DOI: 10.1111/cgf.13230

Eurographics Symposium on Rendering 2017
P. Sander and M. Zwicker
(Guest Editors)

Volume 36 (2017), Number 4

Fiber-Level On-the-Fly Procedural Textiles

Fujun Luan[1] Shuang Zhao[2] Kavita Bala[1]

[1]Cornell University [2]University of California, Irvine

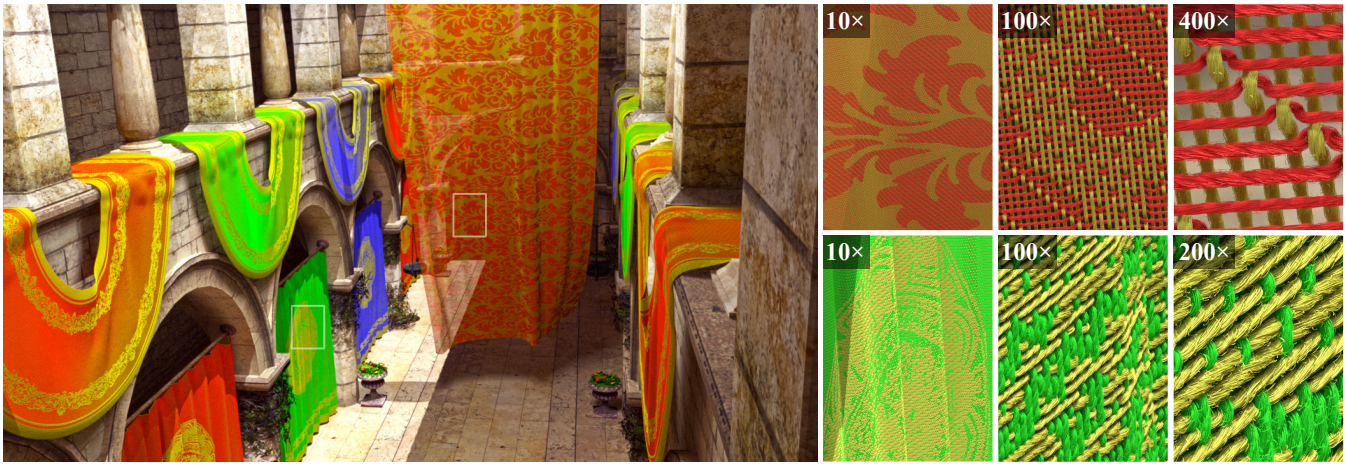


Figure 1: *The new Sponza scene by Crytek with 12 procedural textile models substituted for the fabrics. These models together represent 33 thousand yarns, composed of 8 million procedurally described fibers. Insets on the right show zoomed versions of two subregions (indicated with white rectangles), demonstrating the level of fidelity and detail offered by these models. When fully realized, all the cloth models would take 867 GB of storage, and are practically unrenderable. We introduce a realization-minimizing technique to render such large-scale procedural textiles on a single computer.*

Abstract

*Procedural textile models are compact, easy to edit, and can achieve state-of-the-art realism with fiber-level details. However, these complex models generally need to be fully instantiated (aka. **realized**) into 3D volumes or fiber meshes and stored in memory, We introduce a novel **realization-minimizing** technique that enables physically based rendering of procedural textiles, without the need of full model realizations. The key ingredients of our technique are new data structures and search algorithms that look up regular and flyaway fibers on the fly, efficiently and consistently. Our technique works with compact fiber-level procedural yarn models in their exact form with no approximation imposed. In practice, our method can render very large models that are practically unrenderable using existing methods, while using considerably less memory (60–200× less) and achieving good performance.*

CCS Concepts

•***Computing methodologies*** → ***Rendering;***

1. Introduction

Fabrics and textiles are critical to our daily lives for their beauty and utility, and are ubiquitous in clothing and apparel, fashion, home decor and furnishings. Virtually simulating the rich variety of fabric appearance across different designs and material types is very challenging because textiles exhibit diverse visual cues ranging from anisotropic highlights to fuzzy silhouettes; many of these cues cannot be fully described with simple surface-based representations.

Recently, a family of *micro-appearance models* [ZJMB11, ZJMB12, KSZ*15] have been developed that explicitly consider a

fabric's properties at the fiber-level to achieve state-of-the-art quality in simulated fabric appearance. These small-scale features have been demonstrated to be crucial for realistic renderings at closeup views and, more importantly, to significantly affect a textile's overall appearance (see Figure 2 and [ZLB16]).

These micro-appearance models are generally data-driven and based on measured micro-geometries described as either high-resolution 3D volumes [ZJMB11], or large collections of individual fiber curves [KSZ*15]. These data-intensive approaches are unwieldy to manipulate, and cannot be scaled to full-size models of fabrics and textiles, as shown in Figures 1, 12, and 13. To address these problems, *procedural models* have been introduced to fabrics by Schröder et al. [SZK15] and refined by Zhao et al. [ZLB16]. These models represent fabric micro-geometry procedurally, via specifically tailored probability distributions, and enjoy high fidelity, compact representation, and easy editability.

However, one major problem remains: the procedural models still need to be *fully realized* before being rendered using physically based methods. In other words, to render these models, they have to be converted back to expensive data-driven formats as either high-resolution volume-based representations or detailed polygonal meshes representing large collections of cloth fibers. This need of full realization fundamentally limits the practical use of procedural models since it not only neglects the benefits offered by being procedural, but also causes large textiles (e.g., those with thousands of yarns, and correspondingly millions of fibers) practically unrenderable due to the excessive amount of data required. Figure 1, for instance, displays a large scene with 12 procedural textiles composed of 33 thousand yarns and over 8 million fibers. A full realization of all these textiles would take 867 GB of storage, making this scene unrenderable for existing methods.

Mathematically, the micro-geometry of a procedural textile is fully determined by the underlying model parameters. So, in principle, all properties (e.g., local material density and albedo) needed for physically based rendering of a procedurally described textile can be computed on the fly. Unfortunately, because of the complex relationship between the procedural model parameters and the rendering properties, it is challenging to compute these properties on the fly.

In this paper, we introduce a novel *realization-minimizing* technique for physically based rendering of large-scale procedural textiles. Precisely, our method provides a volumetric representation allowing the material properties (e.g., optical density) to be queried at render time with only a small fraction of the full model realized. These volumes can then be rendered using standard Monte Carlo solutions (e.g., volume path tracing) to the anisotropic radiative transfer problem [JAM*10a].

Technically, our contributions include:

- A novel realization-minimizing framework enabling physically based rendering of large-scale procedural textiles. Our framework retains the full generality of the procedural model by working with its state-of-the-art form exactly, and imposes no approximations;
- New data structures and volumetric search algorithms connecting the procedural model parameters and material optical properties in an efficient and consistent way.

We demonstrate the practical impact of this approach using a variety of virtual textiles, from wovens (Figures 1 and 12) to knits (Figure 13), at widely different scales. The level of fidelity and detail offered by our technique not only creates realistic zoomed renderings, but also enables predictive reproduction of fabric appearance based on their fundamental (fiber-level) properties. Such predictive rendering is critical for design and virtual prototyping applications.

2. Related Work

Cloth appearance: Modeling and reproducing cloth appearance has been an active research topic in computer graphics for decades. Many surface reflectance models, which treat fabrics as infinitely thin 2D sheets, have been proposed (e.g., [IM12, SBDDJ13]). Although these models provide high-quality renderings for fabrics viewed from the far field, they lack the power to closely reproduce the appearance of fabrics at the near field, which is important for applications in design and virtual reality.

Micro-appearance cloth models: Micro-appearance models [ZJMB11, KSZ*15, SZK15, ZLB16] produce superior quality for virtual fabrics by explicitly capturing the geometric and optical properties of a fabric down to the fiber level. The first micro-appearance cloth model was introduced by Zhao et al. [ZJMB11] as a special medium in which volumetric scattering of light is given by the micro-flake model [JAM*10a]. Later, improvements to the micro-flake model were developed [KSZ*15, HDCD15]. Other than volumetric representations, fiber meshes coupled with specialized light scattering models such as bidirectional curve scattering distribution functions (BCSDFs) have been demonstrated to be equally effective in reproducing fabric appearance [SZK15, KSZ*15].

Early micro-appearance cloth models were highly data-intensive, requiring many giga-bytes of data to describe a coin-sized sample, making these models very expensive to store. To address this problem, structure-aware synthesis [ZJMB12] has been developed. This technique builds large models by replicating a small set of example blocks, and the synthesized models can be efficiently rendered using modular flux transfer [ZHRB13]. However, this technique requires the final model to have a regular 2D tiling grid, which is the case for woven fabrics but not for many others such as knitted textiles. Further, the synthesis process uses randomized tiling which can lead to noticeable repeated patterns. Please refer to §6.1 for more details.

Procedural textiles: Procedural models [SZK15, ZLB16] have recently been introduced to graphics. These models offer superior model compactness and editability by leveraging specialized probability distributions to describe the arrangement of fibers within a fabric (see §3.2 for details). Unfortunately, as previous work on procedural cloth modeling has mainly focused on the core model descriptions, how these models can be rendered in practice without full model realization has remained largely ignored.

Recently, Wu and Yuksel [WY17] introduced a real-time GPU-based technique to visualize procedural textiles. They approximate

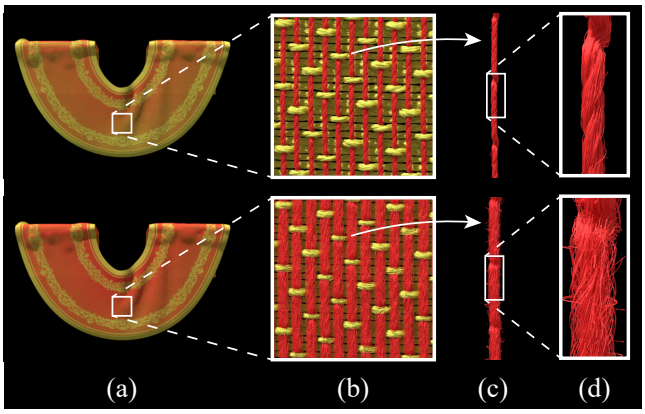


Figure 2: ***Textile structure hierarchy and its effect on fabric appearance:*** *A textile (a) is formed by hundreds to thousands of yarns (b). Each yarn (c), in turn, is composed of tens to hundreds of micro-diameter fibers (d). The small-scale structures (c, d) of a fabric can greatly impact its overall appearance. While the top and bottom fabrics have identical weave patterns and warp/weft colors, since the warp yarns in both textiles have different sizes and fuzziness, the two models have very different overall appearance.*

the procedural model to generate fiber-level geometries at runtime on the GPU, and visualize the resulting model using rasterization. This technique is great for interactively visualizing procedural textiles but has approximations (e.g., baked-in shadows, missing subsurface scattering, limitations on the number of fibers) that impact quality for applications like textile design and virtual prototyping. In contrast, we see our work as the complete solution that these applications will build on for high-fidelity rendering. On the other hand, some technical components of their work such as the core-fiber approximation may be adopted for physically based rendering. Please refer to §6.1 for a detailed discussion.

