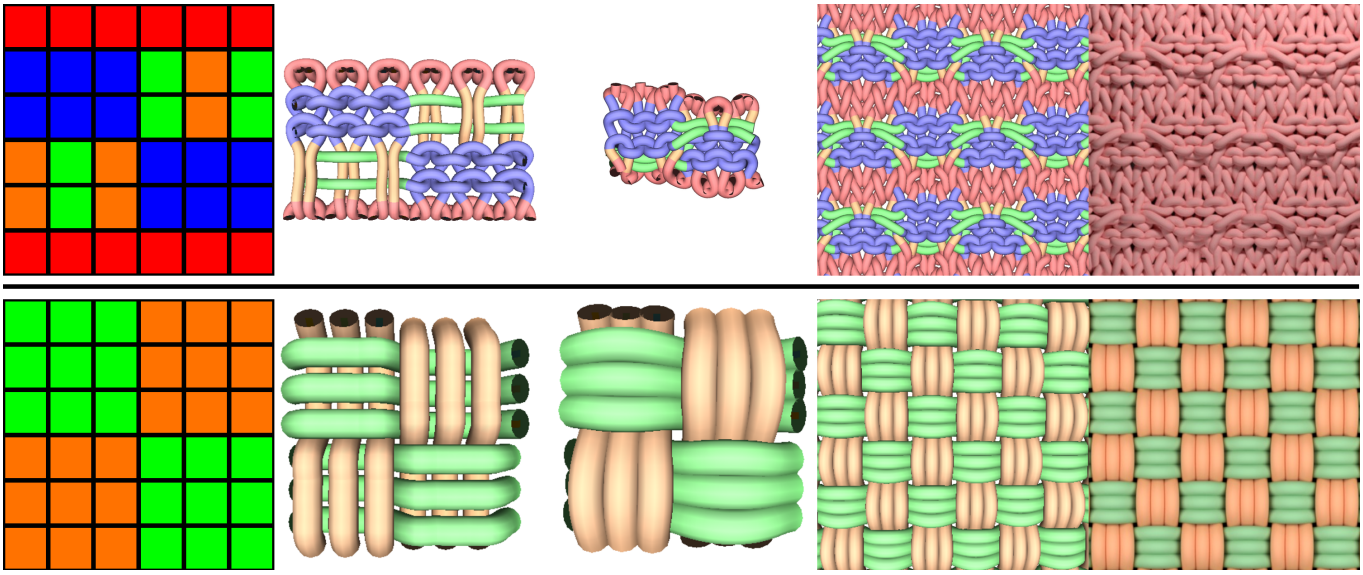
Interactive Design of Periodic Yarn-Level Cloth Patterns

JONATHAN LEAF, Stanford University, USA
RUNDONG WU, Cornell University, USA
ESTON SCHWEICKART, Cornell University, USA
DOUG L. JAMES, Stanford University, USA
STEVE MARSCHNER, Cornell University, USA



(a) Pattern (b) Periodic Simulation (c) Parameter Adjustment (d) Tiled Pattern (e) Final Render

Fig. 1. **Interactive simulation-based design of yarn-level cloth patterns:** The user can (a) draw a color-coded knit (Top) or woven (Bottom) pattern, (b) quickly simulate it by leveraging both periodic boundary conditions and a fast GPU solver, (c) interactively adjust model parameters for immediate design feedback, then (d) tile the pattern for (e) final rendering and texturing.

We describe an interactive design tool for authoring, simulating, and adjusting yarn-level patterns for knitted and woven cloth. To achieve interactive performance for notoriously slow yarn-level simulations, we propose two acceleration schemes: (a) yarn-level periodic boundary conditions that enable the restricted simulation of only small periodic patches, thereby exploiting the spatial repetition of many cloth patterns in cardinal directions, and (b) a highly parallel GPU solver for efficient yarn-level simulation of the small patch. Our system supports interactive pattern editing and simulation, and runtime modification of parameters. To adjust the amount of material used (yarn take-up) we support "on the fly" modification of (a) local yarn rest-length adjustments for pattern specific edits, e.g., to tighten slip stitches, and (b) global yarn length by way of a novel yarn-radius similarity transformation. We demonstrate the tool's ability to support interactive modeling, by novice users, of a wide variety of yarn-level knit and woven patterns. Finally, to validate our approach, we compare dozens of generated patterns against reference images of actual woven or knitted cloth samples, and we release this corpus of digital patterns and simulated models as a public dataset to support future comparisons.

CCS Concepts: • **Computing methodologies** → *Physical simulation*;

Additional Key Words and Phrases: Cloth modeling, yarn-level cloth, physics-based simulation, knitted, woven, interactive design

ACM Reference Format:
Jonathan Leaf, Rundong Wu, Eston Schweickart, Doug L. James, and Steve Marschner. 2018. Interactive Design of Periodic Yarn-Level Cloth Patterns. *ACM Trans. Graph.* 37, 6, Article 202 (November 2018), 15 pages. https://doi.org/10.1145/3272127.3275105

Authors' addresses: Jonathan Leaf, Stanford University, Stanford, USA, jcleaf@stanford.edu; Rundong Wu, Cornell University, Ithaca, USA; Eston Schweickart, Cornell University, Ithaca, USA; Doug L. James, Stanford University, Stanford, USA, djames@stanford.edu; Steve Marschner, Cornell University, Ithaca, USA.

0730-0301/2018/11-ART202 $15.00
https://doi.org/10.1145/3272127.3275105

1 INTRODUCTION

Yarn-level cloth relaxation is a powerful tool for creating detailed fabric models for textile design and manufacturing, and visual texture generation in graphics. Cloth structure and deformations arise intrinsically from the contact-based interactions between many individual yarns. While textiles are highly complex, nearly all fabric is made up of yarn-level patterns based on repeated sections of cloth. Yarn-level cloth patterns can produce a stunning range of 3D shapes and appearances for knit and woven fabrics, and these results are often far from obvious from the input 2D patterns. Traditionally these patterns are devised by trial and error, but this time-consuming process could be greatly reduced with good simulations. With a simulation available, one can explore and design a wide range of possible cloth shapes and textures by adjusting input patterns and using a physics-based relaxation of the input yarn curves to help understand the unpredictable final shape of the cloth. Unfortunately, yarn-level simulations to date have been extremely slow, making them unusable for interactive yarn-level pattern design.

In this paper, we propose techniques that enable interactive yarn-level pattern design for knit and woven cloth. In our system, the user can simply draw an input pattern, and in seconds see the resulting cloth shape appear during relaxation. The speed of our system is enabled by two acceleration schemes for yarn-level pattern relaxation:

(1) *GPU Relaxation:* We greatly improve relaxation speed by developing a GPU implementation of yarn-level cloth, and can achieve a 20× speed-up over our CPU implementation.
(2) *Periodic BCs:* We exploit the spatial redundancy of many patterns by only simulating the smallest repeated yarn patch subject to *periodic boundary conditions* on yarn spline curves, with yarn-yarn collisions resolved properly between the patch's yarn curves and those of the virtual periodic yarns.

Together these two accelerations can enable orders-of-magnitude speed-ups over prior yarn-level simulation implementations, and enable interactive yarn-level pattern design.

Our prototype interactive design tool allows naive users to author periodic yarn-level patterns of knit and woven cloth using an intuitive drawing interface. Users can then adjust high-level material parameters during an interactive relaxation phase to explore the design space of possible pattern mechanics.

One important aspect we allow the user to control is how much yarn material is allocated to the pattern, and where, since yarn length is not known a priori—patterns essentially specify stitch topology but usually not yarn lengths or gauge. Since the amount of material in a pattern dramatically influences the look and shape of the pattern, we support "on the fly" modification of the amount of yarn used ("take-up") in several ways. First, to handle extreme material disparities generated by certain patterns (for knits, slip stitches generate a considerable disparity), we support local yarn rest-length adjustments for pattern specific edits, e.g., to tighten slip stitches. Second, to support efficient global scaling to pattern yarn length ("total take-up") we introduce a novel yarn-radius similarity transformation.

We demonstrate the tool's ability to support easy interactive modeling of a wide variety of yarn-level knit and woven patterns. Finally, to validate our approach, we compare dozens of generated patterns against reference images of actual woven or knitted cloth samples. We release this corpus of digital patterns and simulated models as a public dataset to support future comparisons.

2 RELATED WORK

2.1 Yarn-Level Cloth Simulation

Yarn-level cloth simulations, treating each thread of a fabric as a connected series of rods/splines and simulating them directly, is a relatively new area for computer graphics simulations. The seminal yarn-level cloth simulation model was proposed in [Kaldor et al. 2008]. Kaldor's work established using a reduced set of degrees of freedom to create an energy based model that allowed many realistic deformations to come naturally from the simulation. This model excels in creating smooth deformations for yarn curves, and enforcing the necessary constraints to mimic real clothing. The primary downside to this approach is computational cost, with contact-related computation being the primary bottleneck. To tackle this issue, Kaldor et al. [2010] developed clever mathematical techniques to improve the performance of the contact-related bottleneck, by linearizing contact matrices until an error threshold is violated. This method was able to improve the performance by 8x over its predecessor, but is still far from interactive. Also the key features of adaptive contact linearization, and nonzero twist angles, were disabled during relaxation. For woven cloth, Cirio et al. [2014] used an approach to discretizing interlacing yarns based on crossing and sliding, which models contacts between yarns implicitly and avoid the high computational cost. For knits, Cirio et al. [2017] handles the contact-related computation cost by creating a reduced order model for contacts. Each contact is treated as a node and yarn segments become edges in a graph. They developed reduced-order dynamics model on this graph to speed up yarn level computation. Jiang et al. [2017] created a general simulation framework for handling frictional contacts by implementing an Eulerian-Lagrangian hybridization of the Material Point Method. This method, while focused on traditional cloth, was also demonstrated to simulate yarn-level cloth. This method shows generality by supporting yarn-level cloth as well as other models, and it is a promising look at improving the computational cost of contacts. Recently, Fei et al. [2018] have handled yarn-level cloth interactions with liquids.

In textile engineering, periodic boundary conditions have been used to measure 3D elasticity using traction interfaces from a simple regular pattern [Dinh et al. 2018], or to simulate micro-geometry of woven textiles [Huang et al. 2013]. Our method uses periodic boundary conditions across splines and supports a considerably broader range of patterns and deformations.

