Real-Time Knit Deformation and Rendering

TAO HUANG*, LIGHTSPEED, USA
HAOYANG SHI*, University of Utah, USA
MENGDI WANG*, LIGHTSPEED, USA
YUXING QIU, LIGHTSPEED, USA
YIN YANG, University of Utah, USA
KUI WU, LIGHTSPEED, USA

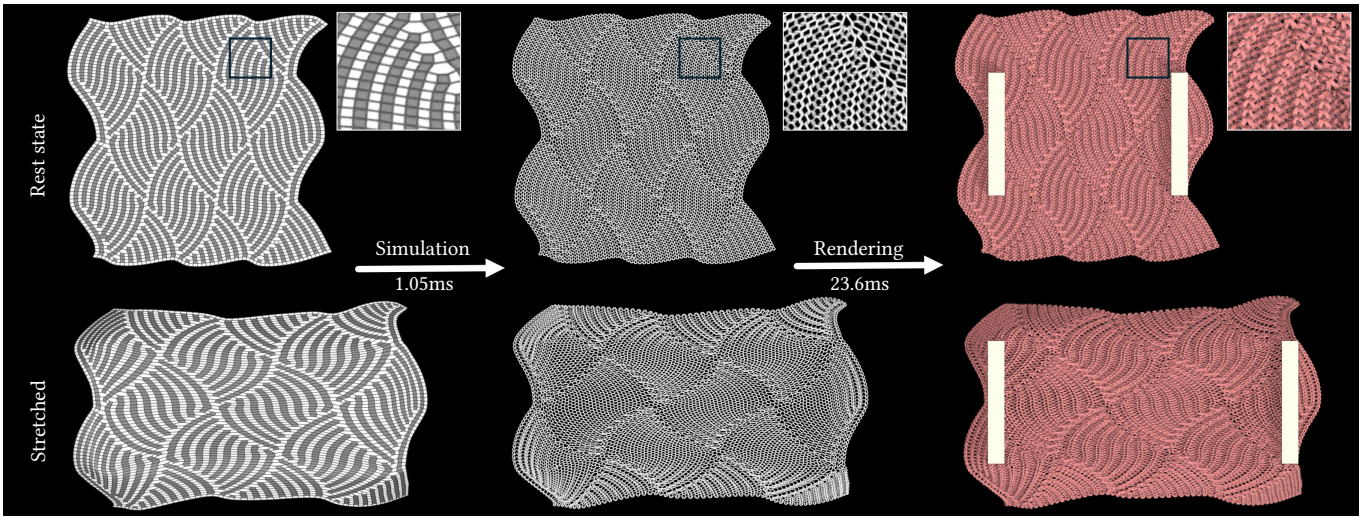


Fig. 1. *Given the animated stitch mesh with a non-periodic Flame pattern (left), our system produces faithful yarn geometry (middle) based on underlying mesh deformation and high-quality knit rendering with fiber-level details under environmental light (right). The simulation and rendering take 1.05 and 23.6 ms, respectively, on an Nvidia RTX3090.*

The knit structure consists of interlocked yarns, with each yarn comprising multiple plies comprising tens to hundreds of twisted fibers. This intricate geometry and the large number of geometric primitives present substantial challenges for achieving high-fidelity simulation and rendering in real-time applications. In this work, we introduce the first real-time framework that takes an animated stitch mesh as input and enhances it with yarn-level simulation and fiber-level rendering. Our approach relies on a knot-based representation to model interlocked yarn contacts. The knot positions are interpolated from the underlying mesh, and associated yarn control points are optimized using a physically inspired energy formulation, which is solved through a GPU-based Gauss-Newton scheme for real-time performance. The optimized control points are sent to the GPU rasterization pipeline and rendered as yarns with fiber-level details. In real-time rendering, we introduce several decomposition strategies to enable realistic lighting effects on complex knit structures, even under environmental lighting, while maintaining computational and memory efficiency. Our simulation faithfully reproduces yarn-level structures under deformations, e.g., stretching and shearing, capturing interlocked yarn behaviors. The rendering pipeline achieves near-ground-truth visual quality while being 120,000× faster than path tracing reference with fiber-level geometries. The whole system provides real-time performance and has been evaluated through various application scenarios, including knit simulation for small patches and full garments and yarn-level relaxation in the design pipeline.

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

Additional Key Words and Phrases: Knit, stitch, yarn-level modeling, fiber details, real-time, mechanic-aware

ACM Reference Format:
Tao Huang, Haoyang Shi, Mengdi Wang, Yuxing Qiu, Yin Yang, and Kui Wu. 2025. Real-Time Knit Deformation and Rendering. *ACM Trans. Graph.* 44, 4 (August 2025), 12 pages. https://doi.org/10.1145/3731184

*Authors contributed equally to this work.

Authors' addresses: Tao Huang, tao_huang@ucsb.edu, LIGHTSPEED, Los Angeles, CA, USA; Haoyang Shi, haoyang.shi@utah.edu, University of Utah, Salt Lake City, UT, USA; Mengdi Wang, mengdi.wang@gatech.edu, LIGHTSPEED, Los Angeles, CA, USA; Yuxing Qiu, yuxqiu@gmail.com, LIGHTSPEED, Los Angeles, CA, USA; Yin Yang, yangzzzy@gmail.com, University of Utah, Salt Lake City, UT, USA; Kui Wu, walker.kui.wu@gmail.com, LIGHTSPEED, Los Angeles, CA, USA.

ACM 0730-0301/2025/8-ART
https://doi.org/10.1145/3731184

1 INTRODUCTION

Knitting is an additive manufacturing technique that creates garments by interlocking yarn loops, widely applied in everyday products such as clothing, home decor, and personal accessories such as hats and bags. By combining different types of stitches, knit designs can achieve intricate patterns, textures, and visual effects that enhance both the functionality and aesthetic appeal of garments and accessories. However, knit design remains a tedious trial-and-error workflow. Artists must wait until a physical sample is manufactured to observe how the stitches deform and how the yarn behaves visually, making it difficult to predict the final appearance and structure during the design process. Industry provides software tailored specifically for their machines, such as Shima Seiki [Shima Seiki 2011] and Stoll [Stoll 2011], as well as software solutions aimed at enhancing design and production processes in knitting, such as PaintKnit [Logica 2020], which rely on physically inspired relaxation and advanced fabric rendering. However, these tools lack physics-based simulation and high-fidelity rendering capabilities, preventing designers from achieving accurate previews.

The knit structure consists of interlocked yarns, where each yarn is made up of multiple plies containing tens to hundreds of fibers twisted together. This intricate geometry, combined with a large number of geometric primitives, poses significant challenges to achieving high-fidelity simulation and rendering in real time. Unfortunately, most previous work suffers from high computational costs and is not suitable for real-time scenarios [Montazeri et al. 2021; Sperl et al. 2020; Yuan et al. 2024; Zhu et al. 2023]. Real-time methods either omit physically based fiber shading and global illumination in rendering [Wu and Yuksel 2017] or support only specific stitch types and simulation scenarios [Sperl et al. 2021].

In this work, we present the first real-time framework that seamlessly integrates yarn-level simulation and fiber-level rendering with details, providing a high-quality knit appearance prediction. Like Sperl et al. [2020], our simulator assumes that the input cloth mesh animation already contains cloth-scale features and adds yarn-level detail purely based on mesh-level deformation. Instead of relying on precomputed data, we propose a knot-based representation to model yarn interactions at contacts. Each knot is associated with a set of control points that define individual yarn splines. During the simulation, knot positions are interpolated from the underlying mesh. Then, with knot positions fixed, a GPU-based Gauss-Newton scheme is deployed to optimize the associated control points, minimizing angles between yarn segments and ensuring consistent yarn lengths so as to effectively preserve the physical accuracy and visual fidelity of the deformed yarn structure. Given the resulting control points from the simulation, we render yarns as camera-facing strips via the GPU rasterization pipeline. Due to the distinct geometric distributions, fiber geometries are categorized into regular and flyaway fibers, and each is represented through a dedicated set of textures. We further decompose the shading process into yarn and fiber levels to efficiently handle multiple scattering with fiber-level details and support environment lighting.

Our simulation pipeline requires no pre-computed data or periodic boundary conditions, making it versatile enough for users to design a wide range of knit patterns interactively. Our knot-based representation achieves a trade-off between efficiency and accuracy by ignoring out-of-plane force and provides 7,680× speedup to full yarn-level simulation. Our rasterization pipeline achieves near-ground-truth visual fidelity while being 120,000× faster than path tracing reference with fiber-level geometries. The whole simulation and rendering system is fast and delivers real-time performance, making it an ideal solution for various application scenarios, such as pattern design and knitted garment animation. We demonstrate the robustness and adaptability of our framework through diverse examples, including knit simulations for small patches and full garments, as well as yarn-level relaxation within the design pipeline.

2 RELATED WORK

This section briefly reviews knit design frameworks and the techniques for yarn-level simulation and rendering.