Procedural granular media: The methods proposed for modeling and rendering granular materials (e.g., sand, snow, and sugar) [MPH*15, MPG*16] involve procedurally generating volumetric representations of grains. However, this technique cannot be applied to our problem as the procedural model for granular media is very different from that for textiles.

3. Terminology and Background

3.1. Hierarchy of Textile Structures

The structure of a textile is complex at many scales (see Figure 2, left to right). A full textile is composed of thousands of threads, or *yarns* (Figure 2-b), via manufacturing techniques such as weaving and knitting. Each yarn (Figure 2-c) in turn is created by "spinning" or twisting together tens to hundreds of micron-radius filaments, or *fibers* (Figure 2-d). Many real-world yarns contain multiple sub-strands, or *piles*, each of which consists of fibers twisted around a common center. Previous research has demonstrated that the shape and arrangement of yarns and fibers greatly affect textile appearance [ZJMB11, IM12, SBDDJ13, SZK15, KSZ*15] (see Figure 2, top vs. bottom row, for an example).

3.2. Procedural Textiles

Given the hierarchy of textile structures, we now describe their mathematical representations. At the finest level, the constituent fibers in a yarn are specified procedurally using previously developed models [SZK15, ZLB16]. Then, individual yarns are combined to form full textiles.

Procedural yarn modeling: To procedurally specify the fiber distribution within a single yarn, the following model has been proposed in textile research and adapted to graphics by Schröder et al. [SZK15] and refined by Zhao et al. [ZLB16]. Assuming the yarn center to be the Z-axis, each fiber j is then modeled as a circular helix $\boldsymbol{h}_j^{\text{reg}}$ parameterized by z:[1]

$$\boldsymbol{h}_j^{\text{reg}}(z) := \left[R_j \cos\left(\theta_j + \tfrac{2\pi z}{a}\right),\ R_j \sin\left(\theta_j + \tfrac{2\pi z}{a}\right),\ z\right], \quad (1)$$

where R_j and a respectively control the radius and pitch of $\boldsymbol{h}_j^{\text{reg}}$. To better capture the migration of fibers exhibited by real-world yarns, R_j in Eq. (1) is further allowed to vary in a sinusoidal pattern via

$$R_j(z) = r_{\min} R_j + \tfrac{r_{\max} - r_{\min}}{2} R_j \left[\cos\left(s\tfrac{2\pi z}{a} + \theta_j^{(0)}\right) + 1\right], \quad (2)$$

where $r_{\min}$, $r_{\max}$ are two scaling factors that determine the range of $R_j(z)$, s specifies the period of one cycle, and $\theta_j^{(0)}$ gives the phase of the variation. In Eq. (1) and (2), a, $r_{\min}$, $r_{\max}$, and s are shared by all fibers in a yarn while θ_j, $\theta_j^{(0)}$, and R_j are per-fiber properties. In practice, θ_j, $\theta_j^{(0)}$ are sampled uniformly and R_j is drawn from a cross-sectional fiber distribution with parameters ε, $\beta \in \mathbb{R}_+$ using rejection sampling upon the (unnormalized) probability density:

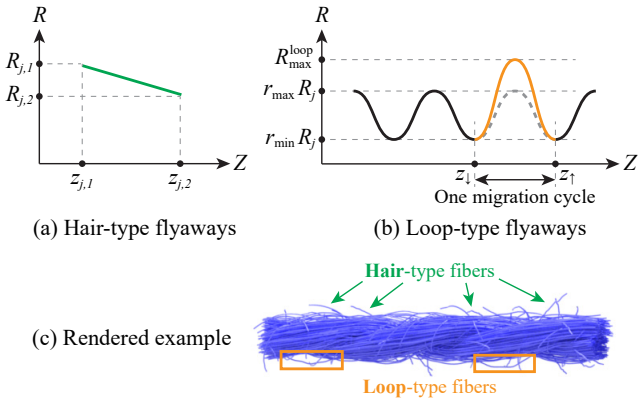
$$p(R) := (1 - 2\varepsilon)\left(\frac{e - e^R}{e - 1}\right)^{\beta} + \varepsilon. \quad (3)$$

Fibers captured by Eq. (1), (2), and (3) are considered *regular* as they closely follow the yarn center. In reality, however, irregularities exist: some fibers may deviate from the main stream and leave open endpoints or loops beyond the main textile surface. Despite being small in number, these *flyaway* fibers are visually significant as they are the main cause for a fabric's fuzzy appearance. In this paper, we follow the model introduced by Zhao et al. [ZLB16] which describes flyaway fibers in two categories: *hair* and *loop*. In particular, given $z_{j,1}, z_{j,2} \in [0, h]$ and $R_{j,1}, R_{j,2}, \theta_{j,1}, \theta_{j,2} \in \mathbb{R}$, each hair-type flyaway fiber j is treated as an arc parameterized by $z \in [z_{j,1},\ z_{j,2}]$:

$$\boldsymbol{h}_j^{\text{hair}}(z) = [R_j^{\text{hair}}(z)\cos\theta_j^{\text{hair}}(z),\ R_j^{\text{hair}}(z)\sin\theta_j^{\text{hair}}(z),\ z], \quad (4)$$

where R_j^{hair} and θ_j^{hair} are linearly interpolated in $R_{j,1}, R_{j,2}$ and $\theta_{j,1}, \theta_{j,2}$, respectively (see Figure 3-a). The loop-type flyaways, on the other hand, are created by mutating existing regular fibers. When creating one loop-type fiber, one migration cycle $(z_\downarrow, z_\uparrow)$ is

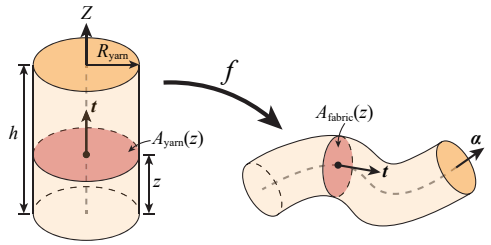
[1] Eq. (1) and (2) assume all fibers in a yarn are twisted around one common center, the Z-axis. In reality, many yarns consist of multiple sub-strands, or *plies*, such that the fibers reside around individual ply centers (which are then twisted around the yarn center). For better readability, we assume all yarns to be single-ply in §3 and most of §4. §4.5 describes how we generalize to handle multi-ply yarns.



(a) Hair-type flyaways (b) Loop-type flyaways

(c) Rendered example

Figure 3: ***Flyaway fiber models:*** *(a) Hair-type flyaways are described as arcs with endpoints specified in z, R, and θ. (b) Loop-type flyaways are obtained by mutating entire migration cycles (marked in orange) of regular fibers; (c) Rendering of a single yarn with both types of flyaways.*



(a) Yarn space (b) Fabric space

Figure 4: ***Yarn space and fabric space:*** *in the yarn space (a), all constituent fibers are assumed to follow one common center: the Z axis. In the fabric space (b), arbitrary smooth curve $\boldsymbol{\alpha}$, parameterized by arc length, is used to specify the yarn center. Transformation $f : \mathbb{R}^3 \to \mathbb{R}^3$ maps locations from the yarn space to the fabric space. In particular, it maps yarn-space cross section $A_{yarn}(z)$ to the fabric-space one $A_{fabric}(z)$.*

selected from a randomly chosen regular fiber j such that $z_\uparrow - z_\downarrow = a/s$ and $R_j(z_\downarrow) = R_j(z_\uparrow) = r_{\min} R_j$. The corresponding fiber segment $\{\boldsymbol{h}^{\text{reg}}_j(z) \mid z \in [z_\downarrow, z_\uparrow]\}$ then has its radius $R_j(z)$ scaled up uniformly to reach some randomly drawn radius $R^{\text{loop}}_{\text{max}}$ at its maximum, creating a visible loop (see Figure 3-b).

Yarn space and fabric space: To compose multiple procedurally described yarns into full textiles, we need to deform them to follow arbitrary center curves instead of the Z-axis. Formally, we seek a transformation $f : \mathbb{R}^3 \to \mathbb{R}^3$ per yarn that converts *yarn-space* coordinates centered around the Z-axis to *fabric-space* ones following an arbitrary smooth curve (Figure 4).

Given a curve $\boldsymbol{\alpha}$ parameterized by arc length, f maps each cross section in yarn space to the corresponding one in fabric space. Precisely, assume without loss of generality that the yarn-space center curve has endpoints $[0,0,0]$ and $[0,0,h]$ for some $h > 0$. Then, f should transform any yarn-space cross section

$$A_{\text{yarn}}(z) := \{(x,y,z) \mid x,y \in \mathbb{R}\}, \tag{5}$$

for some $z_0 \in [0,h]$ to the corresponding one in fabric-space, given by:

$$A_{\text{fabric}}(z) := \{\boldsymbol{r} \in \mathbb{R}^3 \mid \langle \boldsymbol{t}(z),\ \boldsymbol{r} - \boldsymbol{\alpha}(z) \rangle = 0\}, \tag{6}$$

where $\langle \cdot,\cdot \rangle$ is the inner product operator, and $\boldsymbol{t}(z) := (\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\alpha})(z)$ denotes the tangent direction of $\boldsymbol{\alpha}$ at z. This can be achieved by setting:

$$f([x,y,z]) = x\,\boldsymbol{n}(z) + y\,\boldsymbol{b}(z) + \boldsymbol{\alpha}(z), \tag{7}$$

where $\boldsymbol{n}(z)$ denotes the normal direction at $\boldsymbol{\alpha}(z)$ and $\boldsymbol{b} := \boldsymbol{t} \times \boldsymbol{n}$. It is easy to verify that Eq. (7) maps the yarn-space center curve $\{(0,0,z) \mid z \in [0,h]\}$ to the fabric-space one $\boldsymbol{\alpha}$.