2.2 Yarn-Level Cloth Modeling

Researchers have taken a number of approaches to generating yarn-level cloth curves. For knitting, one approach is to replicate manufacturing processes used by knitting machines [Eberhardt et al. 2000] and [Duhovic and Bhattacharyya 2006]. Although these methods produce topologically correct results, they are slow, and therefore impractical for a design tool.

Several works from the textile community generate knit geometry using spline curves and focus on the plain knit stitch [Choi and Lo 2003, 2006; Demiroz and Dias 2000; Göktepe and Harlock 2002; Renkens and Kyosev 2011]. Still others focus on modeling pattern cells [Kurbak 2009; Kurbak and Alpyildiz 2008; Kurbak and Soydan 2009], which can be challenging to combine due to inconsistencies along the boundaries. Yarn and fiber level modeling are also used to simulate the deformation of woven textiles [Huang et al. 2013; Lin et al. 2008; Miao et al. 2008; Sherburn 2007]. For example, TexGen [Sherburn 2007] has been used to create geometric models for woven composites, as well as predict deformation and mechanical properties of the textiles using the finite element method. These works mainly focus on studying the mechanical properties of common woven patterns rather than encouraging users to explore the design space interactively.

Igarashi et al. [2008a] introduced semi-automatic techniques for generating knit models from an input 3D shape. A user interface for authoring knitted yarn topologies from 3D meshes was introduced in [Yuksel et al. 2012]. They define "Stitch Meshes," a mesh-based representation of knitted yarn topologies. Each polygon on the mesh surface contains curve geometry for a single stitch that ensures correct topology. Their work allows users to edit the mesh directly at the level of each individual stitch and synthesize topologically correct yarn curves that are relaxed in an offline simulator to get the correct yarn geometry. More recently, [Wu et al. 2018] have extended stitch meshes to enable automatic yarn topology generation on meshes. Connecting to manufacturing, [Narayanan et al. 2018] developed a method for generating automatic knitting machine instructions from a 3D mesh. One downside of these approaches is that through this representation it is unclear how much yarn material must be allocated for each individual stitch. As we will later discuss, the effects of yarn radius and yarn allocation dominate the relaxation result, and these parameters must be carefully calibrated.

Our yarn generation methodology is most similar to [Kaldor 2011], who developed a semi-automated process of tiling interconnected loops together to form yarn-level cloth. In addition, we adopt the tile abstractions from [Yuksel et al. 2012] to easily enforce consistent topological consistency across tile boundaries. Because our method requires rectangular repeatable patterns, we limit the set of tiles to be repeatable blocks. This methodology ensures users have creative freedom, while always generating topologically valid yarn curves.

2.3 Interactive Design of Cloth Patterns

Graphics research has a long history of developing user interfaces for visualizing and/or modifying cloth structures. Berthouzoz et al. [2013] convert sewing patterns into 3D models for visualization. Bartle et al. [2016] uses cloth simulations to drive the dynamics of the cloth, while allowing a user to edit a simplified representation of the garment. Each of these approaches attempts to solve the complexity of designing macroscopic garment-level behavior. Igarashi et al. [2008b] introduced "Knitty," a sketch-based technique for designing plush toys. McCann et al. [2016] proposed a method to generate knitting instructions for knitting 3D models designed by their custom interface, using primitives such as tubes and sheets. Recently, Wang [2018] have developed improved optimization methods combining cloth simulation for large garment design.

Textile manufacturers use a set of complex CAD tools for designing cloth. Industrial grade knitting machines, such as those from Shima Seiki or Stoll, come with user interfaces that show a color map, where each color corresponds to a stitch instruction for the machine. These maps can be visualized as non-physical demonstrations of the yarn texture, which can lead to highly distorted visualizations. We also provide a color-based interface that will be familiar to designers, but our system previews the texture of the cloth interactively using a full simulation of the yarn-level mechanics.

The ability to visualize physical aspects of a pattern is important to the knitting community. At "stitch-maps.com", knitting instructions are converted to a set of icons, positioned to create a visualization of how the pattern is expected to be shaped. This representation allows users to get some intuition about the shape. Our method helps address this need by allowing users to generate and simulate patterns interactively for a more informative pattern visualization.

3 GPU SIMULATION OF YARN-LEVEL CLOTH

Our simulator is a GPU-based implementation of a yarn-level cloth model heavily influenced by Kaldor et al. [2008]. We use a similar physical model and integration techniques, with some key differences for performance reasons:

(1) Replaced inextensibility constraint projection with an inextensibility penalty energy to reduce solver complexity and improve parallelism.
(2) Used a Spatial Hashing technique for contact detection (instead of a BVH tree) for GPU performance.
(3) Use adaptive timestep restriction to avoid yarn pull-through while ensuring large steps for relaxation.

The implementation of the GPU solver involved specific considerations to ensure high-performance code. In §3.2 we describe how we handle contacts with spatial hashing, and use stencils to represent the computation of intrinsic energies to increase throughput.

3.1 Revised Yarn-Level Cloth Model

Our yarn-level cloth simulation model is based on Kaldor et al. [2008] with some modifications. We re-use equations 3-4, 6-10 which handle contact, bending, and length energies, as well as mass-proportional and collision damping terms; definitions and symbols imported from Kaldor are listed in Table 1, model details in Appendix B, and an illustration in Figure 2. Material parameters are set to mimic yarn behaviors of common knitted and woven material in our examples; for knits we use parameters in Table 1 of [Kaldor et al. 2008].

Allowing Yarn Stretch: One important difference for interactive design is that we replace the hard inextensibility constraints $C_i^{\text{len}} = 0$ on spline segments (which were solved using an ICD projection solve), with penalty terms that drive the spline length to its rest length, ℓ_i:

$$E_i^{\ell} \equiv k_{\ell} \left(C_i^{\text{len}}\right)^2 = k_{\ell}\left(1 - \frac{1}{\ell_i}\int_0^1 \|\mathbf{y}_i'(s)\| ds\right)^2, \qquad (1)$$

Table 1. **Symbols imported from [Kaldor et al. 2008]**

Quantity	Equation	Description
$\mathbf{q}$		Control-point positions
$\dot{\mathbf{q}}$		Control-point velocities
$\mathbf{y}_i(s)$		Position on i^{th} spline segment
$\mathbf{v}_i(s)$		Velocity on i^{th} spline segment
ℓ_i		Rest-length of i^{th} spline segment
$\kappa_i(s)$	4	Unsigned curvature of i^{th} spline segment
E_i^{bend}	4	Bending energy of i^{th} spline segment
C_i^{len}	5	Length constraint on i^{th} spline segment
E_i^{len}	6	Length constraint on interior of i^{th} spline segment
$E_{i,j}^{\text{contact}}$	7	Contact energy of (i, j) spline segments
D_i^{global}	9	Mass-proportional damping energy
$D_{i,j}^{\text{contact}}$	10	Contact damping of (i, j) spline segments

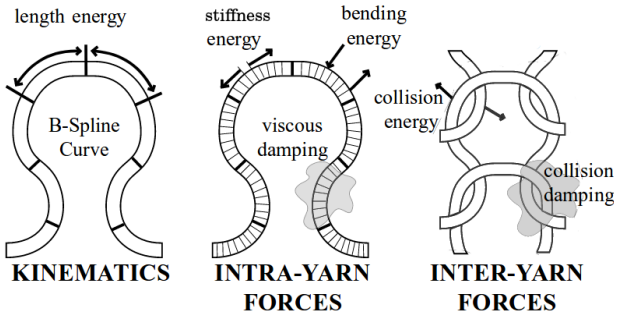


Fig. 2. **Yarn-level model summary:** Yarns are splines with energy terms handling each of the yarns behaviors. Internal forces resist bending, stretching, and intra-segment stretching, and external forces repel colliding splines. Damping forces are applied to yarns and contact points to ensure smooth motion. (Image is a modified version of Figure 4 of [Kaldor et al. 2008]).

where $\int_0^1 \|\mathbf{y}_i'(s)\|\, ds$ evaluates to the measured length of spline segment i. Note that setting k_ℓ to a sufficiently large value, e.g., 1000 $g\,cm^2/s^2$, can reduce length strain in our examples to within a few percent error without noticeable timestep restriction (even up to k_ℓ=10000), and thus achieves a similar effect as the constraint projection step of [Kaldor et al. 2008] without requiring a linear system solve. However, the main reasons we prefer to allow yarn stretch for interactive pattern design are: (a) the yarn lengths attributed to the default stitch curves are somewhat arbitrary, (b) in cases where not much yarn is allocated, hard inextensibility constraints (combined with periodic boundary conditions) can fight against contact constraints, making it harder to find more natural configurations and making the simulator less amenable to interactive design, and (c) we can always reduce excess yarn length later using techniques from §6.

Contact Sampling. Similar to Kaldor et al. [2008] we also use 11 integration samples per spline for all intrinsic energy gradients (∇E^{bend}, ∇E^{len}, ∇D^{global} and now ∇E^{ℓ}). For contact energy integrals ($\nabla E^{\text{contact}}$, $\nabla D^{\text{contact}}$), we opt for a variable number of contact samples per spline to improve performance. To ensure that splines have a sufficient number of samples, we require that the number of samples per spline N_S be odd[1], and that $N_S > \frac{L_{min}}{2r}$, where L_{min} is the spline of smallest length, and r is the yarn (contact-sphere) radius. Without this condition, splines could be insufficiently sampled and allow pull-through, i.e., where one yarn passes through another yarn.

Integration and Timestep Reduction. We use a semi-implicit Euler integrator with a small timestep ($\Delta t_{\text{default}} \approx 0.0004s$ for our default stiffness values), and we further restrict timestep sizes to avoid pull-through at yarn-yarn contacts. Specifically, we estimate the largest timestep value $\Delta t_i(s)$ for each spline segment i and contact sample s such that it does not move more than $\varepsilon\, r$ (a fraction $\varepsilon \in (0, 1]$ of the yarn radius r), then find the smallest Δt_i and use it as the global system timestep size if it is less than $\Delta t_{\text{default}}$. In Appendix A we derive the following per-sample timestep bound:

$$\Delta t_i(s) = \frac{-\|\mathbf{B}_i(s)\,\dot{\mathbf{q}}^n\|_2 + \sqrt{\|\mathbf{B}_i(s)\,\dot{\mathbf{q}}^n\|_2^2 + 4\varepsilon r\|\mathbf{B}_i(s)\,\mathbf{a}^n\|_2}}{2\|\mathbf{B}_i(s)\,\mathbf{a}^n\|_2} \tag{2}$$

where $\mathbf{B}_i(s)$ is the B-spline basis function matrix of the sample on spline i and contact sample s; $\dot{\mathbf{q}}^n$ are control-point velocities at the beginning of the timestep; $\mathbf{a}^n = \mathbf{M}^{-1}(-\nabla E + \mathbf{f})$ are control-point accelerations; E is the sum of model energies, and $\mathbf{f}$ are external forces.