Knit Design. While industrial knitting machine companies such as Shima Seiki [Shima Seiki 2011] and Stoll [Stoll 2011] provide knitting design software tailored specifically for their machines, these tools lack physics-based simulation and high-fidelity rendering capabilities, preventing designers from achieving accurate previews. In academia, Yuksel et al. [2012] first introduced *stitch meshes*, which abstract interlocked stitch structures into faces (typically quads or pentagons) and were initially developed for rendering and animation purposes. This framework has since been extended for hand-knitting [Wu et al. 2019], support for arbitrary input meshes [Wu et al. 2018], machine knittability [Narayanan et al. 2019], wearability [Wu et al. 2022], sensing [Luo et al. 2021; Zlokapa et al. 2022], and pneumatic actuation [Luo et al. 2022]. Significant efforts have also been made to develop more intuitive and automated design pipelines, such as using shape primitives [Kaspar et al. 2019; McCann et al. 2016] and cut-and-sew patterns [Kaspar et al. 2021]. Recently, Twigg-Smith et al. [2024a,b] introduced simplified mass-spring systems to relax *TopoKnit* [Kapllani et al. 2021] structures in 2D. However, these methods are limited to simple 2D stretching and cannot handle more complex 3D scenarios. We propose a knot-based representation of the concept of TopoKnit, but we extend it by incorporating additional contact and control points that allow for reconstructing and optimizing natural yarn shapes when the underlying mesh is deformed.

Knit Simulation. Kaldor et al. [2008, 2010] were the first to simulate knitted structures by explicitly modeling yarn-yarn interactions. Several subsequent works sought to accelerate these simulations through various approaches: assuming persistent contact [Cirio et al. 2014, 2016], resolving collisions with background grids via material point method [Jiang et al. 2017], mixing yarn-level and triangle-based representations [Casafranca et al. 2020], homogenizing knit behavior using thin shells [Sperl et al. 2020; Yuan et al. 2024], and leveraging neural networks for efficiency [Feng et al. 2024]. Sageman-Furnas et al. [2019] use the Chebyshev net to simulate knotted structures instead of considering yarn-yarn contacts. The stitch mesh [Yuksel et al. 2012] introduces a two-stage mesh-based and yarn-level relaxation process to achieve realistic appearances. However, yarn-level relaxation remains computationally expensive due to its reliance on full yarn-level simulation. While Leaf et al.

[2018] introduced a GPU-based parallelization for small yarn-level patch relaxation, it remains computationally challenging to simulate full garments due to complex yarn-yarn collision and high degrees of freedom, making such approaches impractical for real-time applications. Recently, Sperl et al. [2021] proposed using triangle strains to interpolate precomputed yarn geometry in real time. However, their method shares the same limitations as other data-driven approaches, including dependency on precomputed data and limited generalization. Inspired by Hsu et al. [2024], our framework computes yarn deformation on-the-fly rather than relying on precomputed data.

Knit Rendering. Traditionally, knit structures are rendered as thin sheets using data-driven approaches, such as *Bidirectional Texture Functions (BTF)* [Dana et al. 1999] or neural networks [Kuznetsov et al. 2021]. Another category of methods converts entire yarn with fiber geometries into volume data for rendering [Gröller et al. 1995; Jakob et al. 2010; Khungurn et al. 2016], but the costly format conversion limits their efficiency in dynamic scenes. Recent approaches that explicitly render geometries demonstrate a remarkable level of realism, using 3-D spatial projection based on underlying surfaces [Hoffman et al. 2020], fiber-level geometries [Khungurn et al. 2016; Luan et al. 2017], ply with pre-computed textures [Montazeri et al. 2020], or yarns with aggregated shading models [Zhu et al. 2023]. Despite their quality, these methods are computationally and storage-intensive, making them impractical for real-time applications. To address efficiency, Wu and Yuksel [2017, 2019] utilize hardware tessellation to create fiber geometries directly on the GPU, but their method does not account for global illumination and environmental lighting, which limits their visual fidelity. To efficiently approximate complex light scattering between hairs, Zinke et al. [2008] introduced *Dual Scattering* to decompose multiple scattering into global and local scattering using simplified approximations, which is further extended by Ren et al. [2010] to support environment lighting. However, this approach is computationally expensive for knit rendering with millions of fiber segment geometries.

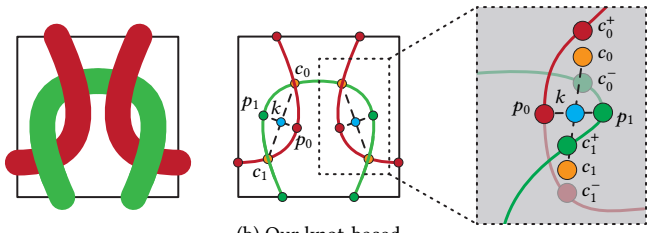
3 PHYSICS-AWARE STITCH OPTIMIZATION

This section first introduces our 2.5D knot-based representation to model yarn structure within the underlying mesh and then describes an optimization scheme that ensures visually plausible yarn geometry when the mesh undergoes deformation.

3.1 Knot-based Yarn Representation

Yarn loop is the fundamental structural element of knitted textiles. A new loop is created when a yarn is drawn through a pre-existing loop. We adopt *Stitch Mesh* [Yuksel et al. 2012] to abstract the knit structure using quad faces. As illustrated in Fig. 2a, the green yarn connects to the bottom edge, serving as the base of the loop, while the red yarn interlocks with the green yarn to form a stitch.

To achieve a tradeoff between efficiency and accuracy, we introduce a 2.5D knot-based representation for the interlocking structure between green and red yarns on the stitch face. The interlocking structure is centered at node k (●) and defined by three contact points, as shown in Fig. 2b. The top and bottom contact points, c_0 and c_1 (●), represent crossings where the two yarns overlap, with one yarn positioned on top of the other. The contact, located at node



(a) Stitch face (b) Our knot-based representation (c) 3D Zoom-in

Fig. 2. *Given the quad stitch mesh face (a) with its associated yarn pieces, we abstract it into a 2D knot-based representation (b), which can then be converted back into a 3D yarn curve (c). In particular, orange circles in (c) indicate contact points on the face plane, and red and green circles are actual yarn control points off the plane along positive and negative normal directions.*

k, separates the two yarns and maintains a stable spacing between them. p_0 and p_1 are control points for the yarn spline corresponding to the contact node k. Node k, contact points c_0 and c_1, and control points p_0 and p_1 lie on the plane of the stitch face. These points jointly define the interactions between yarns, ensuring a stable twisting structure.

Our representation assumes the following geometric properties to simplify the interlocked structure: (1) The distance between control points p_0 and p_1 is constant at $2r$, where r refers to yarn radius. This fixed distance reflects the physical compression of yarns when they contact. (2) The direction $\mathbf{d} = (\mathbf{x}_{c_0} - \mathbf{x}_{c_1})/||\mathbf{x}_{c_0} - \mathbf{x}_{c_1}||$ between contact points c_0 and c_1 is perpendicular to the contact direction $\mathbf{b} = (\mathbf{x}_{p_0} - \mathbf{x}_{p_1})/||\mathbf{x}_{p_0} - \mathbf{x}_{p_1}||$, where p_0 and p_1 on a stitch face plane, such that $\mathbf{b} = \mathbf{d} \times \mathbf{n}$, where $\mathbf{n}$ is the face normal direction and $\mathbf{x}_\bullet$ refers to the position. (3) The knot is a symmetric structure, with its position centered relative to the four planar points c_0, c_1, p_0, and p_1. Hence, given a knot centered at $\mathbf{x}_k$, positions of the planar points are defined as:

$$\begin{aligned} \mathbf{x}_{c_0} &= \mathbf{x}_k + l\,\mathbf{d}, \quad & \mathbf{x}_{c_1} &= \mathbf{x}_k - l\,\mathbf{d}, \\ \mathbf{x}_{p_0} &= \mathbf{x}_k + r\,\mathbf{b}, \quad & \mathbf{x}_{p_1} &= \mathbf{x}_k - r\,\mathbf{b}, \end{aligned} \tag{1}$$

where l is the half distance between crossing points c_0 and c_1. The contact points can be further offset by a distance of r along the direction $\mathbf{n}$ to make control points for the yarn curves, as shown in Fig. 2c. For instance, control points above and below the contact point c_0 can be computed as:

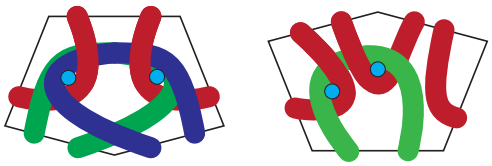
$$\mathbf{x}_{c_0^+} = \mathbf{x}_k + l\,\mathbf{d} + r\,\mathbf{n}, \quad \mathbf{x}_{c_0^-} = \mathbf{x}_k + l\,\mathbf{d} - r\,\mathbf{n}. \tag{2}$$

By connecting yarn control points c_0^+, p_0, and c_1^-, the yarn geometry can be recovered in 3D.

Our knot-based representation can also model other types of stitches, as shown in Fig. 3. For a *decrease* stitch, two loops from the bottom share the same knot position, but each yarn maintains its own contact points and control points. In contrast, *yarn-over* increase introduces an additional loop without pulling it through an existing loop, so it only has two knots, similar to a regular knit.

3.2 Knot-based Yarn Optimization

Given the underlying mesh driven by a cloth simulator, the embedded yarn must also deform to reflect the changes accordingly.



(a) Decrease stitch (b) Increase stitch

Fig. 3. *Knots at contacts for decrease (a) and yarn-over increase (b).*

However, as noted by Sperl et al. [2021], uniformly interpolating yarn control points along with the mesh fails to account for yarn-level physics. Specifically, yarn should contact with each other when the fabric is stretched. On the other hand, we observe that yarn sliding is minimal due to the high friction caused by the fuzzy flyaway fibers around the yarn. As shown in Fig. 4, while yarn geometry cannot be stretched uniformly, the knots exhibit a uniform distribution when stretched, which is another motivation for our knot-based representation.

