Knit Sketching: from Cut & Sew Patterns to Machine-Knit Garments

ALEXANDRE KASPAR, KUI WU, YIYUE LUO, and LIANE MAKATURA, MIT CSAIL, USA
WOJCIECH MATUSIK, MIT CSAIL, USA

Fig. 1. *Different garments knitted with our workflow. The initial garment specifications come from the popular fashion magazine BurdaStyle and were reinterpreted using our system to be knitted with a whole-garment knitting machine.*

We present a novel workflow to design and program knitted garments for industrial whole-garment knitting machines. Inspired by traditional garment making based on cutting and sewing, we propose a sketch representation with additional annotations necessary to model the knitting process. Our system bypasses complex editing operations in 3D space, which allows us to achieve interactive editing of both the garment shape and its underlying time process. We provide control of the local knitting direction, the location of important course interfaces, as well as the placement of stitch irregularities that form seams in the final garment. After solving for the constrained knitting time process, the garment sketches are automatically segmented into a minimal set of simple regions that can be knitted using simple knitting procedures. Finally, our system optimizes a stitch graph hierarchically while providing control over the tradeoff between accuracy and simplicity. We showcase different garments created with our web interface.

CCS Concepts: • **Applied computing** → *Computer-aided manufacturing*.

Additional Key Words and Phrases: garment modeling, computed-aided design, computational knitting

Authors' addresses: Alexandre Kaspar, akaspar@mit.edu; Kui Wu, kuiwu@mit.edu; Yiyue Luo, yiyueluo@mit.edu; Liane Makatura, makatura@mit.edu; Wojciech Matusik, wojciech@csail.mit.edu, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, Massachusetts, 02139, USA..

ACM Reference Format:
Alexandre Kaspar, Kui Wu, Yiyue Luo, Liane Makatura, and Wojciech Matusik. 2021. Knit Sketching: from Cut & Sew Patterns to Machine-Knit Garments. *ACM Trans. Graph.* 40, 4, Article 63 (August 2021), 15 pages. https://doi.org/10.1145/3450626.3459752

1 INTRODUCTION

Textiles are among the most ubiquitous elements of everyday life, as they constitute our clothing, home decor, personal accessories like hats and bags, and even deployable structures such as umbrellas and tents. As a result, textile design is a long-standing, multi-billion dollar industry that operates at incredible economies of scale.

Among the common textile manufacturing methods (e.g., weaving, sewing), weft knitting offers several advantages. It is an additive manufacturing method that constructs garments by interlocking yarn loops using computer-controlled knitting machines. Whole-garment knitting machines are able to turn yarn into complex 3D structures in a wide range of shapes, colors, and textures [Spencer 2001; Underwood 2009]. This reduces fabric waste and manual post-processing, which are common pitfalls of traditional workflows for woven and knitted garments. Furthermore, the continuously inter-locking "loop-through-loop" structure makes knitted fabrics especially deformable and stretchable. Recent advances in manufacturing, functional fibers, and computational design have opened many new opportunities for knitted textiles, including medical sensing, communication, soft robotics, flexible user interfaces, mass customization of garments, and more [Albaugh et al. 2019; Han and Ahn 2017; Luo et al. 2021; Ou et al. 2019; Vallett et al. 2016; Wicaksono et al. 2020]. However, most of these emerging applications are

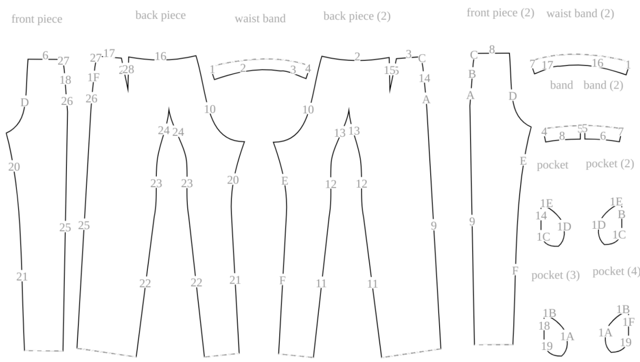


Fig. 2. *The segmentation of a sewing pattern for the rightmost pair of trousers of the teaser figure.*

either experimental or inherently small-scale (i.e., personalized) in nature, so they are severely hindered by the time and expertise required for state-of-the-art garment design processes. To enable these endeavors, it is critical to provide efficient, flexible, and intuitive processes for knitted garment design and manufacturing.

Unfortunately, intuitive design software for whole-garment knitting is still quite limited. Knitting machine manufacturers provide commercial tools for developing and constructing patterns, but only a few garment styles are directly accessible in the form of predefined templates [Shima Seiki 2011; Stoll 2011]. More complex shapes and non-standard patterns must be hand-designed at the stitch level, which requires considerable patience and expertise, as shown in the work of Underwood [2009].

By contrast, cut & sew garment designs are widely accessible and customizable by designers of varying skill levels. In this common design pipeline, several flat panels are cut from 2D fabric and then sewn together along shared seams (see Fig. 2). The 3D structure of the resulting garment (e.g., curvature, topology) can be arbitrarily complex, but it is fully prescribed by the 2D panel boundaries and their connectivity. That is, the panels capture the garment's 3D structure *intrinsically*. It is appealing to work in this lower-dimensional panel space because the intermediate (and resulting) blueprints can be easily edited, and they convey the designer's intention in a simple, compact, and precise manner. There is also a rich collection of sewing patterns available online for various clothing styles (e.g., BurdaStyle, Deer&Do), and customization is straightforward with existing industrial design software, which offers short cycles between design and fabrication [Clo3D 2020; MarvelousDesigner 2020]. However, there is no clear way to design knits directly via a cut & sew pipeline, because whole-garment knitting is a time-dependent fabrication process that requires extra information during the design stage.

In this work, we combine the strengths of whole-garment knitting and the cut & sew design pipeline. For the garment design phase, we develop a user interface based on the powerful, low-dimensional representation from cut & sew. Then, for efficient garment construction, we automatically translate the 2D panels into a full garment that is machine-knittable. Since our approach constructs the garment and its constituent fabric simultaneously, we can offer additional control over the *interior* of each panel, rather than being limited to the boundary.

This workflow presents several technical challenges. Since knit fabric relies on sequentially interlocking loops (discussed in Sec. 2.2), the standard cut & sew panel representation must be augmented with time and direction information. Moreover, some cut & sew patterns do not immediately yield machine-knittable garments: in some cases, the knitting sequence produces undesired artifacts; in other cases, valid knitting sequences may not exist at all. To allow users to iteratively refine such designs, our system must offer rapid inference of the garment's final 3D structure and computation of the high-level knitting sequence. However, existing approaches typically require a full 3D mesh embedding, which can be time-consuming to create and operate on.

To address these challenges, our computational pipeline operates exclusively in the 2D domain shown in Fig. 2, which corresponds to the standard flattened view available in professional garment editing software. In particular, our computational workflow only relies on intrinsic surface metrics and local connectivity, thus bypassing a global 3D embedding (e.g., a 3D mesh) of the desired garment. Although a 3D preview would still be helpful for designers, our work shows that all required knitting information can be inferred and efficiently computed from the intrinsic 2D representation.

Our high-level knitting sequence uses a representation similar to Narayanan et al. [2018], with a time function over the sketch manifold that details the relative fabrication order among different areas of the garment. We develop new ways to solve for the time information and generate low-level stitch placement and knitting programs without the use of a 3D mesh. Lastly, to support a wide range of garment structures, our scheduler provides basic support for mixed planar and tubular structures, which is a challenging problem not addressed by previous works.

Our main contributions are:

- a novel workflow that interprets traditional cut & sew garment patterns into weft knitting machine programs,
- a system that enables interactive editing of both a garment shape and its knitting time function, and
- a stitch sampling algorithm that provides the user with both global and local control over the stitch topology.

Our system allows designers to leverage their existing cut & sew textile knowledge, intuition, and patterns to produce a fundamentally different whole-garment construction. Moreover, by automatically generating machine-knittable instructions, our method reduces the time and manual effort required for physical production. Our system is accessible as an open-source web interface, available at http://knitsketching.csail.mit.edu.

2 BACKGROUND AND RELATED WORK

Before discussing the details of our method, we briefly review the most related prior work on garment design and knitting.

2.1 Garment Design

Interactive physically-based garment design is a challenging problem of long-standing interest [Volino et al. 2005]. Sketch-based design pipelines are particularly prominent [Decaudin et al. 2006;

Igarashi and Hughes 2003; Turquin et al. 2007; Wang et al. 2018], because the familiar 2D-to-3D approach allows designers to use their existing experience and intuition. Other works allow designers to edit the garment directly in 3D space, by sketching the desired fold pattern of the draped fabric [Li et al. 2018] or directly modifying garment shape [Bartle et al. 2016]. After making the desired edits in 3D, the corresponding 2D patterns are generated via a simulation framework that incorporates design constraints. Utilizing design sensitivity analysis, Umetani et al. [2011] presented an interactive tool for garment design that allows interactive bidirectional editing between 2D patterns and 3D draped garment shape. Several methods adjust parametric shape patterns to customize an existing pattern for specific individuals, such as profile template encoding/decoding [Wang et al. 2005], gradient descent method w.r.t. parametric patterns [Montes et al. 2020; Wang 2018], and learning-based methods [Guan et al. 2012; Wang et al. 2018]. Berthouzoz et al. [2013] combine machine learning with integer programming techniques for automatically parsing BurdaStyle [1] sewing patterns and converting them into 3D garment models, whereas Shen et al. [2020] combine sewing patterns and 3D body mesh data using a Generative Adversarial Network. Finally, Huang et al. [2016] generate garment models directly from a pair of front and back images. By contrast, our work transforms sewing patterns into instructions for garment production on weft knitting machines. Our computational workflow bypasses 3D mesh generation completely, as intrinsic metrics are sufficient to describe the knitted garment topology for manufacturing.

2.2 Knitting

Knitting machines work yarn into a grid of *stitches* that form a stable fabric as shown in Fig. 3 left. Continuous yarn is first formed into a row of stitches, called a *course*. Each new stitch is created by pulling a loop of yarn through a pre-existing stitch in the previous course. The new stitch remains on a knitting needle until it is stabilized by the formation of a stitch for the subsequent course. The vertical connections between these stitches form columns, called *wales*. 3D knitted surfaces can also be shaped by irregular stitches, such as *increases*, *decreases*, and *short-rows*. To knit the above stitches, knitting machines only need to perform four basic needle operations—*knit*, *tuck*, *split* and *transfer* — and *rack*, which can shift the back needle bed laterally as a whole. We refer readers to McCann et al. [2016] for more detailed information about machine knitting.

