solid object (i.e. the *target object*) that will wear the garment. The output of our system is a knitting-machine-fabricated garment that can be worn on the target object. The first step of our system is to identify a suitable set of cuts of the garment surface model. The cuts here serve two purposes: to allow the knitting machine to produce the garment (*knittability*), and to allow the garment to be put over the target object (*wearability*). We refer to the resulting cut surface as the *cut-mesh* and the edges that are cut during this process as the *cut-edges*. After fabrication, the cut-edges will either be permanently sewn back together or temporarily laced – as determined by the wearability test result. Our system performs two separate pipelines for the cut-mesh, testing the wearability and ensuring knittability. As shown in Fig. 2, the system includes the following steps:

- **Surface Cutting** (Section 4). A patch-growing algorithm is used to cut the input 3D surface into a topological disc, while no singularities are allowed in the interior of the cut-mesh. Our method also employs two soft criteria, improving cut straightness and lessening 2D overlaps.
- **Wearability Test** (Section 5). The system iteratively tests the wearability of the cut-mesh while eliminating cut-edges until no more cuts can be eliminated.

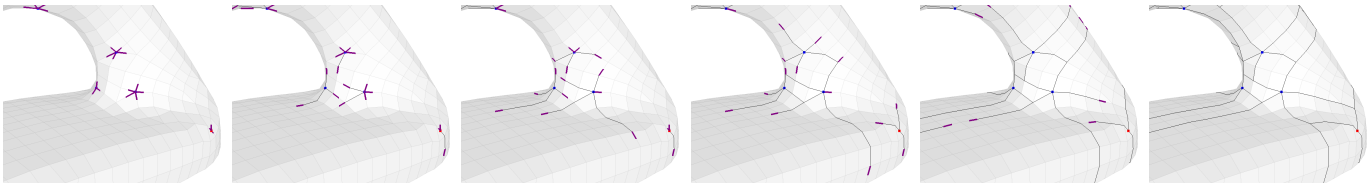


Fig. 4. The motorcycle-graph procedure is used to identify straight sequences of edges, which are then favoured to be elected as part of the cuts (thus encouraging straightness): "motorcycle" particles (black thick segments) are spawned from irregular vertices (dots) and traced over the quad-dominant mesh.

The eliminated cut-edges are labeled for sewing (permanent assembly), while the remainder are labeled for lacing (reversible assembly).

- **Anisotropic Stitch Meshing** (Section 6). Concurrently to the wearability test, our system generates an anisotropic stitch mesh from the initial cut-mesh. Further, our method adjusts the mesh's local connectivity to remove invalid faces and avoid unreliable knitting. This ensures that the fabricated garment is the correct size and can be knitted smoothly.
- **Knitting Instruction Generation** (Section 7). In the final algorithmic step, our system generates machine knitting instructions using our stochastic sheet-based scheduler. Additional cuts (labeled for sewing) may be automatically introduced to guarantee the knittability during this step.
- **Fabrication and Assembly.** The finished garment is produced by running the low-level instructions on a knitting machine. Then, the cuts labeled for sewing are manually sewn. The target object can then be dressed, and the remaining cuts closed by lacing.

4 Surface Cutting

In this section, we describe the algorithm used to produce the initial set of cuts. Our cutting guarantees that:

1) the cut surface has no interior singularities because inconsistent course/wale labeling will preclude knittability;
2) and the cut surface is a single topological disk, which provides a good starting configuration for ensuring wearability.

4.1 Selecting Cuts

The input surface S is first converted into a quad (-dominant) proxy mesh Q, from which a *cross field* [49] – a 2-fold rotationally-symmetric field can be derived. The cross field is used to define the labeling of edges of the cut-mesh as either course or wale. When Q is a quad-dominant mesh, our code splits all pentagons into one quad and one triangle by adding the edge that splits the largest corner. Then, for triangular faces, the most acute corner is labeled as a *vanishing corner*: the two edges it connects are considered to represent a pair of *opposite* wale or course edges rather than one wale edge and one course edge.

We employ a *patch-growing* algorithm, similar to [39], but extended to deal with singularities. The algorithm begins forming a patch with an arbitrary *seed* quad, the edges of which are labeled using field directions and grows it, one face at a time. During this process, the boundary B of the patch is tracked as a circular sequence of half-edges of Q. At the end of this process, B represents the initial cuts (plus any original mesh boundaries). Specifically, each growing step consists of three steps: edge selection, expansion, and fusion.

Edge selection randomly picks any one edge e in B that neighbors a non-visited face f. Then, *expansion* attaches that face into the patch over e; in B, edge e is replaced by all edges of f except e. Edge labels are propagated consistently to the attached face. Last, *fusion* removes one pair of edges if there are two consecutive elements of B on opposite sides of the same edge. Importantly, this move is only allowed if the pair of dissolved half edges is labeled as having *opposite* directions (incidentally, which prevents B from being reduced to an empty set).

Since consistency is enforced during patch growth, no field singularity can exist in the interior of the patch. Furthermore, this process opens a connected Q into a single simply connected disk: after seeding, the visited region consists of a single face, and neither type of move changes its topology. Since each face is only visited once, no loop will be formed.

4.2 Improving Cut Selection

Although our goal is accuracy rather than aesthetics, we still add some guidance to edge expansion. We assign each edge in B a *cuttability score* and have cutting expand over the edge with the lowest cuttability score. This approach is able to accommodate any user-specified criteria for assigning cuttability scores (for example, to discourage cuts over certain regions). In our examples, we always employ a combination of two soft criteria: *improving cut straightness* and *lessening 2D overlaps*, as explained below.

Improving cut straightness

To discourage the sequence of boundaries edges that are zig-zagging, our algorithm assigns higher cuttability scores to certain straight sequences of edges stemming from irregular vertices, identified beforehand using an algorithm (Fig. 4) similar to the generalized version [50] of *motorcycle graphs* [51]. First, our code identifies the *irregular vertices*: the internal vertices of Q surrounded by $n \neq 4$ polygon corners (not counting vanishing corners of triangles). From each irregular vertex, it spawns n particles called *motorcycles*, one along each edge. A motorcycle travels across edges in a topologically straight fashion until it reaches a vertex that is previously visited by a motorcycle or is on the boundary of the mesh. When a motorcycle reaches the vanishing corner of a triangle, it randomly picks one of the two possible "straight" edges. The cuttability score of edges traversed by a motorcycle is increased by a user-defined constant c_s.

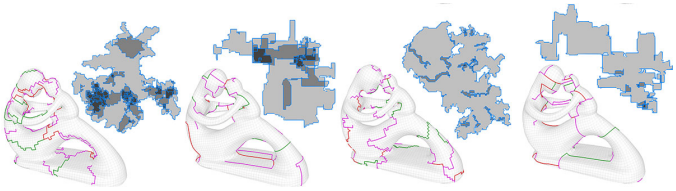


Fig. 5. **Results of the initial cut selection procedure with different edge prioritization strategies:** From left to right, the purely random selection of expansion move, cut-straightening only, overlap avoidance only, and the combination of the two criteria, which we use in all other examples. In each case, we show the cut over the mesh Q. The approximated parametric space (with overlaps in darker grays – on the right, the patch boundary in 2D may appear artificially less straight due to the approximations of parametric position estimation).