To avoid the expensive full yarn-level simulation [Cirio et al. 2014; Leaf et al. 2018], we propose a novel physics-aware knot-based yarn optimization scheme to recover physically plausible yarn curves when deforming. Our scheme begins by interpolating the knot positions from the deformed underlying mesh. Then, we optimize yarn control point positions $\mathbf{x}$ by minimizing the following energies:

$$\underset{\mathbf{x}^i}{\text{argmin}} \quad w_{\text{angle}} \sum_i^m E^i_{\text{angle}} + w_{\text{length}} \sum_i^n E^i_{\text{length}} \tag{3}$$

$$s.t. \begin{cases} (\mathbf{x}^j_{c_0} + \mathbf{x}^j_{c_1})/2 = \mathbf{x}_k \\ (\mathbf{x}^j_{p_0} + \mathbf{x}^j_{p_1})/2 = \mathbf{x}_k \\ ||\mathbf{x}^j_{p_0} - \mathbf{x}^j_{p_1}|| = 2r \\ (\mathbf{x}^j_k - \mathbf{x}^j_{p_0}) \cdot (\mathbf{x}^j_{c_0} - \mathbf{x}^j_k) = 0 \end{cases} \quad \forall \text{ knot } j, \tag{4}$$

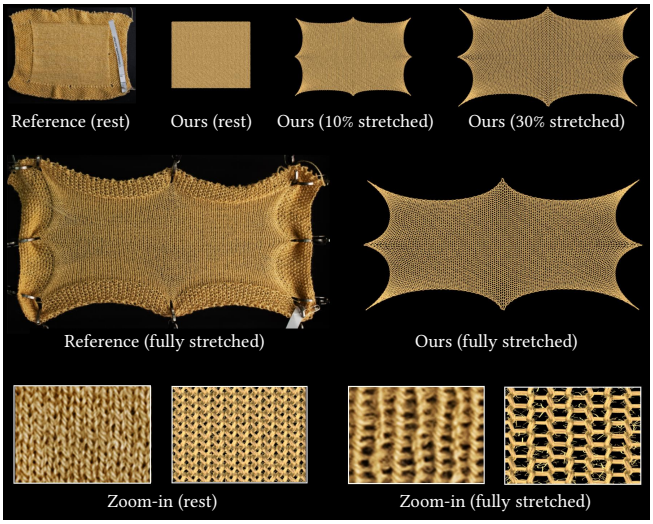


Fig. 4. *Compared to the image of a real knit sample from KnitDB captured by Hofmann et al. [2019] (reference). Our method can reproduce the knit appearance and yarn-level deformation in rest and stretch states under environmental light.*

with $w_\bullet$ being the corresponding weights, where our energies enforce minimal deviations in yarn shape from its original configuration in terms of angle between consecutive yarn segments and segment length:

$$E^i_{\text{angle}} = \left(\frac{\mathbf{x}^{i+1}-\mathbf{x}^i}{||\mathbf{x}^{i+1}-\mathbf{x}^i||} \cdot \frac{\mathbf{x}^{i+2}-\mathbf{x}^{i+1}}{||\mathbf{x}^{i+2}-\mathbf{x}^{i+1}||} - \frac{\mathbf{x}^{i+1}_0-\mathbf{x}^i_0}{||\mathbf{x}^{i+1}_0-\mathbf{x}^i_0||} \cdot \frac{\mathbf{x}^{i+2}_0-\mathbf{x}^{i+1}_0}{||\mathbf{x}^{i+2}_0-\mathbf{x}^{i+1}_0||}\right)^2 \tag{5}$$

$$E^i_{\text{length}} = (||\mathbf{x}^{i+1} - \mathbf{x}^i|| - ||\mathbf{x}^{i+1}_0 - \mathbf{x}^i_0||)^2 , \tag{6}$$

where $\mathbf{x}^i_0$ and $\mathbf{x}^i$ indicate the rest and current position of control point i, respectively. The rest positions of yarn control points are predefined as stitch templates.

This pair of energies is designed to encourage the yarn to preserve its original loop configuration because the yarn tends to retain its curly shape even after unraveling, a phenomenon known as *fiber memory*, as shown in Fig. 5. This plasticity arises when the knit structure is maintained for a period and reinforced further after the fabric undergoes washing and drying processes.

Fig. 5. *Yarn persists in curly shape even after unraveling.*

The optimization is further subject to several geometric constraints (Eq. 4), derived from our knot representation, formalized in Eq. 1 and 2. Note that $\mathbf{x}^i$ and $\mathbf{x}^j_{p_0}$, $\mathbf{x}^j_{p_1}$, $\mathbf{x}^j_{c_0^+}$, and $\mathbf{x}^j_{c_0^-}$ are the same set of out-of-plane control point positions with different coordinates used to constrain the optimization in Eq. 4, and their representations are interchangeable according to Eq. 1 and 2.

Our knot interpolation operates independently on each stitch face, making it straightforward to parallelize. Simple quadratic energies allow for an efficient Gauss-Newton scheme on the GPU. Despite its simplicity, the method ensures smooth and realistic yarn shapes, leveraging physics-aware energy formulations and constraints derived from our knot representation.

4 YARN RENDERING WITH FIBER-LEVEL DETAILS

We review the fiber shading model and dual scattering and describe our real-time rendering pipeline with environment lighting.

4.1 Preliminary

Single fiber shading model. As suggested by Zhu et al. [2023], the interaction between light and a single fiber can be decomposed into reflection R and transmission TT, along with an additional diffuse lobe D to approximate multiple scattering within the fiber. The fiber bidirectional curve scattering distribution function (BCSDF), denoted as f, is defined as:

$$f(\theta_i, \phi_i, \theta_o, \phi_o) = \sum_{p \in \{R,TT,D\}} A_p \; M_p(\theta_h) \; N_p(\phi), \tag{7}$$

where p denotes different types of lobes, $\phi = \phi_o - \phi_i$ is the relative azimuthal angle, $\theta_h = (\theta_i + \theta_o)/2$ is the longitudinal half angle, and A_p is the attenuation function associated with each lobe. M_p and N_p are longitudinal and azimuthal scattering functions, respectively. Subscripts $_i$ and $_o$ indicate incoming and outgoing components.

Dual scattering (DS). To efficiently handle the complicated light scattering between hairs, Zinke et al. [2008] proposed dual scattering to approximate the fraction of light $\Psi_{DS}(\mathbf{x}, \omega_d, \omega_i)$ entering the hair volume from direction ω_d that is scattered inside the hair volume and finally arriving at point $\mathbf{x}$ from direction ω_i with two components, global multiple scattering $\Psi_G(\mathbf{x}, \omega_d, \omega_i)$ and local multiple scattering $\Psi_L(\mathbf{x}, \omega_d, \omega_i)$, as:

$$\Psi_{DS}(\mathbf{x}, \omega_d, \omega_i) = \Psi_G(\mathbf{x}, \omega_d, \omega_i)(1 + \Psi_L(\mathbf{x}, \omega_d, \omega_i)) \,, \tag{8}$$

where the global component represents light arriving at shading point $\mathbf{x}$ after passing through multiple fibers, and the local component estimates the total contribution from nearby fibers due to multiple light bounces.

4.2 Rendering Pipeline

Yarn comprises multiple plies, each containing tens to hundreds of fibers twisted together to maintain stability. The densely packed fibers at the yarn centerline form the core structure, while flyaway fibers, introduced during the yarn fabrication process, exhibit a sparse distribution and cover a larger spatial area. The complex structure and a large number of geometric primitives present a significant challenge for high-fidelity rendering, especially multiple scattering between fibers, in real time.

Hence, given yarn control points from the simulation, we adopt the GPU rasterization pipeline to render yarn as a camera-facing strip with precomputed periodic fiber-level tangent and normal textures and approximate multiple scattering via DS. We further introduce the following decomposition strategies for real-time performance while maintaining rendering quality. First, as shown in Fig. 6, to accommodate the distinct geometric distributions of different types of fiber, we decompose the yarn geometry into the following:

- **Regular fibers**, representing fibers in the core that exhibit a dense and organized arrangement;
- **Flyaway fibers**, representing loosely distributed and irregular fibers that extend outward from the core.

Fig. 6. *Three periodic regular and flyaway textures connected seamlessly. Gray indicates regular fibers, and flyaway fibers are colored based on tangent direction. Dashed lines refer to texture borders.*

Second, approximating multiple scattering and handling environment lighting at the yarn level is fast but misses fiber details. Individual fibers demand extremely high-resolution framebuffers, e.g., deep opacity maps for DS, as fibers are substantially thinner compared to the overall size of the knit structure. Therefore, we decompose the shading process into yarn- and fiber- levels.

4.3 Regular Fibers

The twisted fiber structure, combined with migrated and flyaway fibers, challenges the layered structure assumption used in DS. Additionally, the low roughness of the R/TT lobes reduces the proportion of light that traverses through the primary light paths, which DS approximation focuses on. As a result, as shown in Fig. 7 fourth row,

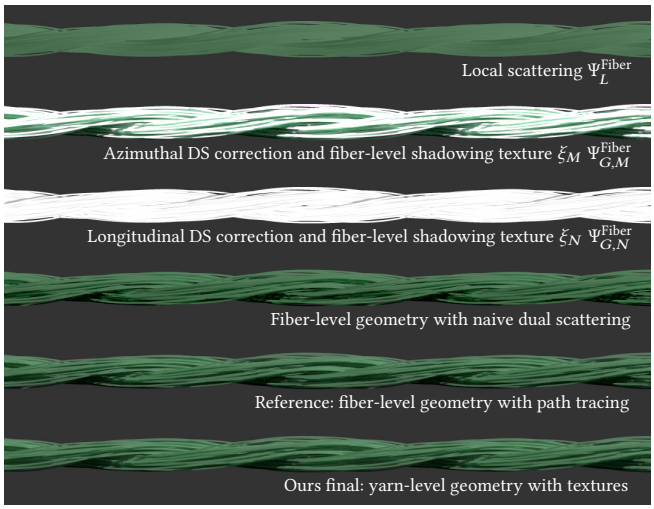


Fig. 7. *With pre-computed DS correction and fiber-level shadowing textures, our yarn-level geometry achieves results nearly indistinguishable from the reference with fiber-level geometry and path tracing.*

significant deviations arise between the multiple scattering approximation Ψ_{DS} by DS and the reference Ψ_* obtained by path tracing, especially when the angular between the incident and outgoing light directions is large.