Traditional design tools developed by knitting machine manufacturers [Shima Seiki 2011; Stoll 2011] work in the construction space of the machine (called needle×time space), requiring designers to determine stitch types, connectivity, and construction time and order at the same time. McCann et al. [2016] recognized the lack of tools for creating intricate and seamless 3D knitted surfaces with a knitting machine. They presented a design system based on shape primitives, together with an algorithm that generates low-level instructions that can be stored in the Knitout file format [McCann 2017] in a machine-independent way. Later, Kaspar et al. [2019a] proposed an interactive interface that allows designers to compose customized knitted garments with simple high-level primitives.

[1] http://www.burdastyle.com

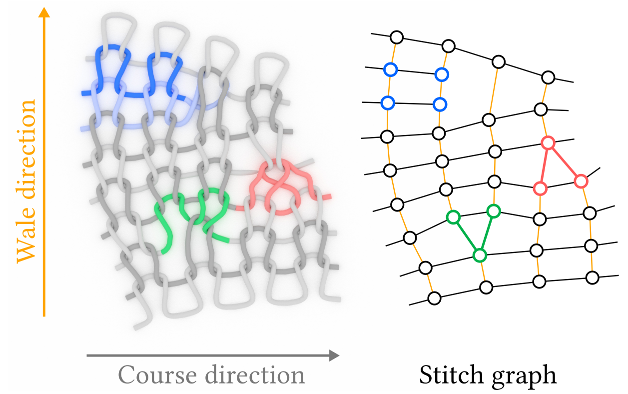


Fig. 3. *Important knitting concepts: Left,* courses *and* wales *as rows and columns; the corresponding* stitch graph *whose nodes represent stitches, and whose edges form the stitch connectivity. Irregular structures for shaping include the stitch* increase *with two next wales, the stitch* decrease *with two previous wales, and* short-rows *that only cover a portion of the base course. Right, each stitch node has one or two adjacent* course *neighbors on the same row. The* wale *neighbors can be separated into two groups,* next wale *neighbors and* previous wale *neighbors and each stitch can have zero to two neighbors for each group. The stitch is classified as* regular*, or (1-1), if they have exactly one* next *and one* previous *wale neighbor, while* increase *and* decrease *are* (1-2) *and* (2-1)*, respectively.*

Others have attempted to construct knitting surfaces from 3D surfaces directly. Igarashi et al. [2008a,b] first presented a semi-automatic design assistant that peels the surface with a winding strip and finds areas where increases or decreases are needed. *Stitch meshes* [Yuksel et al. 2012] abstracts different interlocked stitch structures into different stitch mesh faces. Users are allowed to explicitly author patterns on the 3D mesh. That work was extended to support hand knitting [Wu et al. 2019] and to allow conversion from arbitrary 3D surface automatically [Wu et al. 2018]. Narayanan et al. [2018] use the stitch mesh's dual structure, the *knit graph*, to represent the knitting structure. Each node in a knit graph represents two knit loops; the nodes are connected to each other based on the actual yarn geometry. Narayanan et al. [2018] also proposed a computational approach that can automatically transform 3D meshes into machine knitting instructions, while Popescu et al. [2018] manually segment the complex surface before converting it into a knit graph. Similar to the knit graph [Narayanan et al. 2018], we specify the neighborhood (wale and course connections) of each stitch using a *stitch graph* (Fig. 3 right).

Recently, Narayanan et al. [2019] also introduced an augmented stitch meshes framework for machine knitting design, in which each stitch mesh face is embedded with low-level knitting machine instructions. Similar to Kaspar et al. [2019a], by modifying how individual units/faces are interpreted, pattern instructions can be effectively generated after scheduling. There are also many works focusing on efficient transfer planning for flat knitting patterns [Lin et al. 2018], patch-level knitting ordering [Wu et al. 2021], automatic knitting program generation [Kaspar et al. 2019b; Scheidt

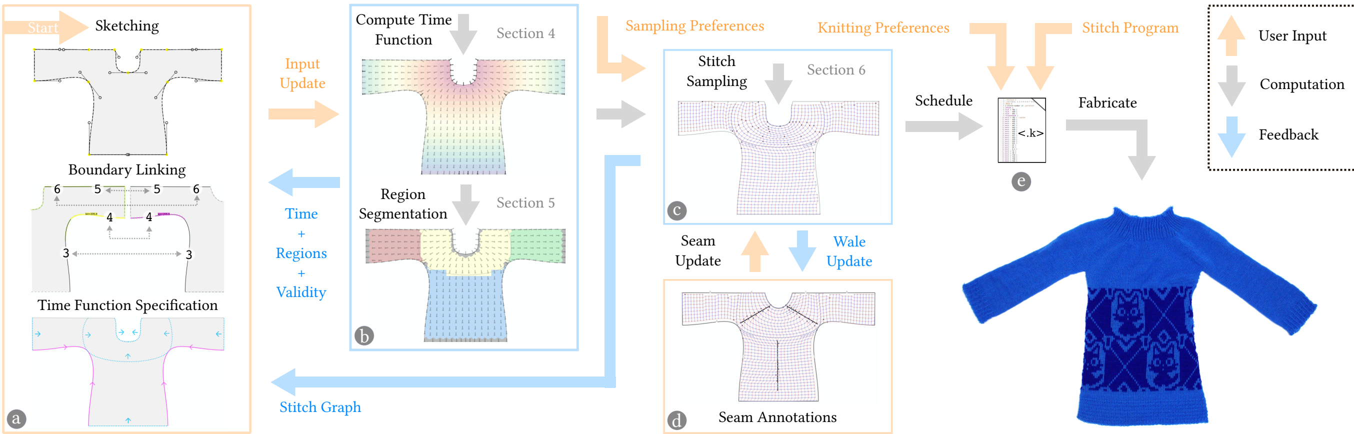


Fig. 4. *Summary of our workflow: (a) the user sketches a garment, links its boundaries and specifies time constraints; (b) the corresponding time function is computed, and its regions segmented; (c) given user sampling preferences (size and course/wale ratios), a stitch graph is sampled; (d) the user potentially provides additional seam annotations to influence the wale distribution until satisfied; (e) given knitting preferences, a schedule is generated and the physical artifact can then be knitted.*

et al. 2020], and interpreting hand-knitting patterns for knitting machines [Hofmann et al. 2019]. Although our work extends the scheduling from Narayanan et al. [2018], we restrict the editing domain to 2D garment patterns to enable interactive editing of both the garment shape and its associated knitting time process. We also introduce additional user constraints and novels user controls, including simplicity tradeoffs for the optimized stitch graph.

3 WORKFLOW

Our approach hinges on the fact that complex 3D garments can be partitioned into a set of simple, closed regions that can be embedded within the 2D plane as in the traditional cut & sew workflow. In differential geometry, each region of the garment's 3D manifold is called a *chart*, and the 2D embedding is the chart's image under the flattening function. For notational simplicity, we use the term *chart* to refer to the flattened 2D domain. As in differential geometry, the collection of 2D charts that fully prescribes a given garment is called an *atlas*.

This section provides an overview of our design process for a given atlas, as illustrated in Fig. 4.

Sketching. The user starts by inputting the desired atlas. Each chart is specified by its boundary shape, which is given by a closed poly-Bezier curve. The user can either draw the charts from scratch or import external SVG files (e.g., from existing cut & sew patterns).

Boundary Linking. Users must also indicate the charts' intended connectivity by annotating boundary segments that should be linked in the final garment. Practically, each Bezier curve along the chart boundary is a linkable boundary segment. A pair of boundary segments should be considered *linked* if they are co-located on the assembled garment. We restrict the design of the base shape to be 2-manifold so that any boundary segment can be linked to at most one other segment.

Time Function Specification. Knitting is a time-dependent process. The time function characterizes many important features about the garment, i.e., stitches' locations and course/wale orientations. The user can design their own knitting time process by specifying different types of time constraints over the atlas.

Time and Region Computations. Once the user has provided a linked atlas with the desired time function constraints, our system automatically solves for the time t over each chart (Sec. 4). Our system also provides feedback about the feasibility of the time function w.r.t. the knitting program space, and reports any notable physical issues, i.e., excessively large local time stretch which may lead to yarn breakage. Based on this time function, our system decomposes the atlas into simple regions (e.g., tubes and sheets) that are straightforward to knit (Sec. 5). These often coincide with semantically meaningful portions of the garment, such as the sleeves, torso, and yoke of a sweater. These steps are solved at an interactive frame rate, which allows the user to get continuous feedback as they adjust their desired shape and constraints.

Stitch Sampling. Once the user converges to a garment specification, they set the desired sampling size (from sketch space to physical units) as well as course and wale sizes. Our system then constructs a *stitch graph* that, when knitted, will yield the desired garment (Sec. 6). The user can further tweak a set of weights to control the relative tradeoff between the size *accuracy* of the garment and the topological *simplicity* of the final stitch graph.

Scheduling and Fabrication. Given the stitch graph, our system traces the yarn, schedules needles, and outputs machine-independent instructions. This takes into account any user preferences for fabrication (e.g., the type of increase stitch to use, and the cast-on/off procedures), and a list of user-specified *stitch programs* that map from stitch to knitting instructions, enabling colorwork and surface texture. The resulting Knitout file [McCann 2017] can be compiled for the target knitting machine before actually knitting it.

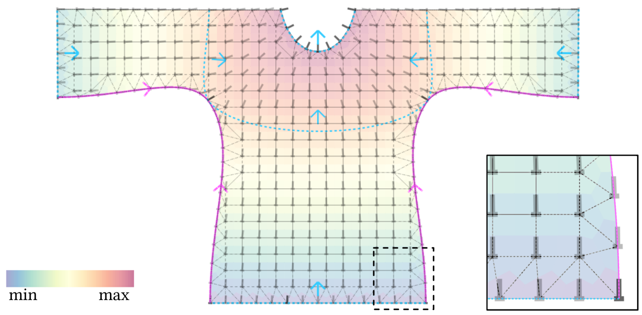


Fig. 5. *Color visualization of the time function over the back of a sweater, together with the underlying mesh illustrating the mixed quad-triangle neighborhoods. Each sample is annotated with a small tack ⊥ representing the flow direction ϕ. The purple arrowed curves are flow direction constraints; the light blue curves with orthogonal arrows are time isoline constraints, with a given flow orientation.*

4 COMPUTING THE KNITTING TIME FUNCTION

Given a garment atlas with multiple linked charts, the first computational step is to determine the knitting time over the domain. In particular, we must determine the order in which the garment is knit, the orientation of each stitch course (row), and the wale (column) connections between rows. We define a continuous scalar field of the *time* t over the garment atlas to represent when the knitting process happens locally. The courses align with the *time isoline* curves, along which t remains constant. The wale connections follow the direction of the time gradient $\phi = \nabla t / \|\nabla t\|$. This direction field should be as smooth as possible, because variations in the field imply local stretching or contraction between stitches. Excessive deviations cause visual artifacts and potential failures during the knitting process, so they must be avoided.