3.3. Micro-Appearance Textile Models

Two types of micro-appearance models can be used to render textiles with procedurally described fiber-level details. *Volumetric* models [ZJMB11, ZJMB12] treat textiles as anisotropic participating media [JAM*10a, HDCD15] which can be rendered using the radiative transfer framework [Cha60]. Alternatively, *fiber-based* representations [KSZ*15, SZK15, ZLB16] rely on Bidirectional Curve Scattering Distribution Functions (BCSDFs) attached to individual fibers which then can be rendered using standard ray tracing.

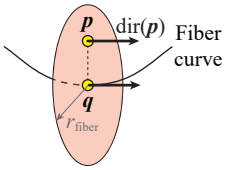
Khungurn et al. [KSZ*15] compared these approaches and demonstrated that both the volumetric and the fiber-based models offer similar levels of quality. In this paper, we use the volumetric representation to render procedural textiles because (i) without highly detailed fiber surfaces, these models render faster; and (ii) the rendering process only requires accessing material properties locally, which is desirable for developing solutions with minimal model realization.

The volumetric formulation requires several parameters that must be specified at each 3D location inside a fabric's bounding volume: the extinction coefficient σ_t, single-scattering albedo α, and the phase function f_p. In the context of cloth rendering, these parameters are generally derived from the local density, tangent direction, and the cloth fibers' optical properties (see Appendix A for more details).

4. Realization-Minimizing Model Lookups

We now present the key ingredient of our technique: accessing procedurally described textiles without fully realizing fiber geometry. At each location $\boldsymbol{p} \in \mathbb{R}^3$ in fabric space, our textile model provides local material density $\text{den}(\boldsymbol{p}) \in \mathbb{R}_+$ and direction information $\text{dir}(\boldsymbol{p}) \in \mathbb{S}^2$, which in turn determine the radiative transfer parameters needed for rendering (§3.3).

In principle, $\text{den}(\boldsymbol{p})$ and $\text{dir}(\boldsymbol{p})$ can be determined based on the fabric-space locations of individual fibers (see the adjacent figure). Given $\boldsymbol{p}$, if there exists a fiber curve such that the minimal distance from



Algorithm 1 Outline of our approach.

Input: Yarn curves $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots$; per-yarn procedural model parameters and fiber scattering profiles

Output: Whether a query point $\boldsymbol{p}$ is inside a fiber; data block $\boldsymbol{b}$ that contains volume density, albedo and orientation information

```
1:  function LOOKUP(p, b)
2:      I ← LOOKUPYARN(p)                                  ▷ §4.2
3:      for all i ∈ I do
4:          p^yarn ← TRANSFORM(i, p)                       ▷ §4.2
5:          if LOOKUPREGULAR(i, p^yarn, b) then
6:              return true
7:          end if                                         ▷ §4.3
8:          if LOOKUPFLYAWAY(i, p^yarn, b) then
9:              return true
10:         end if                                         ▷ §4.4
11:     end for
12:     return false
13: end function
```

$\boldsymbol{p}$ to this curve is below the fiber's radius, then $\text{den}(\boldsymbol{p})$ equals the density of this fiber; otherwise, $\text{den}(\boldsymbol{p}) = 0$. When $\text{den}(\boldsymbol{p}) > 0$, we further compute the projection $\boldsymbol{q}$ of $\boldsymbol{p}$ on the fiber curve and set $\text{dir}(\boldsymbol{p})$ to the tangent direction at $\boldsymbol{q}$. In this way, computing material density and direction at any given location $\boldsymbol{p}$ boils down to searching for a fiber curve lying close enough to this point and finding its projection $\boldsymbol{q}$ on that curve (if there exists one), as outlined in Algorithm 1.

We introduce a novel solution to this problem **without** creating complete model realizations. Our objective is to render large-scale procedural textile models exactly (i.e., not approximating the model itself) while minimizing the amount of data needed to be realized and preserving, as much as possible, rendering performance.

4.1. Input

The input to our technique includes a set of n yarn curves $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_n$ and per-yarn parameters for (i) the procedural fiber geometry used by Eq. (1), (2), (3), and (4); and (ii) the material's optical properties including density and single-scattering albedo (see Appendix A for more details).

Notice that, although the input contains explicitly stored yarn curves, the amount of data involved is negligible compared to a fully realized model. This is because for each yarn, the input contains only a single curve while the fully realized version would involve hundreds of fiber curves. Allowing explicitly described yarn centers also enjoys high flexibility. For instance, we can directly take yarn-based simulation results [KJM08, CLMMO14] and add procedural fiber-level details to these models to produce high-quality renderings (see Figure 13).

4.2. Yarn Lookups

To determine the local fiber density and direction at a given fabric-space point $\boldsymbol{p}$, we first need to determine the yarn to which this point belongs (if any) so that further checks based on the procedural model depicted in §3.2 can be performed (Line 2 of Algorithm 1).

Identifying relevant yarns: Since each yarn in the fabric space occupies a pipe-shaped volume determined by the yarn's center curve and radius (Figure 4-b), a yarn contains $\boldsymbol{p}$ if and only if the minimal distance from this point to the yarn's center falls below its radius. To efficiently identify all such yarns, we leverage a point Kd-tree containing the vertices of all yarn curves. We postpone detailed descriptions of this step to §5.2.

Fabric-to-yarn-space transformation: For each yarn i containing $\boldsymbol{p}$, we need to check if this point is indeed covered by a constituent fiber. Since all fibers are described procedurally in yarn space via Eq. (1), (2), and (3) for regular ones as well as Eq. (4) for flyaways, it is desirable to perform the check in this same space. We, therefore, transform $\boldsymbol{p}$ back to the corresponding yarn space location $\boldsymbol{p}^{\text{yarn}}$ (Line 4 of Algorithm 1) by inverting the transformation specified by Eq. (7). Namely,

$$\boldsymbol{p}^{\text{yarn}} = f_i^{-1}(\boldsymbol{p}), \tag{8}$$

where the subscript i emphasize that f varies across different yarns. Let $\boldsymbol{p}^{\text{yarn}} = [x, y, z]$, Eq. (7) then gives

$$x\,\boldsymbol{n}_i(z) + y\,\boldsymbol{b}_i(z) = \boldsymbol{p} - \boldsymbol{\alpha}_i(z),$$

where $\boldsymbol{\alpha}_i$ denotes the center curve of yarn i, and $\boldsymbol{n}_i$, $\boldsymbol{b}_i$ are respectively the normal and bitangent directions along $\boldsymbol{\alpha}_i$. Here we assume $\boldsymbol{t}_i$, $\boldsymbol{n}_i$, and $\boldsymbol{b}_i$ are all given with $\boldsymbol{\alpha}_i$. In practice, we compute these directions using $\boldsymbol{\alpha}_i$. Please refer to §5.1 for more details.

Since $\boldsymbol{n}_i(z)$ and $\boldsymbol{b}_i(z)$ are orthogonal to each other, it holds that

$$x = \langle \boldsymbol{p} - \boldsymbol{\alpha}_i(z),\ \boldsymbol{n}_i(z) \rangle, \quad y = \langle \boldsymbol{p} - \boldsymbol{\alpha}_i(z),\ \boldsymbol{b}_i(z) \rangle, \tag{9}$$

and evaluating Eq. (8) boils down to finding a proper z.

Let $\boldsymbol{q}$ be the projection of $\boldsymbol{p}$ on $\boldsymbol{\alpha}_i$, then $\boldsymbol{p}$ and $\boldsymbol{q}$ belong to the same fabric-space cross section. Since f maps yarn-space cross sections (Eq. (5)) to the corresponding ones in fabric space (Eq. (6)), $\boldsymbol{p}^{\text{yarn}}$ and $\boldsymbol{q}^{\text{yarn}}$ should come from the same yarn-space cross section and share identical z coordinates. In practice, $\boldsymbol{q}$ is obtained when computing the minimal distance between $\boldsymbol{p}$ and the yarn center (see §5). After finding $\boldsymbol{q} = \boldsymbol{\alpha}_i(z)$, $\boldsymbol{p}^{\text{yarn}}$ can be obtained using Eq. (9) as

$$\boldsymbol{p}^{\text{yarn}} = [\langle \boldsymbol{p} - \boldsymbol{\alpha}_i(z), \boldsymbol{n}_i(z) \rangle,\ \langle \boldsymbol{p} - \boldsymbol{\alpha}_i(z), \boldsymbol{b}_i(z) \rangle,\ z].$$

With $\boldsymbol{p}^{\text{yarn}}$ determined, we then check if it is covered by a regular (§4.3) or a flyaway fiber (§4.4) based on the procedural model described in §3.2.

4.3. Regular Fiber Lookups

Regular fibers form the main body of a yarn, and properly capturing their structure is crucial for reproducing a textile's overall appearance, from specular highlights to textures (Figure 5 shows how critical fiber-level details are for visual realism.).

Thus, our goal is to develop a realization-minimizing technique to look up regular fibers. This is not too difficult if there is no fiber migration (i.e., R_j in Eq. (1) remains constant), but it is challenging with migration which adds important visual irregularity. We now discuss how we address this problem.

The procedural model describes regular fibers as helices using

Eq. (1). Since $\boldsymbol{h}_j^{\text{reg}}$ is parameterized with z, we determine whether a given yarn-space location $\boldsymbol{p}^{\text{yarn}} = [x, y, z]$ is covered by a regular fiber in the cross sections A_{yarn} under the same parameterization. Let $\hat{\boldsymbol{h}}_j^{\text{reg}}(z)$ denote the cross-sectional location of fiber j, which equals the sub-vector of $\boldsymbol{h}_j^{\text{reg}}(z)$ involving its X and Y components. Then, $\boldsymbol{p}^{\text{yarn}}$ is covered by a regular fiber j if and only if $[x, y]$ and $\hat{\boldsymbol{h}}_j^{\text{reg}}(z)$ lie close enough (see Figure 6-(a)). We now introduce an efficient approach to determine the existence of such j.