Dual scattering correction. To ensure results align more closely with reference from all viewing angles, we introduce a correction term ξ such that

$$\Psi_*(\mathbf{x}, \omega_d, \omega_i) \approx \xi(\mathbf{x}, \omega_d, \omega_i)\ \Psi_{DS}(\mathbf{x}, \omega_d, \omega_i) \,, \tag{9}$$

where $\xi(\mathbf{x}, \omega_d, \omega_i)$ can be pre-computed over all directions ω_d and ω_i and positions $\mathbf{x}$ at given yarn with fiber-level geometry. For efficiency purposes, this 7D data is further decomposed into the product of azimuthal and longitudinal contributions as:

$$\xi(\mathbf{x}, \omega_d, \omega_i) \approx \xi_M(\mathbf{x}, \theta)\ \xi_N(\mathbf{x}, \phi) \,, \tag{10}$$

where $\theta = \theta_d - \theta_i$ and $\phi = \phi_d - \phi_i$ are the relative azimuthal longitudinal angle between ω_d and ω_i.

Fiber-level shadowing. Since the fiber geometries are baked into a yarn texture, the actual fiber geometry is no longer explicitly available for shadowing. To address this issue, an additional global scattering term $\Psi_G^{\text{Fiber}}(\mathbf{x}, \omega_d, \omega_i)$ is introduced at the fiber level. During rendering, global scattering between yarns Ψ_G^{Yarn} can be computed based on a yarn-level shadowing pass, while occlusion between fibers within each yarn Ψ_G^{Fiber} is queried from a precomputed texture. To further reduce storage space, this fiber-level texture is decomposed into azimuthal $\Psi_{G,M}^{\text{Fiber}}(\mathbf{x}, \theta)$ and longitudinal $\Psi_{G,N}^{\text{Fiber}}(\mathbf{x}, \phi)$ components. Combining everything together, the dual scattering function Eq. 8 comes as:

$$\Psi(\mathbf{x}, \omega_d, \omega_i) = \xi_M\ \xi_N\ \Psi_G^{\text{Yarn}}\ \Psi_{G,M}^{\text{Fiber}}\ \Psi_{G,N}^{\text{Fiber}} \left(1 + \Psi_L^{\text{Fiber}}\right) . \tag{11}$$

4.4 Flyaway Fibers

Flyaway fibers are a crucial component of textiles. Their distribution is more dispersed and irregular, adding significant subpixel-level

details to the rendering results and contributing to the inherent fuzziness of textiles. During runtime, we use a separate pass to overlay flyaway fibers on the rendering results of regular fibers.

Procedural flyaway model. To generate realistic flyaway fibers, we adopt a procedural yarn geometry model [Zhao et al. 2016]. The vertex position (x, y, z) of each flyaway fiber is parameterized by $t \in [0, 1]$ as:

$$x(t) = R(t)\cos\theta(t),\ y(t) = R(t)\sin\theta(t),\ z(t) = z_0 + z_e t\ , \quad (12)$$

where $R(t) = R_0 + R_e t$ and $\theta(t) = \theta_0 + \theta_e t$. To enhance the realism, we introduce two modifications to this model:

- **Random perturbations**: Noise is added to the progressive direction of the fibers to introduce random variations, emulating natural irregularities.
- **Curvature adjustment**: An α parameter is introduced to adjust the curvature of the fibers, particularly improving the appearance for side view, as:

$$R(t) = R_0 + R_e(1 - (1 - t)^{\alpha})\ . \quad (13)$$

Flyaway texture baking. Since the flyaway texture is directly applied to the camera-facing strips during rendering, performing a precise depth test between the flyaway and regular fibers is impractical due to the limited hardware floating point accuracy. To address this challenge, we incorporate depth testing during the texture generation phase. During texture baking, we rasterize the regular fibers to create a depth stencil in the first pass. Then, flyaway fibers are rasterized as any occluded by a regular fiber in the same yarn are discarded. The tangent information of the remaining visible flyaway fibers is stored in the texture. Note that we create three periodic flyaway textures that seamlessly connect at their boundaries, as shown in Fig. 6, allowing for a smooth transition when repeated. During rendering, one of the three periodic textures is chosen randomly for each period along the yarn to avoid pattern artifacts arising from reusing a single periodic texture.

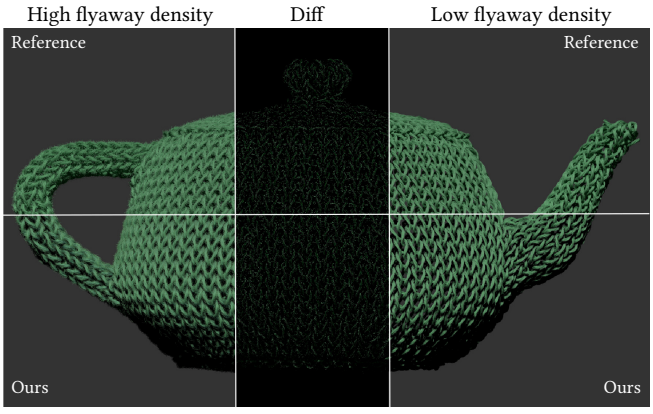


Fig. 8. *Knit teapot with same regular fibers but flyaway fibers with different densities. The middle image shows the difference between these results, highlighting how the flyaway density influences the overall brightness.*

Flyaway rendering. Since the depth test between flyaway and regular fibers has already been baked into the texture during its generation, rendering flyaway fibers on top of regular fibers during run-time becomes trivial. It is also worth noting that when the density of flyaway fibers increases, the radiance received by regular fibers undergoes noticeable changes. Fig. 8 demonstrates that variations of flyaway density result in consistent brightness changes between regular fibers. To address this, we adjust the intensity of local scattering for DS to account for radiance variations.

4.5 Environmental Lighting

To support environment lighting in real-time, we follow Ren et al. [2010] to use a set of *Spherical Radial Basis Functions (SRBF)* to fit the environment map as:

$$L(\omega_i) \approx \Sigma_j W_j G(\omega_i; \omega_j, \lambda_j)\ , \quad (14)$$

where $G(\omega_i; \omega_j, \lambda_j) = e^{\lambda_j(\omega_i \cdot \omega_j - 1)}$ represents the Gaussian SRBF kernel centered at ω_j with bandwidth λ_j. For brevity, we use $G(\omega_i)$ to denote an SRBF. The scattering integral under incident environment lighting can be written as follows:

$$L(\omega_o) = \int_{\Omega} L(\omega_i) T(\omega_i) f(\omega_i, \omega_o) \cos\theta_i d\omega_i\ , \quad (15)$$

where f is the bidirectional scattering function defined in Eq. 7 and the backward scattering function $T(\omega_i)$ is the transmittance in the incident direction. As suggested by Ren et al. [2010], approximating transmittance T with effective transmittance $\tilde{T}$ yields Eq. 15 to

$$L(\omega_o) = \Sigma_j W_j\ \tilde{T}(\omega_j, \lambda_j) \int_{\Omega} G_j(\omega_i) f(\omega_i, \omega_o) \cos\theta_i d\omega_i\ , \quad (16)$$

where $\tilde{T}(\omega_j, \lambda_j)$ is average attenuation of the SRBF lighting j and the integration $\int_{\Omega} G_j(\omega_i) f(\omega_i, \omega_o) \cos\theta_i d\omega_i$ can be pre-computed as a 4D table $I_M(\cos\theta_j, \cos\theta_o, \cos(\theta_j - \theta_o), 1/\lambda_j)$.

Runtime rendering. Unfortunately, computing a fiber-level *Deep Opacity Depth Map (DODM)* as suggested by Ren et al. [2010] is computationally expensive, as mentioned before. To achieve an efficient yet accurate approximation of the average attenuation $\tilde{T}$ for each SRBF lighting j at runtime, we decompose the computation into two components. We first interpolate the fiber-level shadowing texture to estimate the occlusion between the fibers within the same yarn. Additionally, instead of using DODM, we directly use the yarn-level shadow map generated from the SRBF center direction ω_j, as the light direction. The attenuation $\tilde{T}$ at the shading point is then approximated using *Percentage Closer Soft Shadows (PCSS)* [Fernando 2005] for soft shadowing. The complete lighting equation is expressed as:

$$L(\omega_o) = \Sigma_j \xi_M\ \xi_n\ W_j\ \tilde{T}^{\text{Yarn}}(c_j, \lambda_j) \Psi^{\text{Fiber}}_{G,M}\ \Psi^{\text{Fiber}}_{G,N} I_m\ . \quad (17)$$

This approach simplifies the computation of $\tilde{T}$ by combining fiber-level shadowing and yarn-level PCSS, thereby reducing the storage footprint while preserving the soft shadowing quality.

5 IMPLEMENTATION DETAILS

This section outlines the preprocessing stages, followed by a detailed description of our run-time pipeline implementation.

