Mechanics-Aware Deformation of Yarn Pattern Geometry

GEORG SPERL, IST Austria, Austria
RAHUL NARAIN, Indian Institute of Technology, India
CHRIS WOJTAN, IST Austria, Austria

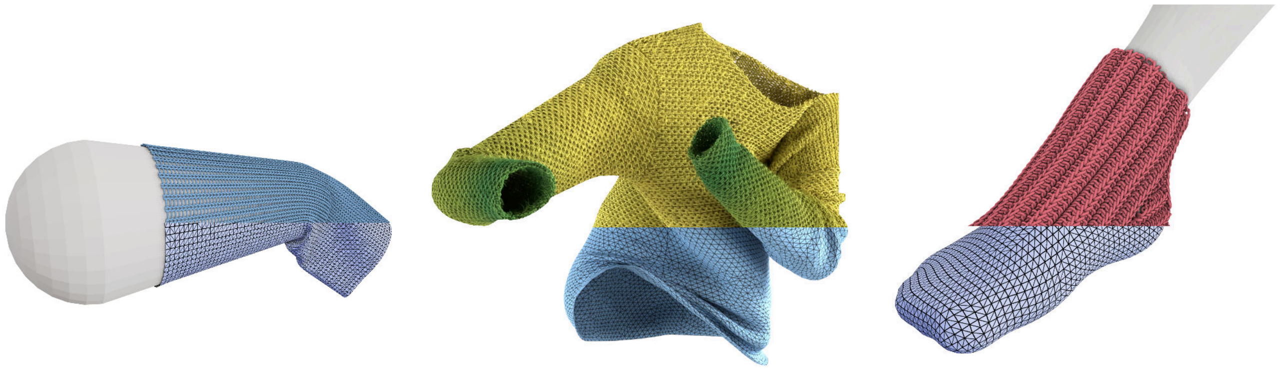


Fig. 1. Our method uses an underlying cloth mesh (bottom) to animate yarn-level cloth (top) in real-time by approximating the yarn-level response in a data-driven fashion based on the deformation of the mesh. It reproduces the stretching behavior of knits (left), animates large garments with millions of yarn vertices (middle), and combines with real-time cloth simulation for end-to-end interactive animation of yarn-level cloth (right). We render the sweater (middle) using pathtracing and with hair particles.

Triangle mesh-based simulations are able to produce satisfying animations of knitted and woven cloth; however, they lack the rich geometric detail of yarn-level simulations. Naive texturing approaches do not consider yarn-level physics, while full yarn-level simulations may become prohibitively expensive for large garments. We propose a method to animate yarn-level cloth geometry on top of an underlying deforming mesh in a mechanics-aware fashion. Using triangle strains to interpolate precomputed yarn geometry, we are able to reproduce effects such as knit loops tightening under stretching. In combination with precomputed mesh animation or real-time mesh simulation, our method is able to animate yarn-level cloth in real-time at large scales.

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

Additional Key Words and Phrases: mechanics-aware, yarn-level cloth, real-time, knitted, woven, cloth simulation

ACM Reference Format:
Georg Sperl, Rahul Narain, and Chris Wojtan. 2021. Mechanics-Aware Deformation of Yarn Pattern Geometry. *ACM Trans. Graph.* 40, 4, Article 168 (August 2021), 11 pages. https://doi.org/10.1145/3450626.3459816

1 INTRODUCTION

The intricate geometry of knitted and woven fabric gives rise to both visual and physical complexity. For example, knit loops tighten when the fabric is being stretched (Figure 2 right), or fabric might curl depending on the yarn pattern [Cirio et al. 2016; Sperl et al. 2020].

Authors' addresses: Georg Sperl, IST Austria, Klosterneuburg, Austria, georg.sperl@ist.ac.at; Rahul Narain, Indian Institute of Technology, New Delhi, India, narain@cse.iitd.ac.in; Chris Wojtan, IST Austria, Klosterneuburg, Austria, wojtan@ist.ac.at.

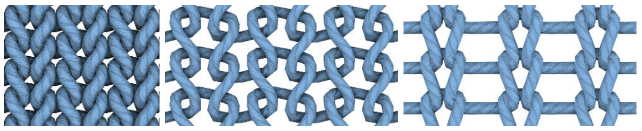


Fig. 2. Deforming a yarn pattern (left) with embedded deformation causes it to stretch uniformly (center), which ignores yarn-level physics that should make the yarn loops tighten (right).

Existing research focuses either on the level of individual yarns, or on overall cloth behavior. Direct simulation of individual yarns [Bergou et al. 2010; Cirio et al. 2016; Kaldor et al. 2008; Sánchez-Banderas et al. 2020] produces detailed yarn behaviors, which can be scaled up to accurately simulate large-scale cloth behaviors albeit at great computational expense. On the other hand, spring- or mesh-based cloth simulation [Baraff and Witkin 1998; Müller et al. 2007; Sperl et al. 2020; Terzopoulos et al. 1987] efficiently approximate the behavior of fabric at the large scale, but they ignore the behavior of individual yarns.

This work proposes to bridge this gap by adding yarn-level deformations to mesh-based cloth simulation. We first precompute the behaviors of a periodic yarn pattern based on the large-scale deformation of the underlying cloth. We then interpolate these deformed yarn patterns at runtime based on the deformation state of the cloth mesh, resulting in richly detailed yarn-level geometry which rearranges in accordance with yarn-level mechanics in real-time. Figure 1 highlights some examples achieved with our method.

We also introduce an efficient way of linearizing the bending response of yarn patterns in terms of stretching, and we propose a way to clamp compression, allowing the user to tune the extent of yarn buckling. We implement the yarn deformation procedure as compute shaders on the GPU.

Overview. We start by discussing related work in Section 2. Then we detail the two main stages of our method: data generation and real-time displacement. Section 3 explains the data generation step shown in Figure 3, where we optimize for the rest configuration of yarn patterns under a range of large-scale deformations, and compute local displacements relative to these deformations. Section 4 details the real-time displacement phase illustrated in Figure 4, where we map the precomputed yarn geometry onto a deforming triangle mesh. Section 5 explores our method's results. Section 6 discusses limitations and advantages, and concludes with a summary and future work.

2 RELATED WORK

Yarn-Level Cloth. Kaldor et al. [2008, 2010] first simulated cloth at the yarn level, and researchers later reduced collision detection costs with persistent sliding contacts [Cirio et al. 2014, 2015, 2016], improved the bending model [Pizana et al. 2020], and handled contact between multiple layers of cloth [Sánchez-Banderas et al. 2020]. Leaf et al. [2018] developed a tool for designing yarn patterns under tension.

Researchers recently combined yarn-level and mesh-level cloth simulators. Casafranca et al. [2020] enriched a mesh-based cloth simulation with yarn-level simulation in regions of interest, and Sperl et al. [2020] used precomputed yarn patterns to develop a homogenized continuum material model. The latter use texture maps to visualize yarn geometry, so the individual strands do not react to the deformation of the cloth. Our work enhances these ideas by adding mechanics-aware yarn-level geometry to any mesh animation at negligible cost.

Embedded Detail. A simple and effective way to give the illusion of detailed physics is to embed fine geometric detail into a coarser control mesh. After the idea of dynamic free-form deformations were introduced by Faloutsos et al. [1997], researchers embedded geometric detail into animations of elasticity [Sifakis et al. 2007], fracture [Muller et al. 2004], viscoelasticity [Wojtan and Turk 2008], fluids [Wojtan et al. 2009], and articulated characters [Rumman and Fratarcangeli 2016]. Researchers also use this idea to deform woven yarn pattern geometry [Montazeri et al. 2020; Zhao et al. 2016]. Hoffman et al. [2020] map detailed knit, crochet, and sequin geometry onto cloth meshes with support for fly-away fibers. Stitch Meshes [Yuksel et al. 2012] used a simplified mechanical sheet model to deform 3D yarn-level cloth models to reduce relaxation times. Because these techniques apply coarse deformations to fine-scale geometry, they cannot reproduce small-scale physical effects. Our method addresses this in the context of yarn-level cloth, by displacing the yarn geometry before mapping it onto the deformed mesh such that the combined deformation actually captures the yarn-level physical effects.