As in Kaldor et al. [2008], $\mathbf{q}$ is defined as the vector of control-point positions, $\dot{\mathbf{q}}$ are control-point velocities, and $\mathbf{M}^{-1}$ is the lumped inverse mass matrix. However, the exact layout of these terms, as well as details relating to periodic boundary conditions (including variables $\mathbf{i_R}$, $\mathbf{i_V}$, $\mathbf{d}$) will be described later in §5. For reproducibility of our method, please see Algorithm 1.

3.2 GPU Implementation

Efficient Contact Force Evaluation. As discussed in Kaldor et al. [2010], the bottleneck of contact force and damping evaluation ($E_{i,j}^{\text{contact}}$ and $D_{i,j}^{\text{contact}}$) involves an all-pairs sphere-sphere overlap test involving contact spheres (of radius r) sampled along each spline segment at equispaced locations. Simply determining overlapping sphere-sphere pairs (from different segments) is a major task that occurs every tiny timestep, followed by the actual force evaluation for contacting pairs. In this work we use the GPU to accelerate contact pair finding (as we describe briefly), but since that is still slow due to the tens of thousands of timesteps, we use a space-time bound to cache nearby sphere-sphere pairs. Doing both of these things is key to rapid force evaluation for our interactive tool.

Contact-pair Detection using Spatial Hashing. Kaldor et al. [2010] found contact pairs using a BVH tree, however, for a GPU implementation, we use spatial hashing since it is particularly efficient for finding overlaps between spherical particles of identical radii [Macklin et al. 2014]. Our GPU spatial hashing is inspired by Green [2010], but avoids a global grid data structure for memory efficiency. The (gridCellID,sphereID) pairs are then hashed into a GPU-based multi-value Cuckoo hash table implementation [Harris et al. 2007], with hash-table collisions corresponding to sphere center locations. We then radially search by querying nearby grid cells for sphere IDs, and construct the necessary pairs. [Bell and Hoberock 2011].

[1] Simpson's Quadrature for integration requires odd sample counts.

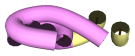
Algorithm 1: Simulation Algorithm

Input: Control points $\mathbf{q}$, Control Point Velocities $\dot{\mathbf{q}}$, Inverse Mass Matrix $\mathbf{M}^{-1}$, Yarn Radius r, Yarn Radius Fraction ε, Total Steps, Virtual Indices $\mathbf{i_V}$, Real Indices $\mathbf{i_R}$, Offset Vectors $\mathbf{d}$
Output: Control points $\mathbf{q}_f$, Control Point Velocities $\dot{\mathbf{q}}_f$
contactPairSet = SpatialHash(q)
MaxMov = 0
repeat
  if *MaxMov* $\geq \varepsilon r$ **then**
    contactPairSet = SpatialHash(**q**)
    MaxMov = 0;
  end
  $\nabla E^{\text{contact}}$ = contactEnergy(**contactPairSet**, **q**)
  $\nabla D^{\text{contact}}$ = contactDamping(**contactPairSet**, $\dot{\mathbf{q}}$)
  $\nabla E = \nabla E^{\text{bend}}(\mathbf{q}) + \nabla E^{\text{len}}(\mathbf{q}) + \nabla E^{\ell}(\mathbf{q}) + \nabla E^{\text{contact}} + \nabla D^{\text{contact}} + \nabla D^{\text{global}}(\dot{\mathbf{q}})$
  Δt = computeMaxValidTimestep($\dot{\mathbf{q}}^n, \nabla E, \mathbf{f}$)
  $\dot{\mathbf{q}}^{n+1} = \dot{\mathbf{q}} + \Delta t\ \mathbf{M}^{-1}\ (-\nabla E + \mathbf{f})$
  $\mathbf{q}^{n+1} = \mathbf{q} + \Delta t\ \dot{\mathbf{q}}^{n+1}$
  $\mathbf{y}$ = computeSampleMovement($\dot{\mathbf{q}}^{n+1}, \Delta t$);
  MaxMov += $\|\mathbf{y}\|_\infty$
  q, $\dot{\mathbf{q}}$ = PeriodicBoundaryConditions($\mathbf{q}^{n+1}$, $\dot{\mathbf{q}}^{n+1}$, $\mathbf{i_V}$, $\mathbf{i_R}$, **d**)
  Step++
until *Step* $\geq$ *Total Steps*;
$\mathbf{q}_f = \mathbf{q}$
$\dot{\mathbf{q}}_f = \dot{\mathbf{q}}$

Amortized Estimation of Contact Pairs. GPU hashing is fast, but because of numerous timesteps it is the bottleneck. Inspired by Kaldor et al. [2010], we use a simple space-time bound (not contact linearization) to exploit the fact that potentially colliding contact pairs remain nearby for many timesteps. The candidate set of collision pairs is kept around as state (contactPairSet in Algorithm 1), and is used for contact force evaluation at subsequent steps. In a similar fashion, our method tracks the maximum sample movement at each step. If the sum of all maximum movements is below a threshold, the contactPairSet is retained; otherwise it is regenerated. The threshold is set to be a fraction of the contact radius, to ensure we re-evaluate contacts before pull-through can occur. Using this method, we can amortize the cost of contact-pair finding across several timesteps (e.g., 5) to remove the bottleneck.

Compute Stencils for Parallel Evaluation of Internal Forces. Each intrinsic energy term—bending, stiffness, length—is defined for each spline segment in terms of splines degrees of freedom. Consequently, the energy of each spline segment will influence the forces (energy gradients) at its four control points. Since adjacent splines share control points, if we parallelize force contributions across spline segments there is a write-after-write scheduling conflict. To avoid this issue, we order the computation instead by the output point, reading the spline data points that contribute to a single control point's energy gradient and computing all the force terms affecting that point. Although there is some redundant computation using this structure, the control-point force evaluation workload becomes perfectly parallel across control points, with each operation requiring a total of 7 control point reads and 1 control point write. We also experimented with using atomic adds to handle write conflicts, to parallelize elements across splines rather than control points, but found the performance was worse. For example, on the Basketweave Rib Stitch pattern, atomicAdds caused a 30% slowdown in mean step time.[2]

Table 2. **Knit Tile Encodings:** Each knitting instruction (or stitch) is encoded by a "knit tile" that has a code, color, and associated spline curves.

Stitch	Code	Color	Visualization
Knit	K		
Purl	P		
Front Stitch	F		
Back Stitch	B		
Yarnover(Y) + Slip Slip Knit	YS		
Knit 2 Together + Y	2U		
Y + Slip Knit Two Pass + Y	Y3Y		

4 PATTERN REPRESENTATION

We define patterns as repeated rectangular sections of knit or woven cloth. All patterns must be designed so that they behave as intended when tiled edge to edge, and the simulation must ensure consistency across the boundaries.

Woven cloth patterns are traditionally encoded using a matrix of 1s and 0s. Each weft yarn is assigned a row, and each warp yarn is assigned a column. A matrix value of $W_{i,j} = 1$ indicates that the ith weft yarn must be above the jth warp yarn. If $Wi, j = 0$, then the warp yarn is placed above the weft yarn. Since all yarns in a periodic weave connect to repeats of themselves, the boundaries are well-defined.

We represent a pattern in knitted cloth as a rectangular grid of tiles/blocks that encodes the pattern's topology. To ensure that patterns are always repeatable, we require that all tiles must also be rectangularly tile-able. A lexicon of knit tiles, with their corresponding geometry, is listed in Table 2. Each tile is identified using a set of 1 to 3 ASCII characters, where the number of loops along that tile corresponds to the number of ASCII characters present. Also, character strings are defined as row-based blocks. Therefore, stacking "Y3Y" into a column would not be a valid configuration.

Using this pattern description language, pattern files mimic the spatial layout of the pattern, and show up as rectangular patterns of text (see Figure 3). Creating a repeatable pattern then becomes

[2]Experiment using CUDA 8.

a packing problem of tiling blocks into a rectangular shape. By constructing patterns using this representation, we guarantee that any tiling will produce a valid knitted pattern that can be simulated using periodic boundary conditions.

$$\begin{bmatrix} K & K & K & K & K & K \\ P & P & P & F & B & F \\ P & P & P & F & B & F \\ B & F & B & P & P & P \\ B & F & B & P & P & P \\ K & K & K & K & K & K \end{bmatrix} \Longleftrightarrow$$

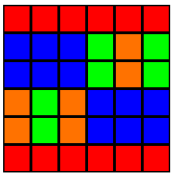


Fig. 3. **Text and Color Representations of a Knit Pattern:** (same pattern as in Figure 1 (Top))

5 PERIODIC BOUNDARY CONDITIONS

Since patterns in our system are always periodic, the behavior of the simulation should also be periodic—the simulation of one copy of the pattern should show the behavior that would be seen if the model were surrounded by identical tiled copies, all relaxing in the same way. Our approach to enforcing periodicity is to ensure that all control points in the simulation experience exactly the same yarn and collision forces that they would in the periodic case.

To achieve this, our method creates a set of *virtual* spline segments, which are offset versions of their corresponding *real* spline segments and provide the neighborhood that is needed to simulate points near the boundary in the same way as interior points. After each step of the simulation, each real control-point's position is shifted by the correct period and overwrites the virtual control-point's position. The real control-point's velocity directly overwrites the virtual control-point's velocity. To do this we (1) append virtual points to the rectangular pattern, (2) automatically identify real-virtual control-point pairs and displacement offsets, and (3) apply update rules for the simulation to efficiently enforce these boundary conditions.

First, using our tiling system we generate one complete set of yarn curves from the pattern. We then extract 8 unique sub-patterns: the 4 long edges, and each of the 4 corners. For a pattern to be periodic, each edge tile must be adjacent to the pattern tile present at a fixed distance on the opposite side of the pattern. For each subpattern, a virtual copy of the subpattern is placed opposite to the original and merged with the global model. A visualization of the result of this process is shown in Figure 4 for knits, where virtual degrees of freedom are visualized in gray. From this model we define $\mathbf{q}$ as a vector of its control point positions, and $\dot{\mathbf{q}}$ as a vector of its control point velocities, both of which include real and virtual (gray) degrees of freedom. It is important to note that each merge of a subpattern into the overall model changes the shape of $\mathbf{q}$ and $\dot{\mathbf{q}}$ by appending virtual degrees of freedom to the already present yarns.