5.1 Preprocessing

Creating stitch template. Our framework requires a yarn template for each stitch type as input. To generate the template, we first perform a mesh-based relaxation following the method in Yuksel et al. [2012]. Specifically, we apply stretching forces by assuming equal edge lengths along the wale and course directions. For pentagonal faces (increase and decrease stitches), we construct shear forces by connecting the top and bottom edge vertices. After face reaches the quasistatic state, we fix the yarn control points along the edges of the stitch mesh face and iteratively relax the interior yarn control points to reach a quasistatic equilibrium state that incorporates yarn contact, similar to Leaf et al. [2018].

Texture baking. To generate the texture set described in Sec. 4.3, we uniformly sample eight directions in both azimuthal and longitudinal dimensions, ensuring smooth directional transitions and close visual alignment with path-traced references. For each sampled direction, we use an offline path tracer to compute both the reference and DS results for a single yarn with fibers, excluding global scattering effects. Additional flyaway, normal, and tangent textures that do not require path tracing are precomputed through a separate OpenGL rasterization pass.

5.2 Runtime Pipeline

At runtime, the animated stitch mesh can be generated on the CPU via mesh-based relaxation or position-based dynamics. The vertex positions are then transferred to the GPU, where we perform knot-based simulation in CUDA to compute the yarn control points. Then, these control points are transferred to the OpenGL rendering pipeline through CUDA-OpenGL interoperability functions.

Simulation. At runtime, the 2D deformation gradient of each stitch face is computed from the mesh vertex positions. Using this, yarn control points are uniformly interpolated as the initial guess for our Gauss-Newton optimizer, which runs in parallel across all stitch faces. Taking the *knit* pattern as an example, the optimization parameters are the tilt angles and half-lengths, denoted as $\Theta_{\text{knit}} = (\theta_1, \theta_2, l_1, l_2)^T$. The residuals from the target lengths and angles are computed by iterating over all yarn control points, along with their Jacobians $J_i = \frac{\partial r_i}{\partial \Theta}$, where $r_i = \sqrt{E_i}$ corresponds to the bending and stretching energy centered at each vertex. The Gauss-Newton update step $\Theta^{(k+1)} = \Theta^{(k)} + \Delta\Theta$ is then formulated as the linear system $A\,\Delta\Theta = b$, with $A = \Sigma J_i J_i^T$, $b = -\Sigma J_i r_i$. A direct solver is used to invert the per-stitch matrix A. After three iterations, the resulting 2D control point positions are lifted to 3D by applying a template-prescribed offset along the normal direction and then written into the rendering buffer via stored indices. This optimization is relatively lightweight and is performed efficiently on a single CUDA thread per stitch. For a *knit* tile, the optimizer optimizes the positions of 8 control points, with another 6 points at the boundary fixed at the interpolated positions. Finally, the control points are up-sampled with 4 additional points per segment, resulting in 56 rendering vertices per stitch face in total.

Rendering. Our rendering pipeline consists of three sequential passes: shadow, regular shading, and flyaway shading. In each pass, yarns are converted into camera-oriented strips via a geometry shader. In fragment shader of regular shading, normal, tangent, and AO textures are periodically mapped onto the strip based on the arc length along the yarn. For flyaway shading, the RGB channels of the flyaway texture encode tangent information, while the alpha channel indicates whether a given location requires shading. This design prevents incorrect occlusion of certain flyaway fibers when rendering multiple overlapping layers using high mipmap levels.

6 RESULTS

We conducted all experiments on a system equipped with an AMD Ryzen Threadripper 3970X 32-core CPU, 256 GB of memory, and a NVIDIA RTX 3090 GPU. The offline path tracer was implemented on the CPU and accelerated using Intel oneAPI Threading Building Blocks (oneTBB) with 64 threads. 16 SRBFs were used to fit the environment map.

6.1 Simulation

Comparison with interpolation. Fig. 9 demonstrates a knit patch under stretch and shear. Naively interpolating yarn control points results in zigzagging yarn curves when the mesh deforms. However, interpolating knot positions and mapping yarn control points along with knots exhibit unrealistic curvature when the mesh is under shearing. In contrast, our knot-based yarn optimization effectively resolves these issues, producing visually plausible yarn curves, as highlighted in the zoomed-in view.

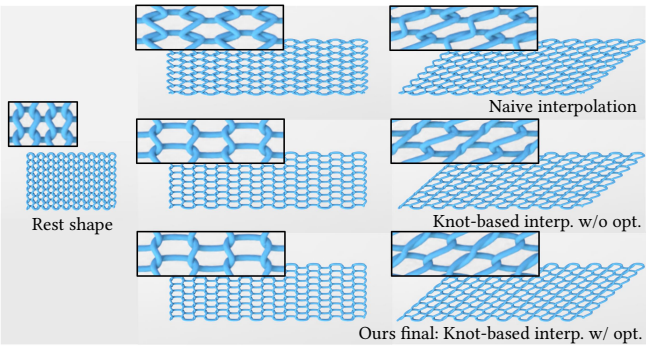


Fig. 9. *A knit patch under stretch and shear with naive and knot-based interpolation, with and without yarn optimization.*

Comparison with full yarn-level simulation and [Yuan et al. 2024]. We use a Gauss-Newton scheme with three iterations, which creates yarn-level hole deformation, as demonstrated in Fig. 10. The computation is parallelized across stitch faces, and, benefiting from our simplified 2.5D representation, the runtime performance achieves approximately 1 ms per frame, achieving four orders of magnitude speed-up compared to full yarn-level simulation (7680 ms), as it involves extensive 3D computation. Furthermore, while our approach is two orders of magnitude faster than the volumetric homogenization method [Yuan et al. 2024] (96 ms), our simplified representation cannot model out-of-plane forces, such as curl-up or flattening effects. Note that, except ours, simulation times are obtained from Yuan et al. [2024] directly without running their code, but timings are all measured on Nvidia RTX3090.

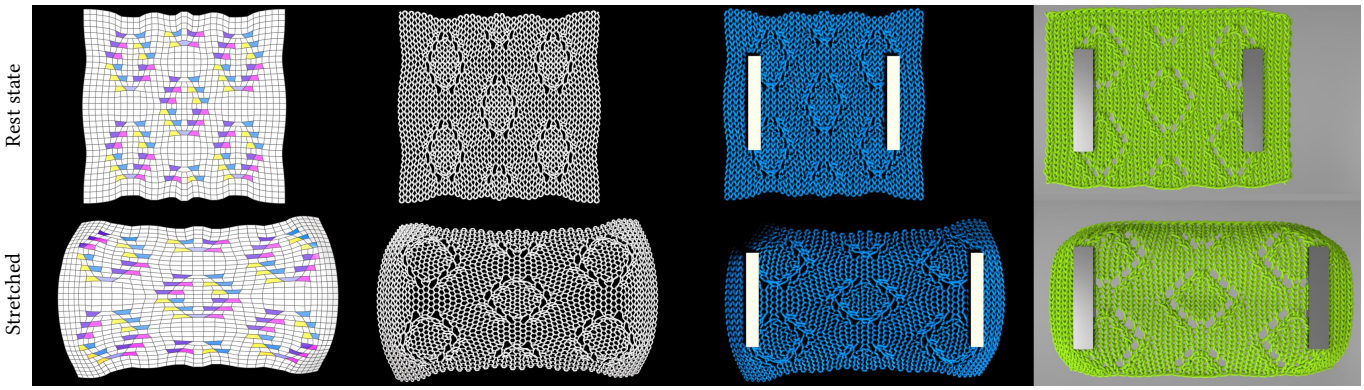


Fig. 10. *From left to right: stitch mesh, yarn geometry after our knot-based optimization, our final rendering result, and images of full yarn-level simulation result from Yuan et al. [2024]. Our simulation only takes 1 ms per frame, while full yarn-level simulation and volumetric homogenization [Yuan et al. 2024] take 7680 ms and 96 ms, respectively, per time step.*

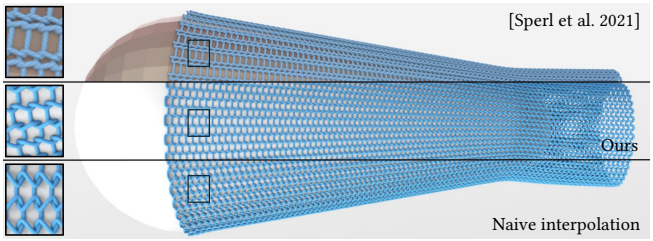


Fig. 11. *A sleeve with Sperl et al. [2021], ours, and naive interpolation.*

Comparison with Sperl et al. [2021]. Fig. 11 presents an animated knit sleeve to highlight the visual difference between ours and that of Sperl et al. [2021]. The top image, taken directly from Sperl et al. [2021], shows tightened stitches under stretching due to their assumption of periodic boundary conditions and elastostatic behavior without friction during full yarn-level simulation. In contrast, based on our observations from a real-world yarn-stretching dataset [Hofmann et al. 2019], yarn sliding is minimal due to the significant friction between fuzzy yarns. So, the interlocking yarns at each knot exert opposing forces that tend to balance each other, resulting in a more uniform stretching of the knots across the fabric in our simulation. Note that naively interpolating yarn control points result in loose stitches with noticeable gaps. Performance-wise, the data-driven mechanics-aware method [1] takes 0.91 ms in total (0.62 for CPU strains, 0.16 for GPU displacement, and 0.13 for GPU mapping). Our optimization only takes 1.03 ms on GPU, comparable to their data-driven approach. Lastly, while our 2.5D representation limits the ability to capture curl-up effects and the flattening behavior of knits under tension, our method remains applicable to a wide range of patterns. Notably, it is not constrained to periodic boundary conditions, such as Flame (Fig. 1), and local variations, such as Montague (Fig. 10).