Example-Based & Mechanics-Aware Deformation. Our work animates yarn physics in real-time by interpolating from precomputed examples. Researchers also use example geometry to bias physics simulators toward preferred results [Gao et al. 2019; Koyama et al. 2012; Martin et al. 2011; Schumacher et al. 2012; Wampler 2016]. Similar to Ma et al. [2008]'s interpolation of displacement textures based on the deformation of a face mesh, we interpolate yarn geometry based on the deformation of a cloth mesh. Montazeri et al. [2019] deform yarn cross-sections by learning a model from precomputed simulations; our approach operates at one scale higher, by animating the reconfiguration of yarn *patterns* based on a database of precomputed examples.

Researchers use similar ideas to add fine-scale wrinkle details to a coarse cloth or flesh simulation. Procedural wrinkle methods rely on an underlying strain field [Hadap et al. 1999; Rohmer et al. 2010; Zuenko and Harders 2019] or explicit simulation of detail [Müller and Chentanez 2010], while data-driven wrinkle methods instead train local operators or pose-dependent systems [Kavan et al. 2011; Wang et al. 2010; Zurdo et al. 2012], with recent works building on recurrent or convolutional neural networks [Chentanez et al. 2020; Jin et al. 2020; Santesteban et al. 2019; Vidaurre et al. 2020]. Our work assumes that the input cloth mesh animation already contains all cloth-scale features including wrinkles, and then adds detail purely on a yarn scale.

3 DATA GENERATION

We want our embedded yarn details to deform realistically, so we prescribe their motion based on yarn-level simulations. This section describes the physics precomputation phase of our algorithm, illustrated in Figure 3.

3.1 Deformation Optimization

We use the method of Sperl et al. [2020] to simulate how each yarn pattern deforms. This method takes as input a particular yarn pattern in an undeformed configuration (e.g. the "stockinette" pattern in Figure 3), and a large-scale surface deformation encoded as the first $\mathbf{I}$ and second $\mathbf{II}$ fundamental forms. The method uses this large-scale deformation to define boundary conditions, and then optimizes for the elastostatic equilibrium configuration of the yarn pattern. Sperl et al. [2020] then reduced this geometric information down to a single scalar energy density used in a hyperelastic material model. In contrast, we will use the yarn geometry directly.

We model yarns as discrete elastic rods [Bergou et al. 2010], represented as connected lists of vertices with positions $\boldsymbol{x}$ along the centerline. Each edge also stores a twist angle θ and a reference director $\underline{\boldsymbol{d}}_1$.[1] We concatenate positions and twists for each vertex into a four-dimensional vector $\boldsymbol{q} = \left(\boldsymbol{x}^\top, \quad \theta\right)^\top$, where θ corresponds to one incident edge. (Note that the reference directors $\underline{\boldsymbol{d}}_1$ are not degrees of freedom.)

[1]The subscript in $\underline{\boldsymbol{d}}_1$ refers to its definition as one of two reference directors: one for the edge normal, and one for the edge binormal.

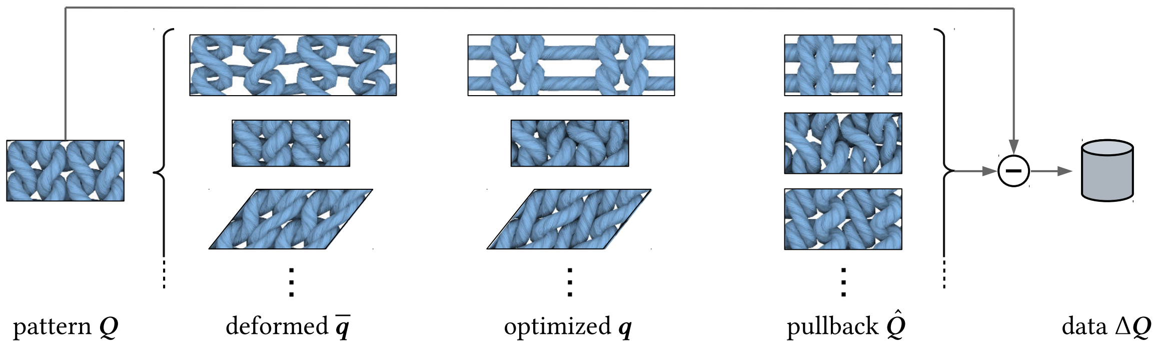

pattern $\boldsymbol{Q}$
deformed $\overline{\boldsymbol{q}}$
optimized $\boldsymbol{q}$
pullback $\hat{\boldsymbol{Q}}$
data $\Delta\boldsymbol{Q}$

Fig. 3. From left to right: We start with a periodic yarn pattern $\boldsymbol{Q}$, and we apply a range of large-scale surface deformations to get the deformed state $\overline{\boldsymbol{q}}$. Using the optimization method of Sperl et al. [2020], we optimize for the elastostatic rest shape $\boldsymbol{q}$ subject to the respective deformation. We then pull the resulting geometry back into the undeformed material space to get $\hat{\boldsymbol{Q}}$. Finally, we subtract the initial state to compute displacements $\Delta\boldsymbol{Q}$, building a mapping from large-scale deformation to the associated local yarn-level deformation.

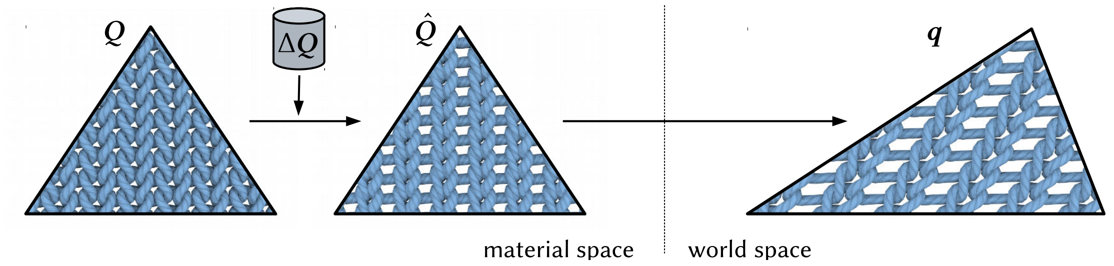


Fig. 4. Our algorithm for animating yarn geometry in real-time: We start with undeformed yarn-level geometry $\boldsymbol{Q}$ tiled over a triangle in material space (left), and we apply the appropriate material displacement $\Delta\boldsymbol{Q}$ from the database to get deformed yarn geometry $\hat{\boldsymbol{Q}}$ (middle). This geometry is then mapped along with the triangle to get the current world-space deformation $\boldsymbol{q}$.

We recall from [Sperl et al. 2020] that during the optimization the kinematics of yarn vertex positions are defined as

$$\boldsymbol{x}(X) = \overline{\boldsymbol{x}}(X) + \tilde{\boldsymbol{u}}(X), \tag{1}$$

$$\overline{\boldsymbol{x}}(X) = \boldsymbol{\varphi}(X_1, X_2) + X_3\, \boldsymbol{n}(X_1, X_2). \tag{2}$$

Here, $\boldsymbol{X} = (X_1, X_2, X_3)^\top$ are the *material-space* coordinates of the undeformed yarn pattern, with (X_1, X_2) the orthogonal and periodic directions along the pattern and X_3 the height coordinate. $\overline{\boldsymbol{x}}$ denotes the *large-scale* deformation constructed from the input fundamental forms $\mathbf{I}$ and $\mathbf{II}$, which encode in-plane and bending deformations respectively. From these fundamental forms, we construct the mid-surface $\boldsymbol{\varphi}$ with normal $\boldsymbol{n}$, as described in [Sperl et al. 2020] and in Appendix A, such that $\mathbf{I} = \nabla\boldsymbol{\varphi}^\top\nabla\boldsymbol{\varphi}$ and $\mathbf{II} = -\nabla\boldsymbol{\varphi}^\top\nabla\boldsymbol{n}$. We then solve for the yarn configuration which minimizes elastic energy. The optimization variables are the fluctuations $\tilde{\boldsymbol{u}}$, which describe local displacements relative to the large-scale deformation, and the twists θ. Using Θ for undeformed twists, the concatenated coordinates are undeformed $\boldsymbol{Q} = (\boldsymbol{X}^\top, \Theta)^\top$, large-scale-deformed $\overline{\boldsymbol{q}} = (\overline{\boldsymbol{x}}^\top, \Theta)^\top$, and optimized $\boldsymbol{q} = (\boldsymbol{x}^\top, \theta)^\top$. Here, we assume that twists Θ are unaffected by the large-scale mapping, which is true for pure in-plane deformations. In general, bending may induce local twists that depend on $\mathbf{II}$ and the orientation of yarns relative to the curvature direction. In our experiments, we assume that this effect is small.