From an optimization perspective, we are seeking a function t whose gradient is intrinsically smooth, i.e., minimizing

$$\int_M \|\nabla \cdot (\nabla t)\|^2 = \int_M \|\Delta t\|^2 \tag{1}$$

over the garment atlas M, subject to user constraints (either *soft* or *hard*). The first set of user constraints are *direction constraints*; these are curves whose tangents or normals dictate the orientation of the direction field ϕ. The second set are *time equality constraints*, which specify individual time isoline curves. In contrast to Narayanan et al. [2018], we provide control on intermediate isolines and the direction field, with an emphasis on interactive editing of the sketch domain and constraints.

We solve for a suitable t automatically using a series of optimizations over increasingly fine chart discretizations.

4.1 Discretization

To discretize each chart into a mesh with a given resolution, we first generate *grid samples*, which are uniformly distributed throughout the interior of the chart using a regular grid with spacing Δ_s. Then, we generate a set of *boundary samples* to capture the boundary of each chart. These are distributed along the boundary as uniformly as possible while adhering to several guidelines.

In particular, we always require boundary samples at the start and end point of each boundary segment. If the boundary segment is not linked to any other segment, we sample the rest of it using a uniform arc-length sampling that matches the local grid cell size, Δ_s. However, if the boundary segment *is* linked (i.e., it is co-located with another segment in the final garment), the linked boundaries must have a consistent representation that can be used to reconcile field values across the charts. To ensure this, we require a bijection between the samples on each linked boundary. Each pair of *linked samples* given by this bijection must be co-located in the final garment. With respect to the final garment, the boundary samples are distributed according to the larger of the linked charts' sampling rates. The spacing of the boundary samples may differ in each local chart embedding as the edge lengths of linked sketch borders are not required to match exactly.

The resulting boundary and grid samples are then connected to their neighboring samples in order to create the mesh over each chart. On the interior of the mesh, we use quads to connect the grid samples with their neighboring samples. The more complex boundary region between the interior and the border uses a Delaunay triangulation, as visualized in Fig. 5.

For the sake of brevity, we introduce the following notation:

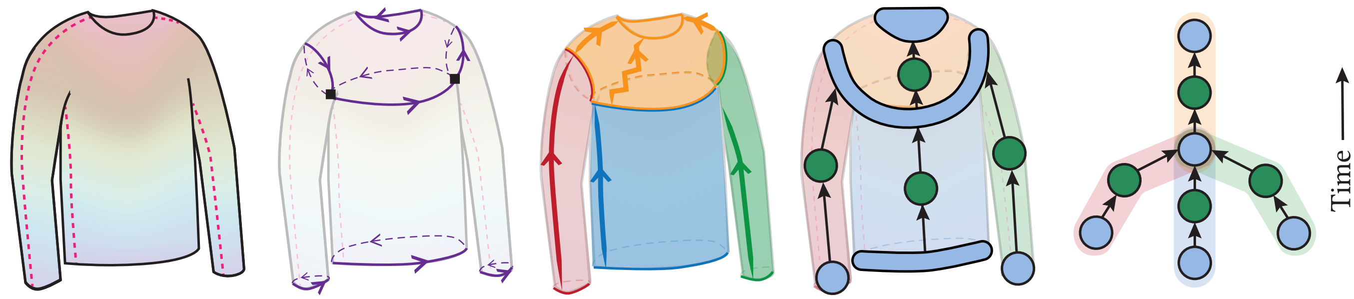
- A *vertex* v refers to some location on the inferred (but never explicitly instantiated) garment manifold. Each v corresponds to one or more samples embedded in the charts.
- $\mathcal{L}(v)$ is the set of samples that are images of v within the charts. $\mathcal{L}(v)$ has one element if v corresponds to an unlinked sample, or more than one if v corresponds to linked samples.
- $\mathcal{N}(s)$ is the set of neighboring samples that share an edge with sample s. All samples in $\mathcal{N}(s)$ must belong to the same chart, and cannot cross any boundary segments (including linked segments that belong to the same chart).
- $C(s) = \{c | s \in \text{supp}(c)\}$ is the set of constraints which s is in the support of (i.e., c affects s directly).

Every sample s has an associated value for each of the two fields: the time $t(s)$ and the direction $\phi(s)$. The quantities associated with linked samples are independent from one another, but we reconcile the values to ensure consistency. Both fields are extended over the entire chart domain by interpolation: linear over sample mesh edges, barycentric over triangles and bilinear over quads.

4.2 Computing Time and Direction Fields

Our strategy is to successively solve for the direction and time fields in a coarse-to-fine manner over meshes of increasingly higher resolution to ensure fast convergence. At each level, we solve for the direction field and integrate to get the time function. To ensure interactivity and fast visual feedback, each optimization is done using Gauss-Seidel iterations that update the quantity at each sample. The updates are done first in the interior of the charts, and then along the border samples. The optimization stops once early termination criteria are satisfied or the maximum number of iterations have occurred. Then, the time and direction fields are upsampled for the next higher-resolution mesh, and the process repeats.

4.2.1 Solving for the Direction Field. We use the normalized direction averaging strategy of Jakob et al. [2015] to efficiently solve for



(a) *Input time function on the garment manifold* **(b)** *Traced isoline segments and separating vertices (■)* **(c)** *Computed regions from dependency paths* **(d)** *Equivalent views of the bipartite region graph relating the garment regions and isoline interfaces over time*

Fig. 6. *Illustration of the steps of our region computation: (a) we start from the time function defined on the garment atlas, (b) we trace a set of isolines that is sufficient to segment the sketch domain into simple regions to knit, each isoline being further decomposed into different oriented segments, (c) we create regions on each side of the isoline segments and merge them by following dependency paths along the sketch manifold, and (d) we create the corresponding bipartite graph with 2-coloring separating nodes into regions and isoline interfaces.*

the direction field $\phi(s)$ at each sample s, namely:

$$\phi(s) \leftarrow \frac{\sum_{s_\mathcal{N} \in \mathcal{N}(s)} w_{s_\mathcal{N}} \phi(s_\mathcal{N}) + \sum_{c \in C(s)} w_c \phi_c}{\sum_{s_\mathcal{N} \in \mathcal{N}(s)} w_{s_\mathcal{N}} + \sum_{c \in C(s)} w_c}, \quad \phi(s) \leftarrow \frac{\phi(s)}{\|\phi(s)\|}, \tag{2}$$

where $w_{s_\mathcal{N}} = 1/\|p(s_\mathcal{N}) - p(s)\|$ and $w_c = \gamma_c/\|\Pi(s,c) - p(s)\|$ with γ_c being a per-constraint, positive, user-tunable weight. ϕ_c is the fixed direction that constraint c enforces on s; $p(s)$ refers to the position of s in local chart coordinates; and $\Pi(s,c)$ is its Euclidean projection onto the curve of c. For all samples in the support of *hard* constraints, we set $w_{s_\mathcal{N}} = 0$.

After each iteration over the full atlas, the directions across linked samples are reconciled to ensure a consistent solution: the samples must have the same orientation, but can have either the same direction (through-flow) or an opposite one (source or sink).

To compare direction vectors across different charts, a common coordinate system is necessary. Given vertex v, all the linked directions are rotated into the domain of one of the charts associated with v. Then, the average orientation is computed and transformed into individual directions that are rotated back to the local domains of the corresponding linked samples.

4.2.2 *Integrating the Knitting Time.*

After the direction field has converged, we iteratively propagate the time over the atlas. On each iteration, the time is computed by (1) integrating it locally over the full atlas, (2) enforcing the time isoline constraints, and (3) averaging the time across linked samples across charts.

Before starting, we select one seed sample s_{seed} with a large neighborhood to propagate from, and fix its time to be $t(s_{\text{seed}}) = 0$.

Step 1. The time integration uses the converged direction field ϕ to update the time at vertex v based on its neighbors' time values:

$$t(v) \leftarrow \frac{1}{|\mathcal{L}(v)|} \sum_{s \in \mathcal{L}(v)} \frac{1}{|\mathcal{N}(s)|} \sum_{s_\mathcal{N} \in \mathcal{N}(s)} [t(s_\mathcal{N}) + dt(s_\mathcal{N} \to s)]. \tag{3}$$

The $|\cdot|$ operator is the set cardinality and $dt(s_\mathcal{N} \to s)$ is the expected local time difference, computed as the dot-product $(\cdot)$ between the average direction and the position difference:

$$dt(s_\mathcal{N} \to s) = \frac{1}{2}[\phi(s_\mathcal{N}) + \phi(s)] \cdot [p(s) - p(s_\mathcal{N})]. \tag{4}$$

Step 2. The time isoline constraints are enforced by averaging the contribution of all samples within their support, and then back-propagating that average time to the individual samples. We average the expected time *after projecting the samples* onto the isoline curve:

$$t(c) \leftarrow \frac{1}{|\text{supp}(c)|} \sum_{s \in \text{supp}(c)} \left[t(s) + dt[s \to \Pi(s,c)]\right], \tag{5}$$

$$t(s) \leftarrow t(c) - dt[s \to \Pi(s,c)]. \tag{6}$$

Step 3. Finally, the time is averaged across linked samples:

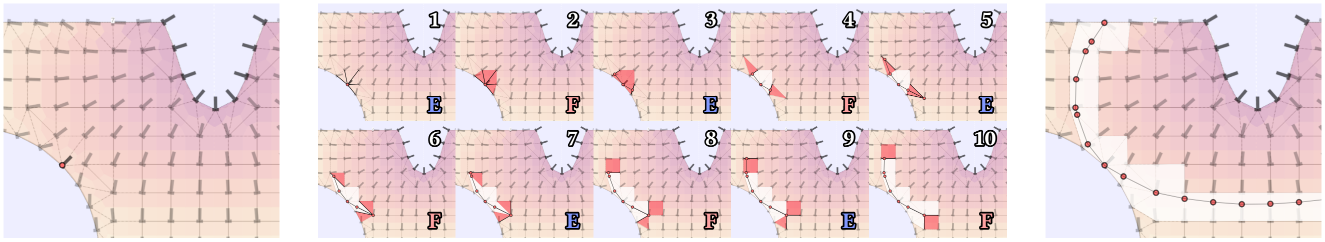
$$t(v) \leftarrow \frac{1}{|\mathcal{L}(v)|} \sum_{s \in \mathcal{L}(v)} t(s), \quad \text{then} \quad t(s) \leftarrow t(v)\big|_{s \in \mathcal{L}(v)}. \tag{7}$$

4.2.3 *Termination, Validation & Normalization.*

For each field, we measure the variation of the field among the samples inside of the charts at the end of each iteration I and stop the computation when it is below a given threshold, i.e., $\max_s |1 - \phi_I(s) \cdot \phi_{I-1}(s)| < \epsilon_\phi$ and $\max_s |t_I(s) - t_{I-1}(s)| < \epsilon_t$, respectively. Once the time function has been solved over our finest resolution sample mesh, the system checks if any local time extrema are found within the sketches. If so, the field is deemed invalid and this feedback is provided to the user. Finally, any source/sink on linked sketch boundaries are topologically opened. This normalization simplifies the treatment of boundaries which can now all be viewed as *open*. The supplement provides details on both the user feedback and the opening strategy.