Lessening 2D overlaps

We discourage expansion moves that would cause the patch to grow, in 2D, over other areas of itself, especially when the extent of the overlap or the number of overlaid layers is large, because *excessive* overlaps can force later knit scheduling steps to add additional cuts (Section 7). To this end, our algorithm estimates the parametric 2D positions of each face of Q inside the patch, which it uses to compute an estimated overlap for each expansion move (and, thus, modifies its cuttability score).

Specifically, when inserting a face q_j in Q, we determine its 2D integer parametric position (u_i, v_j). The initial seed is assigned, arbitrary, e.g. to $(0, 0)$. During an expansion move, the newly included face's position is obtained as the position of the starting face, increased or decreased by 1 along u or v axis, according to the field direction of the traversed edge. Thus, our parameterization treats each face as an axis-aligned unit square, disregarding, as a further approximation, the effect of the presence of triangles.

Our algorithm stores an *overlap count* per face to indicate the number of faces with the same 2D parametric position. Each face added to the patch with an expansion is also assigned an *overlap score*, which estimates the minimal number of overlapping faces that one needs to cross to reach that face from the seed face. When a face q_j is added to the patch with an expansion from face q_i, the overlap score of q_i is assigned as the overlap score of q_i plus the overlap count of q_j minus 1. When a fusion is performed dissolving the edge between faces q_i and q_j, both faces' overlap scores are re-assigned to the minimum of their current overlap scores, increased by the occupancy grid value at the respective parametric position minus 1. Finally, the cuttability score of each edge in B is increased by the overlap score of the face that would be added to the patch, multiplied by a user-defined constant c_o. Thus, expansion favors picking faces that do not lead to deeper overlaps.

In our experiments we use $c_o \ll c_s$ (specifically, $c_s = 10^3$ and $c_o = 1$), to reflect that cut straightness is more important for our purposes and that minor overlaps can be tolerated. Fig. 5 shows an example of the individual and combined effect of these two heuristics.

Last, our method doesn't rely on any specific meshing algorithm. We include more cut examples with different input quad-dominant meshes generated by previous quad meshing techniques in the supplemental document.

5 Wearability Verification

So far, our system has identified a set of cuts which make the garment *knittable*. In this stage, our system ensures the wearability of the cut surface while removing as many cuts as possible. These removed cuts can be permanently *sewn* in the fabrication phase. The other cuts instead have to be opened and resealed every time the garment is put on or off and is implemented by lacing.

In computational design for fabrication projects, the penalties for creating a non-working object are high (100% loss of time and material, must restart process) while the penalties for fabricating a slightly-less-than-optimal but working object are low (some fraction of material or fabrication time may be saved). Thus, we design a conservative wearability test that never overestimates wearability.

Our strategy is as follows: first, our wearability test verifies that the initial set of cuts make the garment wearable. Our method then splits the set of cuts into *cut-arches*, straight sequences of cut edges. It tentatively removes one cut-arch at a time and reinserts it if the resulting design is determined, by the test below, not to be wearable. Each cut-arch will be tested once and marked for permanent sewing if it can be successfully removed in the process.

To quickly determine whether or not a given design is wearable, our system deploys a physically-based simulation of the process of wearing the garment. One problem is that the act of wearing a garment (or putting it on a passive wearer) can require a complex sequence of precise movements. For example, human dressing strategies require deep reinforcement learning with tens of hours of training [52]. Therefore, instead of searching for a strategy for *putting on* the garment, our system follows a similar strategy used in [53] to checks if the garment can be *stripped* off by a set of simple external forces, starting from a configuration where it is already worn over the subject.

Two types of external stripping forces are employed. The *local forces* repels the garment away from the wearer (Fig. 6b); our code applies a force to each garment mass in position p, in the direction of the gradient of the signed distance to the wearer at p, and with a magnitude inversely proportional to the square of that distance.

Meanwhile, a *global force*, of constant magnitude, drags all the garment masses in an arbitrary direction, which is changed randomly every fixed number of simulation steps. This force helps disentangle the garment from complex shapes, such as tubular structures (for example, Fig. 6, c–f). Fig. 13 demonstrates that our physical wearability test aligns well with the simulation.

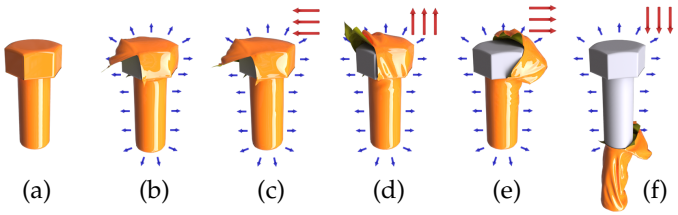


Fig. 6. **An example of our wearability test:** (a) the Bolt model wearing the cut-mesh, (b) simulation with local force (blue arrows) only, and (c)–(f) additional global force (red arrows) that changes direction periodically until garment is removed.

The wearability test is passed as soon as the garment is detected to be sufficiently far from the target object. If that does not happen within a fixed span of simulated time, the wearability test is considered to have failed.

Note that the initial cut surface does not necessarily guarantee wearability even it is a disk-like shape. For example, it is trivially impossible to insert a sphere inside a closed surface featuring a single cut. Furthermore, even a disc-shaped soft object can be crimped between corrugated surfaces in such a way that no near-isometric deformation can remove it. Hence, a wearability test needs to be performed after generating initial cuts. Fortunately, if the initial cut mesh is not wearable, we can generate a different set of initial cuts, as the procedure is stochastic.

6 Anisotropic Stitch Meshing

Our system generates a stitch mesh from the cut-mesh using the remesh-and-label approach of [26], with a simplified labeling stage (enabled by our cut-mesh generation step).

6.1 Anisotropic Remeshing

The first step is to convert the cut-mesh into a quad-dominant mesh in which there are only triangles, quads, and pentagons, and all faces have the same size and the same ratio between the course and wale edge lengths. These lengths are determined by the configuration of the knitting machine and the yarn material used for fabrication, so any mismatch between the mesh and the target lengths will result in inaccurate fabrication.

Conveniently, the course and wale directions are already determined during surface cutting, so any field-aligned anisotropic remeshing pipeline (e.g., [54], [55]) can be employed for this step. We chose to use the local anisotropic parameterization method of [56], coupled with the mesh extraction approach described in previous work [26] to produce an anisotropic quad-dominant mesh with relatively few triangles and pentagons.

6.2 Labeling

Then, our system assigns a knitting direction to each edge of the generated anisotropic mesh. Recall that our surface cutting process assigns directions to every edge of the cut-mesh during the patch-growing process (Section 4). To label the anisotropic mesh, all our system needs to do is to transfer these knitting directions from the cut-mesh to the anisotropic mesh. This transfer is done by first interpolating local knitting directions on the cut-mesh to each vertex in the anisotropic mesh; then averaging these values to compute local knitting direction at each edge of the anisotropic mesh; and, finally, comparing each edge's orientation to the local knitting directions so computed in order to determine a course or wale label.

6.3 Local Mesh Modification

In our system, a valid stitch mesh face must be either a triangle with a single wale edge or a polygon that has a degree of four or higher with only two disconnected wale edges. However, our remeshing (rarely) introduces inconsistent wale and course labels because of its local nature. Further, the resulting stitch mesh can still pose challenging cases for machine knitting, even when no invalid face exists. To address these problems, our system performs a local mesh modification step after anisotropic mesh generation.