Crucially for our application, we found that the optimization of Sperl et al. [2020] contains a nullspace that allows yarns to slide: a periodic yarn curve $\boldsymbol{x}(s)$ parametrized by s can slide by a parametric shift Δs without changing its elastic energy E, i.e. $E(\boldsymbol{x}(s)) = E(\boldsymbol{x}(s + \Delta s))$. Geometrically, such a shift corresponds to tangential sliding of yarn while maintaining the same periodic shape. To the optimizer any such state is equivalent, and the actual result may depend arbitrarily on numeric solver parameters. This nullspace does not affect the homogenized energies in [Sperl et al. 2020] by definition. However, *interpolating* between two shifted parameterizations may produce completely different shapes, as shown in Figure 5. In our experience, this nullspace creates distracting interpolation artifacts even for nearly-identical deformations. The problem remains as we sample the deformations more densely, manifesting as sharp discontinuities in deformation space.

We eliminate this parametric yarn sliding by adding a constraint to the optimization, effectively removing the nullspace. Specifically, we fix one vertex per periodic yarn to remain on the boundary of the pattern:

$$\tilde{\boldsymbol{u}} \cdot (\nabla\boldsymbol{\varphi}\, \boldsymbol{N}) = 0, \tag{3}$$

where $\boldsymbol{N}$ is the undeformed normal to the respective pattern boundary, either $\boldsymbol{N} = (1, 0)^\top$ or $\boldsymbol{N} = (0, 1)^\top$. This sparse set of vertex constraints efficiently and effectively removes the aforementioned

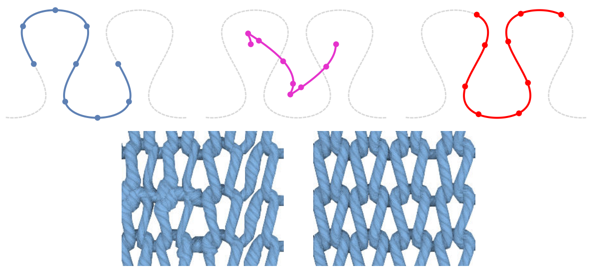


Fig. 5. Top: Interpolating two periodic curves (left and right) that are identical up to a shift in parametrization can result in arbitrary interpolated shapes which look very different from the original data (center).
Bottom: Interpolating yarn geometry without accounting for sliding creates unrealistic artifacts like self-collisions and floating loops (left); our sliding constraint eliminates these artifacts (right).

interpolation artifacts. This strategy allows us to find a physically realistic yarn shape for a given large-scale deformation described by **I** and **II**.

3.2 Sampling

Now that we can produce yarn geometry for a given deformation, we want to precompute a series of representative samples and interpolate them at runtime. **I** and **II** are each symmetric 2×2 tensors, so parameterizing deformation by **I** and **II** directly yields a 6-dimensional function which is expensive to precompute, store, and read at runtime. We aim to reduce this dimensionality by parameterizing the deformation with as few variables as possible.

Following Sperl et al. [2020], we reparameterize **I** as a 3-dimensional function using the in-plane strains

$$s_x = \sqrt{\mathrm{I}_{11}} - 1, \qquad s_a = \frac{\mathrm{I}_{12}}{\sqrt{\mathrm{I}_{11}\mathrm{I}_{22}}}, \qquad s_y = \sqrt{\mathrm{I}_{22}} - 1. \tag{4}$$

II can be parameterized similarly by introducing additional curvature variables and increasing the dimensionality of the dataset. However, we show in Section 4 that bending deformation can be reasonably approximated in terms of stretching variables alone. This strategy lets us sample the entire large-scale deformation space with only three variables s_x, s_a, and s_y, significantly reducing memory and computation overhead. We sample these deformations on a regular 3D grid, and we discuss the performance, data size, and quality for different numbers of samples in Section 5.

3.3 Material-Space Displacements

At runtime, our interpolated yarn geometry will be mapped onto a deformed triangle mesh, where it will naturally inherit the deformation of its triangle (the mapping from material-space to world-space in Figure 4). Thus, we want to store all of the deformation *except* the large-scale deformation in our precomputation as material-space displacements (the "pullback" column in Figure 3).

To do this, we need to find the modified material space coordinates $\hat{X}$ which give us our desired world-space deformations $\boldsymbol{x}$ when deformed by only the large-scale deformation $\overline{\boldsymbol{x}}(\hat{X})$: in other words, find $\hat{X}$ s.t. $\boldsymbol{x} = \overline{\boldsymbol{x}}(\hat{X})$. We perform this solve using Newton's method and provide more details in Appendix B.

Algorithm 1 Real-time Yarn Animation

Input: mesh animation, yarn pattern, displacement data ΔQ
Output: deformed yarn geometry $\boldsymbol{q}$

```
 1: procedure ANIMATE YARNS
 2:     $Q \leftarrow$ yarn pattern tiled over mesh
 3:     EACH FRAME:
 4:         compute I, II per face
 5:         interpolate I, II to mesh vertices
 6:         for each yarn vertex do                              ▷ on the GPU
 7:             interpolate I, II from mesh
 8:             compute linearized bending $\mathbf{I}(X_3)$          ▷ (7)
 9:             clamp compression                                ▷ Section 4.3
10:             compute strains $s_x, s_a, s_y$                  ▷ (4)
11:             look up displacements $\Delta Q$                 ▷ Section 3.3
12:             displace: $\hat{Q} \leftarrow Q + \Delta Q$       ▷ (8)
13:             map: $\boldsymbol{q} \leftarrow (\boldsymbol{x}(\hat{X})^\top, \hat{\Theta})^\top$   ▷ Section 4.5
14:         end for
15:         tessellate and render $\boldsymbol{q}$                ▷ Supplementary S2
16: end procedure
```

Concatenating $\hat{Q} = (\hat{X}^\top, \theta)^\top$, we subtract the initial material state to get displacements

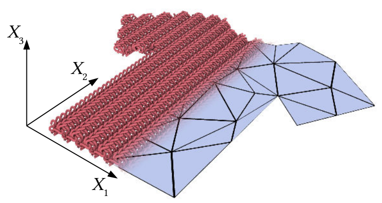
$$\Delta Q = \hat{Q} - Q. \tag{5}$$

Note that at the rest pose, $\mathbf{I} = \mathrm{Id}$, $\mathbf{II} = 0$, and $\Delta Q = 0$.

To summarize, we can now build a database of material-space yarn displacements ΔQ for each vertex i of a pattern and for a range of in-plane deformations j: $\Delta Q_i(s_{xj}, s_{aj}, s_{yj})$. This database can be interpreted as a grid of example deformations, or similarly as a 3D displacement texture per yarn vertex. Interpolating between these samples takes a planar deformation s_{xj}, s_{aj}, s_{yj} and maps it to a yarn-level displacement map. Note that this will recover the exact deformed yarn pattern for the strains sampled in the database and approximate patterns for intermediate strains.

4 REAL-TIME DISPLACEMENT

Now that we have a database of yarn pattern displacements for a range of deformations, we apply them to a yarn pattern tiled over an animating triangle mesh. In the following discussion, we denote Q as the undeformed (material space) coordinates, $\hat{Q}$ as the coordinates deformed by *local* displacements, and $\boldsymbol{q}$ as the final world-space coordinates, as illustrated in Figure 4. Algorithm 1 outlines our procedure.



As a precomputation, we first create the initial undeformed yarn mesh corresponding to the undeformed triangle mesh (inset). We generate a 2D background grid in the mesh's UV-coordinates with cells the size of the periodic pattern, and we copy the yarn geometry into every cell that overlaps an undeformed triangle. We then remove yarn vertices that do not lie within a triangle and delete yarn fragments shorter than a user-specified length for aesthetic purposes as described in the supplementary (Section S1). Finally, we precompute the material-space barycentric coordinates for each yarn vertex.

4.1 Mesh Strains

Each animation frame, we compute discrete fundamental forms for each triangle as in [Sperl et al. 2020]:

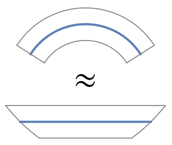
$$\mathrm{I} = \boldsymbol{F}^\top \boldsymbol{F}, \qquad \mathrm{II} = \boldsymbol{F}^\top \boldsymbol{\Lambda} \boldsymbol{F}, \tag{6}$$

where $\boldsymbol{F}$ is the triangle deformation gradient and $\boldsymbol{\Lambda}$ is the triangle-averaged shape operator from [Grinspun et al. 2006]. We then distribute them to triangle vertices using modified Shepard weights from Phong interpolation [James 2020], and finally interpolate them to each yarn vertex.