Second, for every virtual control point, we record the real-virtual control-point pair and its tile-specific displacement offset. This offset is the local displacement with respect to the tile's reference frame, which we store for all control points in an array $\mathbf{d}$. The indices of the virtual and real control point pairs are stored in the arrays $\mathbf{i_V}$ and $\mathbf{i_R}$ respectively. These periodic BCs and virtual offsets are shown for a woven pattern in Figure 5.

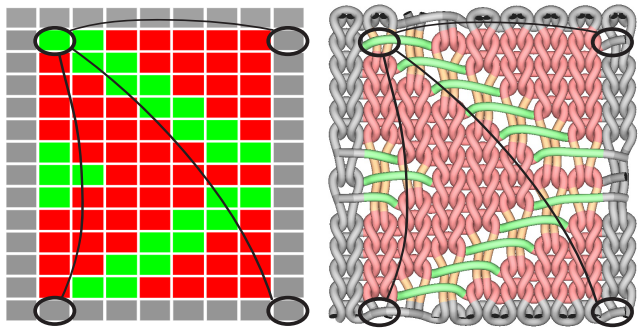


Fig. 4. **Knit Periodic Boundary Conditions:** (Left) Colored pattern tiles with additional gray border tiles representing virtual material; the correspondence between a real corner tile and virtual tiles are indicated. (Right) Yarn curves following relaxation, with gray portions corresponding to the virtual degrees of freedom that track with the corresponding real (non-gray) portion of the pattern. The correspondence between real and virtual control-point curves are indicated.

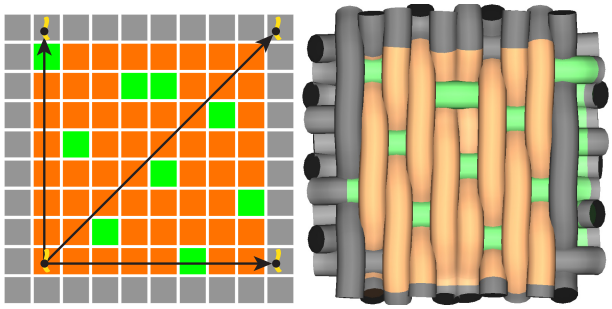


Fig. 5. **Woven Periodic Boundary Conditions:** (Left) Colored pattern tiles, with additional gray border tiles representing virtual material. (Right) Yarn curves following relaxation, with gray portions corresponding to the virtual degrees of freedom that track with the corresponding real (non-gray) portion of the pattern. Each yarn's virtual control-point tracks its corresponding real control point with a tile-specific displacement offset applied. Offset arrows are shown in (Left) for three virtual control points that map to a single real control point.

Finally, using $\mathbf{q}$, $\dot{\mathbf{q}}$, $\mathbf{i_V}$, $\mathbf{i_R}$, and $\mathbf{d}$, we can efficiently apply gather, scatter, and add operations standard to GPU libraries to efficiently update the virtual degrees of freedom after each step. See Algorithm 2 for details.

Discussion. It is critical to derive virtual points from entire tiles of the pattern, not just a few points at the ends of the real splines. This is because the virtual points immediately on the edge of the boundary should have the same energy environment as the real control points they correspond to so that the contribution to the adjacent real points is accurate.

6 ADJUSTING YARN LENGTHS

When relaxing a garment, it is unclear at first how much material to allocate for each tile and stitch, especially as the yarn take-up

Algorithm 2: Periodic Boundary Conditions

Input: Positions $\mathbf{q}$, Velocities $\dot{\mathbf{q}}$, Virtual Point Indices $\mathbf{i_V}$, Real Point Indices $\mathbf{i_R}$, Offset Vectors $\mathbf{d}$
Output: Updated Positions $\mathbf{q}$ and Velocities $\dot{\mathbf{q}}$

```
p = Gather(q, i_R)      // Get real-point positions (has duplicates)
u = Gather(q̇, i_R)      // Get real-point velocities (has duplicates)
p += d;                 // Displace real positions by virtual offset
q ← Scatter(p, i_V)     // Set virtual-point positions
q̇ ← Scatter(u, i_V)     // Set virtual-point velocities
```

of a given stitch varies dramatically from pattern to pattern. We introduce two ways to adjust yarn lengths: (1) local yarn rest-length adjustments for individual spline segments, e.g., to size slip stitches, and (2) global yarn-length adjustment by way of a yarn-radius scaling similarity transformation.

6.1 Rest-length scaling for stitch-level yarn control

Many patterns require local control on how much yarn is used, such as for patterns with *slip stitches* which may either be pulled tight or left as long strands. A good illustration of this issue is the difference between Honeycomb, Slip Stitch Rib, and Jacquard [3]. Each of these patterns has a different relationship between vertical and horizontal slip stitch material, which naturally comes from the pattern description itself. Because there are many ways to adjust these yarn-length parameters, they can produce a wide range of styles for a given pattern (as shown in our examples). Unfortunately the rectangular color-based tiling system encodes topology with default yarn lengths, since we do not know *a priori* how much material a designer would want to allocate for each stitch.

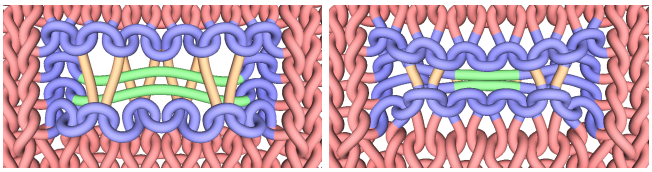


Fig. 6. **Rest-Length Scaling:** We demonstrate before and after reducing the length of slip stitches. Green and Orange yarns correspond to horizontal and vertical slip-stitch material, respectively.

To address this problem, our design tool allows interactive control of how much material is allocated along each horizontal or vertical slip stitch. In our implementation, we scale the amount of yarn allocated to a stitch, by allowing the user to scale the rest lengths, ℓ_i, of associated spline segments. Results are shown in Figure 6. We also adopt this technique for controlling the lengths of yarns in the warp and weft directions of woven cloth.

6.2 Radius scaling for global yarn-length adjustment

Sometimes we want to change the *total* amount of yarn used to make the pattern, i.e., its "take up," in order to loosen or tighten the structure. Because we want to make these changes in an interactive design tool setting, changes must be fast, and not significantly slow relaxation. One could scale the rest length of every yarn spline segment (as in §6.1) but the large position-level motions that result can slow relaxation. Alternately we could just add/remove spline segments to increase/decrease the overall yarn length, but that complicates interactivity and implementation, and also slows relaxation.

Instead, to globally reduce the pattern's yarn length from L' to L'/γ (assumingwithout loss of generality that $\gamma > 1$), we propose a radius-scaling transformation that artificially increases the yarn radius from the physical value, r, to γr. Mathematically, we associate our inflated-radius simulation model (with primed variables) with a smaller physical model (with unprimed variables) that has a shorter yarn length L (our goal) as a result of a uniform spatial scaling by $1/\gamma$ that scales control-point positions $\mathbf{q}$, spline lengths ℓ_i, etc.:

$$L = L'/\gamma, \quad \mathbf{q} = \mathbf{q}'/\gamma, \quad \ell_i = \ell_i'/\gamma. \tag{3}$$

The transformations are illustrated in Figure 7.

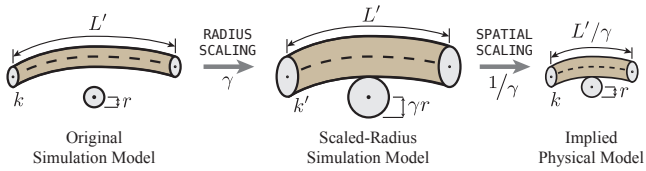


Fig. 7. **Radius-scaling transformation for yarn-length adjustment:** (Left) Given a simulation model with too much yarn and loose contacts, we can (Middle) tighten up the pattern by simulating a scaled-radius version with modified stiffness parameters ($k \to k'$), and thus efficiently simulate a pattern with reduced yarn length (Right) subsequently obtained by a post-simulation uniform spatial scaling.

For this parameter transformation to be physically consistent, the two primed/unprimed models should deform the same up to a spatial scaling. We enforce energy term equality by modifying the simulator's (primed) stiffness parameters as follows:

$$\begin{aligned} k'_{\text{bend}} &= \gamma\, k_{\text{bend}} & k'_{\text{contact}} &= \gamma^{-2}\, k_{\text{contact}} \\ k'_{\text{global}} &= \gamma^{-2}\, k_{\text{global}} & k'_{\text{dt, dn}} &= \gamma^{-4}\, k_{\text{dt,dn}} \\ k'_{\ell} &= k_{\ell} & k'_{\text{len}} &= k_{\text{len}}. \end{aligned} \tag{4}$$

For example, the bending stiffness must increase by γ to simulate a smaller physical model ($\ell = \ell'/\gamma$) with higher curvature ($\kappa = \gamma\kappa'$), if $E_i^{\text{bend}} = k_{\text{bend}}\ell_i \int_0^1 \kappa^2 ds = k'_{\text{bend}}\ell'_i \int_0^1 \kappa'^2 ds$. For a derivation of these modified scaling parameters, see Appendix D. Following relaxation, we can scale the simulated model by $1/\gamma$ to obtain shortened yarn geometry consistent with the original physical yarn gauge and stiffness parameters, k.

Discussion. This transformation has two favorable attributes in practice: (1) it changes the effective length of the yarn curves while keeping the number of spline segments constant, and (2) the yarn curves undergo relatively little motion (unlike for rest-length rescaling) and so established yarn contacts are maintained and continue relaxing efficiently. As a result, radius scaling provides a useful "knob" for interactive adjustment of how much material was used for a particular pattern. We later demonstrate that this transformation allows us to tune pattern take-up to better match reference images from Nalhcib [2018] (see Figure 8). We also experimented

[3] Visualizations may be found in supplemental material.

with using a length-scaling approach (where we jointly scaled the yarn rest lengths, periodic boundary offsets, and control point positions), and without including memory I/O computational cost of the length-scaling approach, it took twice as long on average to converge (see (5)) compared to our radius-scaling approach. Practically, the energy gradient norm struggles to settle because of the contention between the discontinuous periodic boundary constraint updates that shrink the perimeter, and our energy minimization that must deal with stiff contact energies. To accommodate for this, changes to the boundary conditions can be made with smaller steps, which require additional time to converge to the desired result.