Without fiber migration: We first consider the special case where fiber migration Eq. (2) is ignored. In this case, for each regular fiber j, the cross-sectional location $\hat{\boldsymbol{h}}_j^{\text{reg}}$ stays at a constant distance R_j from the yarn center (i.e., the Z-axis). Since all fibers share the same pitch a, all cross-sectional fiber locations remain static relative to each other. In other words, all fiber locations $\hat{\boldsymbol{h}}_1^{\text{reg}}(z)$, $\hat{\boldsymbol{h}}_2^{\text{reg}}(z), \ldots$ are simply a rotated version of $\hat{\boldsymbol{h}}_1^{\text{reg}}(0)$, $\hat{\boldsymbol{h}}_2^{\text{reg}}(0), \ldots$ (see Figure 6-(a2)). In particular, for all j, it is easy to verify that

$$\hat{\boldsymbol{h}}_j^{\text{reg}}(z)M(z) = [R_j \cos\theta_j,\ R_j \sin\theta_j] \equiv \hat{\boldsymbol{h}}_j^{\text{reg}}(0), \tag{10}$$

where $M(z)$ is a 2×2 rotation matrix given by

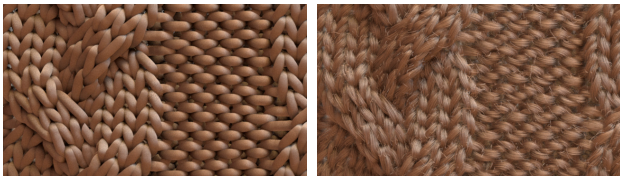
$$M(z) = \begin{bmatrix} \cos(2\pi z/a) & -\sin(2\pi z/a) \\ \sin(2\pi z/a) & \cos(2\pi z/a) \end{bmatrix}.$$

Therefore, to check if $\boldsymbol{p}^{\text{yarn}} = [x, y, z]$ is contained in a fiber, we can rotate $[x, y]$ by applying the same $M(z)$ as Eq. (10) to obtain $[x', y'] := [x, y]M(z)$ and determine if $[x', y']$ is covered by a cross-sectional fiber in the cross section at $z = 0$ (Figure 6-(a2)).

To check if this is the case, we store $\hat{\boldsymbol{h}}_j^{\text{reg}}(0)$ for all fibers in all yarns. Then, the whole search for fibers covering $\boldsymbol{p}^{\text{yarn}}$ becomes a 2D nearest neighbor (NN) lookup around $[x', y']$ in the cross section $A_{\text{yarn}}(0)$. This is a standard problem that has been thoroughly studied and can be solved very efficiently. Notice that, compared to fully realizing all fibers which requires storing complete fiber curves, our method consumes orders of magnitude less memory as it only stores two floating numbers per regular fiber.

With fiber migration: As demonstrated in prior work [SZK15, ZLB16], real-world fibers generally do not stay at a constant distance from the yarn center. This irregularity is visually important, but is challenging to model without a completely instantiated model.

In our case, the migration of fibers is modeled procedurally via Eq. (2). Then, Eq. (10) no longer holds: instead, $\hat{\boldsymbol{h}}_j^{\text{reg}}(z)M(z)$ now



(a) Yarn curve only (b) With fiber-level details

Figure 5: ***Fiber-level details*** *are critical for producing realistic silhouettes and close-up views.*

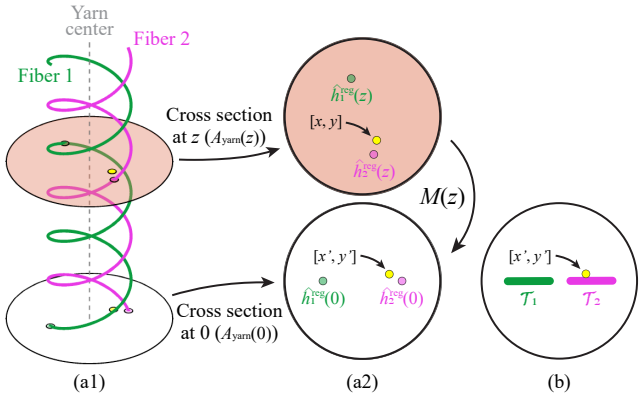


Figure 6: ***Regular fiber lookups:*** *A yarn-space point $\boldsymbol{p}^{yarn} = [x, y, z]$ is covered by a regular yarn if there is a fiber center $\hat{\boldsymbol{h}}_j^{reg}(z)$ located close enough to $[x, y]$ in the cross section $A_{yarn}(z)$. (a) Without fiber migration, all fiber locations are identical at each cross section up to a global rotation M. Thus, one can determine if the rotated version $[x', y'] := [x, y]M(z)$ is close enough to some $\hat{\boldsymbol{h}}_j^{reg}(0)$ within $A_{yarn}(0)$. (b) With fiber migration, we reason about the trajectory $\mathcal{T}_j$, namely all possible locations $\boldsymbol{h}_j^{reg}(z)M(z)$ can take, for each fiber j. This allows us to quickly prune all fibers whose trajectories have stayed far from $[x', y']$.*

varies with z:

$$\hat{\boldsymbol{h}}_j^{\text{reg}}(z)M(z) = [R_j(z) \cos\theta_j,\ R_j(z) \sin\theta_j], \tag{11}$$

where $R_j(z)$ is given by Eq. (2). Since Eq. (11) does not equal $\hat{\boldsymbol{h}}_i^{\text{reg}}(0)$ in general, whether $\boldsymbol{p}^{\text{yarn}} = [x, y, z]$ is covered by a regular fiber can no longer be determined by checking if $[x, y]M(z)$ lies close enough to $\hat{\boldsymbol{h}}_j^{\text{reg}}(0)$ for some fiber j.

To tackle this problem, we consider all possible values $\hat{\boldsymbol{h}}_j^{\text{reg}}(z)M(z)$ can take. If $[x, y]M(z)$ stays far away from all these values, $[x, y, z]$ cannot be covered by fiber j. Precisely, let $\mathcal{T}_j \subset \mathbb{R}^2$ be the *trajectory* of $\hat{\boldsymbol{h}}_j^{\text{reg}}(z)M(z)$ for all $z \in \mathbb{R}$. Then,

$$\begin{aligned} \mathcal{T}_j &:= \{\hat{\boldsymbol{h}}_j^{\text{reg}}(z)M(z) \mid z \in \mathbb{R}\} \\ &= \left\{[R\cos\theta_j, R\sin\theta_j] \;\middle|\; R \in [r_{\min}R_j, r_{\max}R_j]\right\}, \end{aligned} \tag{12}$$

Algorithm 2 Regular fiber lookups.

```
1:  function LookupRegular(i, p^yarn, b)
2:      [x', y'] ← [x, y]M(z)              ▷ Assuming p_ply = [x, y, z]
3:      Finding the bin B_k containing θ' := atan2(y', x')
4:      for all T_j associated with B_k do
5:          Realize fiber j on the fly
6:          if fiber j covers p^yarn then
7:              Store local material information (e.g., density) in b
8:              return true
9:          end if
10:     end for
11:     return false
12: end function
```

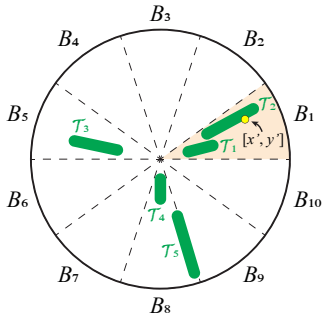


Figure 7: ***Handling fiber migration:*** *As preprocessing, we partition the yarn-space cross section $A_{yarn}(0)$ into a number of bins $B_1, B_2, \ldots$ based on the azimuthal angle θ. For each bin B_k, we store the list of fibers whose trajectories lie close enough to B_k. At run-time, given the query point $[x', y']$, we can quickly find the bin containing this point and obtain the associated fibers. In this example, $[x', y']$ belongs to B_1, and $\mathcal{T}_1$ and $\mathcal{T}_2$ are the two fiber trajectories intersecting B_1.*

where the second equality follows Eq. (2) and (11). As $\cos\theta_j$ and $\sin\theta_j$ are fixed in Eq. (12), $\mathcal{T}_j$ represents a line segment with slope $\tan\theta_j$ and intercept zero (see Figure 6-(b)). Since $\mathcal{T}_j$ captures *all* possible values of $\hat{\boldsymbol{h}}_j^{\text{reg}}(z)M(z)$, a yarn-space point $[x,y,z]$ cannot be covered by fiber j if the minimal distance between $[x,y]M(z)$ and $\mathcal{T}_j$ is greater than the fiber radius. Thus, we can pre-compute and store $\mathcal{T}_j$ as tuples of $(r_{\min}R_j,\ r_{\max}R_j,\ \theta_j)$ for all fibers. Notice that, compared to the migration-free case where $\hat{\boldsymbol{h}}_j^{\text{reg}}(0)$ is used, $\mathcal{T}_j$ only takes one extra floating number per fiber to describe.

At render time, given $\boldsymbol{p}^{\text{yarn}} = [x,y,z]$, we then select all fibers j with trajectories $\mathcal{T}_j$ lying close enough to $[x',y'] := [x,y]M(z)$. To do this efficiently, we partition the domain of θ_j, namely $[0,2\pi)$, into a few equal sized components $B_1, B_2, \ldots$. For each component B_t, we then associate to it a 2D interval $(r_{\min}R_j,\ r_{\max}R_j)$ for all j satisfying $\theta_j \in B_t$ as preprocessing.[2] At runtime, given $[x',y']$, we rewrite it in polar coordinates as $[r',\theta']$. Let $B \in \{B_1, B_2, \ldots\}$ be the component containing θ'. We can then retrieve all intervals associated with B that contain r' and examine each of them to see if the corresponding fiber actually covers the given point (Figure 7).

The full process of our regular fiber lookup technique is summarized in Algorithm 2. Notice that Lines 4–6 realize all fiber j associated with B_k to determine if any of them covers $\boldsymbol{p}^{\text{yarn}}$. It is possible to further accelerate this step by introducing acceleration structures over the trajectories $\mathcal{T}_j$ (so that those faraway from $\boldsymbol{p}^{\text{yarn}}$ can be quickly pruned). However, we find this unnecessary in practice since a fine discretization of θ guarantees each bin B_k to be associated with a small number of trajectories.

[2] In practice, since each fiber has some non-zero radius r^{fiber}, we make each component B_t to store the list of fibers whose trajectories lie close enough to B_t. In the example illustrated in Figure 7, for instance, fiber 5 is associated with both B_8 and B_9.

4.4. Flyaway Fiber Lookups

As described in §3.2, flyaway fibers capture naturally arising irregularities and contribute significantly to visual realism. Compared to regular fibers that share many common properties (e.g., pitch and migration period), the flyaways are much more different individually, causing significant challenges for realization-free solutions.