5 REGION GRAPH CONSTRUCTION

The next step is to automatically decompose the linked garment into a minimal set of regions that are simple to knit, such as tubes and flat sheets, while conforming to the time and direction fields. A *simple region* must be knittable using only traditional forms of shaping, including stitch increases/decreases and short-rows. In particular, simple regions cannot contain non-trivial topological features like



(a) *Initial vertex* **(b)** *Sequence left-to-right, top-to-bottom of finding edges (E) and faces (F)* **(c)** *Final isoline trace*

Fig. 7. *Illustration of isoline tracing: starting from a location (here a vertex), we alternate between the adjacent edges (E) that contain the given isoline time, and their adjacent faces (F), until we've traced the whole isoline domain. Here the color of the mesh visualizes its time function.*

splits or merges. The time isolines that serve as *interfaces* between a garment's minimal set of simple regions are called *critical isolines*.

These *critical isolines* include all isolines that coincide with:

- a topological split or merge,
- a topological change (from flat to tubular, or vice versa), or
- a boundary of the final garment manifold (e.g., the edge of a cuff or neckline).

Note that regions may be bounded by some *portion* of a given isoline if the isoline coincides with a topological split, merge, or change, e.g., when joining the sweater sleeves and trunk into the yoke region in Fig. 6b. Thus, we denote each simple region with a pair of lower and upper *isoline segment sets* ($\mathcal{S}^{\text{low}}, \mathcal{S}^{\text{up}}$), in which each set $\mathcal{S}$ contains a number of isoline segments $\{\sigma_0, \sigma_1, \ldots\}$ at time isoline L^{low} and L^{up}, with time values $t(L^{\text{low}}) < t(L^{\text{up}})$, respectively.

We aim to compute the set of critical isolines and simple regions, along with the *knitting time dependency* between them. However, it is difficult to directly extract all critical isolines, as topological structures are not always apparent from the linked atlas alone. Instead, we follow the 3-stage process outlined in Fig. 6. Given a time-annotated garment, we first identify the collection of candidate isolines that are likely to serve as delimiters between neighboring regions (Sec. 5.1). Then, we build the directed connections between all simple regions (Sec. 5.2). Finally, we transform the region and interface dependencies into a directed acyclic bipartite graph (Sec. 5.3).

5.1 Tracing Candidate Isolines

We begin by identifying the set of vertices $\mathcal{V}$ from which we should trace candidate isolines. This includes vertices that are located at (1) *corners* (start or end of chart boundary segments), or (2) *local time extrema* along their boundary segment. The corner vertices are where the most common topological splittings and mergings happen; the time extrema along boundaries capture the remaining topological events (including flow sources and sinks). Note that $\mathcal{V}$ is *complete*, but not all its elements are *necessary*, e.g., subdividing a sketch boundary does not necessarily indicate a change in topology.

Beginning from vertex $v \in \mathcal{V}$ with time $t(v)$, an isoline is traced by alternating between two operations: (1) from a given neighborhood (vertex, edge, or face), find all adjacent edges that contain the given time t, and (2) from a given edge, find all faces that are adjacent to it. This generates a continuous isoline path over the garment manifold, as illustrated in Fig. 7.

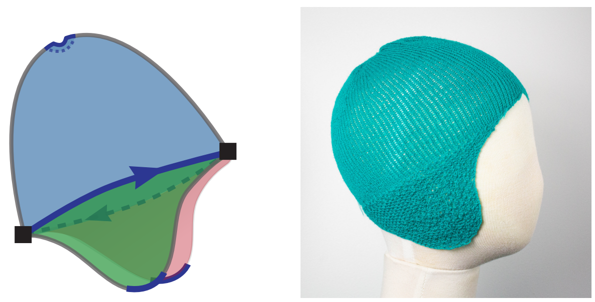


Fig. 8. *Examples of separating vertices (■) at the boundaries of the garment manifold. The central isoline is split into two segments separated by vertices that form transitions between being* inside *the shape (above each ear flap), and at its* boundary *(between both ear flaps). The top isoline surrounds a pointwise sink of the time function, which was topologically opened.*

Any isoline paths that encompass non-trivial topological features (e.g., topological split, merge, or change) are then subdivided into multiple segments. The isolines are partitioned at the topologically critical points, or *separating vertices*, which are given by one of two scenarios. In the first case, a separating vertex coincides with more than two edges of the isoline for time t (as in the armpit of Fig. 6b). The second case indicates the point at which an isoline path transitions between being on the *interior* of the garment manifold and being on its *boundary*. This is illustrated by the central isoline of the beanie, which separates the ear flaps from the main body in Fig. 8. The isoline is primarily on the garment's interior, but it intersects the boundary at each of the two separating vertices (■). This is interpreted as a topological change over this isoline: the two lower flat regions merge into the upper circular body region.

5.2 Computing Regions from Dependency Paths

The next step is to uncover the simple regions by inferring the connectivity between the candidate isolines. Each simple region must be bounded by two sets of isoline segments — $\mathcal{S}^{\text{low}}$ and $\mathcal{S}^{\text{up}}$ — that can be connected along a continuous path through the garment without passing through any other candidate isoline. Thus, the set of regions and their extents can be determined by tracing paths

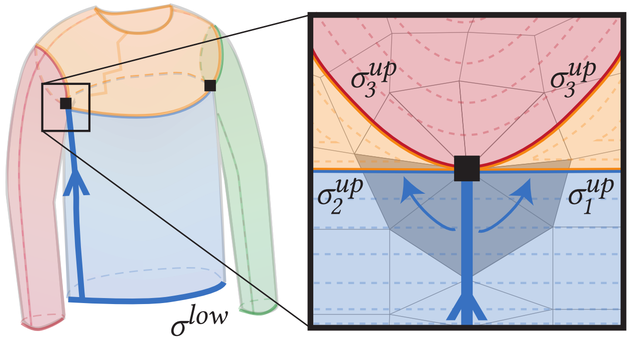


Fig. 9. *When a dependency path (blue line) reaches a candidate isoline at a separating vertex (■), we must take extra steps to determine which of the incident isoline segments (σ_1^{up}, σ_2^{up}, or σ_3^{up}) bound the region in question (blue). We decide this by traversing the triangle fan that surrounds the vertex, until reaching (or crossing) the nearest candidate isoline segment in each direction (σ_1^{up} and σ_2^{up}).*

from each isoline segment to its next reachable neighbor, in order to confirm their local connectivity.

Each isoline segment σ_i can be associated with at most two regions: its *preceding* region (for which $\sigma_i \in \mathcal{S}^{\text{up}}$), and its *subsequent* region (for which $\sigma_i \in \mathcal{S}^{\text{low}}$). Initially, no other members of $\mathcal{S}^{\text{low}}$ or $\mathcal{S}^{\text{up}}$ are known, so it is only possible to allocate a partially-known region for each side of σ_i. Then, a dependency path is traced from each lower segment σ_i, eventually reaching another isoline segment σ_j (with $t(\sigma_i) < t(\sigma_j)$). This confirms that the subsequent region of σ_i and the preceding region of σ_j are identical and allows us to merge them into a single region with the union of the corresponding isoline segments on either side.

Our system generates dependency paths by following edges of the mesh in a specific time direction until some isoline segment is *reached. Reaching* essentially means that the last edge e of the path intersects an isoline. This intersection may occur within e or at an end vertex of e. In the former case, the dependency always indicates a single isoline segment σ_j. If the latter case occurs at a separating vertex v, there may be several incident isoline segments that partition the local neighborhood around v into sectors representing distinct regions, as shown in Fig. 9. In such a case, the dependency path only *reaches* the isoline segment(s) that delimit the sector from which the path originated.

5.3 Building the Bipartite Region Graph

Armed with the garment's topological structure, non-critical isolines are filtered out to produce the desired minimal set of simple regions. The *non-critical* isolines are those which (1) connect a single preceding region to a single subsequent region, **and** (2) have topologically identical structures on both sides (i.e., flat to flat, or circular to circular). Note that both criteria are necessary for pruning. For instance, if an isoline has a single previous and a single next region but its topology changes (from flat to circular or vice versa), the isoline is considered critical. Once a non-critical isoline is removed, its preceding and subsequent regions are merged.

After resolving the minimal set of regions, we construct a bipartite *region graph* that represents the final garment decomposition. This graph has a node set $\mathcal{I}$ to represent interfaces (critical isolines), another node set $\mathcal{R}$ to represent each simple region, and a *directed* edge set $\mathcal{E}$ to connect related isolines and regions. Each directed edge $\mathbf{e}_i \in \mathcal{E}$ corresponds to an isoline segment set $\mathcal{S}_i = \{\sigma_0, \sigma_1, \ldots\}$. Moreover, for a given interface node η, any incident edges in $\mathcal{E}_\eta^{\text{in}}$ originate at a preceding region/interface, and those in $\mathcal{E}_\eta^{\text{out}}$ lead to a subsequent one. This yields the final graph $G = (\{\mathcal{I}, \mathcal{R}\}, \mathcal{E})$ as shown in Fig. 6d. In the supplement, we describe how the user can interactively control the complexity of this graph so as to prune excessively small regions.