We only allow three types of stitch mesh faces:

- A quad with two disconnected wale edges,
- A pentagon with two disconnected wale edges, indicating an increase or a decrease, and
- A triangle with a single wale edge, indicating a short-row.

We consider all other face labeling configurations as *invalid*. Such invalid configurations can appear during labeling due to multiple positional singularities on the surface with close proximity. In order to convert the invalid faces to valid faces, we follow the post-processing procedure of [26], which first cuts invalid polygons into triangles and then merges some of these new triangles with the neighboring faces to form valid ones. It is important to note that the output stitch mesh here is guaranteed to be machine-knittable (details can be found in the supplemental document).

At this point, our generated yarn topology is valid and stable, meaning that it would fabricate perfectly on an ideal knitting machine using perfect yarns. However, in reality, some mesh configurations might lead to instabilities during knitting because of the real-world machine and yarn variations. Hence, we introduce local mesh modifications to encode "rules of thumb". We arrived at these rules by knitting test patterns and by using "Knitting Assist" in the Shima Seiki KnitPaint software [28], which generates warnings for structures that don't match industry best practices. In particular, we have identified four such cases, as shown in Fig. 7. A detailed explanation of our solution to these cases can be found in the supplemental document.

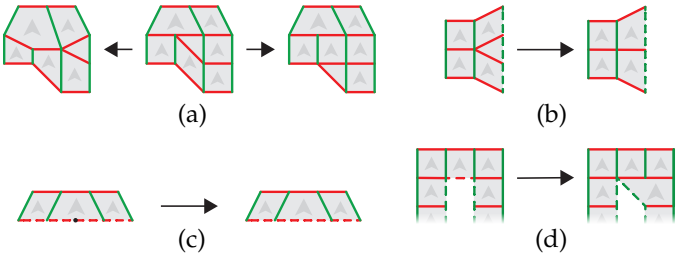


Fig. 7. Examples of the mesh topology causing unstable yarn loops. Dashed lines indicate mesh boundaries.

7 Knitting Instruction Generation

Given a stitch mesh, each stitch mesh face can be translated into low-level machine knitting instructions [5]. However, the instructions must be assigned to needle locations in order to execute properly. This is known as the *knit scheduling* problem. Unfortunately, existing schedulers only work on tubes [2], [5]; and our stitch mesh is a flat sheet.

The primary challenge faced by any scheduling algorithm is that loops stacked on the same needle at the same time can never be separated. This means that each independent part of an in-progress knit sheet needs to be held at a different needle location. This resource conflict is easy to see if the machine is visualized with a needle $\times$ time

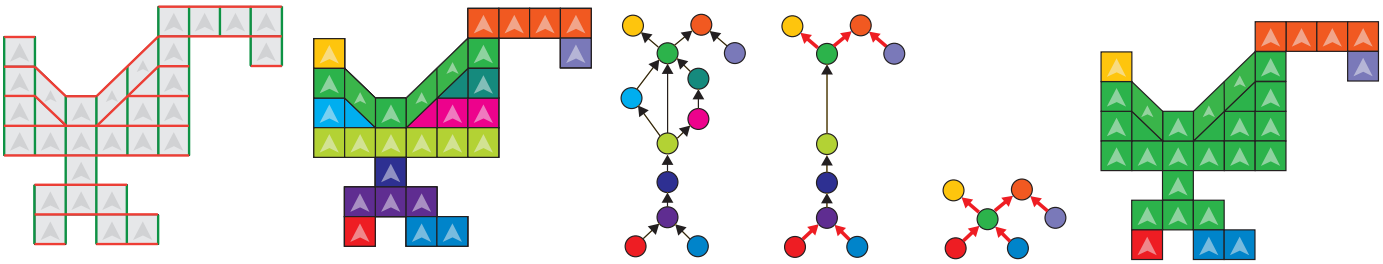


Fig. 8. **Grouping faces into patches:** (a) the stitch mesh, (b) faces grouped into rows, (c) the graph induced by the rows and their connections, (d) redundant paths from short rows merged, (e) serial chains merged (avoiding red split/merge edges), and (f) the final patch assignment.

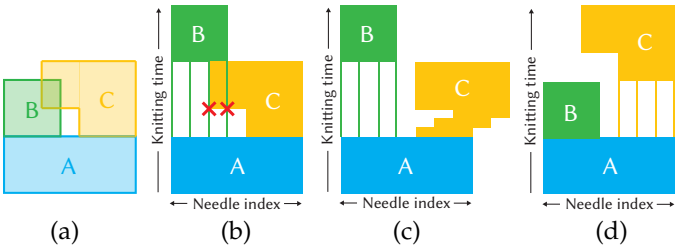


Fig. 9. **Knitting order and knittability:** (a) a garment flattened into 2D with two knitted pieces B and C connected with A, (b) the needle × time "bed view" for knitting the garment. If C is knit before B, loops held for B interfere with the needles needed by C. This can be resolved by, e.g., (c) shifting C one needle per row to make it knittable, or (d) knitting B after A.

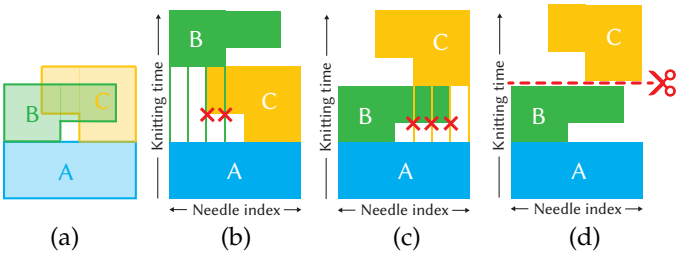


Fig. 10. **More examples of knitting order affecting the knittability:** A garment (a) in which no order (b,c) is knittable, however, adding a cut (d) makes it knittable.

"bed view" (Fig. 9), which shows which needles are holding loops at which points during the construction time.

Our scheduler's job is to assign needles and knitting orders in order to avoid these resource conflicts. Unfortunately, as with most resource scheduling problems, it is impossible to find globally optimal solutions in practice, given the exponential spaces of needle locations and knitting orders. We provide a practical method to find the locally optimal solution by using two reductions:

- grouping stitch mesh faces into *patches* (Section 7.1);
- and assigning needle locations without considering overlap (Section 7.2).

Following these reductions, our system can determine the patch knitting order based on assigned needle locations (Section 7.3), subdivide and trace the stitch mesh (Section 7.4), and produce machine instructions (Section 7.5).

7.1 Grouping Faces

Our knitting instruction generation begins by grouping faces into *patches* which can be naively scheduled without overlap. Given a stitch mesh (Fig. 8a), our system first groups all faces into *rows*, where each row contains the faces that are placed side-by-side along the course direction (Fig. 8b). These rows are viewed as a graph where each node corresponds to a row, and edges are induced by connections between rows (Fig. 8c). Graph paths are removed if removing would not lead to disconnected graphs (Fig 8d). Finally, all serial chains can be collapsed (Fig. 8e), resulting in the desired patch grouping (Fig. 8f).