4.2 Linearized Bending

As alluded to in Section 3.2, we can approximate the effect of bending behavior by adding stretching and compression depending on the surface curvature. We show in Appendix C.1 that we can enhance the first fundamental form with one that varies along the surface normal:

$$\mathrm{I}(X_3) \approx \mathrm{I} - 2\, X_3\, \mathrm{II}, \tag{7}$$



We illustrate this idea in the inset figure to the right, which approximates the extruded volume around a curved blue midsurface (top) with a linearized volume that is stretched above and compressed below the midsurface (bottom). We compare this idea to other bending models in Section 5.

4.3 Compression Clamping

Like most elastic materials, cloth *buckles* out of plane when compressed. The chaotic nature of this buckling can make our yarn optimization reach multiple visually distinct configurations that do not vary smoothly over deformation space (Figure 6, top). Sperl et al. [2020] dealt with this issue regularizing the fit of their output energies. We similarly regularize compression by clamping the eigenvalues of I to a lower bound before looking up the yarn displacement, which reduces buckling in a user-tunable way.

Using the method of Deledalle et al. [2017], we set a minimum value $\lambda_{\min}$ for the eigenvalues of $\mathrm{I}(Z)$. (Unless stated otherwise, all of our experiments use $\lambda_{\min}$=0.8, where λ<1 is compression.) Because $\Delta\boldsymbol{Q}\!\longrightarrow\!0$ as $\mathrm{I}\!\longrightarrow\!\mathrm{Id}$, our technique for clamping compression will only reduce local deformations, but it will still deform due to the triangle embedding. Figure 6 (bottom) shows how this clamping reduces buckling while preserving the large-scale deformation.

4.4 Local Displacement

After clamping I, we convert it to strains s_x, s_a, s_z (Equation (4)), trilinearly interpolate the yarn displacement $\Delta\boldsymbol{Q}(s_x, s_a, s_z)$, and compute the deformed material space yarn coordinates

$$\hat{\boldsymbol{Q}} = \boldsymbol{Q} + \Delta\boldsymbol{Q}. \tag{8}$$

We clamp strains outside of the sampled range to their nearest neighbor in the $\Delta\boldsymbol{Q}(s_x, s_a, s_z)$ dataset. Similar to compression clamping,

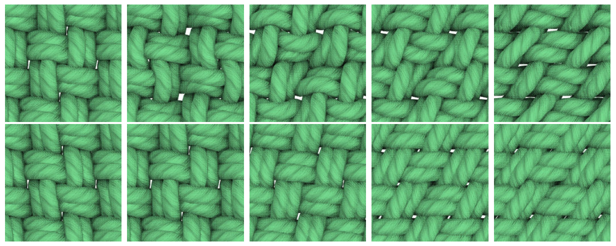


Fig. 6. Top: From left to right, a yarn pattern buckles into different configurations under increasing shearing deformation. Bottom: Adding a lower bound $\lambda_{\min}$=0.7 to the eigenvalues of the in-plane deformation I allows a user to tune the extent of buckling during animation.

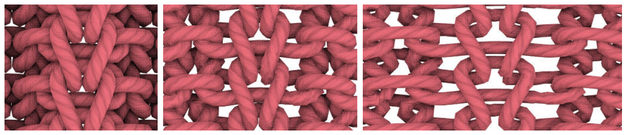


Fig. 7. The yarns of a rib pattern adapt to increased stretching from an initial state (left) until the limits of the precomputed data are reached (middle). Deformation beyond the sample limits (right) do not result in further local displacements, but instead fall back to purely embedded deformation.

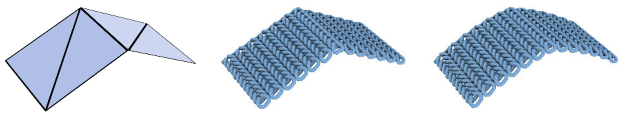


Fig. 8. Barycentric embedding of yarn geometry may create sharp edges (center), whereas Phong deformation smooths out obvious mesh resolution artifacts (right).

this constant extrapolation will limit the local deformations while still inheriting large-scale deformations from the mesh embedding (See Figure 7).

4.5 World-Space Mapping

To map the yarn vertices to world space $\boldsymbol{x}$, we use:

$$\boldsymbol{x}(\hat{\boldsymbol{X}}) = \boldsymbol{\varphi}(\hat{X}_1, \hat{X}_2) + \hat{X}_3\, \boldsymbol{n}(\hat{X}_1, \hat{X}_2), \tag{9}$$

which corresponds to extruding the world-space mesh surface $\boldsymbol{\varphi}$ along its normal $\boldsymbol{n}$. To avoid piecewise linear embedding artifacts, we employ Phong deformation [James 2020] and interpolated vertex normals to generate a smoother surface $\boldsymbol{\varphi}$ and shell-volume (Figure 8).

To map yarn twists, we simply copy the updated twist values θ=$\hat{\Theta}$, and we co-transform the edge normals $\underline{\boldsymbol{d}}_1$ using an approximate mapping of the Jacobian of (9), as detailed in the supplementary material Section S2.1.

The computation of yarn vertex deformation and world-space mapping are trivially parallel, so we implement them as GPU compute shaders. The interpolation of $\Delta\boldsymbol{Q}$ results in a single 3D texture interpolation per yarn vertex.

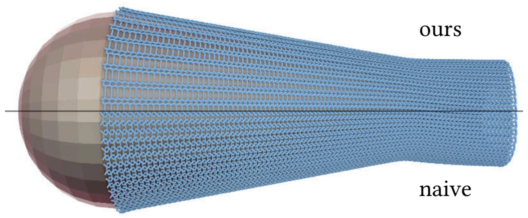


Fig. 9. Comparison of our mechanics-aware yarn animation (top) against naive embedding (bottom). The yarn geometry differs substantially on the left, where the deformation is large.

4.6 Real-Time Rendering

Computer graphics researchers have developed a number of algorithms for rendering yarns and fibers [Montazeri et al. 2020, 2019; Wu and Yuksel 2017]. For the examples in this paper, we tessellate the deformed yarns as cylindrical meshes in a geometry shader. We approximate ply- and fiber-level detail with procedurally twistable normal maps and ambient occlusion maps, and we approximate volume conservation by locally rescaling yarn radii when they are stretched. We provide the full rendering details in the supplementary document Section S2.

5 RESULTS

We will now discuss examples generated by our mechanics-aware yarn animation technique. Figure 9 shows how our algorithm compares to a more typical embedded approach to animating yarn-level geometry detail. Our method naturally reacts to the deformation of the underlying cloth by causing loops to rearrange and tighten up as the tension is increased. As a natural consequence, our approach also reproduces the tendency of knitted and mesh fabrics to become more transparent when stretched. Figure 10 shows how we can easily apply different yarn-patterns to any cloth mesh, producing visually distinct geometry which depends on both the deformations of the mesh and the precomputed yarn mechanics. Our method can add yarn-level details onto *any* deforming triangle mesh: examples in this paper use deforming cloth meshes from ArcSim [Narain et al. 2013, 2012], position-based dynamics [Müller et al. 2007], and Blender [2020].

We enhance an offline cloth simulation solver which reproduces the large-scale effects of knitted garments [Sperl et al. 2020]. Figure 11 compares this approximation to ground-truth yarn-level simulations, yielding visually plausible recreations of the local pattern deformation. We note that the accuracy of our method highly depends on the accuracy of the underlying triangle deformation, so we see higher discrepancy where the FEM cloth simulation deviates from yarn-level simulation.

We also used our approach to add yarn-level details to a position-based dynamics cloth solver [Bender et al. 2015], to approximate yarn-level cloth simulation in real-time. Figure 12 shows how a user can perform an interactive dressing operation by pulling a knitted sock onto a foot. Note that our method still produces plausible yarn-level deformations, despite the fact that the default position-based elastic material model is quite far from a model derived from yarn physics.