6 HIERARCHICAL STITCH SAMPLING

In order to generate machine knittable instructions from the region graph, a *stitch graph* must be instantiated. Our stitch graph computation is formulated as a global hierarchical optimization over the region graph and sketch atlas. Unlike previous works, each phase of our optimization accounts for both (1) the garment size accuracy and (2) its topological simplicity. These goals are fundamentally conflicting, because irregular topologies (shaping or short-rows) often improve size accuracy, but this typically happens to the detriment of a simple topology (regularity of stitches). Our system solves optimization problems that allow the user to navigate the tradeoff between garment size accuracy and topological simplicity. Moreover, our formulation enables the user to interactively control the wale alignment using seam annotations.

Our optimization approach has multiple stages illustrated in Fig. 10. First, we optimize the number of stitches at each interface (Sec. 6.1). We then optimize the number of full courses and short-rows within each region and the number of stitches placed along each full course (Sec. 6.2). Next, we create all course stitches with their course connectivity, and optimize the wale connectivity across the interfaces and within each region, while taking the user's seam annotations into account (Sec. 6.3). Finally, we insert short-row stitches (Sec. 6.4) and convert the final stitch graph into a knitting program (Sec. 6.5).

6.1 Interface Sampling

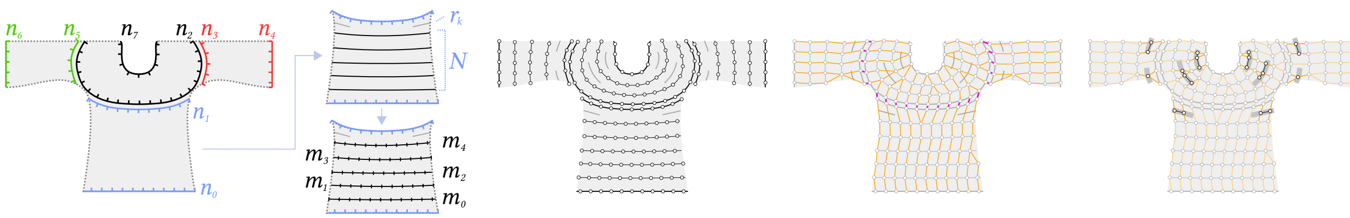
To determine the stitch count n_i at each edge $\mathbf{e}_i \in \mathcal{E}$ within the bipartite region graph $G = (\{\mathcal{I}, \mathcal{R}\}, \mathcal{E})$, we formulate an Integer Quadratic Programming problem (IQP) with linear constraints:

$$\begin{aligned} \arg\min_{\mathbf{n}} \ & \lambda_{\text{crs}} \sum_{\mathbf{e}_i \in \mathcal{E}} E_{\text{crs}}(n_i) + \lambda_{\text{smpl}} \sum_{(\mathbf{e}_i, \mathbf{e}_j) \in \mathcal{R}} E_{\text{smpl}}(n_i, n_j) \\ \text{s.t.}\ & \forall\, \eta \in \mathcal{I}_{\text{internal}}, \sum_{\mathbf{e}_i \in \mathcal{E}_\eta^{\text{in}}} n_i = \sum_{\mathbf{e}_j \in \mathcal{E}_\eta^{\text{out}}} n_j. \end{aligned} \tag{8}$$

The first term E_{crs} measures the per-edge *course accuracy* for the stitch count n_i along $\mathbf{e}_i$:

$$E_{\text{crs}}(n_i) = \left| n_i - \frac{\omega_i}{D_{\text{crs}}} \right|^2, \tag{9}$$

where D_{crs} is the expected distance between the center of adjacent course-connected stitches and ω_i is the user's desired course width, as indicated by the scaled atlas.



(a) *Interface sampling* **(b)** *Region sampling* **(c)** *Course connectivity* **(d)** *Wale connectivity* **(e)** *Short-row Insertion*

Fig. 10. *Illustration of the steps of our sampling algorithm: (a) optimizing stitch numbers at region interfaces (Sec. 6.1), (b) optimizing course number, short-row densities and stitch numbers in each region (Sec. 6.2), (c) creating stitch courses (Sec. 6.3.1), (d) pairing stitches between adjacent courses across interfaces (Sec. 6.3.2) and within regions (Sec. 6.3.3), and (e) generating short-row stitches (Sec. 6.4).*

The simplicity term E_{smpl} penalizes large differences in stitch counts (n_i, n_j) between the beginning and end of a given region $(\mathbf{e}_i, \mathbf{e}_j)$ so as to encourage simple regions with minimal shaping:

$$E_{\text{smpl}}(n_i, n_j) = |n_i - n_j|^2. \tag{10}$$

The constraints in Eq. 8 ensure that courses on either side of an *internal* interface (i.e., those with $\mathcal{E}_\eta^{\text{in}}, \mathcal{E}_\eta^{\text{out}} \neq \emptyset$) have the same number of stitches. The user-specified weights λ_{crs} and λ_{smpl} control the trade-off between course accuracy and simplicity.

6.2 Region Sampling

After optimizing the stitch count n for each of the interface edges, we optimize the sizing along the wale and course directions, respectively, within each region. All regions can be solved in parallel.

6.2.1 Sizing Along the Wale Direction. To ensure that each region has the desired measurements along the wale direction, we minimize an energy penalty for the wale size accuracy across the region. In particular, we subdivide the region by tracing N isolines uniformly along its time extents and accumulating the local wale error across those while accounting for a number of additional short-rows $\mathbf{r}$ to fill the distance in between. The subdivision produces N isoline segment sets $\mathcal{S}_i \in \mathcal{U}$ for $N+1$ sub-regions $(\mathcal{S}_i, \mathcal{S}_j) \in \mathcal{A}$. We optimize for both the number of subdivisions N and the local short-rows $\mathbf{r}$ between each sub-region:

$$\underset{N,\,\mathbf{r}}{\arg\min} \sum_{(\mathcal{S}_i, \mathcal{S}_j)\in\mathcal{A}} (\lambda_{\text{wale}} E_{\text{wale}}(\mathcal{S}_i, \mathcal{S}_j) + \lambda_{\text{srs}} E_{\text{srs}}(\mathcal{S}_i, \mathcal{S}_j)), \tag{11}$$

where the energy term E_{wale} measures the size accuracy along the wale direction and the short-row simplicity term E_{srs} penalizes adjacent short-row densities that change too fast.

To measure E_{wale} and E_{srs}, K sample pairs $(s_{i,k}, s_{j,k})$ are uniformly distributed along $\mathcal{S}_i$ and $\mathcal{S}_j$, respectively. We let r_k be the number of *additional* short-rows between each sample pair. For efficiency, our value of K is typically much smaller than the final number of stitches on the courses. In particular, K is computed based on the curve lengths $\ell(\cdot)$ and the distance Δ_s between adjacent grid samples at the finest mesh resolution: $K = \lceil \max(\ell(\mathcal{S}_i), \ell(\mathcal{S}_j))/\Delta_s \rceil$. Then, E_{wale} and E_{srs} can be defined in a discretized form as follows:

$$E_{\text{wale}} = \sum_{k=1}^{K} \left| \frac{\mathrm{G}(s_{i,k}, s_{j,k})}{D_{\text{wale}}} - 1 - r_k \right|^2, \quad E_{\text{srs}} = \sum_{k=1}^{K} |r_k - r_{k-1}|^2, \tag{12}$$

where $\mathrm{G}(s_{i,k}, s_{j,k})$ is the geodesic distance between samples $s_{i,k}$ and $s_{j,k}$, and the -1 term accounts for the implicit wale step that happens between $\mathcal{S}_i$ and $\mathcal{S}_j$.

Full courses are preferable to short-rows wherever possible, as the latter tend to increase knitting complexity. To enforce this, we require that at least one sample pair from every sub-region ends up with no intermediate short-row density– i.e., $\exists\, k,\; r_k = 0$ between each $(\mathcal{S}_i, \mathcal{S}_j)$. By optimizing Eq. 11 subject to this constraint, we bias the solution toward full-course isolines (large N, small r_k) rather than relying on short-rows (lower N, large r_k).

6.2.2 Sizing Along the Course Direction. Given the best value of N, we optimize for the number of stitches m_i along each $\mathcal{S}_i \in \mathcal{U}$. This is formulated as a similar constrained IQP problem to that of Eq. 8, with a tradeoff between course accuracy and simplicity:

$$\underset{m}{\arg\min}\; \lambda_{\text{crs}} \sum_{\mathcal{S}_i\in\mathcal{U}} E_{\text{crs}}(m_i) + \lambda_{\text{smpl}} \sum_{(\mathcal{S}_i,\mathcal{S}_j)\in\mathcal{A}} E_{\text{smpl}}(m_i, m_j)$$
$$\text{s.t.}\ \forall\, (\mathcal{S}_i, \mathcal{S}_j) \in \mathcal{A}, \quad \lceil m_j / F_{\max} \rceil \leq m_i \leq \lfloor m_j\, F_{\max} \rfloor. \tag{13}$$

The constraint enforces a user-defined maximum shaping factor $F_{\max} \in (1, 2]$, which limits the rate at which stitch counts can change between adjacent courses. The bounds on $F_{\max}$ ensure that stitch counts can be instantiated into a valid stitch graph, where each stitch has at most two next wales, and at most two previous wales. Because the stitch counts n_i, n_j at the extents of each region have already been fixed by the interface sampling step, the value $F_{\max}$ also implies a minimum value of N that must be respected for a given region: $N_{\min} = \left\lceil \log_{F_{\max}}[\max(\frac{n_i}{n_j}, \frac{n_j}{n_i})] \right\rceil - 1$.

6.3 Stitch Connectivity

After determining the number of courses and stitches in each region, stitches are sampled uniformly along their corresponding isoline. Then, course and wale connections between them are computed to form an initial stitch graph.

6.3.1 Course Connectivity. Adjacent stitches on the same isoline segment set are connected first. The process is trivial for singleton sets, as the sequence of neighboring stitches is clear. For multi-segment sets, it is necessary to determine a *course path* over the segments first, to ensure that the stitch sequence is well-defined. The course path traces a consistently-oriented Eulerian path over the isoline segments, where the *orientation* is defined as the sign of the cross-product between the local displacement and the local time direction field between two subsequent locations in the same sketch. The arrows in Fig. 6b illustrate the default positive orientation.

6.3.2 Connectivity across Interfaces. After connecting stitches on each course within the regions, all regions are connected together by computing a 1-1 wale assignment between the stitches on either side of an interface. Our system optimizes for the alignment between the paired stitches, while enforcing that the adjacent regions have a valid layout on the final needle bed for scheduling.