7.2 Assigning Needle Locations

Before exploring the space of knitting orders, our system assigns needle locations without considering potential overlaps of different patches on the needle bed. It begins by processing each patch independently and then shifts the needle locations of patches, as needed, to connect the consecutive patches (i.e., to make sure that needle locations along the boundaries of consecutive patches match).

Needle indices are assigned to course edges instead of faces. The bottom course edge indicates where the previous loop is positioned, and one for its top course edge, specifying where the new loop will be placed after the stitch is formed.

Our system assigns needle locations from its bottom row within each patch, starting with one face on the bottom row chosen randomly. The course (both the bottom and the top) edges of the chosen face is assigned to an arbitrary needle. The course edges of the neighboring faces along the course direction (i.e., on the same row) are placed on neighboring needles. Then, to minimize the total transfer distance (i.e. the shift of the row along the needle bed), which would make the fabrication process more stable, we shift the top edges' needle locations such that the total difference between top and bottom edge locations are minimal. Note that, for the row containing short-row face, we align the needle locations based on the short-row face, so the needle locations of edges of short-row dace are aligned. The bottom edge locations of the next row are simply copied from the top edge locations of the current row. We process all rows of the patch until all edges get assigned their needle locations.

After that, our system traverses all patches and aligns the needle locations along patch boundaries by shifting the next patch's needle locations accordingly. This forms a global needle assignment for the entire stitch mesh.

7.3 Determining the Knitting Order

As mentioned above, resource conflicts can cause stitch mesh to be unknittable. It is possible to avoid this problem in some cases by simply shifting loops along the machine bed and separating the overlapped pieces (Fig. 9c). However, this can lead to over-stretched yarn and broken stitches. Instead, our system avoids resource conflicts by adjusting the order of patches that are scheduled to be knit (Fig. 9d) or by adding additional cuts (Fig. 10d).

Our solution is to test a fixed number of randomly-generated knitting orders. For each knitting order, our system performs a knitting simulation on the needle bed to identify overlaps. When an overlap is detected, the system introduces a cut between consecutive patches as shown in Fig. 10d. After testing a user-specified number of knitting orders, the system picks the one that introduces the smallest number of cuts. Details can be found in the supplemental document.

7.4 Subdivision and Tracing

After the patch knitting order and needle locations are determined, our system subdivides the stitch mesh and specifies its faces' knitting order.

The subdivision operation splits each row into two rows by converting each quad face into two faces, each triangle into a quad (i.e., a *short-row face*), and each pentagon into two or three faces, as shown in Fig. 11. This converts each short row into a pair of short rows, which reduces the number of separate yarn pieces needed for fabrication; more discussion can be found in prior works, which use a similar tactic [2], [26]. Note that all course edges in the resulting subdivided stitch mesh inherit the needle locations from the original stitch mesh.

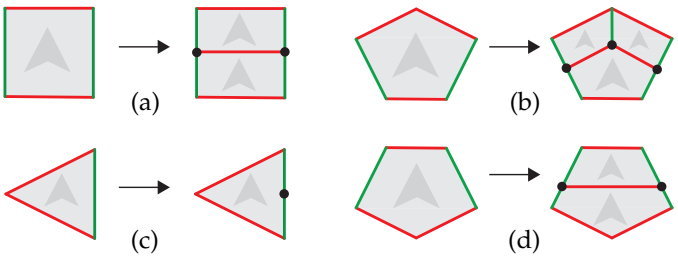


Fig. 11. **Subdivision:** (a) each quad is subdivided into two quads; (b) each increase face is subdivided into two quads and one increase; (c) each triangle is converted into a special quad (short-row face) by adding one vertex on the wale edge; and (d) each decrease face is subdivided into one quad and one increase.

Our system *traces* the resulting mesh to generate the knitting order for its faces. Our system traces along the row until reaching the wale boundary and then switches to the next row and traces in the opposite course direction. Short-rows, however, require special consideration. When our code reaches a face below a short-row face that is on the right end of a short-row, it modifies the local mesh topology as shown in Fig. 12. The goal is to alter the yarn path to perform a "tuck and turn" (Tracing Rule 4 in [2]) and trace the short-row before continuing to the next face on the same row. Since subdivision converts each row to two rows, it guarantees that the yarn path comes back after finishing the short-row.

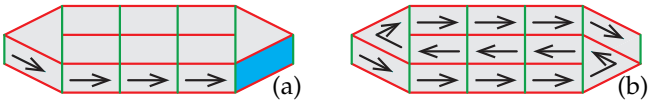


Fig. 12. When upper face is short-row end face, modify the mesh locally. (a) when tracing, reach a face (blue) right below a short-row face that is on the right end of a short-row and the pair of short-rows form a closed loop, which is not knittable. (b) shift the short-row face upward and trace the short-row before continuing to the next face. Black arrows here indicate tracing direction.

7.5 Scheduling

Our system converts each face type into low-level machine instructions using the bottom course edges' needle locations. At the end of each row, a transfer pass is used to move the loops to the new locations recorded in the top course edges. Transfers are planned using the Schoolbus algorithm [32]. Note that extra passes for cast-on and bind-off stitches are required for the top course edges and bottom course edges for the course edges on the boundary, respectively.

8 Results

We test our pipeline on a set of 3D objects with varying topology and geometry complexities. Once fabricated, seams marked for sewing are hand-sewn, the garments are placed on 3D printed target objects, and the seams marked for lacing are hand-laced with additional yarn in a contrasting color. Lacing our examples takes 10 mins – 1 hr. The computational steps are run using a single thread on a desktop machine with an Intel(R) Core(TM) i7-7700 CPU @ 3.60Hz and 64 GB of memory. Knitting instructions were executed by a Shima Seiki SWG091N2 industrial knitting machine with 15 needles per inch. All examples were knit from a 2-ply acrylic yarn (Tamm Petit). Machine knitting instructions for all examples are included in the supplementary material.

8.1 Knitted Results

In order to avoid the curling that can happen in all-knits fabric, our system uses the "Seed" pattern, a checkerboard of "knit" and "purl" stitches that is balanced (i.e., lies flat). Anisotropic meshing edge lengths were determined by measuring a 40×40 stitch seed pattern knit on our target machine – $4.2\text{cm} \times 11.5cm$ with our knitting machine settings and selected yarn.

We first test our system with a simple but interesting model, Bolt. The input surface is first cut into a topological disc. Through the wearability test, most of the cut-arches are sewn without compromising wearability. As shown in Fig 13, only a small hole is left in order to put the garment on. Fig. 6 reports the physical simulation used to verify wearability, and Fig. 13 depicts the process in reality.

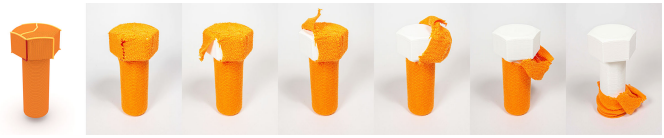


Fig. 13. **The Bolt model**: rendered result and photographs of the real garment being removed following the wearability simulation (Fig. 6).