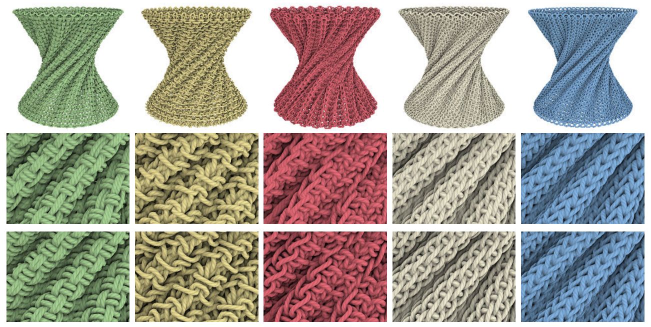


Fig. 10. Different yarn patterns mapped onto a twisted cloth with our method react differently to the mesh deformation. The second row shows a zoom and compares it to naive embedding in the third row. Our method captures the subtle tightening in all cases, while the naive method results in unrealistic gaps between yarns.

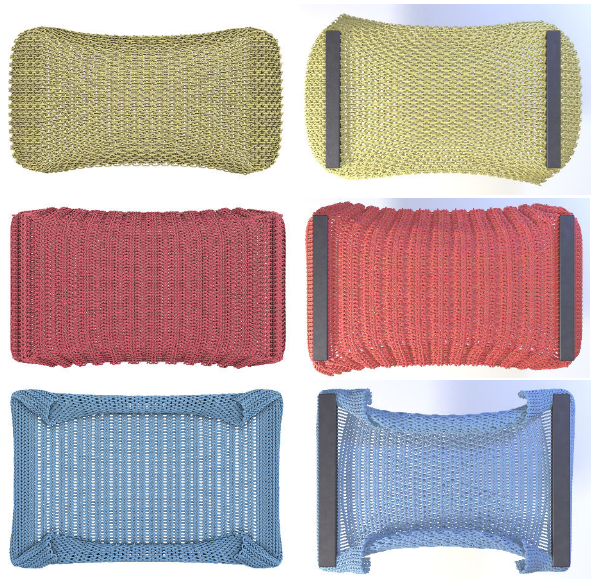


Fig. 11. Comparison of yarns animated with our method applied to a precomputed cloth simulation (left) against full yarn-level simulation (right, courtesy of Sperl et al. [2020]). Our yarn level deformations differ the most where the triangle-level cloth simulation is least accurate.

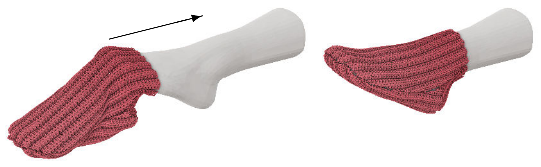


Fig. 12. We use position-based dynamics to simulate a sock being pulled over a foot in real-time, where the user can interactively control the force.

Figure 13 shows we can render our yarn geometry offline to produce higher quality path-traced scenes with "fuzz" from a procedural particle system. Figure 14 shows how our GPU-based yarn

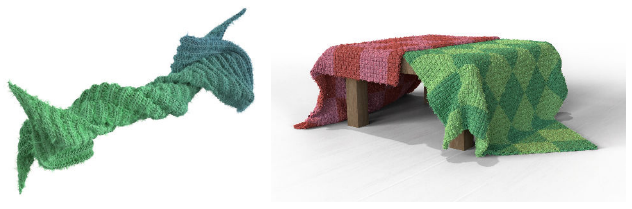


Fig. 13. Two examples of our yarn geometry that were rendered offline using path tracing and hair particles.

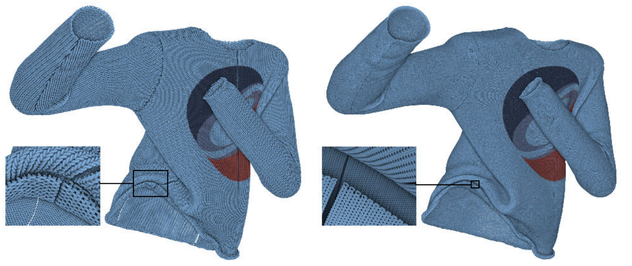


Fig. 14. Our method scales favorably with increasing yarn density. We deform yarn geometry for a sweater with 0.8 million vertices at well over 100 FPS (left), and a sweater with 42.7 million vertices at over 14 FPS (right).

displacement algorithm allows for the animation of garments with millions of individual yarn vertices at interactive rates. Please refer to our supplementary video for additional highly detailed results. For reproducibility, we release the source code and data used to generate our results (https://git.ist.ac.at/gsperl/MADYPG).

Performance. We ran our method on a desktop computer with an Intel Core i7-7820X processor and an NVIDIA GeForce GTX 1080 Ti graphics card and gathered performance statistics. Table 1 breaks down the computational cost of our algorithm for each of the various examples in this paper. Almost all examples in this paper perform both geometry generation and rendering in real-time (well over 60 fps). To stress-test our method, we created the knitted sweater in Figure 14 (right), which has more than 40 million yarn vertices and generates yarn geometry at over 14 fps. The total time including rendering (depending on the complexity of the renderer and the amount of geometry per pixel due to camera zoom) is around 4 fps for this example. Please see our submission video for more detailed demonstrations of our method.

Dataset Size. The number of samples in our precomputed yarn database only modestly affects the visual quality of the animation. Figure 15 shows a deformed rib pattern with a varying number of database samples and a visualization of the associated interpolation error. We also provide animations with several sampling densities in our supplementary material. In practice, we found that a small number of samples was sufficient to capture strong effects like loops tightening under tension, and we noticed surprisingly few obvious interpolation errors (like interpolated threads leading to self-collisions).

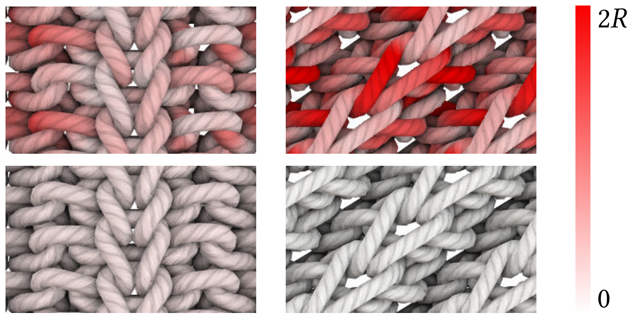


Fig. 15. Comparison of different data sampling densities. The top and bottom rows show results generated with $5 \times 5 \times 5$ and $31 \times 31 \times 31$ samples respectively. In the top images, we color-code the L^2 error in vertex positions between the coarse and fine model, compared to the yarn radius R; red color corresponds to a larger difference. The coarse model produces plausible geometry even in the presence of larger errors.

On the other hand, a large number of database samples led to unwanted noise in the animations, like sudden 'pops' from one configuration to another, instead of gradual ones. We believe our observations are consistent with those of Sperl et al. [2020], who noted that a dense sampling rate (especially in the compressive regime where fabric buckles chaotically) required explicit filtering to remove high-frequency noise. In our case, simply removing samples from the database acts as an effective low-pass filter, and our compression clamping technique strongly reduces popping.

The complexity of our yarn database has only a small affect on runtime. Table 2 relates the number of precomputed yarn-level simulations in our database to the memory and runtime of our method. Notably, a 238× increase in database size caused a proportional increase in memory but only a 2.8× increase in runtime. Unless stated otherwise, the results shown in this paper use datasets of 9^3 samples sampled uniformly over the ranges $s_x, s_y \in [-0.2, 1.0]$ and $s_a \in [-0.7, 0.7]$.

Bending Models. Finally, we compare the effect of our linearized bending approximation in Figure 16. We compare our linearized model described in Section 4.2 to a bending model that explicitly captures combined stretching and bending. (See Appendix C for details on these comparison models, and see our supplementary videos for an animated version of this and similar tests.) First of all, we found that the importance of bending depends on the knit pattern: "thick" patterns like the rib (Figure 10 red, center) exhibit more differences than nearly planar ones like the stockinette (Figure 10 blue, right). The further away the geometry is from the surface $\boldsymbol{\varphi}$, the more it is locally stretched or compressed by bending. Second, even for a thick pattern, the differences between the two models are minor and localized to regions of strong curvature. Most importantly, the linearized model is much more efficient — when comparing CPU-based implementation, our linearized model ran roughly 8× faster.

An extra benefit of the linearized bending model is that it transforms bending-induced buckling into compression, which can be filtered with our compression clamping algorithm (Section 4.3). Otherwise, it would be non-trivial to filtering both compression- and bending-related buckling in a compatible and coherent manner.

