Neural inverse procedural modeling of knitting yarns from images

Trunz, Elena; Klein, Jonathan; Müller, Jan; Bode, Lukas; Sarlette, Ralf; Weinmann, Michael; Klein, Reinhard

DOI
10.1016/j.cag.2023.12.013

Publication date
2024

Document Version
Final published version

Published in
Computers and Graphics (Pergamon)

Computers & Graphics 118 (2024) 161–172

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Neural inverse procedural modeling of knitting yarns from images ☆

Elena Trunz [a], Jonathan Klein [a,b], Jan Müller [a], Lukas Bode [a], Ralf Sarlette [a], Michael Weinmann [c,*], Reinhard Klein [a]

[a] University of Bonn, Germany
[b] KAUST, Saudi Arabia
[c] Delft University of Technology, Netherlands

ARTICLE INFO

Keywords:
Inverse procedural modeling
Model fitting
Yarn modeling
Neural networks

ABSTRACT

We investigate the capabilities of neural inverse procedural modeling to infer high-quality procedural yarn models with fiber-level details from single images of depicted yarn samples. While directly inferring all parameters of the underlying yarn model based on a single neural network may seem an intuitive choice, we show that the complexity of yarn structures in terms of twisting and migration characteristics of the involved fibers can be better encountered in terms of ensembles of networks that focus on individual characteristics. We analyze the effect of different loss functions including a parameter loss to penalize the deviation of inferred parameters to ground truth annotations, a reconstruction loss to enforce similar statistics of the image generated for the estimated parameters in comparison to training images as well as an additional regularization term to explicitly penalize deviations between latent codes of synthetic images and the average latent code of real images in the encoder's latent space. We demonstrate that the combination of a carefully designed parametric, procedural yarn model with respective network ensembles as well as loss functions even allows robust parameter inference when solely trained on synthetic data. Since our approach relies on the availability of a yarn database with parameter annotations and we are not aware of such a respectively available dataset, we additionally provide, to the best of our knowledge, the first dataset of yarn images with annotations regarding the respective yarn parameters. For this purpose, we use a novel yarn generator that improves the realism of the produced results over previous approaches.

1. Introduction

Due to their ubiquitous presence, fabrics have a great importance in domains like entertainment, advertisement, fashion and design. In the era of digitization, numerous applications rely on virtual design and modeling of fabrics and cloth. Besides the use of fabrics in games and movies, further examples include online retail with its focus on more accurately depicting the appearance of the respective clothes in images, videos or even virtual try-on solutions, as well as virtual prototyping and advertisement applications to provide previews on respective product designs.

The accurate digital reproduction of the appearance of fabrics and cloth relies on a fiber-level-based modeling to allow the accurate representation of light exchange in the fiber and yarn levels. However, due to the structural and optical complexity imposed by the arrangement of fibers with diverse characteristics within yarns and the interaction between yarns in the scope of weave and knitting patterns – where small changes in the fiber and yarn arrangement may result in significant appearance variations – as well as due to the numerous partial occlusions of the involved fibers and yarns, capturing and modeling the appearance of yarns, fabrics and cloth remains a challenge. In the context of reconstructing yarns, Zhao et al. [1] addressed the difficulty of scanning the self-occluding fiber arrangements based on computer-tomography (CT) scans to get accurate 3D reconstructions of the individual yarns. However, this imposes the need for special hardware. Instead, in this paper, we aim at the capture and modeling of the appearance of yarns by inferring individual yarn parameters from a single photograph depicting a small part of a yarn.

To address this goal, we investigate the capabilities of neural inverse procedural modeling. Whereas directly optimizing all the parameters that determine a yarn's geometry (including flyaways, i.e. fibers that migrate from the yarn, thereby contributing to the fuzziness of the yarn) with a single neural network may seem an intuitive choice, the complexity of the depictions of yarns, where twisting characteristics

☆ This article was recommended for publication by Professor M Daoudi.

* Correspondence to: Visual Computing Department, FriedrichHirzebruch-Allee 5, 53115 Bonn, Germany.
E-mail addresses: trunz@cs.uni-bonn.de (E. Trunz), kleinj@cs.uni-bonn.de (J. Klein), muellerj@informatik.uni-bonn.de (J. Müller), lbode@cs.uni-bonn.de (L. Bode), sarlette@cs.uni-bonn.de (R. Sarlette), M.Weinmann@tudelft.nl (M. Weinmann), rk@cs.uni-bonn.de (R. Klein).

https://doi.org/10.1016/j.cag.2023.12.013
Received 28 February 2023; Received in revised form 18 October 2023; Accepted 24 December 2023
Available online 27 December 2023

dominate the appearance in the yarn center and flyaway statistics dominate the appearance in the yarn's border regions, imposes that the network has the capacity to understand where which parameters can be predominantly inferred from. This observation might indicate that other strategies such as training separate networks for inferring the structural parameters for the main yarn and the characteristics of flyaways or even using an ensemble of networks, where each of these networks is only responsible for estimating a single parameter of the underlying yarn model, could be reasonable alternatives. Therefore, we investigate the potential of these approaches for the task of inverse yarn modeling from a single image. Furthermore, we investigate the effect of different loss functions including a parameter loss to penalize the deviation of inferred parameters to ground truth annotations, a reconstruction loss to enforce similar statistics of the image generated for the estimated parameters in comparison to training images as well as an additional regularization term to explicitly penalize deviations between latent codes of synthetic images and the average latent code of real images in the encoder's latent space. Thereby, we also analyze to what extent such models can be trained from solely using synthetic training data.