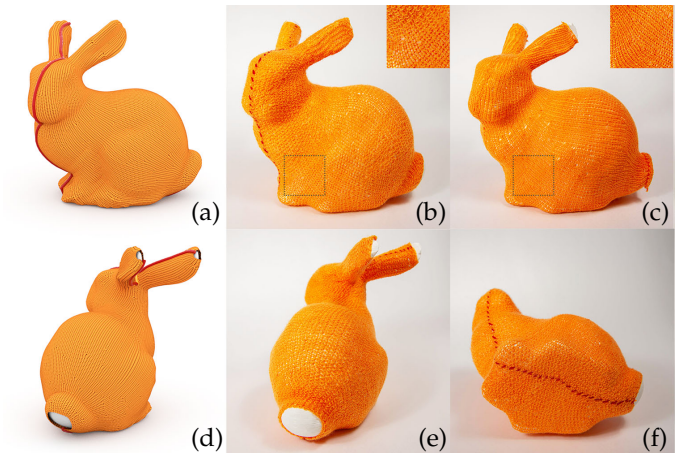


Fig. 14. 3D printed rigid Bunny wearing our knitted results: (a) and (d) rendered bunny from different views, (b), (e), and (f), front, back, and bottom views of knitted bunny with "Seed" pattern, and (c) knitted bunny with "Plain" pattern.

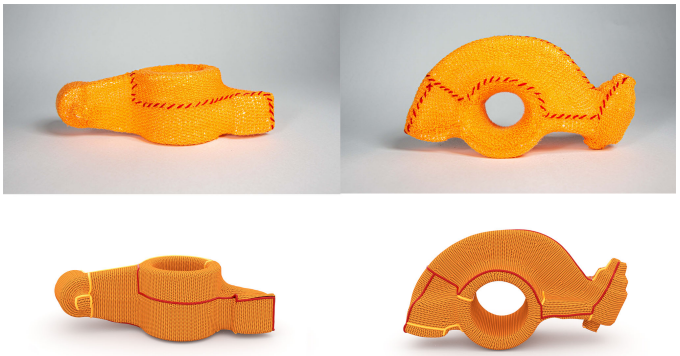


Fig. 15. Two different views for the Rockerarm model wearing our knitted garment.

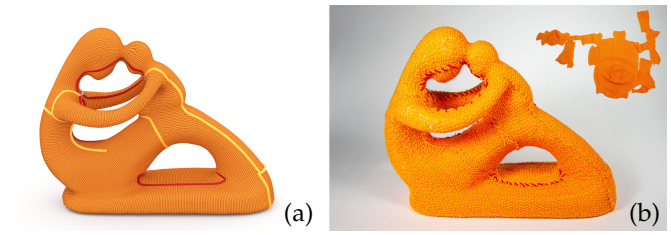


Fig. 16. **The Fertility model:** (a) the stitch mesh with yarn texture and (b) the real knitted garment on the 3D printed model and the same knitted but flatted garment as shown on top right.

Fig. 17. **The Centaur model:** left, wearing a knitted sweater, and right, wearing a knitted skirt.

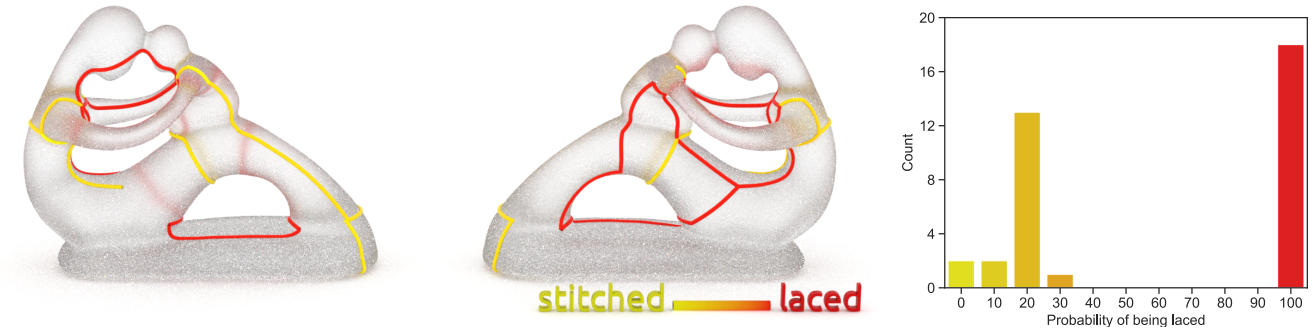
We create a wearable garment for the Stanford Bunny, which is printed as a rigid body. In this case, our system is given a partial covering of the original shape, with three holes designed around ears and tail. As shown in Fig. 14, the knitted garment fits the input shape as intended. Fig. 15 shows another physically realized example, the RockerArm model. Observe how the garment can reproduce the shape of the physical object, including in the strongly concave regions.

The fertility model is a challenging shape for both cutting and scheduling. Due to the UV overlap minimization of the surface-cutting algorithm, our system could knit the whole garment without any additional cuts introduced from scheduling. The shape presents high curvature regions reflected in either irregular vertices or triangles in the quad-dominant mesh. In the physical produced garment, the curvature is recreated in part by the re-sewing of the cuts during assembling, and, importantly, in the interior of the knitwork, by virtue of the computed knitting pattern, as shown in Fig. 16.

The garment mesh doesn't need to follow the target mesh closely. For example, we used our system to produce both a tight-fitting "sweater" and a loose "skirt" for the Centaur model (Fig. 17). In Fig.19 we show several other knitted designs obtained with our framework, including a pair of "pants" for a Cat model, a covering for an Aquadom model, a "sweater" for a Dragon model, and a covering for the Sculpt model. In order to resolve two-layer overlaps, during machine instructions generation, Aquadom covering, Dragon sweater, and Sculpt covering have to be separated into 2, 5, and 6 pieces, respectively.

Our framework can directly incorporate the knitting texture design system proposed by [5], as exemplified in Fig. 14.

Comparison with [2] One clear difference is that our pipeline outputs a garment wearable by a rigid wearer, while the knits in these previous papers [2], [5] are shown on soft foam, because – despite the stretchiness of knit fabric – their tubular knit covering cannot be fit on a solid model. Further, knitting a flat sheet saves knitting time owing to more efficient transfer plans. For example, on the same knitting machine with the same speed settings, the knitted "plain" bunny in Fig. 1 of [2] takes 43 mins using data provided by those authors, while our result for the same-sized bunny model (Fig. 14c) only needs 16 mins. Even our "seed"-pattern bunny (Fig. 14b), which requires transferring stitches front and back each row, still takes only 33 mins to finish. Of course, some of this saved knitting time is lost to sewing as a post-process.



(a) (b)

Fig. 18. (a) the Fertility model with the cuts and different colors indicating the probability of being laced, based on 300 full wearability tests on the same set of cuts with different cuts testing order and (b) histogram of all cuts respecting to the probability of being laced.

Fig. 19. **Extra results generated using our method:** From left to right, columns are front, back, and knitted pieces. From top to bottom, Cat pant, Aquadom, Dragon sweater, and Sculpt. Note that knitted piece photographs have been hue-adjusted with software to match renders.