Ablation study on energy weights. We evaluate the effectiveness of different weights for the angle and length energy terms in our optimization framework. Specifically, we fix the angle energy weight to 1 and vary the length energy weight from 10^2 to 10^3. Fig. 12 demonstrates that the bending (angle) energy primarily governs the overall shape preservation of the yarn under stretched, while the stretching (length) energy influences the tightness of the yarn knots. We use $w_{\text{length}} = 10^2$ and $w_{\text{angle}} = 1$ for all other results.

[1]We used the implementation at https://git.ista.ac.at/gsperl/MADYPG

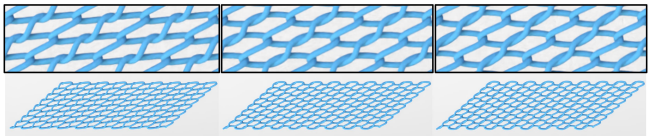


Fig. 12. *A shear knit patch with different weights for length energy.*

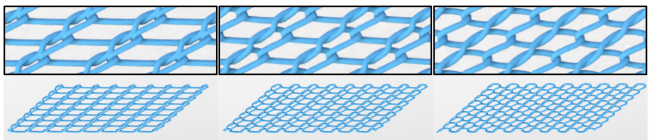


Fig. 13. *A shear knit patch where knots distance can be adjusted by a blending parameter β.*

Non-uniform knot interpolation. Assuming the original undeformed distance between knots is d_0 and the interpolated distance is d_1, our knot interpolation scheme also enables user-defined control over knot spacing through a blending parameter α, such that the resulting distance is given by $d = d_0 + \beta(d_1 - d_0)$. As illustrated in Fig. 13, setting $\beta = 0$ recovers a visual appearance similar to the results of Sperl et al. [2021].

6.2 Rendering

Comparison with reference and SOTAs. Fig. 14 demonstrates a knit glove rendered by our method under directional light, with a screen resolution of 1520×960. Reference is obtained by offline path tracing with explicit fiber-level geometry. Directly using dual scattering [Zinke et al. 2008] in the full fiber geometry underestimates irradiance due to oversimplified assumptions about the fiber structure. However, applying the aggregated ply shading model [Zhu

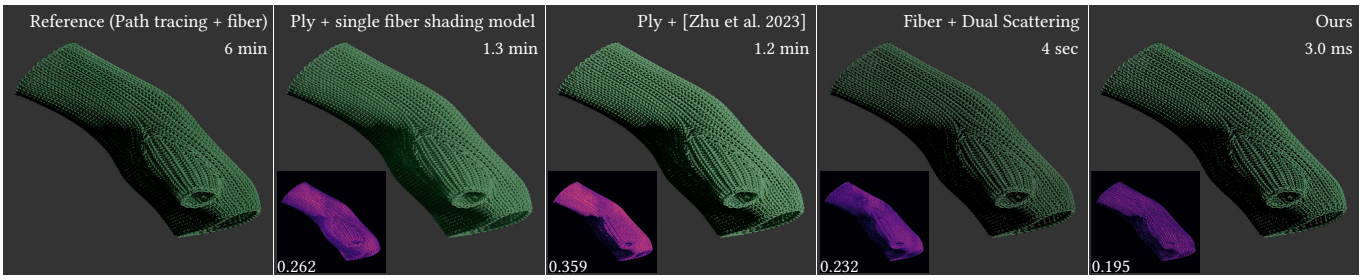


Fig. 14. *Comparison to previous rendering solutions. ꟻLIP error maps are displayed at the bottom-left corner. Our method is at least three orders of magnitude faster and produces results nearly identical to ground truth with fiber-level geometry and path tracing with the lowest ꟻLIP error.*

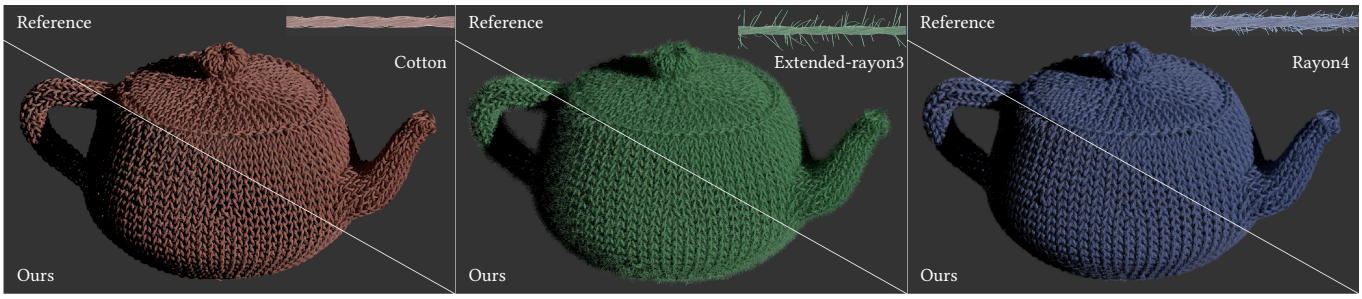


Fig. 15. *Knit teapot with three distinct types of flyaway configurations, cotton, extended-rayon3 (flyaway length is tripled for a fuzzier appearance), and rayon4. The bottom left and top right of each image compare our method with the reference. These models comprise 51M, 55M, and 61M fiber segments, respectively. Reference takes 9, 8.3, and 7 min, while ours only needs 5.1, 8.1, and 7.3 ms.*

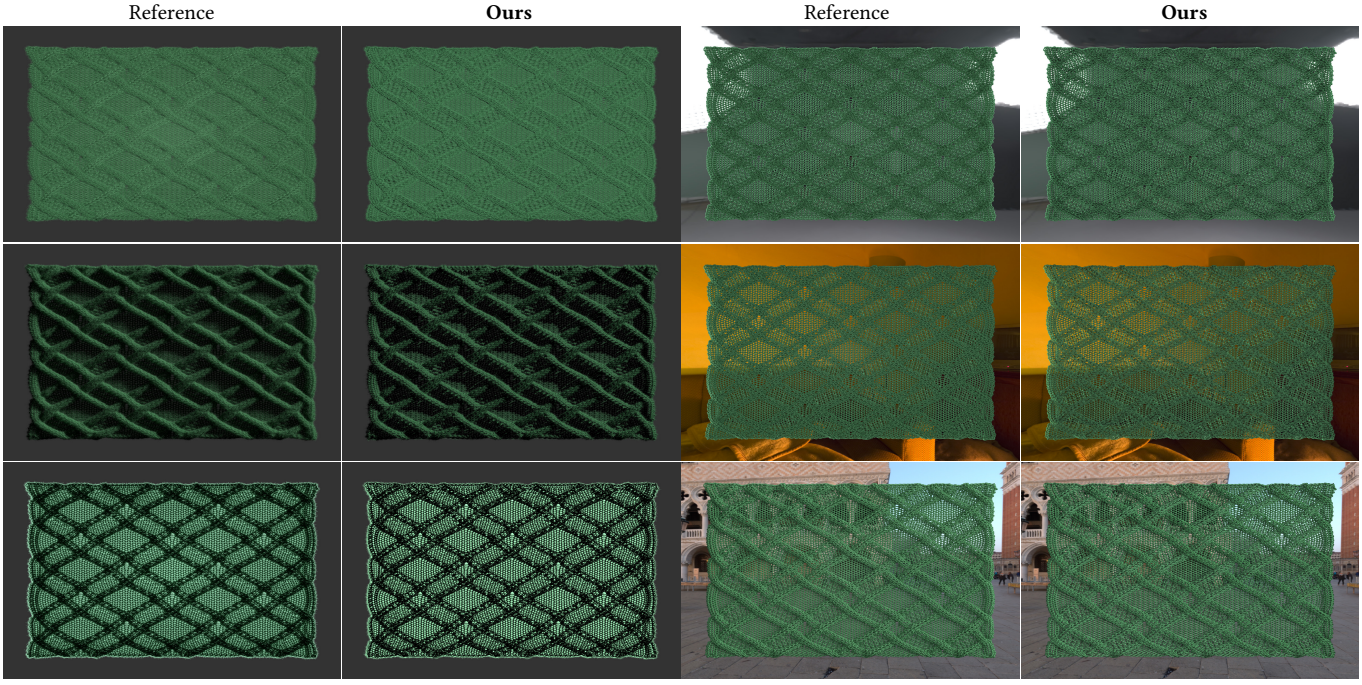


Fig. 16. *Our approach produces rendered results with equal quality to the reference for view and light direction angles of 15°, 75°, and 180°(left column), and under different environmental lighting (right column). The reference employs full fiber geometry with 118M segments and takes around 15 and 28 mins for directional and environment lights, while ours utilizes only 100K segments with only 9.4 and 20.6 ms, respectively.*

et al. 2023] or the single-fiber shading model at the ply level results in over-brightness due to inaccuracies in the approximation of fiber-level scattering within the ply. Our method produces results that are nearly indistinguishable from the reference. Beyond its accuracy, our method is significantly faster than previous approaches. Reference, ply-level methods, and dual scattering require offline path tracing with 512, 64, and 20 samples per pixel, respectively, to achieve equal-quality (EQ) results. In contrast, our method only takes 3 ms to achieve the same quality using an OpenGL rasterization pipeline with 4× multisample antialiasing (MSAA). In particular, our approach is 120,000× faster than the reference and at least three orders of magnitude faster than any existing knit rendering technique with the lowest ꟻLIP error listed in the lower left corner of each sub-figure in Fig. 14. Note that the error in our results arises primarily from the geometry mismatch between our camera-oriented stripes and the actual yarn geometries.