A greedy wale distribution approach is used to ensure that any circular structures sandwiched between other structures end up split evenly across both knitting beds. For the general case, our system binds N lower courses to M upper courses. We reduce this to a pair of simpler interfaces (an N-to-1 interface followed by a 1-to-M interface), both of which can be solved in a symmetric manner. Our base case is a course that needs binding to M courses, for which we greedily search the best 1-to-1 stitch alignment by

(1) selecting an ordering $(\pi)_{k=1}^{M}$ of the M upper courses, then
(2) sequentially searching for the best layout of the course π_k, which minimizes the geodesic distance between existing stitches after left-to-right packing of courses π_1 to π_k, and
(3) using the overall best ordering π and its wale assignments.

The left-to-right packing assumes that intermediate circular courses get split evenly between front and back. If more than one intermediate course is circular, it may end up with irregular odd packing as described in Narayanan et al. [2018]. To avoid this, our system enforces the optimization in Sec. 6.1 to produce even-parity stitch counts for any interface of $M > 3$ courses.

This approach enables a wide array of practical garment topologies. However, scheduling constraints can be arbitrarily complex for intricate garments, and the general case remains an open problem.

6.3.3 Wale Connectivity. To assign the remaining wale connections between stitches in the region interiors, we extend the *Dynamic Time Warping* strategy of Narayanan et al. [2018] with a modified penalty function E_{penalty} and apply it between each pair of adjacent courses independently. The modified penalty between a source stitch Ω^{src} and a target stitch Ω^{trg} is defined as follows:

$$E_{\text{penalty}} = \lambda_{\text{dist}}\, E_{\text{dist}}(\Omega^{\text{src}}, \Omega^{\text{trg}}) + \lambda_{\text{seam}} \sum_{\Omega \in \{\Omega^{\text{src}}, \Omega^{\text{trg}}\}} \chi(\Omega)\, E_{\text{seam}}(\Omega)\,, \tag{14}$$

where $\chi(\cdot)$ is an indicator of the stitch's irregularity: $\chi(\Omega) = 1$ if Ω is the source of a 1-2 connection, and $\chi(\Omega) = 1$ if Ω is the target of a 2-1 connection; otherwise, $\chi(\Omega) = 0$.

The first term E_{dist} is the normalized squared geodesic distance between Ω^{src} and Ω^{trg} on the garment manifold:

$$E_{\text{dist}}(\Omega^{\text{src}}, \Omega^{\text{trg}}) = \left(\frac{\mathrm{G}(\Omega^{\text{src}}, \Omega^{\text{trg}})}{D_{\text{wale}}}\right)^2. \tag{15}$$

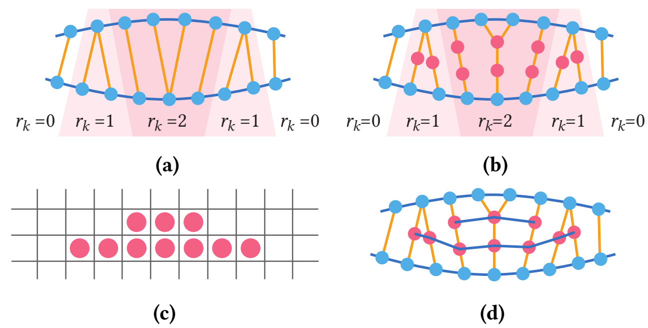


Fig. 11. *Short-row formation by splitting wales: (a) setup with initial wales and short-row densities, (b) uniform distribution of stitches over wales, (c) short-row stitch grid given user alignment (bottom), and (d) the final short-row connectivity.*

The second term E_{seam} is introduced to gather irregular wale connections around the user-specified seam annotations, by penalizing irregular wales that occur far away from any seam location:

$$E_{\text{seam}}(\Omega) = \min\left(\alpha_{\text{seam}}, \frac{\Delta_{\text{seam}}(\Omega)}{D_{\text{crs}}}\right), \tag{16}$$

where $\alpha_{\text{seam}} = \Delta_s \sqrt{2}$ is the interaction support of any seam annotations, Δ_s is the distance between adjacent grid samples at the finest mesh resolution, and $\Delta_{\text{seam}}(\Omega)$ is the Euclidean distance between stitch Ω and the closest seam location in its 2D chart.

After computing the wale connection, users are allowed to further edit their seam annotations interactively. To incorporate the new annotations, the wale distribution optimization described above has to be repeated. To expedites this process, our system preemptively caches the geodesic distances between each stitch pair Ω^{src} and Ω^{trg} during the initial pass of the wale connection optimization. This dramatically reduces the evaluation time for $E_{\text{dist}}(\Omega^{\text{src}}, \Omega^{\text{trg}})$. Note that $E_{\text{seam}}(\Omega)$ cannot be cached because the seam distances must be recomputed with respect to the new annotations, but the Euclidean distance evaluations are fast enough to support interactive editing.

6.4 Short-row Insertion

After connecting all stitches along full courses, short-row stitches are inserted according to r_k from Eq. 11, which indicates the number of short-rows to be instantiated between the sampled pair $s_{i,k}$ and $s_{j,k}$. Our system considers each wale connection between full course stitches $(\Omega_u^{\text{src}}, \Omega_u^{\text{trg}})$, and subdivides the wale into r_u stitches, as shown in Fig. 11. Since the number of stitch pairs generally exceeds the number of sample pairs, the density r_u between $(\Omega_u^{\text{src}}, \Omega_u^{\text{trg}})$ takes on the value r_k from the closest sample pair $(s_{i,k}, s_{j,k})$.

For a 1-1 wale connection, the wale is subdivided into r_u uniformly-spaced stitches. The same process applies for 2-1 wale connections, except stitches are added to *both* wales in this manner. For 1-2 wales, short-row stitches are uniformly placed along the wale path between Ω^{src} and the average location $\overline{\Omega}^{\text{trg}}$ of the two target stitches. The wale connections from the source up to the upper-most short-row stitch are 1-1; only the upper-most short-row stitch has a 1-2 wale connection to the original target stitches.

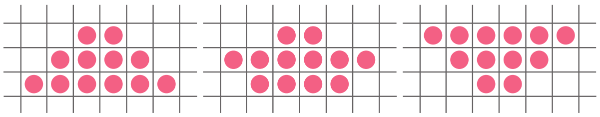


Fig. 12. *Different vertical alignments: from left to right,* bottom, mid-dle *(biased towards bottom) and* top.

After the short-row stitches have been inserted within the wales, they are connected into courses based on a user-defined vertical alignment in a virtual grid. The contiguous stitches in each row of the grid get course-connected, forming the final short-row topology. Our system supports three different vertical alignments (*bottom*, *middle*, and *top*), as illustrated in Fig. 12.

6.5 Knitting Instruction Generation

To convert the final stitch graph into a knitting program (Knitout file), our system executes double tracing and scheduling, similarly to Narayanan et al. [2018]. Unlike previous work, our system also supports some basic mixed circular/flat layouts, as described in the supplementary document.

Although the current system focuses on garment shape specification, it also supports additional colorwork and textures. In particular, each stitch graph node is associated with a *stitch program*, which specifies a mapping from stitch node to knitting instructions. A typical program consists in (1) selecting target stitches using queries similar to Kaspar et al. [2019a], and (2) binding local knitting programs from Narayanan et al. [2019]. See the supplement for examples of the programs used within our results.

7 RESULTS

All results are knit on a Shima Seiki SWG091N2 machine with 15 gauge needles and a 2/30 1-ply acrylic yarn. They are knit in half-gauge, with the following size measurements on a tubular swatch: D_{crs} = 300 mm/100 stitches and D_{wale} = 135 mm/100 stitches.

Most of our garment patterns are created by manually redrawing on top of original patterns selected from BurdaStyle. The only exception are: the first *sweater*, which we drew from scratch to showcase the capabilities of our system and to serve as a simple introductory design, and the *beanie*, which is based on the *Joyful baby bear hat* from Joy Kelley at howjoyful.com. The supplementary document contains larger scale versions of these results together with their respective user parameters, constraints, time function, region decomposition, stitch graphs, and stitch programs.

Young boy garments. Fig. 13 shows the larger-scale examples we knitted for a 4-foot-tall boy mannequin, including three garment pieces. These results verify our pipelines' ability to scale to human-sized garments. The primary constraint preventing a full adult-scale garment is our knitting machine target. Keeping in mind that we knit in half-gauge, our largest example, the sweater, takes over 309 needles of our knitting machine bed, out of 541 available.

The *beanie* with earflaps showcases a mixed flat/tubular structure. Both earflaps use a garter pattern over their entire structure to avoid curling and folding, which is particularly pronounced with

Fig. 13. *Our larger examples on a 4-foot boy mannequin, together with top-down views of the individual garment pieces and a zoom on one of the inseam pockets of the trousers which are knit as inside-out tubular structures merging with the body.*

flat Jersey fabric. The upper section uses a fair-isle pattern that is tiled horizontally and floats the background yarn inside.

The *sweater* includes partial rib patterns at the wrists, a waffle pattern at the base of the trunk, and a radial rib pattern for the neck. It also includes fair-isle colorwork in the center section.

The pair of *trousers* uses ribs around the waist and garter patterns on the ankles. The original trousers pattern did not have any pockets. The inseam pockets on the side of our trousers were added by cutting and pasting the segmentation of a pair of pockets from a different garment. This illustrates that multiple existing sketches can be reused to build more complex ones.

Both the trousers and the shirt exhibit yarn breakage in the armpit regions unless short-rows are used.

Smaller mannequin garments. Fig. 14 shows the top-down views of the complex upper garment patterns from the teaser, together with a visualization of their sketch atlas with linking. They are scaled to fit on a 16-inch wooden mannequin. All garment patterns use patterning at their extremities, typically ribs or garter stitch, to ensure that they don't curl or fold.

The *cardigan* is knitted flat from top to bottom, to avoid having to split the yarn between three sections (front left, front right, and the back). Splitting can lead to yarn breakage unless each section is knit in parallel; our scheduler is sequential, so we do not support this. For the same reason, we do not link the top section, but bind it manually instead. Since the whole structure is flat, we use a global garter stitch pattern to prevent it from curling and folding.