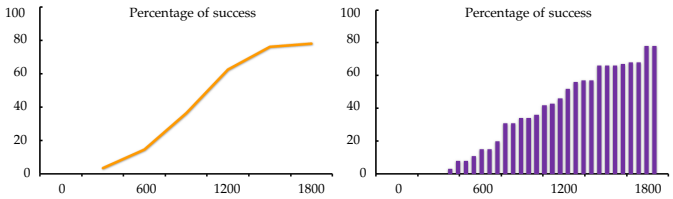


Fig. 20. Left, probability of wearability test success (taking the garment off the target object) for the same set of laced and sewn cuts with different force directions using a different number of iterations; right, the Cumulative Distribution Function (CDF) of the probability of wearability test success the same set of laced and sewn cuts with different force directions with 1800 as the maximum number of iterations.

8.2 Physically-based Simulation for Wearability Test

We modeled the garment as a mass-spring system and the wearability test uses NVIDIA Flex – a GPU implementation of Position Based Dynamics (PBD) [57]. In particular, we use three types of springs: along each edge for stretch resistance, along the diagonals of quads for shear resistance, and connecting every other vertex along the wale and course directions to provide bending resistance. The stiffness parameters for these three types of springs are manually set, based on our experiments with small real-world samples. Since knit cloth is stretchable, our simulations do not include any inextensibility constraint. The global force is aligned to an axis-aligned direction and its direction is switched after every 300 steps to another axis-aligned direction. Experiments (Fig. 20) suggest that similar results are obtained for wide ranges of used constants. Observe this test only needs to be conservative: failure to detect a potential way to strip off a garment only results in under-sewing, not over-sewing, in the final object, meaning it will still be wearable. For all the tested models, the combined computational time for the surface cutting, stitch meshing, and knitting code generation stages are around a few minutes. The wearability test process takes from a few minutes to less than 20 minutes, depending on the number of tested cut-arches. The slowest stage is the knitting fabrication process itself, which takes from 0.5 to 1.5 hours.

Our system tests cut-arches in a random order, although user preferences could be easily accounted for. Surprisingly, experiments (Fig. 18) suggest the choice of the order has only a limited impact on the probability of a given arch to wind up as laced or sewn.

Note that the PBD simulator we use can be easily replaced with another simulation system. We picked a GPU-based PBD implementation because of its high performance. On the other hand, our simulator does not accurately model the anisotropic behavior of knit cloth. A higher fidelity simulation can be achieved using a more comprehensive data-driven approach with an anisotropic constitutive material model [58] or a yarn-level simulation system [59], but they would have higher computation cost.

9 Discussion

As we explain in this paper, automated fabrication of wearable knit artifacts poses a complex and challenging problem space. Therefore, the framework we present in this paper involves multiple components, some of which provide effective but sub-optimal solutions. Nonetheless, each component is carefully designed for the target problem and preferable to similar approaches in prior work in this context.

For example, prior work includes sophisticated undressing strategies that ensure wearability [19], [53], [60]. However, [19] relies on convex hulls, unsuitable when the wearer has complex geometry/topology. The gradient-based method introduced by [60] highly depends on the initial guess by nature, making it unsuitable for our problem. Also, with complex shapes, locking (when the flux field force is balanced by the stretching force) and tangling issues are common. Finally, the strategy of [53] starts from multiple pieces and requires extra cuts introduced in the wearability test, which is undesirable in our case. In contrast, our method offers a simple and effective solution, though it is sub-optimal, providing a baseline with potential for future innovation.

Our work also fills in an important missing part of the automatic knitting design space: topological-disc-shaped knitting. Our sheet-based scheduler makes complementary algorithmic choices to those made by [2]. Our scheduler works with only sheets, is stochastic, and always quickly finds a valid solution (by adding cuts if needed); thus, it can be used as an "any-time" algorithm, running longer for more optimal results or returning quickly with feasible results. In contrast, the scheduler of [2] works with only tubes, is deterministic, and fails (after quite a few minutes) if the pattern cannot be embedded.

Though our method is able to generate garments for an arbitrary 3D shape, it is still restricted by physical constraints of the knitting machine, e.g., the curvature of the shape, stitch size, yarn materials, and garment resolution, which poses challenges when knitting models with high curvature features, such as the fingers in Armadillo or claws in Dragon. For example, in Fig. 21a, there are 157 singularities and causes 166 cut-arches. Stitch-meshing this detail requires a mesh resolution beyond what is feasible to knit. It would be interesting to investigate the maximum surface curvature that can be represented with a given stitch mesh face size.

As mentioned in Section 5, designing an effective dragging strategy to take the garment off from the target object is a challenging problem. A global directional force can cause the garment to intertwine with the target object if the model is complicated. For instance, the shape of the Elk model can easily form locks that prevent the garment from being torn off, as shown in Fig. 21b. Designing an efficient way to take off the garment from a complicated shape would be a very interesting future direction.

Fig. 21c demonstrates that a complicated shape such as the Botijo model would cause tens of separated knitted pieces, which would be challenging to sew together manually. It would be interesting to automate the sewing process in the future. Zippers or buttons would be a good alternative to lacing, though it would require moving to a larger scale of knitting and adding selvage to edges (e.g., during the local mesh modification phase).

The layout of the cuts in the knitted garment is largely determined by input mesh. While we use quad meshes from [61], [62], [63] for our surface cutting since they have few singularities, tailoring a meshing technique specifically for knitting purposes is an interesting direction.

The framework we describe in this paper is limited to fabricating coverings for *rigid* objects. Targeting *non-rigid* objects, like plush toys, would involve accurately measuring and simulating their behavior of real-world objects, which can be challenging. Articulated rigid bodies do not include this additional challenge; however, as demonstrated by Clegg et al. [52], with a sequence of body and limb movements, wearability can be preserved using fewer cuts than rigid objects. Therefore, extending our framework to articulated models and humanoids/animals requires a more sophisticated motion planning and unclothing strategy.

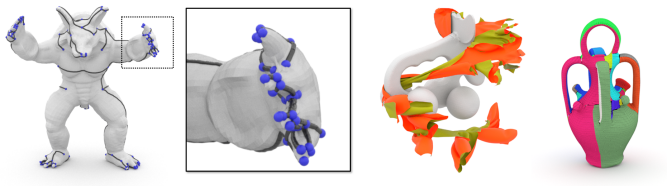


Fig. 21. **Failed cases:** (a) the Armadillo model with hundred of singularities (blue), which leads tens of cuts (black), (b) the geometry of the Elk model preventing the garment being tore off, and (c) the Botijo model with 26 separated pieces shown with different colors.

Finally, our paper focuses on automatic and generic covering generation rather than the design of human garments, meaning that it does not implement aesthetics and affordances from garment design. That being said, our system can accommodate several degrees of user control (for example, painting areas where cuts are undesired or manually selecting cuts for earlier testing, making them more likely to end up sewn rather than laced). Building a semi-automatic interface to assist expert clothing designers would make for good future work.

10 Conclusion

In this paper, we proposed an automatic method that produces knits that are ensured to be both wearable and machine knittable. The effectiveness of our pipeline has been verified on a set of 3D models that are typically challenging for machine knitting.

Acknowledgments

We thank Professor Wojciech Matusik for offering all the resources needed and giving his full support for Kui Wu in this paper. We also thank Zhiyuan Zhang and Ella Moore for 3D printing and Alexandre Kaspar for the insightful discussions. This work was supported in part by NSF-IIS-1910486. Kui Wu was supported in part by University of Utah Graduate Research Fellowship

References

[1] J. McCann, L. Albaugh, V. Narayanan, A. Grow, W. Matusik, J. Mankoff, and J. Hodgins, "A compiler for 3d machine knitting," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 49:1–49:11, Jul. 2016.