4.4.1. Loop-Type Fibers

We first focus on loop-type flyaways. As described in §3.2, these loops are created by mutating the regular ones. For each yarn i, assume there are m constituent fibers each with k migration cycles (see Figure 3-(b)). Let ρ^{loop} denote the density of loops. Then, the expected number of loops in this yarn is $m^{\text{loop}} := \rho^{\text{loop}}h$ where h is the length of the yarn center. Since the loops are uniformly chosen among all migration cycles, each cycle has a probability of $p := m^{\text{loop}}/(mk)$ to be mutated. Given this probability, we can then easily extend our regular fiber lookup routine to occasionally scale up a fiber's radius (according to randomly sampled $R_{\max}^{\text{loop}}$). In particular, when realizing a regular fiber j on the fly (Line 5 of Algorithm 2), we further determine if the corresponding migration cycle is mutated (see §5 for more details on how this can be done in a consistent way). If so, we then use a different R_j (given by $R_{\max}^{\text{loop}}$ from the procedural model) to check if the mutated fiber covers $\boldsymbol{p}^{\text{yarn}}$.

4.4.2. Hair-Type Fibers

With the loop fibers properly handled, we now focus on hair-type flyaways described using Eq. (4). Compared to loop-type fibers, the hair-type ones are even more irregular and challenging to handle in a realization-minimizing way. In particular, unlike regular fibers that are valid for the entirety of $z \in [0,h]$, a hair fiber j is only defined between $z_{j,1}$ and $z_{j,2}$ (see Eq. (4) and Figure 3-(a)). It follows that, given $\boldsymbol{p}^{\text{yarn}} = [x,y,z]$, every hair fiber j covering $\boldsymbol{p}^{\text{yarn}}$ should have $z_{j,1}$ and $z_{j,2}$ with $z \in [z_{j,1} - r^{\text{fiber}},\ z_{j,2} + r^{\text{fiber}}]$. Notice that we extended the interval of $[z_{j,1}, z_{j,2}]$ using the fiber radius r^{fiber} since $\boldsymbol{p}^{\text{yarn}}$ with z slightly smaller than $z_{j,1}$ or greater than $z_{j,2}$ can still be within the range of r^{fiber} to the fiber center. Let $\mathcal{J}(z)$ denote the set of hair fibers satisfying this condition:

$$\mathcal{J}(z) := \{j \mid z \in [z_{j,1} - r^{\text{fiber}},\ z_{j,2} + r^{\text{fiber}}]\}. \tag{13}$$

Then, obtaining $\mathcal{J}(z)$ becomes an important step for looking up hair-type flyaways at $\boldsymbol{p}^{\text{yarn}}$.

We now introduce two complementary solutions to obtain $\mathcal{J}(z)$ at render time. First, we present a purely realization-free solution where no per-fiber information has to be pre-generated or stored. Second, we introduce a solution which stores partial information for each hair fiber but offers significantly superior performance.

Fully realization-free solution: According to Eq. (13), each j in $\mathcal{J}(z)$ should satisfy $z \geq z_{j,1} - r^{\text{fiber}}$ and $z \leq z_{j,2} + r^{\text{fiber}}$. In other words, $z_{j,1} \leq z + r^{\text{fiber}}$ and $z_{j,2} \geq z - r^{\text{fiber}}$. Thus, by embedding each hair fiber j into a 2D Euclidean space at $\boldsymbol{z}_j := (z_{j,1}, z_{j,2})$, we know that

$$\boldsymbol{z}_j \in D(z) \quad \text{where} \quad D(z) := [0, z + r^{\text{fiber}}] \times [z - r^{\text{fiber}}, h].$$

Computing $\mathcal{J}(z)$ then becomes finding all hairs embedded within

Algorithm 3 Detecting all hair fibers active at z (realization-free).

function ACTIVEFIBERS(z, I, Ω)
  $n \leftarrow |I|$
  if $n = 0$ or $\Omega \cap D(z) = \emptyset$ **then**
    return $\emptyset$
  else if $n = 1$ **then**
    Draw $\boldsymbol{z}_j$ ▷ Assuming $I = \{j\}$
    return $\boldsymbol{z}_j \in D(z)$? $\{j\}$: $\emptyset$
  else
    Subdivide Ω into four sub-domains $\Omega_1, \Omega_2, \Omega_3, \Omega_4$
    $p_k \leftarrow \mathbb{P}[\boldsymbol{z} \in \Omega_k \mid \boldsymbol{z} \in \Omega]$ for $k = 1,2,3,4$
    Draw (n_1, n_2, n_3, n_4) from multinomial distribution with n trials and probabilities (p_1, p_2, p_3, p_4)
    Partition I into I_1, I_2, I_3, I_4 according to n_1, n_2, n_3, n_4
    return $\cup_{k=1}^{4}$ ACTIVEFIBERS(z, I_k, Ω_k)
  end if
end function

the rectangle $D(z)$. Since $z_{j,1}$ and $z_{j,2}$ for all j are independent and identically distributed, one can leverage the framework introduced by Jakob et al. [JHY*14] to obtain all $\boldsymbol{z}_j$ in $D(z)$ as follows.

The basic idea is to traverse an (implicit) point hierarchy in which each node corresponds to an axis-aligned bounding box. At a node corresponding to $\Omega \subset \mathbb{R}^2$ in which a set of n hair fibers are embedded, one can split Ω into four equal sized sub-domains $\Omega_1, \Omega_2, \Omega_3, \Omega_4$. According to the procedural model, the probability p_k for a fiber to be further contained in Ω_k can be evaluated for $k = 1, \ldots, 4$. Then, by sampling a multinomial distribution, the number n_k for fibers belonging to Ω_k can be obtained, which then allows this process to be recursively applied for each Ω_k and n_k until reaching a leaf with $n \leq 1$. We summarize this entire process in Algorithm 3. Notice that if we seed the random number generator used by each node consistently, the outcome of Algorithm 3 will be fixed for any given z, I, and Ω. Please refer to §5 for more details on consistently seeded random number generation. Lastly, we have $\mathcal{J}(z) = \text{ACTIVEFIBERS}(z, \{1,2,\ldots,m\}, [0,h]^2)$ where m indicates the total number of hair fibers in the current yarn.

Hybrid solution: Although the fully realization-free solution requires no pre-stored information, several steps of Algorithm 3, such as sampling multinomial distributions, are computationally expensive. Thus, we present an alternative solution providing a good balance between performance and memory consumption. We use this solution to generate all results in this paper.

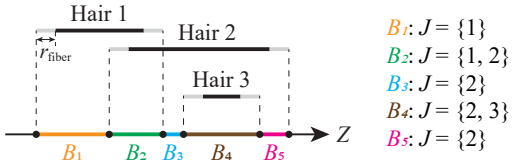


Figure 8: ***Hybrid approach:*** *discretizing the Z dimension based on the two endpoints (i.e., $z_{j,1} - r^{\text{fiber}}$ and $z_{j,2} + r^{\text{fiber}}$) of each hair-type fiber j. In this example, there are three hair-type fibers that yield a discretization of five bins $B_1, B_2, \ldots, B_5$. The set of active fibers $\mathcal{J}$ remains constant in each bin.*

In particular, we pre-generate the Z coordinates for all hair-type fibers (while leaving the other dimensions realized on demand). Then, to efficiently determine $\mathcal{J}(z)$ for a given yarn, we pre-discretize the Z dimension using $z_{j,1} - r^{\text{fiber}}$ and $z_{j,2} + r^{\text{fiber}}$ for each constituent hair-type flyaway fiber j (see Figure 8). For each bin with endpoints $z_\downarrow$ and $z_\uparrow$, we further store a list indicating active hair fibers (i.e., all j with $z_{j,1} \leq z_\downarrow$ and $z_{j,2} \geq z_\uparrow$). At runtime, we can then quickly select the bin containing z using binary search, which in turn gives $\mathcal{J}(z)$.

When $\mathcal{J}(z)$ is determined (using either the full or the hybrid approach), we then realize other parameters used in Eq. (4) for each active hair fiber using consistently seeded random number generators (§5). Lastly, we check if any of them indeed covers $\boldsymbol{p}^{\text{yarn}}$.

4.5. Discussion

The main dimension: In §4.4.2, both our solutions use dimension Z to reason about fiber activeness via Eq. (13). In principle, since hair-type flyaway fibers can also be parameterized using R and θ, each of these dimensions (or even a combination of multiple of them) can be used to define the set $\mathcal{J}$ of active fibers. We choose Z as the "main" dimension because the hair fibers are generally short and sparsely distributed in this dimension. In fact, according to our experiments, our realization-minimizing solution is even faster than the fully realized version where all hair fibers are pre-generated and looked up using a Kd-tree at run-time. This is because although our method may have to examine more candidate fibers, the fact that each discretized bin only contains a small number of active fibers makes the overhead of our approach lower than that introduced by the Kd-tree traversals.

Multi-ply yarns: Up to this point in this section, we have assumed Eq. (1), (2), and (3) describe fibers in entire yarns. In case of multi-ply yarns, however, these equations actually capture fibers in individual plies which in turn are twisted to form yarns. We chose to not describe this further complexity earlier for ease of explanation. Our technique can be easily extended to handle textiles with multi-ply yarns. Given a point $\boldsymbol{p}^{\text{yarn}} = [x, y, z]$ in the yarn space, one can further transform it to $\boldsymbol{p}_{\text{ply}}$ in the ply space by finding the cross-sectional ply center closest to $[x, y]$. The resulting $\boldsymbol{p}_{\text{ply}}$ can then be fed into the following steps of our approach for regular and flyaway lookups.

5. Implementation Details

We now describe implementation specifics that complete the description of our approach. In §5.1, we describe our formulation of yarn curves and how the normal and binormal needed by Eq. (7) are obtained. Then, §5.2 presents how to efficiently find the closest point on a set of yarn curves from any given point, which is needed by our yarn lookup step (§4.2). Lastly, we discuss in §5.3 how random numbers can be generated in a consistent way, an important ingredient for our flyaway lookups (§4.4).