The *princess dress* is knit in two variants. The first version in Fig. 14 is knit as a single piece, showcasing one of the potential advantages

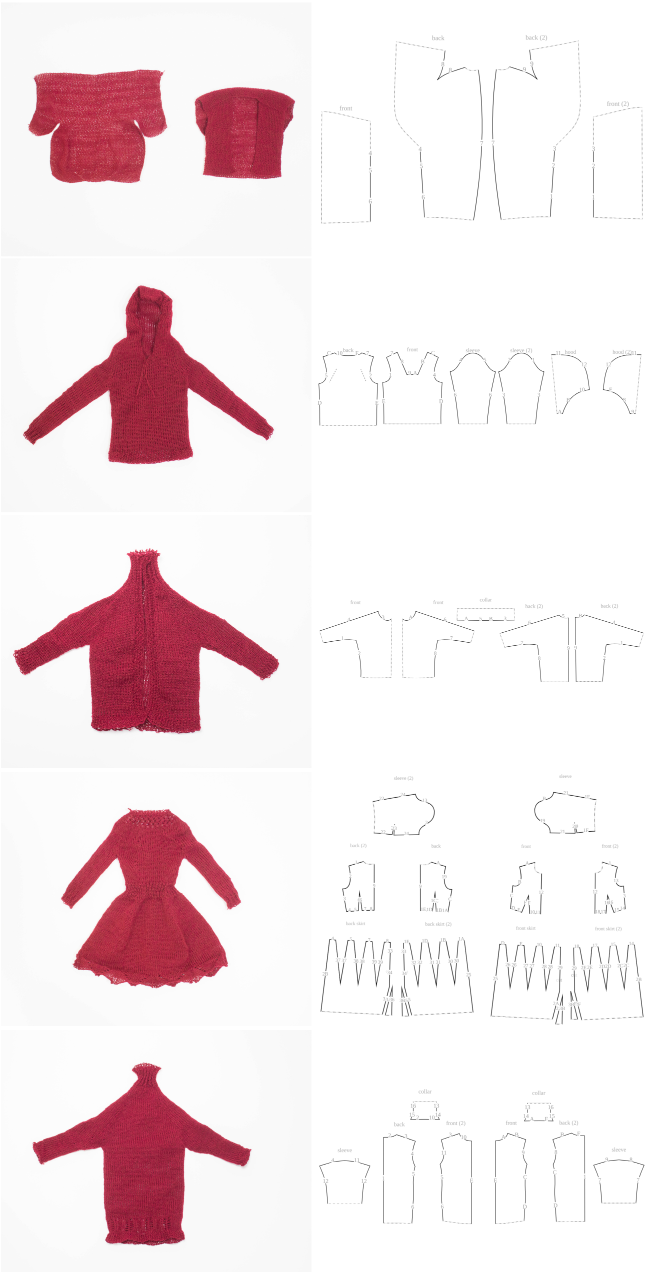


Fig. 14. *On the left: top-down views of the dresses from the teaser; on the right: their linked sketch atlas. From top to bottom: the cardigan, the hoodie, the jacket, the princess dress and the turtleneck dress.*

of whole-garment knitting. The pleats found in the original pattern are non-trivial to knit automatically, so we substitute a series of darts at the interface between the skirt and the body. We attract irregular stitches to the dart edges via seam annotations, and use rib patterns above that waist interface to strengthen the visual impact of the folds. Near the top of the neck and the bottom of the skirt, we showcase different tiled lace patterns. The second version of the dress, shown in Fig. 15, features the original pleated pattern, which can be knit in two sections and bound manually.

Fig. 15. *Two-parts version of the princess dress, with manual binding done with box pleats.*

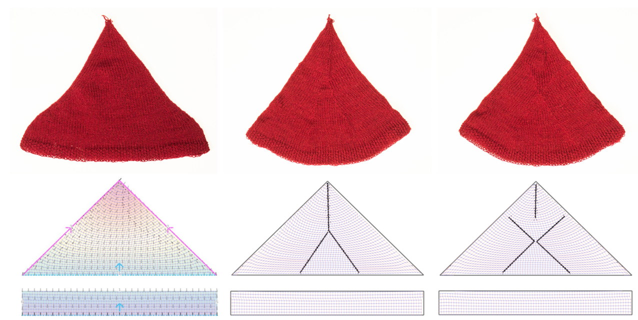


Fig. 16. *Illustration of the impact of seam annotations with the corresponding irregular stitch placement. The bottom figures show the corresponding stitch graphs. Darker stitches correspond to irregular stitches. The left sample has no seam annotation, and we show the color-coded time visualization on top. The right samples highlight the seam placements and irregular stitches are attracted to their vicinity.*

The *hoodie* and *jacket* examples both showcase c-shaped knitting layouts for which one side of the panels are not linked. The *turtleneck dress* was originally opened at the top of the back to make it easier to put on. However, we closed the opening to simplify its manipulation given its physical scale.

Seam Placement. Fig. 16 illustrates the impact of the irregular stitches and how our wale penalty deals with their specific placement. More examples are shown in the supplement. While the location of the singular stitches is reasonably clear in such samples, one limitation of our penalty-based editing is that the wale distribution is done independently per course pair. Thus, we do not have any notion of the alignment of irregular stitches across subsequent courses. Although this global alignment is important in practice, it is not fully controllable in our system.

Sketch	Charts	Levels	Regions	Stitches	Comp. Time	Comp. Region	Sampling	Scheduling
Beanie	2	3	3	13184	0.2	0.1	22.0	1.2
Sweater	2	3	4	47624	0.1	0.1	47.0	5.0
Trousers	12	3	6	57254	0.3	0.2	65.4	5.8
Cardigan	4	3	4	12290	0.1	0.1	6.8	0.0*
Dress	14	2	4	17238	0.8	0.4	29.6	2.1
Hoodie	6	3	5	12874	0.8	0.2	70.9	17.7
Jacket	5	3	4	11252	0.5	0.1	41.2	0.7
Turtleneck	8	3	4	13426	0.8	0.1	26.5	1.6
Shorts	6	2	3	2842	0.1	0.1	8.4	0.6
L trousers	12	3	6	11104	0.2	0.4	19.1	22.5
W trousers	6	3	3	14804	0.6	0.2	23.7	0.4

Table 1. *Runtimes using a single computation thread. The column values correspond either to number counts or time measurements in* seconds. *(*) The cardigan layout is fully flat and without complex branching. This leads to a trivial search space for the scheduler.*

Fig. 17. *Example of knitting failures due to failing needle transfers: the* left *example failed at large decreases above the crotch due to non-ideal schedule alignments; the* right *example had catastrophic failures due to overlapping loop transfers during shaping transfers.*

Interactive System. Our system is implemented as a web browser application in Javascript and WebAssembly. The constrained IQP problems of Section 6 are solved using NLOpt [Johnson 2014] together with a limited time-budget branch-and-bound strategy. The geodesic distance is adaptively computed, making use of Geometry Central [Sharp et al. 2019] for the precomputation, and the exact geodesic algorithm of Surazhsky et al. [2005] for refinement. The supplementary document provides implementation details.

Table 1 summarizes the timings of our system for the different garment results, using an Intel Xeon i7 CPU with 32GB of RAM, as a single-threaded web worker computation beside the UI thread.

As shown in the first timing column, the time and region computation all occur in less than a second. The visual feedback for the knitting time and direction fields can be done in real-time due to the hierarchical and iterative nature of the computation. In practice, we throttle it to some fixed frame rate (e.g., 60FPS) as web-worker transfers induce a noticeable overhead on the total computation.

The second timing column shows that the sampling stages and scheduling are typically not interactive, unless the examples are considered at a relatively small scale. Fortunately, the results are cached after the first pass. This allows the subsequent seam edits to be made at an interactive framerate until we finally converge on a design, then generate the schedule and knitting programs. Those numbers also show high variance because the structure of the input and its symmetries end up having a large impact on both sampling and scheduling. For example, the cardigan is scheduled completely flat and allows for a trivial initial solution that helps the branch and bound exploration finish very quickly. The supplementary document provides details on the timing and convergence of each step.

8 DISCUSSIONS

This section highlights some of the design implications of our workflow, its limitations, and potential extensions.

8.1 Scheduling Algorithms

Existing scheduling algorithms [Lin et al. 2018; Narayanan et al. 2018; Wu et al. 2021] either work with tubular or flat fabric, but not both. To support the scheduling for some of our mixed flat and tubular designs, we extend the work of Narayanan et al. [2018], which parameterized the needle bed layout of tubular structures. We add two different representations for flat structures: *single-fold* and *c-shaped* layouts, further detailed in the supplementary document. The main take-away is that scheduling becomes, perhaps counter-intuitively, harder. Flat structures can be folded in different ways, and their parameter-varying extents substantially increase the search space. While some of the structures may appear simpler locally, their interactions become more complex.

One major issue we encountered with existing schedulers is that they rely on the assumption that transferring stitches around is fine as long as excessive slack and unwanted loop overlaps are avoided. Our experience seems to indicate that large stitch cycle transformations typically lead to some form of failure (due to transfers). Similarly, the current general-purpose transfer procedure *Collapse-Shift-Expand* [McCann et al. 2016] enforces slack and overlap constraints, but allows unrestricted *overlapping loop transfers* for loops that have the same target needle. While having a same target needle is necessary (i.e., for decrease shaping), overlapping loop transfers are a common source of failure. Fig. 17 shows failure cases caused by both issues.

We envision that part of the scheduling should be guided by the user similarly to how our workflow allows control of the directions and isolines of the knitting time process. Current schedulers have enabled many applications, but they would be more practical if the user could interactively manipulate their process.

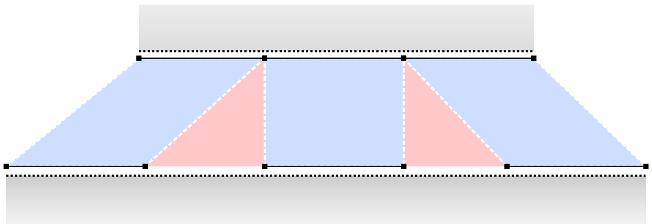


Fig. 18. *Pleat binding: blue regions are links between the two panels (in gray), red regions are the intermediate regions to fold / bind off.*

8.2 Binding Fabric

In this work, we primarily assume that the binding of garment panels should be done continuously in a direct manner (i.e., by following the connectivity from the stitch graph). This provides a very simple and intuitive support for *darts*, which our system simply considers as direct links from one side of the fabric to another (no fabric is actually cut or folded). However, cut & sew supports various other means of manually binding pieces of fabric together, including pleats, ruffles, zippers, or other non-manifold bindings of multiple fabric layers together.