Flyaways. We test our method with three distinct types of flyaway configurations [Zhao et al. 2016] to demonstrate the compatibility of our method with various flyaway densities, fiber color, and structures to consistently deliver high-quality rendering close to the reference (see Fig. 15).

Lighting conditions. Fig. 16 demonstrates that our method can produce results with an equal quality to the reference for angles of view and light direction, 15°, 75°, and 180°. Additionally, our method can effectively capture environmental lighting information, delivering consistent rendering outcomes, including high dynamic range (HDR) images for both outdoor and indoor scenarios.

6.3 Demos

Compared to photographs. We evaluate our method through a qualitative comparison between our results and images of a real knit sample from KnitDB [Hofmann et al. 2019] in both the rest and the stretched states (Fig. 4). We create a stitch mesh with the same number of rows and columns of stitches as in the real sample. The mesh is pre-simulated using ARCSim [Narain et al. 2012] to match the overall shape under stretch. By tweaking the fiber shading parameters, stitch shape, and environmental light, our method can closely replicate the knit appearance in its rest state. Furthermore, our knot-based approach optimizes the yarn curve throughout stretching with 1.58ms per frame, effectively predicting the overall appearance change caused by yarn-level deformation.

Stretching patches with various patterns. We demonstrate four knit patches with Flame (Fig. 1), Montague (Fig. 10), Stockinette (Fig. 18), and Ribbing (Fig. 19) patterns animated based on the underlying stitch mesh with yarn-level deformation details as well as high-quality fiber-level rendering under environmental light. Simulation consistently takes around 1 ms per frame. The rendering takes 23.6 ms for Flame due to a large number of yarn control points and 16 ms for the other two patterns. The entire simulation and rendering cost is less than 25 ms. Please refer to the accompanying video for the whole animation sequence.

Full garment relaxation. Our system also supports yarn-level relaxation in the knit design pipeline, which has been the bottleneck in previous work [Yuksel et al. 2012]. As shown in Fig. 17, after mesh relaxation, our yarn optimization takes only 2.5 ms to relax the yarn to a visually plausible state. In contrast, it took several hours, as reported in Yuksel et al. [2012].

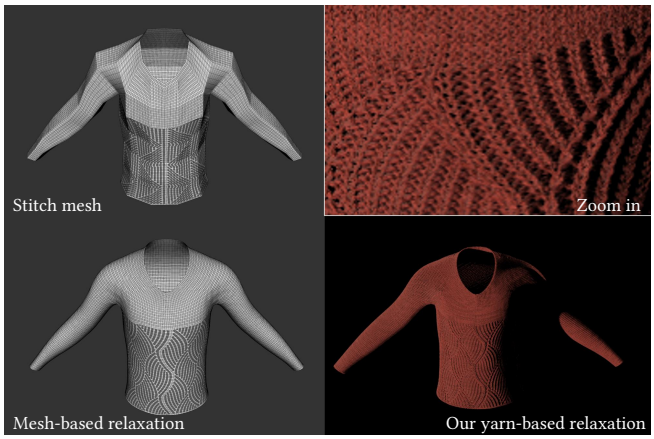
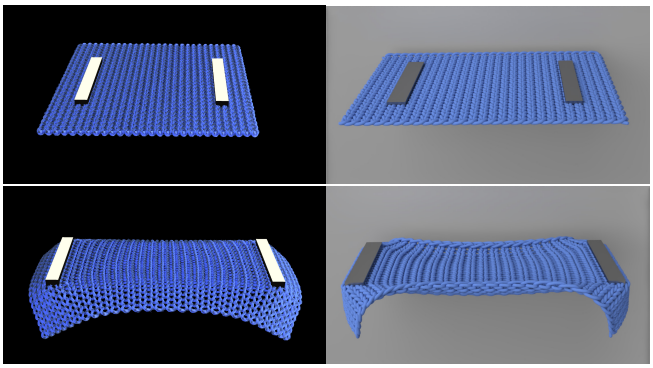


Fig. 17. *Given the mesh-based relaxation, our knot-based optimization can also be used for relaxing yarn geometry to obtain a physically plausible appearance, as highlighted in the zoom-in view. The whole relaxation only takes 2.5 ms on GPU.*

Performance. Table 1 lists the performance of our framework, highlighting its efficiency and scalability for both real-time simulation and rendering. Knot-based optimization is implemented using CUDA, and the resulting yarn control points are seamlessly transferred to the OpenGL rasterization pipeline via CUDA-OpenGL interop for rendering. This fully GPU-based pipeline eliminates overhead caused by GPU-CPU data transfers. Note that enabling environmental lighting significantly increases rendering costs due to the need to approximate the environment lighting for 16 SRBFs. Additionally, generating all the textures required for rendering a specific configuration of fiber takes approximately 5–7 mins on CPU, depending on the fiber-level geometry complexity. On the simulation side, 3 iterations for the Gauss-Newton scheme already provide comparable results to full yarn-level simulation four orders of magnitude faster than full yarn-level simulation.

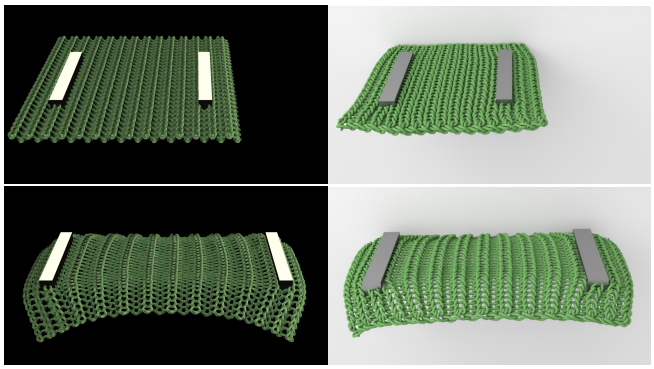
Table 1. Statistics. All examples were timed on an Nvidia RTX 3090 for simulation and rendering time. Yarn CPs, fiber seg., and env. refer to yarn control point, fiber segments, and environmental light. Screen size is 1920 × 1280 for the front six and 1300 × 1300 for the rest.

Model	Fig.	Yarn CPs #	Fiber Seg. #	w/ Env.	Ren. (ms)	Sim. (ms)
Flame	1	200K	29M	Y	23.6	1.05
Photograph	4	172K	150M	Y	31.1	1.58
Sleeve	11	44K	–	–	–	1.03
Ribbing	19	38K	6.5M	Y	15.7	1.02
Montague	10	52K	7.3M	Y	16.5	1.02
Relaxation	17	319K	180M	N	22.8	2.50
Glove	14	60K	60M	N	3.0	–
Teapot (red)	15	59K	51M	N	5.1	–
Teapot (green)	15	59K	55M	N	8.1	–
Teapot (blue)	15	59K	61M	N	7.3	–
Cable	16	130K	118M	N	9.4	–
Cable	16	130K	96M	Y	20.6	–



Ours Yarn-level simulation

Fig. 18. *Our method relies on the underlying mesh to capture the high-level deformation of the knit patch. However, for structures such as Stockinette, our method alone cannot reproduce curl-up effects under stretching, as demonstrated by yarn-level simulations.*



Ours Yarn-level simulation

Fig. 19. *While the Ribbing patch with our yarn optimization and fiber-level rendering under environmental lighting requires only 16.7 ms, our method is unable to capture the flattening behavior of the yarn-level structure under tension.*

7 LIMITATIONS

Our method can effectively produce knit appearances with fiber-level detail under various lighting conditions with notable yarn-level deformation. There is still a notable gap between our virtual results and real-world imagery.

Simulation. Our knot-based optimization approximates 3D knit structures as a net-like representation with a fixed topology, compromising physical accuracy for real-time performance. Furthermore, unlike previous data-driven methods [Sperl et al. 2021; Yuan et al. 2024], our method does not capture curl-up effects (Fig. 18) and flattening behavior (Fig. 19) of the yarn-level structure under tension since our 2.5D representation does not account for out-of-plane forces. Lastly, while our method can be generalized to various stitch types, such as increases and decreases, as shown in Fig. 10, and to other single-layer planar fabric structures, including woven fabrics, lace, and crochet, it also has inherent limitations. Specifically, our current representation does not accommodate layered knit structures, such as cables, where stitches cross over each other to create twisted or braided effects. Additionally, it is not suitable for fabric structures with complex yarn placement that cannot be captured solely through yarn contacts, such as the Cartridge Rib stitch, where slipped yarns float across the back of a regular ribbing pattern.

Rendering. Our method retains common issues associated with billboard rendering as we cannot achieve true radial changes when physically rotating the yarn. Secondly, the fibers within the yarn structure become denser and more compact when stretched. These microstructural changes profoundly influence the macroscopic appearance. Specifically, stretched knits exhibit noticeable differences in brightness and shininess compared to their relaxed state. However, due to the time required for texture generation, we cannot reproduce this kind of light transport behavior in real time.

8 CONCLUSION

We have introduced a real-time framework that seamlessly integrates yarn-level simulation and fiber-level rendering, delivering a high-fidelity yet efficient system for knitted structures. Our key contributions are twofold. We proposed a novel 2.5D knot-based stitch representation. Coupled with a highly parallelizable GPU optimization, our system generates visually plausible yarn curve geometry even as the underlying mesh undergoes deformation. Given yarn control points from knot-based simulation, we introduce a GPU rasterization pipeline incorporating novel decomposition strategies to render yarn with fiber-level geometry, accommodating multiple scattering effects and environmental lighting. Our simulator is lightweight, highly parallelizable, and requires no pre-computed data, while our rendering pipeline is at least three orders of magnitude faster than existing methods while producing results nearly identical to ground truth. This comprehensive system is perfectly suited for a variety of application scenarios, including knit pattern design and animating full garments with detailed knit structures.