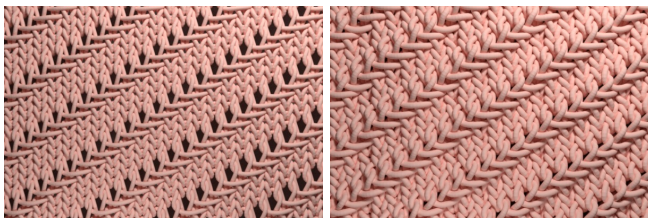


Fig. 8. **Radius-Scaling Transformation:** The "Right Diagonal" pattern (Left) is a loose knit with many visible holes ($\gamma = 1.075$), whereas (Right) reducing yarn take-up with radius scaling $\gamma = 1.5$ produces a tighter knit. This pattern updates interactively at approx. 710 timesteps/sec using our GPU relaxer with periodic BCs.

7 INTERACTIVE PATTERN DESIGN

The full range of possible yarn configurations is enormous. To allow directable exploration of this large space of patterns, we provide a pattern editor that allows users to author patterns, and a user interface for the simulation tool to drive yarn-level relaxation interactively and adjust design parameters, such as yarn lengths and pattern dimensions.

7.1 Pattern Editor

We provide image-based and text-based methods for describing patterns. In a separate window a user is displayed a grid-based image with a set of colors. Each color in the palette corresponds to a single ASCII character in the pattern representation described in Section 4. Using a fill-tool, users can specify grid cell colors from a provided palette to create a pattern. The number of colors in the palette for knits is 8, one for each character. The number of colors in the palette for single-layer wovens is 2, to encode the binary matrix representation.

Once a pattern file has been edited, the simulator can generate yarn curves with matching stitch topology, as shown for a knit pattern in Figure 9, and single- and double-layer woven patterns in Figure 10. This new model can then be simulated and modified via the controls described in Section 7.2.

7.2 User Interface for Yarn Relaxation Tool

Our tool provides an OpenGL visualization of the current pattern, and a number of user controls for driving the pattern exploration. These controls are roughly separated into three categories: simulation control parameters, visualization parameters, and pattern material allocation parameters. An example of the user interface is shown in Figure 11.

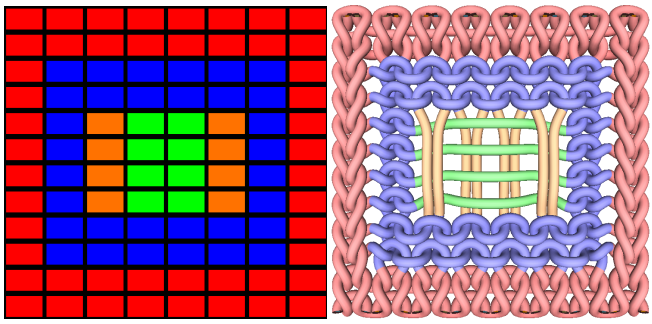
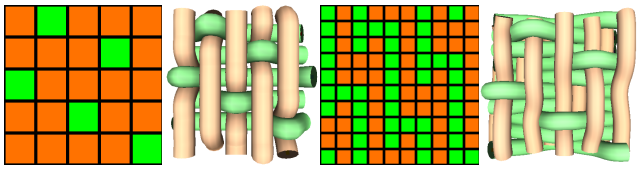


Fig. 9. **Pattern-Editor Paint Tool:** (Left) Knit patterns are represented using grid-based color images based on the color encoding of Table 2), and image edits will update the (Right) generated yarn-curve geometry with matching stitch topology.



(a) (b) (c) (d)

Fig. 10. **Woven pattern:** On the left we show a 2-3 satin (b) and its pattern (a). On the right we show a two layer cloth (d) and its pattern (c).

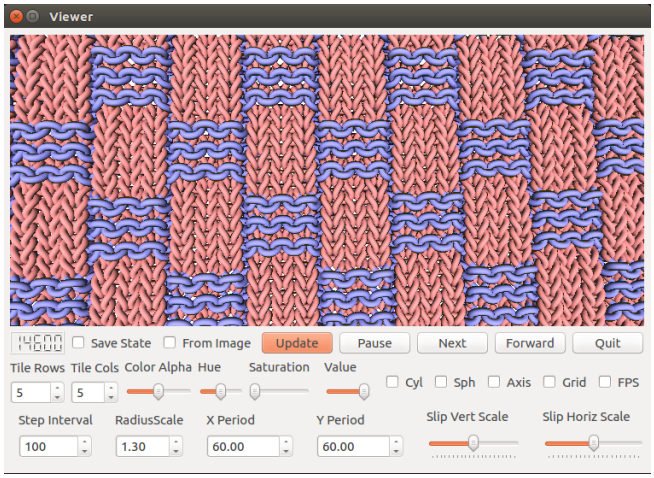


Fig. 11. **Yarn-Relaxation User Interface** provides controls for simulation, visualization, and pattern parameters: (First Row) timestep count, pattern I/O, and simulation timestepping; (Second Row) controls for visual appearance (horiz./vert. spatial tiling, pattern color, graphics styles); (Third Row) timesteps per frame, and pattern controls (radius scaling, spatial period, and rest-length scaling). The rendered pattern has been 5×5 tiled to show the pattern's overall appearance.

The simulation control parameters allow the userto import state from the image-based or text-based pattern descriptions described in

Section 7.1. The user is also able to stop, step, or start the relaxation process to fine tune other parameters dynamically.

The visualization parameters we provide include tiling, pattern based coloring, and yarn coloring. Users can choose a number of rows or columns to repeat the pattern in the rendering. Tiling a pattern involves copying the generated geometry in each of the cardinal directions, and doesn't involve any simulation. We allow users to use a number of color sliders to change how strongly the pattern coloring is shown compared to the HSV color that the user desires for the yarn aesthetic.

The material allocation parameters we provide are listed as follows: radius transformation via γ, x repeat distance, y repeat distance, and slip stitch material scaling parameters in the x and y directions. When a user sets γ, the simulator gradually updates γ (as defined in §6.2) until the user-set value for γ is reached. Both parameters that set the x and y repeat distances are provided to allow the user to stretch or compress the current pattern. Once the user sets the desired repeat period, the simulation smoothly changes the repeat distance of the periodic boundary conditions in the simulation to match the desired repeats set by the user.

The slip stitch material scaling parameters allow users to affect how much material is allocated for slip stitches in knitting patterns. Specifically, a slider is provided that will either shrink or increase the material at each interval. The lengths are restricted to be larger than 10% of their original length, and less than 200% of their original length. These sliders are also used to modify the warp and weft yarn lengths of woven cloth independently.

8 RESULTS

We have modeled and simulated a large corpus of yarn-level patterns that are listed in Table 3, along with performance timings for GPU-based relaxation. Histograms of simulation costs for knit and woven patterns are shown in Figure 12 for all patterns in the dataset. In addition to the results shown here, please see the supplemental document which shows all patterns, and reference photographs for comparison. Please also see our supplemental video for interactive pattern design sessions, relaxation animations, and other supporting footage.

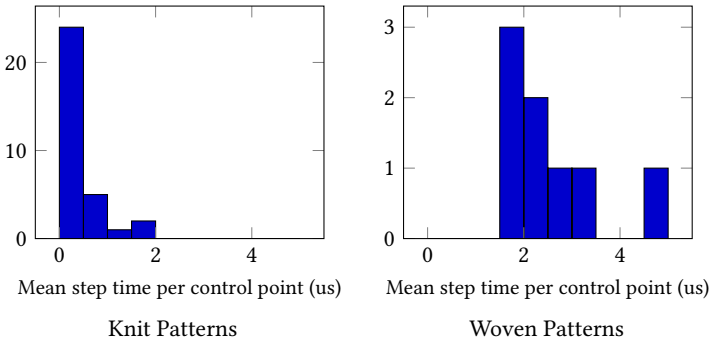


Fig. 12. **Histograms of Simulation Cost per Control Point:** Histograms showing distribution of mean wall-clock time per control point (per timestep) of (Left) knit patterns, and (Right) woven patterns from Table 3. Woven patterns were more expensive on average due to, e.g., many adjacent parallel yarns in contact.

Table 3. **Pattern Statistics and Relaxation Performance:** For each pattern, we run the relaxation to convergence as defined in (5). **Type** indicates either a knit (K) or woven (W) pattern. $T_{Converge}$ is the time to converge from the initial pattern state. T_{Step} is the average time to compute a single step of the simulation. $T_{ControlPoint}$ is the average step time per control point. Lastly, we report the number of control points simulated in each pattern.

Pattern	$T_{Converge}$ [s]	T_{Step} [ms]	$T_{ControlPoint}$ [us]	Type	Control Points
Alternating Diagonals	2.18	0.95	0.25	K	3794
Basketweave Rib Stitch	0.79	0.58	0.34	K	1714
Block1	0.09	1.17	0.20	K	5686
Block2	0.06	0.78	0.38	K	2054
Cartridge Belt Rib	0.45	0.44	1.03	K	423
Double Basket Pattern	3.00	0.88	0.18	K	4860
Garter Block	0.13	0.61	0.30	K	2054
Honeycomb	0.33	0.77	0.23	K	3392
Jacquard1	0.38	0.50	0.33	K	1504
Jacquard2	0.18	0.54	0.38	K	1433
Jacquard3	0.29	0.59	0.28	K	2106
King Charles Brocade	1.36	0.58	0.24	K	2394
Lattice With Seed Stitch	0.55	1.12	0.15	K	7280
Left Diagonal	0.05	0.50	0.70	K	723
Left Diagonal Rib	0.68	0.58	0.42	K	1374
Long Slip Textured	0.10	0.42	0.93	K	456
Parallelograms	0.19	0.64	0.24	K	2732
Pie Crust Basketweave	0.15	0.65	0.65	K	986
Pyramid Stitch	0.68	0.58	0.26	K	2198
Rhombus Texture	1.30	0.60	0.25	K	2394
Right Diagonal	0.37	0.44	0.61	K	723
Right Diagonal Rib	0.45	0.55	0.40	K	1374
Seersucker	1.09	0.49	0.40	K	1230
Slip Stitch Honeycomb	0.27	0.43	1.50	K	284
Slip Stitch Rib	0.43	0.41	1.64	K	248
Stockinette And Garter	26.54	1.52	0.14	K	10890
Thunderbird	10.91	1.08	0.15	K	7280
Tumbling Moss Blocks	0.02	1.14	0.35	K	3214
Woven Transverse Herringbone	0.61	0.59	0.29	K	1996
Basket 2x2	0.09	0.49	2.70	W	180
Basket 3x3	0.11	0.50	1.64	W	304
Plain	0.16	0.42	4.81	W	88
Satin 2x3	0.25	0.48	2.02	W	238
Satin 3x2	0.23	0.44	1.85	W	238
Twill 2x1	0.19	0.44	3.37	W	130
Twill 3x2	0.12	0.51	2.14	W	238
Twill 4x1	0.38	0.45	1.90	W	238

Implementation Details. All pattern simulations were run on a single NVIDIA GeForce GTX TITAN X (see specifications in Table 4), on a workstation with a 4-core Intel Xeon CPU E5-2637 v3 @ 3.50GHz processor. Our implementation is written in C++ and CUDA. To simplify writing CUDA code for the energy gradient computation, we created a pipeline for generating CUDA code directly from Mathematica expressions; see Appendix C for details.