[2] V. Narayanan, L. Albaugh, J. Hodgins, S. Coros, and J. McCann, "Automatic machine knitting of 3d meshes," *ACM Trans. Graph.*, vol. 37, no. 3, pp. 35:1–35:15, Aug. 2018.

[3] M. Popescu, M. Rippmann, T. Van Mele, and P. Block, "Automated generation of knit patterns for non-developable surfaces," in *Humanizing Digital Reality*, D. R. K. et al., Ed. Singapore: Springer, 2018.

[4] A. Kaspar, L. Makatura, and W. Matusik, "Knitting skeletons: A computer-aided design tool for shaping and patterning of knitted garments," in *Proceedings of the 32Nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '19. New York, NY, USA: ACM, 2019, pp. 53–65.

[5] V. Narayanan, K. Wu, C. Yuksel, and J. McCann, "Visual knitting machine programming," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 63:1–63:13, Jul. 2019.

[6] C. Yuksel, J. M. Kaldor, D. L. James, and S. Marschner, "Stitch meshes for modeling knitted clothing with yarn-level detail," *ACM Trans. Graph. (Proceedings of SIGGRAPH 2012)*, vol. 31, no. 3, pp. 37:1–37:12, 2012.

[7] Y. Liu, H. Pottmann, J. Wallner, Y.-L. Yang, and W. Wang, "Geometric modeling with conical meshes and developable surfaces," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 681–689, Jul. 2006.

[8] H. Pottmann and Y. Liu, "Discrete surfaces in isotropic geometry," in *Proceedings of the 12th IMA International Conference on Mathematics of Surfaces XII*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 341–363.

[9] S. Bouaziz, M. Deuss, Y. Schwartzburg, T. Weise, and M. Pauly, "Shape-up: Shaping discrete geometry with projections," *Comput. Graph. Forum*, vol. 31, no. 5, pp. 1657–1667, Aug. 2012.

[10] R. Poranne, E. Ovreiu, and C. Gotsman, "Interactive planarization and optimization of 3d meshes," *Computer Graphics Forum*, vol. 32, no. 1, pp. 152–163, 2013.

[11] Y. Mori and T. Igarashi, "Plushie: An interactive design system for plush toys," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007.

[12] M. Skouras, B. Thomaszewski, P. Kaufmann, A. Garg, B. Bickel, E. Grinspun, and M. Gross, "Designing inflatable structures," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 63:1–63:10, Jul. 2014.

[13] N. Umetani, D. M. Kaufman, T. Igarashi, and E. Grinspun, "Sensitive couture for interactive garment modeling and editing," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 90:1–90:12, Jul. 2011.

[14] A. Bartle, A. Sheffer, V. G. Kim, D. M. Kaufman, N. Vining, and F. Berthouzoz, "Physics-driven pattern adjustment for direct 3d garment editing," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 50:1–50:11, Jul. 2016.

[15] A. Garg, A. O. Sageman-Furnas, B. Deng, Y. Yue, E. Grinspun, M. Pauly, and M. Wardetzky, "Wire mesh design," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 66:1–66:12, Jul. 2014.

[16] J. Pérez, M. A. Otaduy, and B. Thomaszewski, "Computational design and automated fabrication of kirchhoff-plateau surfaces," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 62:1–62:12, Jul. 2017.

[17] C. Schüller, R. Poranne, and O. Sorkine-Hornung, "Shape representation by zippables," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 78:1–78:13, Jul. 2018.

[18] X. Zhang, G. Fang, M. Skouras, G. Gieseler, C. C. L. Wang, and E. Whiting, "Computational design of fabric formwork," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 109:1–109:13, Jul. 2019.

[19] Y. Igarashi, T. Igarashi, and H. Suzuki, "Interactive cover design considering physical constraints," *Computer Graphics Forum*, vol. 28, no. 7, pp. 1965–1973, 2009.

[20] A. Mahdavi-Amiri, P. Whittingham, and F. Samavati, "Cover-it: An interactive system for covering 3d prints," in *Proceedings of the 41st Graphics Interface Conference*, ser. GI '15. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2015, pp. 73–80.

[21] M. Hofmann, L. Albaugh, T. Sethapakadi, J. Hodgins, S. E. Hudson, J. McCann, and J. Mankoff, "Knitpicking textures: Programming and modifying complex knitted textures for machine and hand knitting," in *Proceedings of the 32Nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '19. New York, NY, USA: ACM, 2019, pp. 5–16.

[22] J. Underwood, "The design of 3d shape knitted preforms," Ph.D. dissertation, Fashion and Textiles, RMIT University, 2009.

[23] S.-M. Belcastro, "Every topological surface can be knit: A proof," *Journal of Mathematics and the Arts*, vol. 3, no. 2, pp. 67–83, 2009.

[24] Y. Igarashi, T. Igarashi, and H. Suzuki, "Knitting a 3d model," *Computer Graphics Forum*, vol. 27, no. 7, pp. 1737–1743, 2008.

[25] ——, "Knitty: 3d modeling of knitted animals with a production assistant interface." in *Eurographics*. Aire-la-Ville, Switzerland, Switzerland: The Eurographics Association, 2008, pp. 17–20.

[26] K. Wu, X. Gao, Z. Ferguson, D. Panozzo, and C. Yuksel, "Stitch meshing," *ACM Trans. Graph. (Proceedings of SIGGRAPH 2018)*, vol. 37, no. 4, pp. 130:1–130:14, jul 2018.

[27] K. Wu, H. Swan, and C. Yuksel, "Knittable stitch meshes," *ACM Trans. Graph.*, vol. 38, no. 1, pp. 10:1–10:13, Jan. 2019.

[28] Shima Seiki, "Sds-one apex3," [Online]. Available from: http://www.shimaseiki.com/product/design/sdsone_apex/flat/, 2011.

[29] Stoll, "M1plus pattern software," [Online]. Available from: http://www.stoll.com/stoll_software_solutions_en_4/pattern_software_m1plus/3_1, 2011.

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

[31] M. Meißner and B. Eberhardt, "The art of knitted fabrics, realistic & physically based modelling of knitted patterns," *Computer Graphics Forum*, vol. 17, no. 3, pp. 355–362, 1998.

[32] J. Lin, V. Narayanan, and J. McCann, "Efficient transfer planning for flat knitting," in *Proceedings of the 2Nd ACM Symposium on Computational Fabrication*, ser. SCF '18. New York, NY, USA: ACM, 2018, pp. 1:1–1:7.

[33] L. Albaugh, S. Hudson, and L. Yao, "Digital fabrication of soft actuated objects by machine knitting," in *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '19. New York, NY, USA: ACM, 2019, pp. VS01:1–VS01:1.

[34] J. McCann, "The "knitout" (.k) file format," [Online]. Available from: https://textiles-lab.github.io/knitout/knitout.html, 2017.

[35] W. Jakob, M. Tarini, D. Panozzo, and O. Sorkine-Hornung, "Instant field-aligned meshes." *ACM Trans. Graph.*, vol. 34, no. 6, pp. 189–1, 2015.

[36] M. S. Floater, "Mean value coordinates," *Computer Aided Geometric Design*, vol. 20, no. 1, pp. 19 – 27, 2003.