Of those, *pleats* are likely the most amenable to automation after darts. The current workflow is theoretically able to deal with pleats at least partially: users can bind a larger interface to a small one by splitting the large one into pairs of linked and unlinked sections that cover the binding of the smaller one (see Fig. 18). With dedicated schedules or knitting procedures, one may be able to automate the folded binding of the intermediate regions. A partially manual solution is to bind off the intermediate section, which the user can then fold and link. However, in practice, the main difficulty is that large changes to the number of stitches without coordinated increases/decreases lead to excessive stitch rotations and result in yarn failure during manufacturing.

Another related issue is that of the fabric purpose. In our system, all sketch charts have the same purpose: composing the apparent garment shape. However, cut & sew includes various types of fabric panels, such as *lining* or *facing*. Each typically serves a distinct purpose such as to modify the fabric's appearance, structure, or rigidity. When interpreting a garment pattern purely from the shape perspective, our system would typically discard the additional fabric panels. By contrast, it would be ideal to account for their intended function using compatible weft-knitting techniques. For example, *inlay* interlocks thread in between wales without creating loops, which restricts the stretch of the fabric; similarly, *stitch patterns* can modify the appearance, texture, and tactile feel of the knit fabric. Ideally, those would all be customizable properties of the garment representation.

8.3 Sizing and 3D Preview

Our stitch sampling strategy makes the simplifying assumption that the number of stitches along courses and wales are sufficient to describe the garment size through two constants D_{crs} and D_{wale}. This is a gross approximation and does not account for the impact of the underlying garment curvature or the impact of different stitch operations and surface textures. From a design perspective, we are missing two components: (1) a more accurate simulation of the size, which would inform the sampling strategy (e.g., through fast simulation [Leaf et al. 2018; Wang 2018]), and (2) a means to adjust the desired size along specific target curves directly (either by optimizing the sketch or the stitch graph), rather than searching for it iteratively as in the current workflow.

Finally, our system only tackles the intrinsic aspect of knitted fabric, whereas a full system would benefit from a full 3D garment preview. Flattened shape editing requires a deep understanding of the traditional cut & sew workflow and an intuition for how local pattern editing influences the final shape. A clear next step is to provide an interactive 3D preview and manipulation alongside the 2D pattern editing capabilities, as is already common in professional garment authoring software [Clo3D 2020; MarvelousDesigner 2020].

8.4 Finishing and Local Stitch Control

While this work is the first to provide explicit visual control over the seams induced by the weft knitting process, various other artifacts are important to consider. Local knitting procedures can have a large impact on the final appearance or physical properties of the knitted garment. For example, we found that the type of increase stitch dramatically impacts the appearance of the irregular stitches (see supplementary document). Thus we allow the user to select from different options. Similarly, *binding the yarn on and off* the needles can be done in various ways that change the tightness and appeal of the garment boundary edges. In general, this calls for a more general framework that can explore those customization capabilities intuitively and possibly select them locally given functional specifications from the user (e.g., yarn looseness, tension).

Other important classes of details include local stitch patterns and colorwork. Although we show a proof of concept that our sampled stitch graph can be used to do patterning without requiring local editing of each stitch, our current solution is far from intuitive for non-technical users. We envision that these patterns could be designed graphically by superimposing layers on top of the garment sketches, while using image stencils or stitch patterns from libraries [Donohue 2015; Hofmann et al. 2019; Kaspar et al. 2019b] to induce the stitch-level operations.

9 CONCLUSION

We presented a novel workflow to design garments to be knitted on industrial weft knitting machines. This permits the design of new garments from scratch and provides an initial method for adapting the plethora of existing garment patterns from the traditional cut & sew workflow. We envision that our system could be integrated as part of the flattened pattern editor in existing garment authoring software, thus enabling a completely digital whole-garment knitting workflow from an initial sketch to the machine instructions. To facilitate this vision, we will make available an open-source version of our system's prototype. We hope that this will support and inspire future avenues of research, such as dedicated user-guided schedulers and more intuitive customization capabilities for colorwork and stitch patterns.

ACKNOWLEDGMENTS

We are grateful to James McCann and his group at Carnegie Mellon University for making the necessary tools to process Knitout files available. We thank Kelly Lam for her initial segmentation work, Tom Buehler for the video composition, Ed Chien and David Palmer for the early geometry discussions, Paul Zhang for the later discussion on geometry terminology, Nicholas Sharp for making Geometry Central available, Timothy Erps and Mike Foshey for the general lab maintenance, and Buttercup Foshey for the moral and inspirational support.

REFERENCES

Lea Albaugh, Scott Hudson, and Lining Yao. 2019. Digital Fabrication of Soft Actuated Objects by Machine Knitting *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13.

Aric Bartle, Alla Sheffer, Vladimir G. Kim, Danny M. Kaufman, Nicholas Vining, and Floraine Berthouzoz. 2016. Physics-Driven Pattern Adjustment for Direct 3D Garment Editing. *ACM Trans. Graph.* 35, 4, Article 50 (July 2016), 11 pages.

Floraine Berthouzoz, Akash Garg, Danny M. Kaufman, Eitan Grinspun, and Maneesh Agrawala. 2013. Parsing Sewing Patterns into 3D Garments. *ACM Trans. Graph.* 32, 4, Article 85 (July 2013), 12 pages.

Clo3D. 2020. Clo3D. [Online]. Available from: https://www.clo3d.com.

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

Nanette Donohue. 2015. 750 Knitting Stitches: The Ultimate Knit Stitch Bible.

Peng Guan, Loretta Reiss, David A. Hirshberg, Alexander Weiss, and Michael J. Black. 2012. DRAPE: DRessing Any PErson. *ACM Trans. Graph.* 31, 4, Article 35 (July 2012), 10 pages.

Min-Woo Han and Sung-Hoon Ahn. 2017. Blooming Knit Flowers: Loop-Linked Soft Morphing Structures for Soft Robotics. *Advanced Materials* 29, 13 (2017), 1606580.

Megan Hofmann, Lea Albaugh, Ticha Sethapakadi, Jessica Hodgins, Scott E Hudson, James McCann, and Jennifer Mankoff. 2019. 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*. 5–16.

Ping Huang, Junfeng Yao, and Hengheng Zhao. 2016. Automatic realistic 3D garment generation based on two images. In *2016 International Conference on Virtual Reality and Visualization (ICVRV)*. IEEE, 250–257.

Takeo Igarashi and John F. Hughes. 2003. Clothing Manipulation. *ACM Trans. Graph.* 22, 3 (July 2003), 697.

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

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008b. Knitty: 3D Modeling of Knitted Animals with a Production Assistant Interface.. In *Eurographics (Short Papers)*. The Eurographics Association, Aire-la-Ville, Switzerland, 17–20.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-Aligned Meshes. *ACM Trans. Graph.* 34, 6, Article 189 (Oct. 2015), 15 pages.

Steven G Johnson. 2014. The NLopt nonlinear-optimization package. http://github.com/stevengj/nlopt

Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. 2019a. 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*. 53–65.

Alexandre Kaspar, Tae-Hyun Oh, Liane Makatura, Petr Kellnhofer, and Wojciech Matusik. 2019b. Neural inverse knitting: from images to manufacturing instructions. In *International Conference on Machine Learning*. PMLR, 3272–3281.

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

Minchen Li, Alla Sheffer, Eitan Grinspun, and Nicholas Vining. 2018. Foldsketch: Enriching Garments with Physically Reproducible Folds. *ACM Trans. Graph.* 37, 4, Article 133 (July 2018), 13 pages.

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

Yiyue Luo, Kui Wu, Tomás Palacios, and Wojciech Matusik. 2021. KnitUI: Fabricating Interactive and Sensing Textiles with Machine Knitting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan.) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–12.

MarvelousDesigner. 2020. MarvelousDesigner. [Online]. Available from: https://www.marvelousdesigner.com.

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

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

Juan Montes, Bernhard Thomaszewski, Sudhir Mudur, and Tiberiu Popa. 2020. Computational Design of Skintight Clothing. *ACM Trans. Graph.* 39, 4, Article 105 (July 2020), 12 pages.

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

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

Jifei Ou, Daniel Oran, Don Derek Haddad, Joseph Paradiso, and Hiroshi Ishii. 2019. SensorKnit: Architecting textile sensors with machine knitting. *3D Printing and Additive Manufacturing* 6, 1 (2019), 1–11.

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

Fabian Scheidt, Jifei Ou, Hiroshi Ishii, and Tobias Meisen. 2020. deepKnit: Learning-based Generation of Machine Knitting Code. *Procedia Manufacturing* 51 (2020), 485 – 492. 30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021).

Nicholas Sharp, Keenan Crane, et al. 2019. geometry-central. www.geometry-central.net.

Yu Shen, Junbang Liang, and Ming C Lin. 2020. GAN-based Garment Generation Using Sewing Pattern Images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, Vol. 1. 3.

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

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

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

Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. 2005. Fast Exact and Approximate Geodesics on Meshes. *ACM Trans. Graph.* 24, 3 (July 2005), 553–560.

E. Turquin, J. Wither, L. Boissieux, M. Cani, and J. F. Hughes. 2007. A Sketch-Based Interface for Clothing Virtual Characters. *IEEE Computer Graphics and Applications* 27, 1 (2007), 72–81.

Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. 2011. Sensitive Couture for Interactive Garment Modeling and Editing. *ACM Trans. Graph.* 30, 4, Article 90 (July 2011), 12 pages.

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

Richard Vallett, Ryan Young, Chelsea Knittel, Youngmoo Kim, and Genevieve Dion. 2016. Development of a Carbon Fiber Knitted Capacitive Touch Sensor. *MRS Advances* 1, 38 (2016), 2641–2651.

Pascal Volino, Frederic Cordier, and Nadia Magnenat-Thalmann. 2005. From early virtual garment simulation to interactive fashion design. *Computer-Aided Design* 37, 6 (2005), 593 – 608. CAD Methods in Garment Design.

Charlie C. L. Wang, Yu Wang, and Matthew M. F. Yuen. 2005. Design Automation for Customized Apparel Products. *Comput. Aided Des.* 37, 7 (June 2005), 675–691.

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

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

Irmandy Wicaksono, Carson I Tucker, Tao Sun, Cesar A Guerrero, Clare Liu, Wesley M Woo, Eric J Pence, and Canan Dagdeviren. 2020. A tailored, electronic textile conformable suit for large-scale spatiotemporal physiological sensing in vivo. *npj Flexible Electronics* 4, 1 (2020), 1–13.

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

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

Kui Wu, Marco Tarini, Cem Yuksel, James Mccann, and Xifeng Gao. 2021. Wearable 3D Machine Knitting: Automatic Generation of Shaped Knit Sheets to Cover Real-World Objects. *IEEE Transactions on Visualization and Computer Graphics* (2021), 1–1.

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