Table 1. Performance breakdown of results in this paper. From left to right, we list: the respective figure, the number of animated yarn vertices and average per-frame times of mesh strain computation (Section 4.1), data look-up and local yarn displacements (Sections 4.2 to 4.4), and embedded world-space mapping (Section 4.5). The mesh stage is implemented on the CPU, whereas yarn displacement and mapping are implemented on the GPU. In comparison, the CPU-implemented position-based dynamics step for the sock example averaged to 19.24 ms.

animation		# vertices	strains$^{\text{CPU}}$	displacement$^{\text{GPU}}$	mapping$^{\text{GPU}}$
sleeve	Fig. 9	94k	1.02 ms	0.13 ms	0.10 ms
patterns (averaged)	Fig. 10	82k	0.82 ms	0.09 ms	0.07 ms
honeycomb stretch	Fig. 11 top	74k	0.61 ms	0.08 ms	0.07 ms
rib stretch	Fig. 11 center	154k	0.77 ms	0.16 ms	0.12 ms
stockinette stretch	Fig. 11 bottom	119k	0.59 ms	0.12 ms	0.11 ms
stockinette sweater	Fig. 14 left	862k	2.59 ms	0.81 ms	0.66 ms
fine stockinette sweater	Fig. 14 right	42.7M	2.59 ms	35.34 ms	27.68 ms
rib twist	Fig. 13 left	433k	0.38 ms	0.51 ms	0.38 ms
table cloth	Fig. 13 right	130k	1.50 ms	0.16 ms	0.13 ms
sock	Fig. 12	139k	7.5 ms	0.14 ms	0.11 ms

Table 2. Comparison of rib pattern displacement data for different dataset sizes. "generation" refers to the time needed to precompute the data, and "displacement" refers to the per-frame time of computing $\hat{Q} = Q + \Delta Q(s_x, s_a, s_z)$ on a representative sweater animation with 1.8 million yarn vertices.

# samples	memory	generation	displacement
$5^3 = 125$	0.7 MB	2.2 min	1.58 ms
$9^3 = 729$	4.2 MB	12.7 min	1.82 ms
$15^3 = 3375$	19.2 MB	59.8 min	2.30 ms
$31^3 = 29791$	169.7 MB	533.6 min	4.35 ms

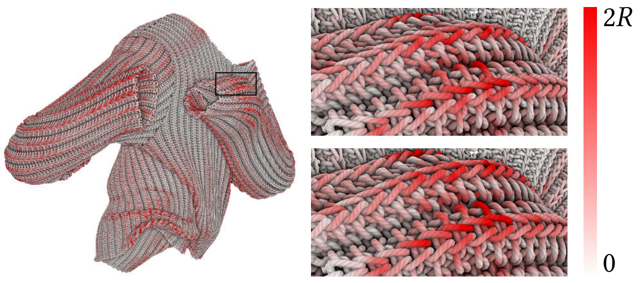


Fig. 16. We compare our linearized bending model against a model incorporating bending strains explicitly on an animated knit sweater. We measure the difference between the models as the L^2 norm of the difference in yarn vertex positions, and we color code them relative to the yarn radius R; red color corresponds to a larger difference. The cut-outs show how knit loops appear to tighten more with the linearized model (top) than the explicit bending model (bottom).

We animated most of the figures and concepts in this paper and included them in a supplementary video .zip file, which is separate from this paper's main explanatory submission video.

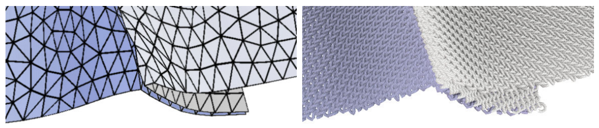


Fig. 17. Approximating a seam by folding one piece of cloth over another on the mesh-level, rather than the thread level.

6 DISCUSSION

Because our method is based on geometric interpolation of yarn vertices rather than an exact simulation of yarn-level geometry, it cannot exactly reproduce all yarn-level effects. Our method fundamentally assumes that the deformations sampled in the database are representative of the deformations in the online simulation. Because our per-vertex mesh strains only communicate deformations on the scale of a single triangle, the method will not be able to accurately react to fine-scale collision events like pulling on an individual thread. Similarly, our method relies on the mesh simulation to handle collisions; it will not exactly resolve object collisions on the level of the individual thread, and it will not exactly resolve collisions for very thick fabrics if the mesh is modeled as infinitely thin.

Our database currently also ignores time-dependent effects like hysteresis and damping, so repeating a mesh deformation will yield exactly the same fine-scale yarn arrangements. Our method interpolates the precomputed behaviors of a periodic yarn pattern, so we cannot yet simulate clothes consisting of completely aperiodic or disorganized threads, and it will be significantly more expensive to simulate ornate patterns with a large number of yarn vertices. We also do not yet handle non-periodic connections between different patches of cloth, so our method cannot yet sew together seams on the individual thread level. However, we *can* approximate larger seams by folding a piece of thin fabric over itself (Figure 17).

On the other hand, our geometric approximation affords us a number of advantages and leads to a few novel challenges. The data-driven approach completely avoids the expense of online collision handling, and its exploitation of periodic structures avoids redundant computations resulting from structurally similar yarn patterns. The method cuts the complexity of brute-force yarn level cloth by several orders of magnitude, and as far as we know, our method is the only way to simulate millions of yarn vertices at interactive rates. The interpolative nature of the approach also guarantees unconditional numerical stability, so the method cannot blow up even in unrealistic video game environments. The speed and detail of our approach for animating knitted garments also creates new research challenges: zooming out from a pattern made of millions of threads can cause aliasing patterns when many of them occupy a single pixel, potentially requiring novel geometry anti-aliasing techniques in the future.

Conclusion. We have presented a method for deforming yarn patterns in a mechanics-aware manner. It reproduces characteristic yarn-level cloth behaviors like knitted loops that tighten when the fabric is stretched. We introduced practical heuristics such as linearizing bending and limiting buckling, which make the method significantly more efficient and tunable by artists. The method is lightweight and GPU-parallelizable, so it is capable of animating millions of yarn vertices at real-time rates.

In the future, we are interested in new research challenges introduced by the massive scale of these yarn simulations. The method could further benefit from level-of detail approaches simplifying yarn geometry where it is not visible. Our technique might also be useful for research into deformation-dependent microfacet rendering or anti-aliasing techniques, for smoothly replacing extremely dense yarn geometry with an analytic or data-driven shading model.

ACKNOWLEDGMENTS

We wish to thank the anonymous reviewers and the members of the Visual Computing Group at IST Austria for their valuable feedback. We also thank Seddi Labs for providing the garment model with fold-over seams.

This research was supported by the Scientific Service Units (SSU) of IST Austria through resources provided by Scientific Computing. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No. 638176. Rahul Narain is supported by a Pankaj Gupta Young Faculty Fellowship and a gift from Adobe Inc.

REFERENCES

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques.* ACM, 43–54.

Jan Bender, Matthias Müller, and Miles Macklin. 2015. Position-Based Simulation Methods in Computer Graphics.. In *Eurographics (tutorials).* 8.

Miklós Bergou, Basile Audoly, Etienne Vouga, Max Wardetzky, and Eitan Grinspun. 2010. Discrete viscous threads. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 1–10.

Juan J Casafranca, Gabriel Cirio, Alejandro Rodríguez, Eder Miguel, and Miguel A Otaduy. 2020. Mixing yarns and triangles in cloth simulation. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 101–110.

Nuttapong Chentanez, Miles Macklin, Matthias Müller, Stefan Jeschke, and Tae-Yong Kim. 2020. Cloth and skin deformation with a triangle mesh based convolutional neural network. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 123–134.

Gabriel Cirio, Jorge Lopez-Moreno, David Miraut, and Miguel A Otaduy. 2014. Yarn-level simulation of woven cloth. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 1–11.

Gabriel Cirio, Jorge Lopez-Moreno, and Miguel A Otaduy. 2015. Efficient simulation of knitted cloth using persistent contacts. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 55–61.

Gabriel Cirio, Jorge Lopez-Moreno, and Miguel A Otaduy. 2016. Yarn-level cloth simulation with sliding persistent contacts. *IEEE Transactions on Visualization and Computer Graphics* 23, 2 (2016), 1152–1162.

Blender Online Community. 2020. *Blender - a 3D modelling and rendering package.* Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

Charles-Alban Deledalle, Loic Denis, Sonia Tabti, and Florence Tupin. 2017. Closed-form expressions of the eigen decomposition of 2 x 2 and 3 x 3 Hermitian matrices. (2017).