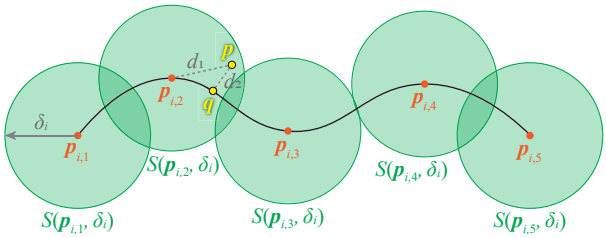


Figure 9: ***Upper bound*** *of the difference between point-to-vertex and point-to-curve distances: when the union of spheres $S(\boldsymbol{p}_{i,k}, \delta_i)$ fully covers the yarn curve $\boldsymbol{\alpha}_i$, for any $\boldsymbol{p}$, the difference between point-to-vertex distance $d_1(\boldsymbol{p})$ (i.e., the minimal distance from $\boldsymbol{p}$ to one of the vertices $\boldsymbol{p}_{i,k}$) and point-to-curve distance $d_2(\boldsymbol{p})$ (i.e., the minimal distance from $\boldsymbol{p}$ to the curve $\boldsymbol{\alpha}_i$) will never exceeds δ_i. Namely, $d_1(\boldsymbol{p}) - d_2(\boldsymbol{p}) \leq \delta_i$ for all $\boldsymbol{p}$.*

5.1. Yarn Curve Representation

Yarn curve representation: To ensure that $\boldsymbol{t}$ and $\boldsymbol{n}$ indeed provide smoothly rotating local frames, we describe the input curve $\boldsymbol{\alpha}$ using cubic Hermite splines [Kre10] with C^1 smoothness. Some previous work represents yarn and fiber curves using piecewise linear polylines [KSZ*15, ZLB16]. We found that as polylines offer only C^0 continuity, they can have artifacts in final renderings.

Rotation minimizing frames: Eq. (7) requires the normal direction $\boldsymbol{n}$ to be specified everywhere on the curve $\boldsymbol{\alpha}$ such that $\langle \boldsymbol{n}(z), \boldsymbol{t}(z) \rangle = 0$ for all t. One simple solution is setting $\boldsymbol{n}(z) = (\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{t})(z) = (\frac{\mathrm{d}^2}{\mathrm{d}t^2}\boldsymbol{\alpha})(z)$. This, unfortunately, is known to have several drawbacks: first, $(\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{t})(z)$ vanishes at locally straight (i.e., zero-curvature) regions; second, it can change rapidly (or even to its opposite) when transitioning from positively to negatively curved areas. These drawbacks can lead to unsatisfactory rendered results. We, therefore, determine $\boldsymbol{n}$ using rotation minimizing frames (RMF) [Klo86, Blo90, WJZL08]. In particular, we obtain $\boldsymbol{n}$ with the rotation-based RMF introduced by Bloomenthal [Blo90].

5.2. Point-to-Curve Projection

A key ingredient of our yarn lookup step (§4.2) is to find all relevant yarns, as well as to project the given fabric-space point $\boldsymbol{p}$ onto each of them. Since the projection is quite expensive computationally, we prune the yarn curves as much as possible beforehand by leveraging nearest-neighbor (NN) searches on all vertices (i.e., endpoints of all spline segments) of the input yarn curves. Specifically, we create a point Kd-tree storing all these vertices during preprocessing.[3] At render time, given $\boldsymbol{p}$, we query the Kd-tree for all points within a certain radius R^{query} around $\boldsymbol{p}$. The yarns corresponding to the outcome of this search are then considered relevant and used for projection computations. What should R^{query} be? We cannot simply set R^{query} to the (maximal) yarn radius because the minimal distance from $\boldsymbol{p}$ to a control point is generally greater than that to the actual splines (Figure 9). Thus, we need to enlarge the lookup radius so that we conservatively find all yarns that might contain $\boldsymbol{p}$.

For each yarn i with vertices $\boldsymbol{p}_{i,1}, \boldsymbol{p}_{i,2}, \ldots$, given $\boldsymbol{p}$, let d_1 and d_2 respectively denote the minimal distances from $\boldsymbol{p}$ to the vertices and to the yarn curve $\boldsymbol{\alpha}_i$. Namely,

$$d_1(\boldsymbol{p}) := \min_k \|\boldsymbol{p} - \boldsymbol{p}_{i,k}\|, \quad d_2(\boldsymbol{p}) := \min_t \|\boldsymbol{p} - \boldsymbol{\alpha}_i(t)\|.$$

Then, if we can estimate an upper bound δ_i of $d_1(\boldsymbol{p}) - d_2(\boldsymbol{p})$ for any $\boldsymbol{p}$, the NN search radius R^{query} can then be set to $R_i^{\text{yarn}} + \delta_i$.

In practice, to obtain δ_i, we consider growing spheres centered at all the vertices $\boldsymbol{p}_{i,1}, \boldsymbol{p}_{i,2}, \ldots$ until their union fully covers the curve $\boldsymbol{\alpha}_i$ (Figure 9). Concretely, we search for $\delta_i \in \mathbb{R}_+$ such that $\boldsymbol{\alpha}_i \subset \cup_k S(\boldsymbol{p}_{i,k}, \delta_i)$, where $S(\boldsymbol{p}_{i,k}, \delta_i)$ denotes the sphere centered at $\boldsymbol{p}_{i,k}$ with radius δ_i. This can be done by examining each spline segment k, computing its bounding box using the two endpoints $\boldsymbol{p}_{i,k}$, $\boldsymbol{p}_{i,k+1}$ and associated tangents, and ensuring that this box is contained in $S(\boldsymbol{p}_{i,k}, \delta_i) \cup S(\boldsymbol{p}_{i,k+1}, \delta_i)$.

Lastly, for each relevant yarn i, we compute the projection $\boldsymbol{q}$ of $\boldsymbol{p}$ onto its center curve $\boldsymbol{\alpha}_i$. This requires solving a piece-wise quintic equation, and we utilize an efficient implementation offered by the SplineLibrary [Mah17].

5.3. Consistently Seeded Random Numbers

A key ingredient of our flyaway lookup (§4.4) is generating random numbers in consistent ways. For instance, when handling loop-type fibers, we need to randomly select fiber migration cycles in each yarn and mutate them. To ensure the same set of cycles are always chosen for a given yarn i, one possibility is to reseed the random number generator using i (or some pre-determined function of i) before the random selection starts. We, however, noted that such reseeding can be expensive in practice. Instead, we use quasi Monte Carlo (QMC) sequences, the van der Corput sequence [KN12] in particular, indexed using the yarn and fiber indices.[4] For instance, if accessing each yarn requires N random numbers, we associate elements with indices $N(i-1)+1$, $N(i-1)+2$, $\ldots$, $N \cdot i$ in the QMC sequence to yarn i.

5.4. Rendering Our Model

We implemented our model as a specialized volume allowing the query of material density and orientation information at individual 3D locations. This volume then defines an anisotropic medium [JAM*10b] that can be rendered using standard Monte Carlo methods. We generated all results in this paper using volume path tracing where the free distance between consecutive scattering

[3] This point Kd-tree has to be rebuilt whenever the yarn curves deform, which needs to be performed per-frame for animated scenes. Fortunately, the computational overhead for building this Kd-tree is negligible compared to rendering a full image.

[4] In principle, the random numbers should be independent, which is not strictly the case for QMC sequences. Fortunately, we observed no visual artifact caused by correlated samples. Further, this correlation can be reduced by taking every n-th sample from the QMC sequence by simply multiplying all indices by n (for some constant n), which is a common practice used by Markov-Chain Monte Carlo methods to generate near-independent samples.

events are sampled using delta tracking [WMHL65]. Notice that our technique is orthogonal to the choice of the phase function. In practice, we used the micro-flake phase function [ZJMB11] for all our results.

6. Results

We demonstrate the effectiveness of our technique through experimental results. In §6.1, we evaluate our technique in comparison with alternatives. Then, §6.2 shows rendered results of procedural textiles that are very challenging for prior methods, and have typically been unrenderable in the past. Last, we discuss limitations of our technique and possible future research topics.

6.1. Evaluations

Performance: In experiments on moderate sized models, we found that our method runs at approximately the same speed[5] as a reference solution where all fiber curves are fully realized, and stored in a Kd-tree (in preprocess) for render-time volumetric lookups. Our extra computation does not make our technique slower than the reference because it exploits the mathematical structure of the procedural models, and mostly works in lower-dimensional domains (e.g., yarn cross sections). This introduces lower overhead compared to the reference which works purely in 3D.

Other reference solutions are possible, including using fully voxelized volumes and fiber meshes. However, full voxelization is impractical even for moderate sized models (without instancing). Fiber meshes are not volumetric, and thus are so different that making an apples-to-apples comparison to these methods is not easy.

Comparison to core-fiber approximation: Wu and Yuksel [WY17] have recently introduced a core-fiber approximation to enable real-time visualization of procedural textiles. This technique uses a "core" fiber, which is effectively a displacement mapped pipe, to approximate a tightly packed collection of regular fibers. In theory, this method can also be adapted for physically based rendering applications. We choose not to take this route for the following reasons.

First, this approximation has difficulties handling general fiber migrations. In their work, Wu and Yuksel used fiber migrations with repeating cycles identical to those of fiber twisting (i.e., letting $s = 1$ in Eq. (3) in Zhao et al.'s work [ZLB16] where s denotes the ratio between a migration cycle and a fiber-twisting cycle). Notice that using $s = 1$ is important for an efficient implementation of the core-fiber approximation because this causes the core to have short identical segments. Significant storage savings can then be achieved by storing only one instance of these segments. Unfortunately, as shown in Figure 10-a, the value of s affects the overall shape of a yarn greatly. When handling models with migration cycles incoherent with fiber twisting, much longer segments have to be stored, reducing the overall efficiency of the core-fiber approximation. For instance, a2 and a3 in Figure 10 requires 2–3 times of the storage for the height map when s equals 0.5 or 1 (compared to $s = 1$). For more general s values (e.g., 1.23), it can be difficult to find a segment on the core that contains integer numbers of cycles for both migration and twisting, leading to even higher overhead.

Further, the use of displacement mapping has difficulties capturing overhang structures resulting from strongly migrated fibers, as demonstrated in Figure 10-b. In this example (generated with $s = 1.2$), when representing 60% of the regular fibers, the core fails to accurately resemble the combined shape of these fibers, changing the appearance at both the micro and the macro scale. Although this problem can be eased by reducing the amount of fibers captured by the core, doing so will inevitably negate the performance gain provided by the core fiber.

In summary, their technique is great for real-time visualizations but largely complementary to our method. Our focus is more on high-fidelity and predictive rendering of procedural textiles.