[37] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, "Least squares conformal maps for automatic texture atlas generation," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 362–371, Jul. 2002.

[38] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. J. Gortler, "A local/global approach to mesh parameterization," in *Proceedings of the Symposium on Geometry Processing*, ser. SGP '08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 1495–1504.

[39] X. Gu, S. J. Gortler, and H. Hoppe, "Geometry images," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 355–361.

[40] J. Smith and S. Schaefer, "Bijective parameterization with free boundaries," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 70:1–70:9, Jul. 2015.

[41] M. Rabinovich, R. Poranne, D. Panozzo, and O. Sorkine-Hornung, "Scalable locally injective mappings," *ACM Trans. Graph.*, vol. 36, no. 2, Apr. 2017.

[42] W. T. Tutte, "How to draw a graph," *Proceedings of the London*, vol. 13, p. 743–767, 1963.

[43] H. Shen, Z. Jiang, D. Zorin, and D. Panozzo, "Progressive embedding," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 32:1–32:13, Jul. 2019.

[44] R. Poranne, M. Tarini, S. Huber, D. Panozzo, and O. Sorkine-Hornung, "Autocuts: Simultaneous distortion and cut optimization for uv mapping," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 215:1–215:11, Nov. 2017.

[45] N. Sharp and K. Crane, "Variational surface cutting," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 156:1–156:13, Jul. 2018.

[46] M. Li, D. M. Kaufman, V. G. Kim, J. Solomon, and A. Sheffer, "Optcuts: Joint optimization of surface cuts and parameterization," *ACM Trans. Graph.*, vol. 37, no. 6, Dec. 2018.

[47] M. Tarini, E. Puppo, D. Panozzo, N. Pietroni, and P. Cignoni, "Simple quad domains for field aligned mesh parametrization," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 142:1–142:12, Dec. 2011.

[48] M. Campen and L. Kobbelt, "Quad layout embedding via aligned parameterization," *Comput. Graph. Forum*, vol. 33, no. 8, pp. 69–81, Dec. 2014.

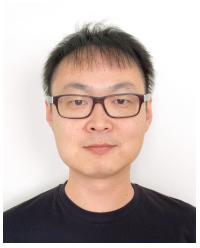
[49] A. Hertzmann and D. Zorin, "Illustrating smooth surfaces," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 517–526.

[50] N. Schertler, D. Panozzo, S. Gumhold, and M. Tarini, "Generalized motorcycle graphs for imperfect quad-dominant meshes," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 155:1–155:16, Jul. 2018.

[51] D. Eppstein, M. T. Goodrich, E. Kim, and R. Tamstorf, "Motorcycle graphs: Canonical quad mesh partitioning," in *Proceedings of the Symposium on Geometry Processing*, ser. SGP '08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 1477–1486.

[52] A. Clegg, W. Yu, J. Tan, C. K. Liu, and G. Turk, "Learning to dress: Synthesizing human dressing motion via deep reinforce-

ment learning," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 179:1–179:10, Dec. 2018.
[53] L. Malomo, N. Pietroni, B. Bickel, and P. Cignoni, "Flexmolds: Automatic design of flexible shells for molding," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 223:1–223:12, Nov. 2016.
[54] N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez, "Periodic global parameterization," *ACM Trans. Graph.*, vol. 25, no. 4, pp. 1460–1485, Oct. 2006.
[55] D. Bommes, H. Zimmer, and L. Kobbelt, "Mixed-integer quadrangulation," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 77:1–77:10, Jul. 2009.
[56] J. Wu, W. Wang, and X. Gao, "Design and optimization of conforming lattice structures," 2019.
[57] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *J. Vis. Comun. Image Represent.*, vol. 18, no. 2, pp. 109–118, Apr. 2007.
[58] H. Wang, J. F. O'Brien, and R. Ramamoorthi, "Data-driven elastic models for cloth: Modeling and measurement," *ACM Trans. Graph.*, vol. 30, no. 4, Jul. 2011.
[59] J. M. Kaldor, D. L. James, and S. Marschner, "Simulating knitted cloth at the yarn level," *ACM Trans. Graph. (SIGGRAPH'08)*, vol. 27, no. 3, p. 65, 2008.
[60] H. Wang, K. A. Sidorov, P. Sandilands, and T. Komura, "Harmonic parameterization by electrostatics," *ACM Trans. Graph.*, vol. 32, no. 5, Oct. 2013.
[61] X. Gao, T. Martin, S. Deng, E. Cohen, Z. Deng, and G. Chen, "Structured volume decomposition via generalized sweeping," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 7, pp. 1899–1911, 2015.
[62] F. Usai, M. Livesu, E. Puppo, M. Tarini, and R. Scateni, "Extraction of the quad layout of a triangle mesh guided by its curve skeleton," *ACM Trans. Graph.*, vol. 35, no. 1, pp. 6:1–6:13, Dec. 2015.
[63] N. Pietroni, E. Puppo, G. Marcias, R. Roberto, and P. Cignoni, "Tracing field-coherent quad layouts," *Comput. Graph. Forum*, vol. 35, no. 7, pp. 485–496, Oct. 2016.



Kui Wu is an postdoctoral associate in the computational design and fabrication group under the guidance of Prof. Wojciech Matusik at MIT CSAIL. He received his PhD degree in Computer Science from the University of Utah in 2019. His research spans most major fields in computer graphics, including geometry processing, physically-based rendering, simulation, and fabrication.

Marco Tarini Marco Tarini (PhD, Università degli Studi di Pisa, 2003) is an Assistant Professor at the Università degli Studi di Milano "La Statale". Marie Curie fellow (2001), he received several awards, including the Young Researcher Award by EUROGRAPHICS (2006). His research activity span several fields within Computer Graphics, Geometry Processing, and applications, such as surface remeshing and parametrization, texture mapping, real time rendering, 3D shape capture, computer assisted fabrication, computer animation, scientific visualization, and video games technologies.

Cem Yuksel is an associate professor in the School of Computing at the University of Utah. Previously, he was a postdoctoral fellow at Cornell University, after receiving his PhD in Computer Science from Texas A&M University in 2010. His research interests are in computer graphics and related fields, including physically-based simulations, realistic image synthesis, rendering techniques, global illumination, sampling, GPU algorithms, graphics hardware, modeling complex geometries, knitted structures, and hair modeling, animation, and rendering.

James McCann is an assistant professor in the Carnegie Mellon University Robotics Institute. Prior to this he worked as a researcher at Disney Research and Adobe. He received his PhD in Computer Science from Carnegie Mellon University in 2010. James leads the Carnegie Mellon Textiles Lab, which advances the start of the art in textiles design and fabrication. More broadly, he is interested in tools that build user intuition and help people create new things.

Xifeng Gao is an assistant professor of the Computer Science Department at Florida State University. Dr. Gao was a PostDoc for two years at the Courant Institute of Mathematical Sciences of New York University. He received his Ph.D. degree in 2016 and won the best Ph.D. dissertation award from the Department of Computer Science at the University of Houston. Dr. Gao has wide research interests that are related to geometry computing, such as Computer Graphics, CAD/CAE, Multimedia Processing, Robotics, and Digital Fabrication.