Petros Faloutsos, Michiel Van De Panne, and Demetri Terzopoulos. 1997. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics* 3, 3 (1997), 201–214.

Lin Gao, Yu-Kun Lai, Jie Yang, Zhang Ling-Xiao, Shihong Xia, and Leif Kobbelt. 2019. Sparse data driven mesh deformation. *IEEE transactions on visualization and computer graphics* (2019).

Eitan Grinspun, Yotam Gingold, Jason Reisman, and Denis Zorin. 2006. Computing discrete shape operators on general meshes. In *Computer Graphics Forum*, Vol. 25. Wiley Online Library, 547–556.

Sunil Hadap, E Bongarter, Pascal Volino, and Nadia Magnenat-Thalmann. 1999. Animating wrinkles on clothes. In *Proceedings Visualization'99 (Cat. No. 99CB37067).* IEEE, 175–523.

Jonathan Hoffman, Matt Kuruc, Junyi Ling, Alex Marino, George Nguyen, and Sasha Ouellet. 2020. Hypertextural Garments on Pixar's Soul. In *ACM SIGGRAPH 2020 Talks.* Association for Computing Machinery, Article 75.

Doug L James. 2020. Phong deformation: a better C0 interpolant for embedded deformation. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 56–1.

Ning Jin, Yilin Zhu, Zhenglin Geng, and Ronald Fedkiw. 2020. A Pixel-Based Framework for Data-Driven Clothing. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 135–144.

Jonathan M Kaldor, Doug L James, and Steve Marschner. 2008. Simulating knitted cloth at the yarn level. In *ACM Transactions on Graphics (TOG).* Vol. 27. 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.

Ladislav Kavan, Dan Gerszewski, Adam W Bargteil, and Peter-Pike Sloan. 2011. Physics-inspired upsampling for cloth simulation in games. In *ACM SIGGRAPH 2011 papers.* 1–10.

Josef Kiendl, Ming-Chen Hsu, Michael CH Wu, and Alessandro Reali. 2015. Isogeometric Kirchhoff–Love shell formulations for general hyperelastic materials. *Computer Methods in Applied Mechanics and Engineering* 291 (2015), 280–303.

Yuki Koyama, Kenshi Takayama, Nobuyuki Umetani, and Takeo Igarashi. 2012. Real-time example-based elastic deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 19–24.

Jonathan Leaf, Rundong Wu, Eston Schweickart, Doug L James, and Steve Marschner. 2018. Interactive design of periodic yarn-level cloth patterns. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–15.

Wan-Chun Ma, Andrew Jones, Jen-Yuan Chiang, Tim Hawkins, Sune Frederiksen, Pieter Peers, Marko Vukovic, Ming Ouhyoung, and Paul Debevec. 2008. Facial performance synthesis using deformation-driven polynomial displacement maps. *ACM Transactions on Graphics (TOG)* 27, 5 (2008), 1–10.

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based elastic materials. In *ACM SIGGRAPH 2011 papers.* 1–8.

Zahra Montazeri, Søren B Gammelmark, Shuang Zhao, and Henrik Wann Jensen. 2020. A practical ply-based appearance model of woven fabrics. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–13.

Zahra Montazeri, Chang Xiao, Yun Fei, Changxi Zheng, and Shuang Zhao. 2019. Mechanics-Aware Modeling of Cloth Appearance. *IEEE transactions on visualization and computer graphics* 27, 1 (2019), 137–150.

Matthias Müller and Nuttapong Chentanez. 2010. Wrinkle Meshes.. In *Symposium on Computer Animation.* Madrid, Spain, 85–91.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

Matthias Muller, Matthias Teschner, and Markus Gross. 2004. Physically-based simulation of objects represented by surface meshes. In *Proceedings Computer Graphics International, 2004.* IEEE, 26–33.

Rahul Narain, Tobias Pfaff, and James F O'Brien. 2013. Folding and crumpling adaptive sheets. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 51.

Rahul Narain, Armin Samii, and James F O'Brien. 2012. Adaptive anisotropic remeshing for cloth simulation. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 152.

José M Pizana, Alejandro Rodríguez, Gabriel Cirio, and Miguel A Otaduy. 2020. A Bending Model for Nodal Discretizations of Yarn-Level Cloth. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 181–189.

Damien Rohmer, Tiberiu Popa, Marie-Paule Cani, Stefanie Hahmann, and Alla Sheffer. 2010. Animation wrinkling: augmenting coarse cloth simulations with realistic-looking wrinkles. *ACM Transactions on Graphics (TOG)* 29, 6 (2010), 1–8.

Nadine Abu Rumman and Marco Fratarcangeli. 2016. State of the Art in Skinning Techniques for Articulated Deformable Characters.. In *VISIGRAPP (1: GRAPP)*. 200–212.

Rosa M Sánchez-Banderas, Alejandro Rodríguez, Héctor Barreiro, and Miguel A Otaduy. 2020. Robust eulerian-on-lagrangian rods. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 59–1.

Igor Santesteban, Miguel A Otaduy, and Dan Casas. 2019. Learning-Based Animation of Clothing for Virtual Try-On. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 355–366.

Christian Schumacher, Bernhard Thomaszewski, Stelian Coros, Sebastian Martin, Robert Sumner, and Markus Gross. 2012. Efficient simulation of example-based materials. In *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*. Citeseer, 1–8.

Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. 2007. Hybrid simulation of deformable solids. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 81–90.

Georg Sperl, Rahul Narain, and Chris Wojtan. 2020. Homogenized Yarn-Level Cloth. *ACM Transactions on Graphics (TOG)* 39, 4 (2020).

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 205–214.

Raquel Vidaurre, Igor Santesteban, Elena Garces, and Dan Casas. 2020. Fully Convolutional Graph Neural Networks for Parametric Virtual Try-On. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 145–156.

Kevin Wampler. 2016. Fast and reliable example-based mesh IK for stylized deformations. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–12.

Huamin Wang, Florian Hecht, Ravi Ramamoorthi, and James F O'Brien. 2010. Example-based wrinkle synthesis for clothing animation. In *ACM SIGGRAPH 2010 papers*. 1–8.

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2009. Deforming meshes that split and merge. In *ACM SIGGRAPH 2009 papers*. 1–10.

Chris Wojtan and Greg Turk. 2008. Fast viscoelastic behavior with thin features. In *ACM SIGGRAPH 2008 papers*. 1–8.

Kui Wu and Cem Yuksel. 2017. Real-time fiber-level cloth rendering. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 1–8.

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), 1–12.

Shuang Zhao, Fujun Luan, and Kavita Bala. 2016. Fitting procedural yarn models for realistic cloth rendering. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.

Evgeny Zuenko and Matthias Harders. 2019. Wrinkles, Folds, Creases, Buckles: Small-Scale Surface Deformations as Periodic Functions on 3D Meshes. *IEEE Transactions on Visualization and Computer Graphics* (2019).

Javier S Zurdo, Juan P Brito, and Miguel A Otaduy. 2012. Animating wrinkles by example on non-skinned cloth. *IEEE Transactions on Visualization and Computer Graphics* 19, 1 (2012), 149–158.

A OPTIMIZATION MID-SURFACE

The fundamental forms $\mathbf{I}$ and $\mathbf{II}$ define the midsurface $\boldsymbol{\varphi}$ used in the optimization of Section 3. We construct $\boldsymbol{\varphi}$ like Sperl et al. [2020] by solving for $\nabla\boldsymbol{\varphi} \approx \boldsymbol{RS}$ in the least-squares sense (with $\nabla\boldsymbol{\varphi} = \boldsymbol{RS}$ exactly for single curvature). To do so, we compute a rotation $\boldsymbol{R}$ describing curvature and an in-plane deformation matrix $\boldsymbol{S}$ from the two input fundamental forms $\mathbf{I}$ and $\mathbf{II}$. The 3×2 matrix $\boldsymbol{S}$ is computed from the principal square root of $\mathbf{I}$ as

$$\boldsymbol{S} = \begin{bmatrix} \multicolumn{2}{c}{\sqrt{\mathbf{I}}} \\ 0 & 0 \end{bmatrix}. \tag{10}$$

To compute the rotation, we first compute normal derivatives

$$\nabla\boldsymbol{n} = \begin{bmatrix} n_{1,1} & n_{1,2} \\ n_{2,1} & n_{2,2} \end{bmatrix} = -\left(\sqrt{\mathbf{I}}\right)^{-\top} \mathbf{II}. \tag{11}$$