Table 4. GPU Specifications

Specification	Value
GPU Clock	1.08Ghz
GPU Memory	12 GB
CUDA Cores	3072
L2 Cache Size	3 MB
Constant Memory	65MB

Convergence. When profiling our system's performance on a variety of patterns, we opted for a convergence term that would be

agnostic to the individual pattern. Specifically, our model has converged at step i if the following inequalities are satisfied:

$$\frac{\|\nabla E_i\|}{\|\nabla E_0\|} \leq \epsilon_1 \qquad (5)$$
$$\left|\frac{\|\nabla E_i\|}{\|\nabla E_{i-1}\|} - 1\right| \leq \epsilon_2$$

Here ∇E_i is the energy gradient at step i, and ∇E_0 is the energy gradient at step 0. For our results, we used $\epsilon_1 = 0.1$ and $\epsilon_2 = 10^{-5}$.

Performance Breakdown. For each of the patterns listed in Table 3 we broke down the performance into three categories: energy gradient computation, contact detection, and integration/timestepping. Over all patterns, the average percent of time spent in each of these sections are 53.5% for gradient computation, 14.5% for contact detection, and 31.5% for integration/timestepping. Because our method amortizes the cost of contact detection across multiple steps, the performance bottleneck is now the core gradient computation. When profiling individual kernels of a single run, we observe that the energy gradient computation for contact pairs makes up 60% of the computation.

Scalability for Larger Patterns. All patterns in Table 3 involve fewer than 11k control points, however our implementation can simulate larger systems efficiently. For example, we ran a relaxation of a jersey knit sweater example with 250k control points, and observed a mean wall-clock time per timestep of 18ms. This corresponds to 0.07 μs per control points per timestep, which is an order of magnitude faster than costs observed for our small periodic patterns (see Figure 12) which achieve worse CUDA core utilization. So while the primary performance bottleneck is contact energy gradient computation for large problems, smaller patterns provide less opportunities for parallelization but are still fast.

Comparison between CPU and GPU implementations with/without Periodic BCs. We compared our GPU CUDA/C++ implementation to a reference parallelized 4-core CPU C++ implementation for eight woven patterns for three model sizes: a small model with periodic BCs (PBC), a larger tiled model, and a very large tiled model. Control point counts and CPU and GPU timings are shown in Table 5. Overall we observe that the relative speedup of the GPU_PBC over the CPU_PBC implementation is modest for the tiniest models, e.g., ≈3x for plain weave with only 88 control points, but increases dramatically for larger problem instances, e.g., ≈21x for the tiled satin3-2. For GPU_Large_Tile, a problem 100x the size of GPU_Tiled, the GPU exhibits excellent parallelism and maintains reasonable step times. In general, larger patterns for which the number of control points exceed the number of CUDA cores (3072 on TITAN X) tend to achieve better speedups.

Scalability of GPU Implementation. We constructed tiled plain weave models of increasing sizes to test the performance of the GPU implementation without periodic BCs. The timing per timestep, as well as per control point, is plotted in Figure 13, and demonstrates roughly linear scaling (except for very tiny models).

Yarn-Length Adjustment (Rest-Length Scaling). We found rest-length scaling (from §6.1) to be indispensible for adjusting slip stitches where the default material allocation from the pattern tiling approach can be highly undesirable. The Honeycomb pattern shown in Figure 14 is a good demonstration of the need for slip-stitch correction. This example also demonstrates how a fixed period approach to periodic boundary conditions produces a stretched version of the natural shape; we therefore also gradually reduce the period until the correct aspect ratio is achieved. We also use rest-length scaling for woven patterns to control the length of warp and weft yarns separately, since that could not be achieved with the global radius scaling transformation.

Table 5. **Control-point count / wallclock time (in ms) per timestep for 8 woven models used in 5 CPU/GPU performance tests**

	CPU_PBC	CPU_Tiled	GPU_PBC	GPU_Tiled	GPU_Large_Tile
plain	88 / 1.38	4288 / 33.29	88 / 0.42	4544 / 2.11	21744 / 14.54
twill2-1	130 / 1.62	4288 / 35.91	130 / 0.44	4544 / 1.90	21158 / 10.78
twill4-1	238 / 2.88	4288 / 31.49	238 / 0.45	4544 / 1.51	21744 / 09.50
twill3-2	238 / 2.88	4288 / 31.35	238 / 0.51	4544 / 1.48	21744 / 10.44
basket2x2	180 / 1.71	3780 / 28.88	180 / 0.49	4020 / 1.40	20580 / 10.82
basket3x3	304 / 3.25	4288 / 34.61	304 / 0.50	4544 / 1.37	19448 / 08.78
satin3-2	238 / 2.85	4288 / 31.50	238 / 0.44	4544 / 1.47	21744 / 10.40
satin2-3	238 / 2.79	4288 / 31.59	238 / 0.48	4544 / 1.42	21744 / 08.39

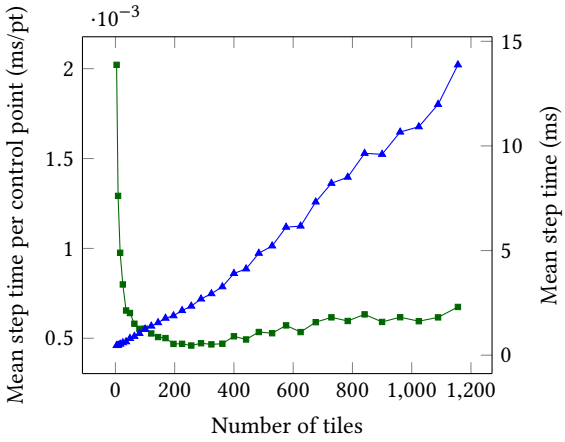
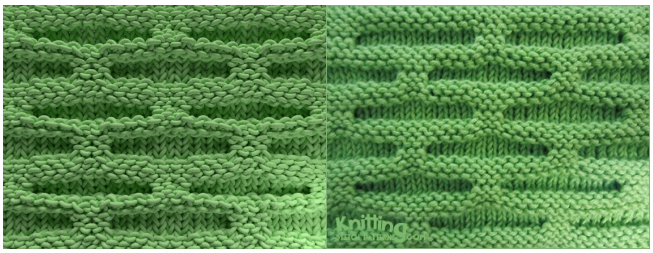


Fig. 13. **GPU performance on tiled plain weave models:** (Green/LeftAxis) Time per timestep per control point and (Blue/RightAxis) time per timestep plotted versus # tiles. On average there are 18.7 control points/per tile, including virtual points.

Yarn-Length Adjustment (Radius Scaling). We used the radius-scaling transformation from §6.2 extensively during interactive design to effectively control essentially every pattern's yarn take-up, and adjust its look. Please see the result shown earlier in Figure 8, as well as the supplemental video for numerous demonstrations of increasing the radius scaling parameter, γ. Since increasing radius scaling introduces new contacts, which slow the relaxation process, we measured a slowdown of the contact detection time that scales with γ.

Knit Pattern Dataset. We synthesized and simulated a total of 32 knit patterns in a dataset that accompanies this paper. These patterns were derived in part from the knitting instructions found at http://knittingstitchpatterns.com [Nalhcib 2018], which provides



(a) Simulated (b) Reference

Fig. 14. **Rest-Length Scaling (Honeycomb pattern):** (Left) Vertical slip stitches were shortened using rest-length scaling to produce the distinctive honeycomb cavities present in (Right) reference photograph of actual knitted samples (image from https://knittingstitchpatterns.com).

a rich collection of patterns using standard knitting instructions. A subset of the simulated patterns are compared to reference photographs in Table 6. Please see the supplemental document for all other examples. Our results closely match the reference images from visual inspection, and benefitted from interactive adjustment of model parameters. Our method also handles lace patterns, as demonstrated in the "Stockinette and Garter Diamonds" pattern.

Woven Pattern Dataset. We also synthesized and simulated a small collection of 8 one-layer woven examples including common basket, twill and satin patterns. For each example we created the bitmap pattern and fabricated the cloth with an industrial loom. We took macro photographs for all of these examples and compared them with our simulation results, some of which are shown in Table 7. Simulated and manufactured patterns have similar structural features, despite different rendering styles. Please see the supplemental video for additional woven examples, including an interactive session designing double-layer woven examples.

9 CONCLUSION

We have introduced the first fully interactive methodology for physics-based stitch pattern design. Our approach is based on core advances for interactive simulation of yarn-level patterns (GPU simulator, periodic boundary conditions), and interactive techniques for on-the-fly pattern adjustment (yarn-length adjustments, sizing, visualization). We have demonstrated the effectiveness of this overall interactive pattern design approach by simulating and adjusting a large corpus of periodic yarn-level cloth models that reasonably match photographs of manufactured samples. By releasing this yarn-level pattern dataset, we hope to inspire research into new methods, and enable the community to more easily compare techniques and results.

Limitations and Future Work. We believe that the implications of this technology are broad, and that in the future, such techniques will fundamentally change the way that yarn-level patterns and textiles are designed, tested, and manufactured by both experts and casual users. While there are several limitations of our specific approach, there are many opportunities for future work.

Regarding our prototype implementation, there are some obvious shortcomings. Every time a pattern is updated during simulation, we completely recreate the yarn curves, discarding any relaxation progress. We opted for this approach due to the complexity of supporting live updates for knit fabrics. Live updates to knit fabric requires tracking contacts and contact relationships, and finding stable transformations that do not cause pull-through or perturb the solution too far from the global optimum. While the periodic-BC GPU relaxer is fast, future work should develop reliable methodologies for on-the-fly modification of local stitch topology for deforming knit and multi-layered woven cloth models. Our radius scaling approach (§6.2) can adjust yarn take-up while keeping a fixed number of spline segments, which works well for increasing the radius, however increasing the yarn length (decreasing the radius) too much may require more degrees of freedom be added to the system and finer contact sampling to avoid pull through. Our current tool only implements a common subset of possible stitches, and others are possible.