Comparison to synthesis-based method: Structure-aware synthesis developed by Zhao et al. [ZJMB12] generates large-scale models based on a small database of example tiles. Beside the inconvenience of requiring many example tiles for variety, there are seams in the synthesized models at the yarn level. This is because adjacent tiles in these models are not guaranteed to be continuous along the boundary. Further, one example tile can appear many times in the resulting model, leading to periodic patterns. Conversely, our method does not rely on tiling and is free of these artifacts (see Figure 11).

6.2. Main Results

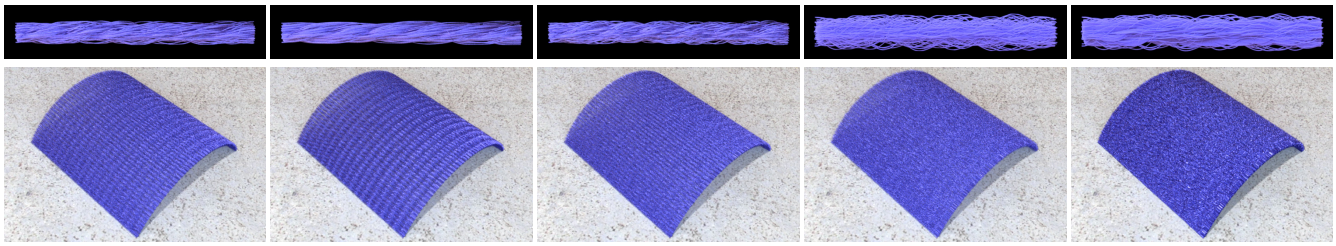
We implemented our procedural textile model as a customized volumetric data source for the Mitsuba renderer [Jak15]. We now show results rendered using our system for a few virtual scenes with large textiles that would be very difficult (if not impossible) to render if fully realized.

Woven fabrics: Figure 12 shows two woven textiles rendered with our technique at four scales. On the top, **Woven 1** is a fabric in which the colors of weft yarns vary continuously from purple to green. This model is challenging for synthesis-based approaches because the continuous change of yarn colors would require too large a set of exemplars. Our technique, on the other hand, can render this model with ease, producing a highly detailed and artifact-free result. Note that the scene has $1,920$ yarns, composed of $549,120$ fibers, and represented by $3,690,086,400$ control points.

The bottom row shows **Woven 2**, a more typical pattern with banding in the background, which would again need too many exemplars. Our result accurately captures the banding variations and remains realistic at all four scales. The fabric is composed of $2,000$ yarns, $488,000$ fibers, and $3,135,283,200$ control points.

Knitted fabrics: Physically accurate knitted fabrics with fiber-level details have been lacking because they normally contain large structures that cannot be easily scanned or synthesized. Leveraging simulated yarn curves, our technique enables physically based rendering of highly realistic knitted fabrics with fiber-level details that are crucial for preserving the fuzziness of these materials. Figure 13 contains two examples. On the top is a sweater from previous work

[5] For example, rendering a scene with 100 yarns and 29,800 fibers using the reference solution takes 15.8 seconds at one million pixel resolution and one sample per pixel, while our methods takes 15.0 seconds.



(a1) $s = 1.0$ (a2) $s = 0.5$ (a3) $s = 1.5$ (b1) Ours (exact) (b2) Core-fiber approx.

Figure 10: ***Comparison to the core-fiber approximation:*** *The core-fiber approximation [WY17] has its limitations when handling fiber migration. First, this method works most efficiently when the migration shares the same period with the twisting of fibers (i.e. $s = 1$). Models shown in (a2) and (a3), on the other hand, would incur higher overheads. Second and more importantly, this representation uses height maps that cannot accurately capture overhang structures, affecting the appearance of a fabric at both micro and macro scales (b). In this figure, all rendered yarns and fabrics contain only regular fibers (and no flyaways).*

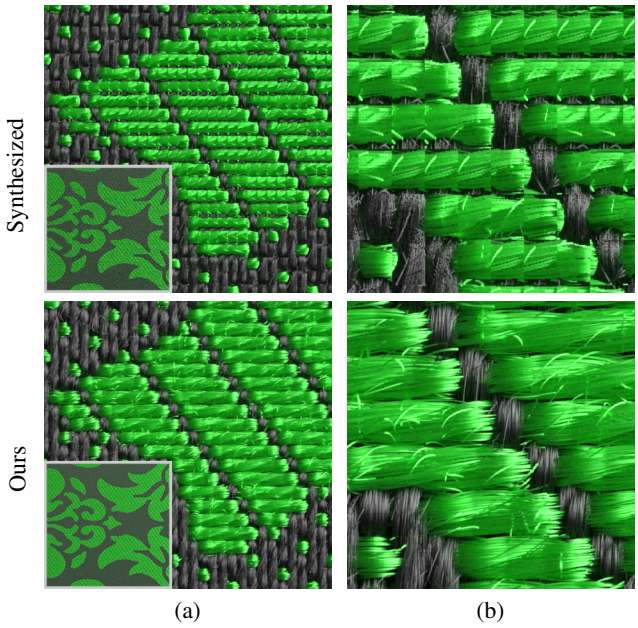


Figure 11: ***Comparison to synthesis-based method:*** *Models synthesized using existing technique [ZJMB12] suffer from seams and repeated patterns at the yarn level. Our model does not rely on instancing, and thus, is free of these artifacts at all scales.*

by Yuksel et al. [YKJM12]. A full realization of this model involves 35,624 fibers with 348,920,000 control points. Our result successfully captures the fuzziness of this material due to the abundance of flyaway fibers. The bottom of Figure 13 shows a larger knitted sample with yarn curves simulated by Kaldor et al. [KJM08]. Fully realizing this model would require storing 286 very long fibers with 1,258,400,000 control points, which takes 15 GB of storage. Our method only requires storing 1 yarn curve with 4,400,000 control points, and is able to generate all fiber-level details on the fly.

Lastly, in Figure 1, we show a modified Sponza scene from Crytek with the fabrics replaced by high-fidelity procedural models. This scene contains 12 textiles composed of 33,302 yarns, with 8,125,688 fibers, and 70,494,393,312 control points. At this scale, a full realization would take 867 GB of storage and is virtually impractical. Our technique, on the other hand, renders this scene on a single server.

Table 1 presents performance and data size results for all models in Figures 1, 12, and 13. Also, in the accompanying video, we show that our method can generate temporally consistent results and seamless, arifact-free zooms across greatly varying scales.

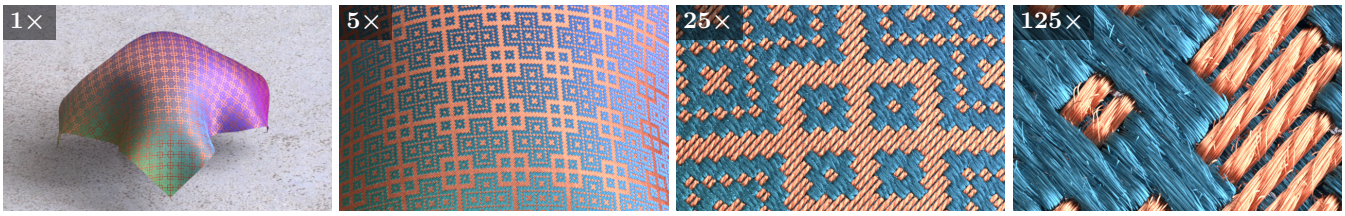
Limitations and future work: Our technique becomes expensive when handling hair-type fibers in extremely large scenes: the pure scheme is impractically slow, while even the hybrid scheme could require too much storage. In the future, improved hair-type fibers models and/or lookup algorithms are worth exploring. In addition, level-of-detail is not yet supported by our method. Given the complexity of procedural textiles, future research along this direction can be highly beneficial.

7. Conclusion

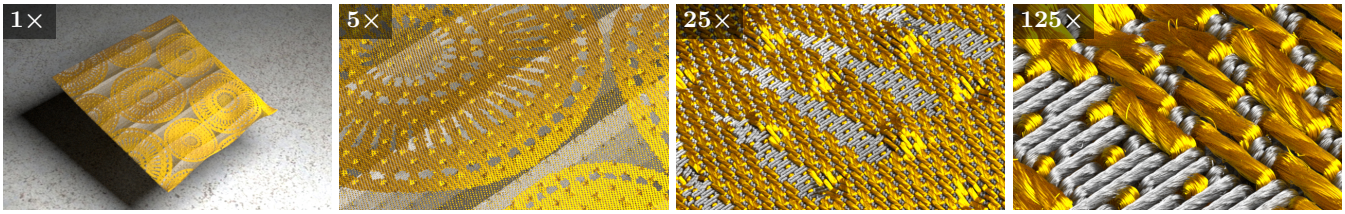
Predictively modeling and reproducing textile appearance is important for many applications in design, prototyping, and entertainment. Procedural models offer high fidelity and detail while enjoying compact representation and easy editability. However, these models have to be fully realized for physically based rendering, causing significant problems when rendering large textiles. We introduce a realization-minimizing technique to address this problem. Our method leverages new algorithms and data structures to look up both regular and flyaway fibers on the fly while storing only a fraction of full model realizations. Our method enables physically based rendering of large virtual textiles that have previously been very difficult, if not impossible. We believe these kinds of approaches are critical to enable the full power of procedural models.

Acknowledgement

The authors would like to thank their funding agencies for their generous support including NSF CHS 1513967, CHS 1617861, Google, Adobe, and AWS Cloud Credits for Research.