Then, with

$$a = \sum_{\alpha=1}^{2} n_{1,\alpha} X_\alpha, \qquad b = \sum_{\alpha=1}^{2} n_{2,\alpha} X_\alpha, \qquad r = \sqrt{a^2 + b^2}, \tag{12}$$

we compute $\boldsymbol{R}(X_1, X_2)$ as

$$\boldsymbol{R} = \begin{bmatrix} 1 - \frac{1}{2}a^2 \operatorname{sinc}(r/2)^2 & -\frac{1}{2}ab \operatorname{sinc}(r/2)^2 & a \operatorname{sinc}(r) \\ -\frac{1}{2}ab \operatorname{sinc}(r/2)^2 & 1 - \frac{1}{2}b^2 \operatorname{sinc}(r/2)^2 & b \operatorname{sinc}(r) \\ -a \operatorname{sinc}(r) & -b \operatorname{sinc}(r) & \cos(r) \end{bmatrix}. \tag{13}$$

Finally, we solve for $\boldsymbol{\varphi}$ with the Poisson equation $\nabla^2\boldsymbol{\varphi} = \nabla \cdot \boldsymbol{RS}$ and natural boundary conditions discretized on a grid. For further details, we refer to the paper and supplementary document of [Sperl et al. 2020]. The sliding constraint (3) is thus $\tilde{\boldsymbol{u}} \cdot (\boldsymbol{RS}\,\boldsymbol{N}) = 0$.

B PULLBACK INTO MATERIAL SPACE

Here, we detail the Newton iteration to transform optimized yarn geometry from Section 3 back into material space. The goal is to find $\hat{\boldsymbol{X}}$ s.t. $\boldsymbol{x} = \overline{\boldsymbol{x}}(\hat{\boldsymbol{X}})$, which we do using Newton's method on $\boldsymbol{f}(\hat{\boldsymbol{X}}) = \boldsymbol{x} - \overline{\boldsymbol{x}}(\hat{\boldsymbol{X}})$, for which we need the gradient $\nabla\boldsymbol{f}$.

The mapping $\overline{\boldsymbol{x}}$ is defined as

$$\overline{\boldsymbol{x}}(\hat{\boldsymbol{X}}) = \boldsymbol{\varphi}(\hat{X}_1, \hat{X}_2) + \hat{X}_3\, \boldsymbol{n}(\hat{X}_1, \hat{X}_2), \tag{14}$$

where $\boldsymbol{\varphi}$ is the deformed midsurface and $\boldsymbol{n}$ its normal. Its gradient is

$$\nabla\overline{\boldsymbol{x}}(\hat{\boldsymbol{X}}) = \left[\nabla\boldsymbol{\varphi}(\hat{X}_1, \hat{X}_2) + \hat{X}_3\, \nabla\boldsymbol{n}(\hat{X}_1, \hat{X}_2), \quad \boldsymbol{n}(\hat{X}_1, \hat{X}_2)\right]. \tag{15}$$

By definition (see [Sperl et al. 2020]) we have

$$\nabla\boldsymbol{\varphi} = \boldsymbol{RS}, \tag{16}$$

and we compute $\nabla\boldsymbol{n}$ with (11). The Newton iterations are

$$\hat{\boldsymbol{X}}_{i+1} = \hat{\boldsymbol{X}}_i - \nabla\boldsymbol{f}(\hat{\boldsymbol{X}}_i)^{-1} \boldsymbol{f}(\hat{\boldsymbol{X}}_i) \tag{17}$$

starting from the rest configuration $\hat{\boldsymbol{X}}_0 = \boldsymbol{X}$.

In our experiments, the iterations converge to a fraction of the yarn radius within three iterations and the cost is negligible compared to the elastostatic optimization. For pure in-plane deformations, $\mathbf{II} = \mathbf{0}$, and thus $\boldsymbol{R} = \mathrm{Id}$ and $\nabla\boldsymbol{f} = -\begin{bmatrix} \sqrt{\mathbf{I}} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$, simplifying the pullback into the constant expression

$$\hat{\boldsymbol{X}} = \boldsymbol{X} + \begin{bmatrix} \left(\sqrt{\mathbf{I}}\right)^{-1} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \tilde{\boldsymbol{u}}. \tag{18}$$

C BENDING MODELS

We briefly outline the two bending models we experimented with: a 4D combination of in-plane and bending deformations (19), and the linearization of bending as stretching (22).

Due to tileability constraints, Sperl et al. [2020] only generate data for singly curved deformations and split the contributions of general bending onto data sampled for bending along either the X_1 or X_2 directions. We apply the same idea to define our "ground-truth" bending model. For brevity, we write $\boldsymbol{s} = (s_x, s_a, s_y)$. Then, we compute two sets of data for combined in-plane deformation with the two bending directions: $\Delta\boldsymbol{Q}_{X_1}(\boldsymbol{s}, \lambda)$ and $\Delta\boldsymbol{Q}_{X_2}(\boldsymbol{s}, \lambda)$. With the eigenvalues λ_1 and λ_2 of $\mathbf{II}$ and the cosine c of the angle between the X_1 axis and the eigenvector corresponding to λ_1, we distribute the bending contributions to get

$$\begin{aligned}\Delta \boldsymbol{Q}(\boldsymbol{s}, \mathbf{II}) = {} & c^2\, \Delta \boldsymbol{Q}_{X_1}(\boldsymbol{s}, \lambda_1) + (1 - c^2)\, \Delta \boldsymbol{Q}_{X_1}(\boldsymbol{s}, \lambda_2) \\ & + c^2\, \Delta \boldsymbol{Q}_{X_2}(\boldsymbol{s}, \lambda_2) + (1 - c^2)\, \Delta \boldsymbol{Q}_{X_2}(\boldsymbol{s}, \lambda_1) \\ & - \Delta \boldsymbol{Q}_{X_1}(\boldsymbol{s}, 0)\end{aligned} \tag{19}$$

Since both datasets include the displacements caused by pure in-plane deformations $\lambda_1 = \lambda_2 = 0$, we have to subtract these displacements once to avoid double counting. This model exactly reproduces the data samples for bending aligned with either X_1 or X_2. However, it requires five 4D-texture look-ups and is thus much more expensive than the linearized model requiring only a single 3D-texture look-up. For the speed comparison mentioned in Section 5, we used two sets of 9^4 samples for the 4D model and 9^3 samples for the linearized model. The linearized model was still 7× faster even when increasing its sample density to 15^3.

C.1 Linearized Bending

For a thin shell represented by a midsurface $\boldsymbol{\varphi}$ with normal $\boldsymbol{n}$, its full domain is

$$\boldsymbol{x} = \boldsymbol{\varphi} + h\,\boldsymbol{n}, \tag{20}$$

with the normal coordinate $h \in [-H/2, H/2]$ for shell thickness H. Its right Cauchy-Green deformation tensor is

$$\nabla \boldsymbol{x}^\top \nabla \boldsymbol{x} = \nabla \boldsymbol{\varphi}^\top \nabla \boldsymbol{\varphi} + h\,(\nabla \boldsymbol{\varphi}^\top \nabla \boldsymbol{n} + \nabla \boldsymbol{n}^\top \nabla \boldsymbol{\varphi}) + O(h^2), \tag{21}$$

which can be interpreted as a first fundamental form $\mathbf{I}(h)$ depending quadratically on h. The quadratic term is commonly neglected (e.g. [Kiendl et al. 2015]). Additionally, we see that $\nabla \boldsymbol{\varphi}^\top \nabla \boldsymbol{\varphi} = \mathbf{I}$ and similarly $\nabla \boldsymbol{\varphi}^\top \nabla \boldsymbol{n} = \nabla \boldsymbol{n}^\top \nabla \boldsymbol{\varphi} = -\mathbf{II}$ are the fundamental forms of $\boldsymbol{\varphi}$. As a result, we get the linearized expression

$$\mathbf{I}(h) \approx \mathbf{I} - 2\,h\,\mathbf{II}. \tag{22}$$

Then, with precomputed data $\Delta \boldsymbol{Q}_{\boldsymbol{s}}(\boldsymbol{s})$ for in-plane deformations $\boldsymbol{s}$, the linearized bending model is

$$\Delta \boldsymbol{Q}(\mathbf{I}, \mathbf{II}) = \Delta \boldsymbol{Q}_{\boldsymbol{s}}(\boldsymbol{s}(\mathbf{I} - 2\,h\,\mathbf{II})). \tag{23}$$