All of these models are trained based on synthetic training data generated using our high-quality yarn simulator that improves upon the generator by Zhao et al. [1] in terms of a more realistic modeling of hair flyaways, fiber cross-section characteristics and the orientation of the fibers' twisting axis. As our approach relies on the availability of a dataset of yarn images with respective annotations regarding characteristic yarn parameters, such as the number of plies, the twisting length, etc., we introduce – to the best of our knowledge – the first dataset of synthetic yarns with respective yarn parameter annotations. Both the dataset and the yarn generator used for the automatic generation of this dataset will be released upon acceptance of the paper. Our approach for neural inverse procedural modeling of yarns exhibits robustness to variations in appearance induced by varying capture conditions such as different exposure times as long as strong over-exposure and under-exposure are avoided during capture.

In summary, the key contributions of our work are:

- We present a novel neural inverse modeling approach that allows the inference of accurate yarn parameters including flyaways from a single image of a small part of a yarn.
- We investigate the effect of different loss formulations on the performance based on different configurations of a parameter loss to penalize deviations in the inferred parameters with respect to the ground truth, a reconstruction loss to enforce the statistics of a rendering with the estimated parameters to match the statistics of given images, and a regularization term to explicitly penalize deviations between latent codes of synthetic images and the average latent code of real images in the encoder's latent space.
- We provide, to the best of our knowledge, the first dataset of realistic synthetic yarn images with annotations regarding the respective yarn parameters.
- We present a yarn generator that supports a large range of input parameters as well as a yarn sampler that guides the selection of parameter configurations for the automatic generation of realistic yarns.

2. Related work

Respective surveys [2–8] indicate the opportunities of computational approaches for the cloth and apparel industries as well as challenges regarding the capture, modeling, representation and analysis of cloth. Some approaches approximate fabrics as 2D sheets. Wang et al. [9] and Dong et al. [10] leverage spatially varying BRDFs (SVBRDFs) based on tabulated normal distributions to represent the appearance of captured materials including embroidered silk satin, whereas others focused on appearance modeling in terms of bidirectional texture functions (BTFs) [11–13].

For scenarios with a focus on efficient simulation and editing or respective manipulation, yarn-based models [14–16] have been shown to be more amenable [17], however, at the cost of not offering the capabilities to accurately capture details of fiber-level structures and the resulting lack of realism. Drago and Chiba [18] focused on simulating the macro- and microgeometry of woven painting canvases based on procedural displacement for modeling the arrangement of the woven yarns (i.e. a spline-based representation) and surface shading. The model by Irawan and Marschner [19] also predicts yarn geometry (in terms of curved cylinders made of spiraling fibers) and yarnwise BRDF modeling to represent the appearance of different yarn segments within a weaving pattern. However, this approach does not model shadowing and masking between different threads. The latter has been addressed with the appearance model for woven cloth by Sadeghi et al. [20] that relies on extensive measurements of light scattering from individual threads, thereby taking into account for shadowing and masking between neighboring threads. However, these approaches are suitable for scenarios where cloth is viewed from a larger distance, since reproducing the appearance characteristics observable under close-up inspection would additionally require the capability to handle thick yarns or fuzzy silhouettes as well as the generalization capability to handle fabrics with strongly varying appearances. To increase the degree of realism, Guarnera et al. [21] augment the yarns extracted for woven cloth in terms of micro-cylinders with adjustments regarding yarn width and misalignments according to the statistics of real cloth in combination with the simulation of the effect of yarn fibers by adding 3D Perlin Noise [22] to the micro-cylinder derived normal map. Several approaches focused on fitting an appearance model like a BRDF [23,24] or a Bidirectional Curve Scattering Distribution Function (BCSDF) [25] to inferred micro-cylinder yarn models in order to simulate the appearance from the fibers within each ply curve extracted for a pattern without explicitly modeling each individual fiber or applying a pre-computed fiber simulation [26]. Extracting yarn paths from image data can be approached by leveraging the prior of perpendicularly running yarns for woven cloth (e.g., [27]) as well as based on the detection of knitting primitives inspired by template matching with a refinement according to an underlying knitting pattern structure [28] or deep learning based program synthesis [29]. While such approaches allow the modeling of the underlying yarn arrangements, the detailed yarn structure including characteristics such as yarn width, yarn composition, yarn twisting, hairiness, etc. is not explicitly modeled.