Future incarnations could connect more directly to manufacturing, and ultimately aim to produce a fully functional and predictive, universal, interactive, textile design system. Such a system would require the design model be able to translate simulation parameters, such as yarn take-up and tension, to low-level machine parameters. For virtual designs to be most useful, more predictive models are necessary for many aspects of simulation. For example, we have experimented with only a simplified single-ply yarn model with circular cross section, but multi-ply yarn models and yarn-yarn contact interaction methods are needed, possibly in a final validation and visualization phase. Better yarn models should be investigated that deform in ways that more realistically match yarn-level renderings with real-world observations. Better approaches should be devised for quantitative validation of simulated patterns against their manufactured counterparts, possibly using CT scans and photographic techniques. Joint garment-pattern design requires more sophisticated models than rectangular periodic patterns. Finally, in graphics, techniques are needed to efficient design and texture animated cloth models for virtual characters that capture the natural appearance and deformation of realistic yarn-level cloth in a production setting.

ACKNOWLEDGMENTS

We would like to acknowledge Raj Setaluri and James WoMa for discussions and digital translation of cloth patterns, Jui-Hsien Wang for video production, Kui Wu and Cem Yuksel for Stitch Mesh code and discussions, Anastasia Onegina and Brooks Hagan for knitting discussions and demonstrations, http://knittingstitchpatterns.com for reference images and pattern descriptions, and anonymous reviewers for helpful feedback. This work is supported in part by the National Science Foundation IIS-1513967, CM-1644490, and donations from Adobe. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

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 (2016), 50–1.

Nathan Bell and Jared Hoberock. 2011. Thrust: A productivity-oriented library for CUDA. In *GPU computing gems Jade edition*. Elsevier, 359–371.

Floraine Berthouzoz, Akash Garg, Danny M Kaufman, Eitan Grinspun, and Maneesh Agrawala. 2013. Parsing sewing patterns into 3D garments. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 85.
K.F. Choi and T.Y. Lo. 2003. An Energy Model of Plain Knitted Fabric. *Textile Research Jour.* 73 (2003), 739–748.
K.F. Choi and T.Y. Lo. 2006. The Shape and Dimensions of Plain Knitted Fabric: A Fabric Mechanical Model. *Textile Research Jour.* 76, 10 (2006), 777–786.
Gabriel Cirio, Jorge Lopez-Moreno, David Miraut, and Miguel A. Otaduy. 2014. Yarn-level simulation of woven cloth. *ACM Transactions on Graphics* (2014). https://doi.org/10.1145/2661229.2661279
G. Cirio, J. Lopez-Moreno, and M. A. Otaduy. 2017. Yarn-Level Cloth Simulation with Sliding Persistent Contacts. *IEEE Transactions on Visualization and Computer Graphics* 23, 2 (Feb 2017), 1152–1162. https://doi.org/10.1109/TVCG.2016.2592908
A. Demiroz and T. Dias. 2000. A Study of the Graphical Representation of Plain-knitted Structures Part I: Stitch Model for the Graphical Representation of Plain-knitted Structures. *Journal of the Textile Institute* 91 (2000), 463–480.
T.D. Dinh, O. Weeger, S. Kaijima, and S.-K. Yeung. 2018. Prediction of mechanical properties of knitted fabrics under tensile and shear loading: Mesoscale analysis using representative unit cells and its validation. *Composites Part B: Engineering* 148 (9 2018), 81–92. https://doi.org/10.1016/j.compositesb.2018.04.052
Miro Duhovic and Debes Bhattacharyya. 2006. Simulating the deformation mechanisms of knitted fabric composites. *Composites Part A: Applied Science and Manufacturing* 37, 11 (2006), 1897–1915.
Bernhard Eberhardt, Michael Meissner, and Wolfgang Strasser. 2000. Knit Fabrics. In *Cloth Modeling and Animation*, Donald House and David Breen (Eds.). A K Peters, Chapter 5, 123–144.
Yun (Raymond) Fei, Christopher Batty, Eitan Grinspun, and Changxi Zheng. 2018. A Multi-scale Model for Simulating Liquid-fabric Interactions. *ACM Trans. Graph.* 37, 4, Article 51 (Aug. 2018), 16 pages. https://doi.org/10.1145/3197517.3201392
O. Göktepe and S. C. Harlock. 2002. Three-Dimensional Computer Modeling of Warp Knitted Structures. *Textile Research Jour.* 72 (2002), 266–272.
Simon Green. 2010. Particle Simulation using CUDA. *NVIDIA Whitepaper* 6 (2010), 121–128.
Mark Harris, John Owens, Shubho Sengupta, Yao Zhang, and Andrew Davidson. 2007. CUDPP: CUDA data parallel primitives library.
Lejian Huang, Youqi Wang, Yuyang Miao, Daniel Swenson, Ying Ma, and Chian-Fong Yen. 2013. Dynamic relaxation approach with periodic boundary conditions in determining the 3-D woven textile micro-geometry. *Composite Structures* 106 (2013), 417–425.
Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008a. Knitting a 3D model. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 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)*. Citeseer, 17–20.
Chenfanfu Jiang, Theodore Gast, and Joseph Teran. 2017. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 152.
Jonathan Kaldor. 2011. *Simulating yarn-based cloth.* Ph.D. Dissertation. Cornell University.
Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2008. Simulating Knitted Cloth at the Yarn Level. *ACM T. Graph. (SIGGRAPH'08)* 27, 3 (2008), 65.
Jonathan M Kaldor, Doug L James, and Steve Marschner. 2010. Efficient yarn-based cloth with adaptive contact linearization. In *ACM Transactions on Graphics (TOG)*, Vol. 29. ACM, 105.
Arif Kurbak. 2009. Geometrical Models for Balanced Rib Knitted Fabrics Part I: Conventionally Knitted 1x1 Rib Fabrics. *Textile Research Jour.* 79, 5 (2009), 418–435.
Arif Kurbak and Tuba Alpyildiz. 2008. A Geometrical Model for the Double Lacoste Knits. *Textile Research Jour.* 78, 3 (2008), 232–247.
Arif Kurbak and Ali Serkan Soydan. 2009. Geometrical Models for Balanced Rib Knitted Fabrics Part III: 2x2, 3x3, 4x4, and 5x5 Rib Fabrics. *Textile Research Jour.* 79, 7 (2009), 618–625.
Hua Lin, Martin Sherburn, Jonathan Crookston, Andrew C Long, Mike J Clifford, and I Arthur Jones. 2008. Finite element modelling of fabric compression. *Modelling and Simulation in Materials Science and Engineering* 16, 3 (2008), 035010.
Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2014. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 153.
James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. 2016. A compiler for 3D machine knitting. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 49.
Yuyang Miao, Eric Zhou, Youqi Wang, and Bryan A Cheeseman. 2008. Mechanics of textile composites: Micro-geometry. *Composites Science and Technology* 68, 7-8 (2008), 1671–1678.
Nalhcib. 2018. "Knitting Stitch Patterns" website. http://www.knittingstitchpatterns.com/
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. https://doi.org/10.1145/3186265
Wilfried Renkens and Yordan Kyosev. 2011. Geometry modelling of warp knitted fabrics with 3D form. *Textile Research Jour.* 81, 4 (2011), 437–443.
Martin Sherburn. 2007. *Geometric and mechanical modelling of textiles.* Ph.D. Dissertation. University of Nottingham.
Huamin Wang. 2018. Rule-Free Sewing Pattern Adjustment with Precision and Efficiency. In *ACM Transactions on Graphics (SIGGRAPH)*, Vol. 37. ACM, 53:1–53:13.
Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. 2018. Stitch Meshing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)* 37, 4 (2018).
Cem Yuksel, Jonathan M Kaldor, Doug L James, and Steve Marschner. 2012. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 37.

A DERIVATION OF TIME-STEP RESTRICTION

To avoid large motion and pull-through, we restrict the largest timestep displacement of any point on a yarn curve to be less than a fraction $\varepsilon \in (0, 1]$ of the yarn radius r. The condition can be written

$$\|\Delta \mathbf{y}_i(s)\|_2 \leq \varepsilon\, r, \quad \forall i \in 1, ..., N, \quad \forall s \in [0, 1]. \tag{6}$$

Rewriting the change in position in terms of the new velocity, the timestep, and the basis matrix for spline i at s, $\mathbf{B}_i(s)$, we find

$$\|\Delta \mathbf{y}_i(s)\|_2 = \|\mathbf{B}_i(s)\, \dot{\mathbf{q}}^{n+1} \Delta t\|_2$$

$$\|\mathbf{B}_i(s)\, \Delta t\, (\dot{\mathbf{q}}^n + \Delta t\, \mathbf{M}^{-1}\, (-\nabla E + \mathbf{f}))\|_2 \leq \varepsilon r$$

$$\|\mathbf{B}_i(s)\, \Delta t\, \dot{\mathbf{q}}^n\|_2 + \|\mathbf{B}_i(s)\, \Delta t^2\, \mathbf{M}^{-1}\, (-\nabla E + \mathbf{f}))\|_2 \leq \varepsilon r$$

$$\Delta t^2\, \|\mathbf{B}_i(s)\, \mathbf{M}^{-1}\, (-\nabla E + \mathbf{f}))\|_2 + \Delta t\, \|\mathbf{B}_i(s)\, \dot{\mathbf{q}}^n\|_2 - \varepsilon r \leq 0$$

$$\Delta t_i(s) = \frac{-\|\mathbf{B}_i(s)\, \dot{\mathbf{q}}^n\|_2 + \sqrt{\|\mathbf{B}_i(s)\, \dot{\mathbf{q}}^n\|_2^2 + 4\|\mathbf{B}_i(s)\, \mathbf{M}^{-1}\, (-\nabla E + \mathbf{f})\|_2 \varepsilon r}}{2\|\mathbf{B}_i(s)\, \mathbf{M}^{-1}\, (-\nabla E + \mathbf{f})\|_2}$$

By minimizing this value over all spline segments, i, and all N_S particle samples for s, we obtain the additional global timestep restriction.

B KALDOR YARN-LEVEL CLOTH MODEL SUMMARY

For convenience we provide a short summary of the model representation as described in [Kaldor et al. 2008]. A yarn is a collection of control points, where the curve is represented as a B-spline interpolation of these points. The dynamics of this model representation are governed by 3 primary energy forces: bending, stretch stiffness, and contact, and 1 constraint: in-extensibility. In the following equations, $s \in [0, 1]$ parametrizes each spline segment, $\mathbf{y}_i(s)$ is the position of the ith spline at parametrized position s.