ACKNOWLEDGMENTS

Yin Yang acknowledges the funding support from NSF 2301040.

REFERENCES

Juan J. Casafranca, Gabriel Cirio, Alejandro Rodríguez, Eder Miguel, and Miguel A. Otaduy. 2020. Mixing Yarns and Triangles in Cloth Simulation. *Computer Graphics Forum* 39, 2 (2020), 101–110.

Gabriel Cirio, Jorge Lopez-Moreno, David Miraut, and Miguel A. Otaduy. 2014. Yarn-level simulation of woven cloth. *ACM Trans. Graph.* 33, 6, Article 207 (Nov. 2014), 11 pages.

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.

Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. 1999. Reflectance and texture of real-world surfaces. *ACM Trans. Graph.* 18, 1 (Jan. 1999), 1–34.

Xudong Feng, Huamin Wang, Yin Yang, and Weiwei Xu. 2024. Neural-Assisted Homogenization of Yarn-Level Cloth. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 80, 10 pages.

Randima Fernando. 2005. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches* (Los Angeles, California) *(SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 35–es.

Eduard Gröller, René T. Rau, and Wolfgang Strasser. 1995. Modeling and Visualization of Knitwear. *IEEE Transactions on Visualization and Computer Graphics* 1, 4 (Dec. 1995), 302–310.

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* (Virtual Event, USA) *(SIGGRAPH '20)*. Association for Computing Machinery, New York, NY, USA, Article 75, 2 pages.

Megan Hofmann, Lea Albaugh, Ticha Sethapakadi, Jessica Hodgins, Scott E. Hudson, James McCann, and Jennifer Mankoff. 2019. KnitPicking Textures: Programming and Modifying Complex Knitted Textures for Machine and Hand Knitting. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 5–16.

Jerry Hsu, Tongtong Wang, Zherong Pan, Xifeng Gao, Cem Yuksel, and Kui Wu. 2024. Real-time Physically Guided Hair Interpolation. *ACM Trans. Graph.* 43, 4, Article 95 (July 2024), 11 pages.

Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. 2010. A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph.* 29, 4, Article 53 (July 2010), 13 pages.

Chenfanfu Jiang, Theodore Gast, and Joseph Teran. 2017. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Trans. Graph.* 36, 4, Article 152 (July 2017), 14 pages.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2008. Simulating knitted cloth at the yarn level. In *ACM SIGGRAPH 2008 Papers* (Los Angeles, California) *(SIGGRAPH '08)*. Association for Computing Machinery, New York, NY, USA, Article 65, 9 pages.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2010. Efficient yarn-based cloth with adaptive contact linearization. *ACM Trans. Graph.* 29, 4, Article 105 (July 2010), 10 pages.

Levi Kapllani, Chelsea Amanatides, Genevieve Dion, Vadim Shapiro, and David E. Breen. 2021. TopoKnit: A Process-Oriented Representation for Modeling the Topology of Yarns in Weft-Knitted Textiles. *Graph. Models* 118, C (Nov. 2021), 19 pages.

Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. 2019. Knitting Skeletons: A Computer-Aided Design Tool for Shaping and Patterning of Knitted Garments. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 53–65.

Alexandre Kaspar, Kui Wu, Yiyue Luo, Liane Makatura, and Wojciech Matusik. 2021. Knit sketching: from cut & sew patterns to machine-knit garments. *ACM Trans. Graph.* 40, 4, Article 63 (July 2021), 15 pages.

Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. 2016. Matching Real Fabrics with Micro-Appearance Models. *ACM Trans. Graph.* 35, 1, Article 1 (Dec. 2016), 26 pages.

Alexandr Kuznetsov, Krishna Mullia, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. 2021. NeuMIP: multi-resolution neural materials. *ACM Trans. Graph.* 40, 4, Article 175 (July 2021), 13 pages.

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 (Dec. 2018), 15 pages.

Logica. 2020. PaintKnit. [Online]. Available from: www.paintknit.com.

Fujun Luan, Shuang Zhao, and Kavita Bala. 2017. Fiber-Level On-the-Fly Procedural Textiles. *Computer Graphics Forum* 36, 4 (2017), 123–135.

Yiyue Luo, Kui Wu, Tomás Palacios, and Wojciech Matusik. 2021. KnitUI: Fabricating Interactive and Sensing Textiles with Machine Knitting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 668, 12 pages.

Yiyue Luo, Kui Wu, Andrew Spielberg, Michael Foshey, Daniela Rus, Tomás Palacios, and Wojciech Matusik. 2022. Digital Fabrication of Pneumatic Actuators with Integrated Sensing by Machine Knitting. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 175, 13 pages.

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

Zahra Montazeri, Søren B. Gammelmark, Shuang Zhao, and Henrik Wann Jensen. 2020. A practical ply-based appearance model of woven fabrics. *ACM Trans. Graph.* 39, 6, Article 251 (Nov. 2020), 13 pages.

Zahra Montazeri, Chang Xiao, Yun Fei, Changxi Zheng, and Shuang Zhao. 2021. Mechanics-Aware Modeling of Cloth Appearance. *IEEE Transactions on Visualization and Computer Graphics* 27, 1 (Jan. 2021), 137–150.

Rahul Narain, Armin Samii, and James F. O'Brien. 2012. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph.* 31, 6, Article 152 (Nov. 2012), 10 pages.

Vidya Narayanan, Kui Wu, Cem Yuksel, and James McCann. 2019. Visual knitting machine programming. *ACM Trans. Graph.* 38, 4, Article 63 (July 2019), 13 pages.

Zhong Ren, Kun Zhou, Tengfei Li, Wei Hua, and Baining Guo. 2010. Interactive hair rendering under environment lighting. *ACM Trans. Graph.* 29, 4, Article 55 (July 2010), 8 pages.

Andrew O. Sageman-Furnas, Albert Chern, Mirela Ben-Chen, and Amir Vaxman. 2019. Chebyshev nets from commuting PolyVector fields. *ACM Trans. Graph.* 38, 6, Article 172 (Nov. 2019), 16 pages.

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

Georg Sperl, Rahul Narain, and Chris Wojtan. 2020. Homogenized yarn-level cloth. *ACM Trans. Graph.* 39, 4, Article 48 (Aug. 2020), 16 pages.

Georg Sperl, Rahul Narain, and Chris Wojtan. 2021. Mechanics-aware deformation of yarn pattern geometry. *ACM Trans. Graph.* 40, 4, Article 168 (July 2021), 11 pages.

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

Hannah Twigg-Smith, Yuecheng Peng, Emily Whiting, and Nadya Peek. 2024a. What's in a cable? Abstracting Knitting Design Elements with Blended Raster/Vector Primitives. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 62, 20 pages.

Hannah Twigg-Smith, Emily Whiting, and Nadya Peek. 2024b. KnitScape: Computational Design and Yarn-Level Simulation of Slip and Tuck Colorwork Knitting Patterns. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 860, 20 pages.

Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. 2018. Stitch meshing. *ACM Trans. Graph.* 37, 4, Article 130 (July 2018), 14 pages.

Kui Wu, Hannah Swan, and Cem Yuksel. 2019. Knittable stitch meshes. *ACM Transactions on Graphics (TOG)* 38, 1 (2019), 1–13.

Kui Wu, Marco Tarini, Cem Yuksel, James McCann, and Xifeng Gao. 2022. Wearable 3D Machine Knitting: Automatic Generation of Shaped Knit Sheets to Cover Real-World Objects. *IEEE Transactions on Visualization and Computer Graphics* 28, 9 (2022), 3180–3192.

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* (San Francisco, California) *(I3D '17)*. Association for Computing Machinery, New York, NY, USA, Article 5, 8 pages.

Kui Wu and Cem Yuksel. 2019. Real-Time Cloth Rendering with Fiber-Level Detail . *IEEE Transactions on Visualization & Computer Graphics* 25, 02 (Feb. 2019), 1297–1308.

Chun Yuan, Haoyang Shi, Lei Lan, Yuxing Qiu, Cem Yuksel, Huamin Wang, Chenfanfu Jiang, Kui Wu, and Yin Yang. 2024. Volumetric Homogenization for Knitwear Simulation. *ACM Trans. Graph.* 43, 6, Article 207 (Nov. 2024), 19 pages.

Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2012. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Trans. Graph.* 31, 4, Article 37 (July 2012), 12 pages.

Shuang Zhao, Fujun Luan, and Kavita Bala. 2016. Fitting procedural yarn models for realistic cloth rendering. *ACM Trans. Graph.* 35, 4 (2016), 51:1–51:11.

J. Zhu, Z. Montazeri, J. Aubry, L. Yan, and A. Weidlich. 2023. A Practical and Hierarchical Yarn-based Shading Model for Cloth. *Computer Graphics Forum* 42, 4 (2023), e14894.

Arno Zinke, Cem Yuksel, Andreas Weber, and John Keyser. 2008. Dual scattering approximation for fast multiple scattering in hair. *ACM Trans. Graph.* 27, 3 (2008), 32.

Lara Zlokapa, Yiyue Luo, Jie Xu, Michael Foshey, Kui Wu, Pulkit Agrawal, and Wojciech Matusik. 2022. An Integrated Design Pipeline for Tactile Sensing Robotic Manipulators. In *2022 International Conference on Robotics and Automation (ICRA)* (Philadelphia, PA, USA). IEEE Press, New York, NY, USA, 3136–3142.