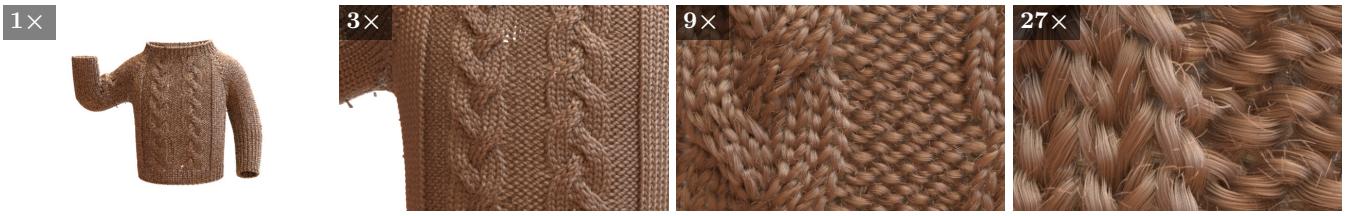


Woven 1

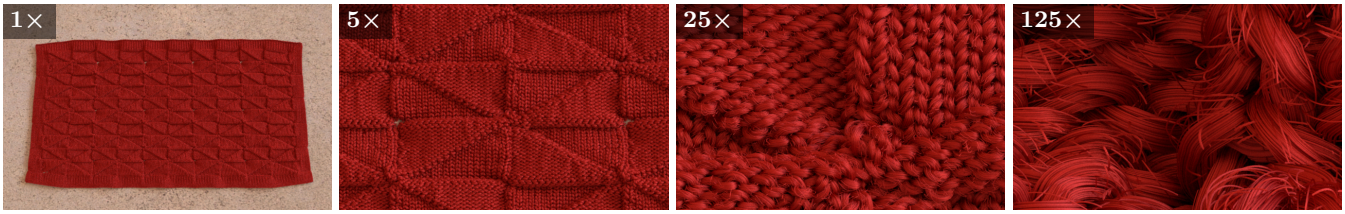


Woven 2

Figure 12: ***Rendered results (woven):*** *we created two large textiles with diverse yarn colors that are very challenging for previous methods to model. Woven 1 (top) and Woven 2 (bottom) both have approximately 2 thousand yarns, 500 thousand fibers, and over 3 billion control points. Procedural models can handle such diversity trivially. Table 1 presents data size and performance information.*



Knitted 1



Knitted 2

Figure 13: ***Rendered results (knitted):*** *fiber-level details are crucial for reproducing the fuzziness of knitted fabrics. Our method enables physically based rendering of these fabrics, which have been largely lacking since they cannot be easily scanned or synthesized. Please see Table 1 for data size and performance statistics.*

References

[Blo90] BLOOMENTHAL J.: Calculation of reference frames along a space curve. *Graphics gems 1* (1990), 567–571. 9

[Cha60] CHANDRASEKHAR S.: *Radiative transfer*. Courier Corporation, 1960. 4

[CLMMO14] CIRIO G., LOPEZ-MORENO J., MIRAUT D., OTADUY M. A.: Yarn-level simulation of woven cloth. *ACM Trans. Graph. 33*, 6 (2014), 207:1–207:11. 5

[HDCD15] HEITZ E., DUPUY J., CRASSIN C., DACHSBACHER C.: The SGGX microflake distribution. *ACM Trans. Graph. 34*, 4 (2015), 48:1–48:11. 2, 4, 13

[IM12] IRAWAN P., MARSCHNER S.: Specular reflection from woven cloth. *ACM Trans. Graph. 31*, 1 (2012), 11:1–11:20. 2, 3

[Jak15] JAKOB W.: Mitsuba physically based renderer, 2015. http://www.mitsuba-renderer.org. 10

[JAM*10a] JAKOB W., ARBREE A., MOON J. T., BALA K., MARSCHNER S.: A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph. 29*, 4 (2010), 53:1–53:13. 2, 4, 13

[JAM*10b] JAKOB W., ARBREE A., MOON J. T., BALA K., MARSCHNER S.: A radiative transfer framework for rendering materials with anisotropic structure. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 53. 9

[JHY*14] JAKOB W., HAŠAN M., YAN L.-Q., LAWRENCE J., RAMAMOORTHI R., MARSCHNER S.: Discrete stochastic microfacet models. *ACM Trans. Graph. 33*, 4 (2014), 115:1–115:10. 8

[KJM08] KALDOR J. M., JAMES D. L., MARSCHNER S.: Simulating

Table 1: ***Model size and rendering times*** *for all our results. The render times are measured by rendering a 1280 pixels wide image, at 128 samples per pixel, on an Intel server with four Xeon X7560 eight-core CPUs.*

Scene		Data size (full)				Data size (ours)					Render time
		Total	Regular	Hair	Loop	Total	(saving)	Regular	Hair	Loop	
Sponza	(Fig. 1)	866.9 GB	806.7 GB	45.2 GB	15.1 GB	18.5 GB	(**46.9**×)	3.4 GB	15.1 GB	0	7.2 h
Woven 1	(Fig. 12)	42.3 GB	42.2 GB	110.3 MB	36.8 MB	189.7 MB	(**223.2**×)	150.7 MB	39.0 MB	0	2.2 h
Woven 2		36.4 GB	35.9 GB	403.5 MB	134.5 MB	287.7 MB	(**126.7**×)	150.7 MB	137.0 MB	0	2.0 h
Knitted 1	(Fig. 13)	4.2 GB	4.0 GB	186.2 MB	62.1 MB	79.3 MB	(**53.6**×)	16.3 MB	63.0 MB	0	4.8 h
Knitted 2		14.8 GB	14.4 GB	325.1 MB	108.4 MB	190.1 MB	(**78.0**×)	50.1 MB	140.0 MB	0	8.0 h

knitted cloth at the yarn level. *ACM Trans. Graph. 27*, 3 (2008), 65:1–65:9. 5, 11

[Klo86] KLOK F.: Two moving coordinate frames for sweeping along a 3D trajectory. *Computer Aided Geometric Design 3*, 3 (1986), 217–229. 9

[KN12] KUIPERS L., NIEDERREITER H.: *Uniform distribution of sequences*. Courier Corporation, 2012. 9

[Kre10] KREYSZIG E.: *Advanced engineering mathematics (10th edition)*. John Wiley & Sons, 2010. 9

[KSZ*15] KHUNGURN P., SCHROEDER D., ZHAO S., BALA K., MARSCHNER S.: Matching real fabrics with micro-appearance models. *ACM Trans. Graph. 35*, 1 (2015), 1:1–1:26. 1, 2, 3, 4, 9, 13

[Mah17] MAHLER E.: Spline Library. https://github.com/ejmahler/SplineLibrary, 2017. 9

[MPG*16] MÜLLER T., PAPAS M., GROSS M., JAROSZ W., NOVÁK J.: Efficient rendering of heterogeneous polydisperse granular media. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) 35*, 6 (Dec. 2016). doi:10.1145/2980179.2982429. 3

[MPH*15] MENG J., PAPAS M., HABEL R., DACHSBACHER C., MARSCHNER S., GROSS M., JAROSZ W.: Multi-scale modeling and rendering of granular materials. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 34*, 4 (July 2015). doi:10.1145/2766949. 3

[SBDDJ13] SADEGHI I., BISKER O., DE DEKEN J., JENSEN H. W.: A practical microcylinder appearance model for cloth rendering. *ACM Trans. Graph. 32*, 2 (2013), 14:1–14:12. 2, 3

[SZK15] SCHRÖDER K., ZINKE A., KLEIN R.: Image-based reverse engineering and visual prototyping of woven cloth. *IEEE Trans. on Vis. and Comp. Graphics 21*, 2 (2015), 188–200. 2, 3, 4, 6

[WJZL08] WANG W., JÜTTLER B., ZHENG D., LIU Y.: Computation of rotation minimizing frames. *ACM Trans. Graph. 27*, 1 (2008), 2:1–2:18. 9

[WMHL65] WOODCOCK E., MURPHY T., HEMMINGS P., LONGWORTH S.: Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Proc. Conf. Applications of Computing Methods to Reactor Problems* (1965), vol. 557, p. 2. 10

[WY17] WU K., YUKSEL C.: Real-time fiber-level cloth rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D 2017)* (2017), ACM. 2, 10, 11

[YKJM12] YUKSEL C., KALDOR J. M., JAMES D. L., MARSCHNER S.: Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Trans. Graph. 31*, 4 (2012), 37:1–37:12. 11

[ZHRB13] ZHAO S., HAŠAN M., RAMAMOORTHI R., BALA K.: Modular flux transfer: Efficient rendering of high-resolution volumes with repeated structures. *ACM Trans. Graph. 32*, 4 (2013), 131:1–131:12. 2

[ZJMB11] ZHAO S., JAKOB W., MARSCHNER S., BALA K.: Building volumetric appearance models of fabric using micro CT imaging. *ACM Trans. Graph. 30*, 4 (2011), 44:1–44:10. 1, 2, 3, 4, 10, 13

[ZJMB12] ZHAO S., JAKOB W., MARSCHNER S., BALA K.: Structure-aware synthesis for predictive woven fabric appearance. *ACM Trans. Graph. 31*, 4 (2012), 75:1–75:10. 1, 2, 4, 10, 11

[ZLB16] ZHAO S., LUAN F., BALA K.: Fitting procedural yarn models for realistic cloth rendering. *ACM Trans. Graph. 35*, 4 (2016), 51:1–51:11. 2, 3, 4, 6, 9, 10

Appendix A: Fiber Scattering Profiles

Previously, a few models have been proposed to describe light scattering within cloth fibers. We briefly recap these models; please refer to the previous work [JAM*10a, ZJMB11, HDCD15, KSZ*15] for more details.

Volumetric: Textiles with fiber-level details can be represented as anisotropic participating media. Under this framework, light transport within a medium is governed by three parameters: the extinction coefficient σ_t, single-scattering albedo α, and the phase function f_p. In case of volumetric cloth, these parameters are further specified using the micro-flake model. Given any point $\boldsymbol{p}$ inside the cloth volume, parameters including local material density $\text{den}(\boldsymbol{p})$, direction $\text{dir}(\boldsymbol{p})$, fiber roughness $\sigma(\boldsymbol{p})$ and albedo $\alpha(\boldsymbol{p})$, are expected to be provided by the user. Then, a distribution $D(\boldsymbol{p},\boldsymbol{m})$ of flake normals $\boldsymbol{m}$ at $\boldsymbol{p}$ (analogous to normal distributions used by micro-facet models) is determined using $\text{dir}(\boldsymbol{p})$ and $\sigma(\boldsymbol{p})$. Lastly, the three radiative transfer parameters σ_t, α, and f_p are obtained using $\text{den}(\boldsymbol{p})$, $\alpha(\boldsymbol{p})$, and $D(\boldsymbol{p},\boldsymbol{m})$ [JAM*10a, HDCD15].

Fiber-based: Alternatively, fiber meshes can be used to describe textiles with rich detail. In this case, bidirectional curve scattering distribution functions (BCSDFs) are generally used to describe how light scatters off individual fibers. The BCSDFs generally involve a few modes capturing light-fiber interactions with varying numbers of reflections and transmissions.

Khungurn et al. [KSZ*15] demonstrated that both volumetric and fiber-based models are similar in quality. Since the former usually renders faster, and is more amenable to realization-minimizing approaches, we chose to use the volumetric representation in this paper.