The bending energy is defined as

$$E_i^{\text{bend}} = k_{\text{bend}} \ell_i \int_0^1 \kappa^2 ds \tag{7}$$

where $\kappa = \frac{\|y_i'(s) \times y_i''(s)\|}{\|y_i'(s)\|^3}$ is the unsigned curvature of the spline segment i. The internal stretch stiffness energy is defined as

$$E_i^{\text{len}} = k_{\text{len}} \int_0^1 \left(1 - \frac{\|\mathbf{y}_i'(s)\|}{\ell_i}\right)^2 ds \tag{8}$$

where ℓ_i is the rest length of spline i. This term ensures the distribution of mass along the spline's parametric curve is uniform. The

contact energy is defined as

$$E_{i,j}^{\text{contact}} = k_{\text{contact}} \ell_i \ell_j \int_0^1 \int_0^1 f\left(\frac{\|\mathbf{y}_i(s) - \mathbf{y}_j(s')\|}{2r}\right) ds ds' \quad (9)$$

where $\mathbf{y}_i(s)$ is the sample of the ith spline at s, $\mathbf{y}_j(s')$ is the sample of the jth spline at s', and r is the radius of the yarn.

$$f(d) = \begin{cases} \frac{1}{d^2} + d^2 - 2 & \text{if } d < 1, \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

As mentioned in their work, only $|i - j| > 1$ are considered pairs for the contact energy, which prevents self-collisions of overlapping sample spheres on the ends of each spline segment. Additionally there are a number of damping terms, but for the purposes of our work we review only the mass-proportional damping and collision damping terms. Mass-proportional damping is defined below:

$$D_i^{\text{global}} = k_{\text{global}} \int_0^1 \mathbf{v}_i(s)^T \mathbf{v}_i(s) ds \quad (11)$$

where k_{global} encodes the mass dependence of the velocity damping. Contact damping is defined below:

$$D_{i,j}^{\text{collision}} = \ell_i \ell_j \int_0^1 \int_0^1 \left(k_{\text{dt}} \|\Delta \mathbf{v}_{ij}\|^2 - (k_{\text{dt}} - k_{\text{dn}})(\hat{n}_{ij}^T \Delta \mathbf{v}_{ij})^2\right) ds ds' \quad (12)$$

where $k_{\text{dt}} \geq 0$ controls damping in the tangential direction, $k_{\text{dn}} \geq 0$ controls damping in the normal direction; $\Delta \mathbf{v}_{ij} = \mathbf{v}_j(s') - \mathbf{v}_i(s)$ is the relative velocity; $\hat{n}_{ij}(s, s')$ is the normalized value of the collision direction, $n_{ij}(s, s') = \mathbf{y}_j(s') - \mathbf{y}_i(s)$.

C PIPELINE FOR GENERATING ENERGY GRADIENTS

CUDA code can be difficult to write for a typical programmer given hardware specific complexity. With multiple energy terms, it is difficult to write gradient or Hessian terms for each energy term to some degree of accuracy in CUDA. To alleviate this issue, we developed a pipeline for developing vectorized computations in Mathematica that can be automatically converted into CUDA code and libraries. A Mathematica user implements a function, and computes its gradient. This gradient is passed to Mathematica's C generation tool. Using some custom scripts in Mathematica and bash, we convert Wolfram's generated C into a CUDA compatible code base, and generate convenient header file descriptions to make calling the generated functions more streamlined. For reproducability these generation scripts accompany this paper. The modified generated code is designed to remove dependencies to Wolfram's libraries to ensure compatibility with CUDA.

D DERIVATION OF RADIUS SCALING TERMS

We derive each of the transformations in (4). For each term, we require that the scaled and unscaled energies must be equal ($E = E'$), where unprimed values are the physical model and the primed values are the radius scaled model. We also observe that the scaled model is related to the physical model via the following transformations: $\ell' = \ell\gamma$, $\kappa' = \kappa/\gamma$, $\mathbf{q}' = \mathbf{q}/\gamma$, and $\dot{\mathbf{q}}' = \dot{\mathbf{q}}/\gamma$. For all examples below we explain the derivation from the primed (radius-scaled) model to the physical model based on these transformation rules.

The bending stiffness energy of (7) depends on rest length ℓ_i and curvature κ. The integral interior is multiplied by γ^{-2}, and the exterior is multiplied by γ. Therefore k_{bend} must be scaled by γ to ensure the radius remains the same.

The contact energy of (9) depends on rest lengths ℓ_i and ℓ_j, radius r, and positions $y_i(s)$ and $y_i(s)$. Since positions linearly depend on q, they are multiplied by γ, and r is multiplied by γ, canceling out in the interior of the integral. On the exterior, both rest lengths are scaled by γ each, therefore the contact stiffness $k_{contact}$ must be scaled by γ^{-2} to ensure the energy remains the same.

The global damping energy of (11) depends only on the dot product of the velocity arrays, which scales by γ^2, therefore the global damping term must be scaled by γ^{-2} to ensure the energy remains the same.

The collision damping energy of (12) depends rest lengths ℓ_i and ℓ_j, and velocity deltas Δv_{ij}. All of these terms scale by γ, so k_{dt} and k_{dn} must be scaled by γ^{-4} to ensure the energy remains the same.

The length constraint energy of (1) and the length stiffness energy of (8) both depend on the rest length ℓ_i and curve tangent y_i'. Both of these terms are multiplied by γ, and cancel each other out. Therefore no scaling is required for these terms.

Table 6. Comparison of Simulated Knit Patterns to Reference Images

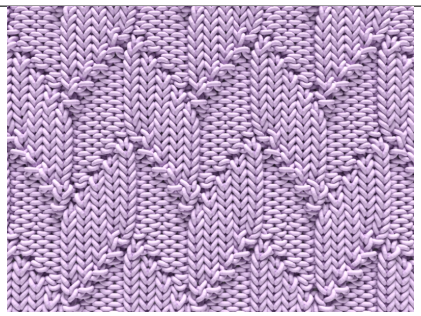
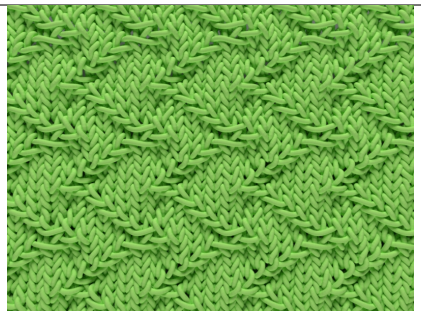
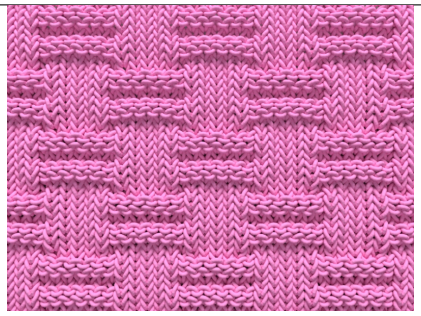
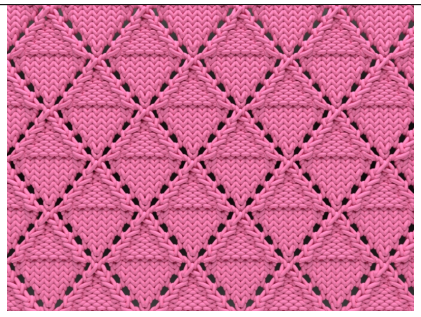
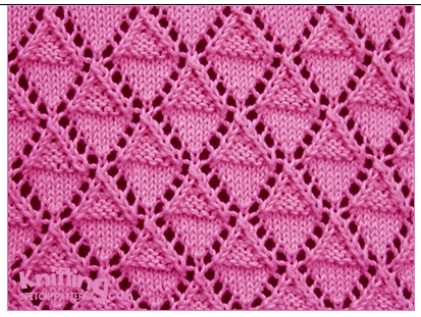
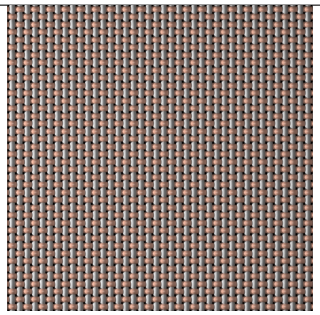
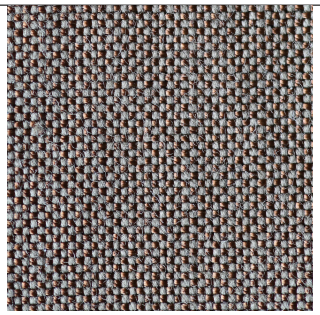
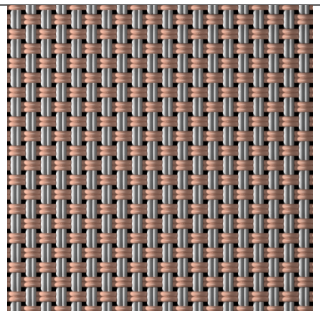
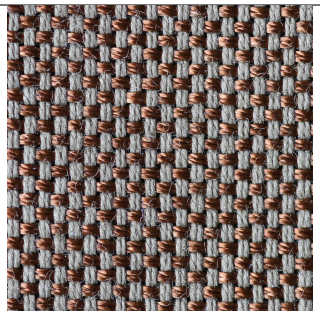
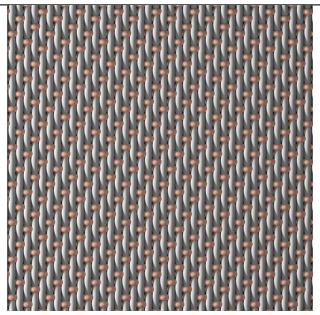
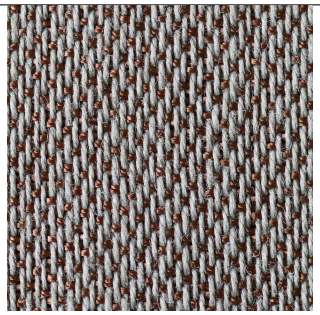
Pattern	Simulated	Reference
Alternating Diagonals		
Jacquard2		
Block1		
Moss Diamond And Lozenge		
Stockinette and Garter		

Table 7. Comparison of Simulated Woven Patterns to Reference Images

Pattern	Simulated	Reference
Plain		
Basket 2x2		
Satin 2x3		
Twill 4x1