Following investigations on the geometric structure of fabrics in the domain of the textile research community [30–33], several works focused on a more detailed modeling of the underlying cloth micro-appearance characteristics to more accurately model the underlying cloth characteristics such as thickness and fuzziness. This includes volumetric cloth models [34–38], that describe cloth in terms of 3D volumes with spatially varying density, as well as fiber-based cloth models [27,39] that represent the detailed 3D structure of woven cloth at the yarn level with its fiber arrangement. Zhao et al. [1,36,37] leveraged a micro-computed tomography (CT) scanner to capture 3D volumetric data. Such a detailed volumetric scan allows tracing the individual fibers and, hence, provides an accurate volumetric yarn model that captures high-resolution volumetric yarn structure. For instance, Zhao et al. [1] presented an automatic yarn fitting approach that allows creating high-quality procedural yarn models of fabrics with fiber-level details by fitting procedural models to CT data that are additionally augmented by a measurement-based model of flyaway fibers. Instead of involving expensive hardware setups such as based on CT scanning, others focused on inferring yarn parameters from images, thereby representing more practical approaches for a wide range of users. Voborova et al. [40] focused on estimating yarn properties like the effective diameter, hairiness and twist based on initially fitting the yarn's main axis based on an imaging system consisting of a CCD camera, a microscope, and optical fiber lighting. Others [41,42] focused

on improving the realism of yarn appearance models by investigating optical and structural properties of real-world cloth fibers, including the effect of modeling elliptical fibers instead of fibers with circular cross-section, and presented respective scattering models. Furthermore, given a fabric's macro geometry in terms of a polygonal mesh with per-vertex UV coordinates and a 2D weave pattern, the yarn appearance model by Montazeri et al. [43] first fits yarn centerline curves and then constructs ply curves as helices around the centerline curves. These ply curves are rendered based on a BSDF model, where fiber-level details are added in a procedural approach per ply by using precomputed fiber normals and shadows. Flyaways are simulated at random locations within the ply. Subsequent work extends this approach to knitted cloth [44].

With a focus on providing accurate models at less computational costs and memory requirements than required for volumetric models, Schröder et al. [27] introduced a procedural yarn model based on several intuitive parameters as well as an image-based analysis for the structural patterns of woven cloth. The generalization of this approach to other types of cloth, such as knitwear, however, has not been provided but still needs further investigation. Saalfeld et al. [45] used gradient descent with momentum to predict some of the procedural yarn parameters used by Zhao et al. [1] from images of synthetically generated yarns. Although the results were promising for some of the parameters, the approach still could not be applied to the real yarn images. Wu et al. [46] estimated yarn-level geometry of cloth given a single micro-image taken by a consumer digital camera with a macro lens based on leveraging prior information in terms of a given yarn database for yarn layout estimation. Large-scale yarn geometry is estimated based on image shading, whereas fine-scale fiber details are obtained based on fiber tracing and generation algorithms. However, the authors mentioned that the use of a single micro-image does not suffice for the estimation of all relevant yarn parameters of complex procedural yarn models like the ones by Zhao et al. [1] or Schröder et al. [27], and, hence, the authors only considered the two parameters of fiber twisting and fiber count. Whereas our yarn generator is conceptually similar to the one by Zhao et al. [1], there is an important difference in how we model the orientation of the twisting axis of the fibers. Similar to [47], instead of using the global z-axis, we align the twisting axis with the relative z-axis of the next hierarchy level, resulting in a more realistic yarn structure. This relative implementation allows adding additional hierarchy levels, i.e. especially for the hand knitting it is common to twist different yarns if they are too thin, thus creating the next level. Our model's realism is further increased by also considering elliptic fiber cross-sections similar to previous work [41,42] due to their occurrence for natural hair fibers like wool and by considering a more natural modeling of flyaways. In addition to considering these characteristics in our model, we also present an approach to derive the respective model parameters from a single input image.

In the context of inferring physical yarn properties from visual information, Bouman et al. [48] estimated cloth density and stiffness from the video-based dynamics information of wind-blown cloth. Others focused on a neural-network-based classification of cloths according how stretching and bending stiffness influence their dynamics. Furthermore, Rasheed et al. [49] focused on the estimation of the friction coefficient between cloth and other objects. Based on the combination of neural networks with physically-based cloth simulation, Runia et al. [50] trained a network to fit the parameters used for simulation to make the simulated cloth match to the one observed in video data. Liang et al. [51] and Li et al. [52] presented approaches for cloth parameter estimation based on sheet-level differentiable cloth models. Gong et al. [53] introduced a differentiable physics model at a more fine-grained level, where yarns are modeled individually, thereby allowing to model cloth with mixed yarns. Their model leverages differentiable forces on or between yarns, including contact, friction and shear.

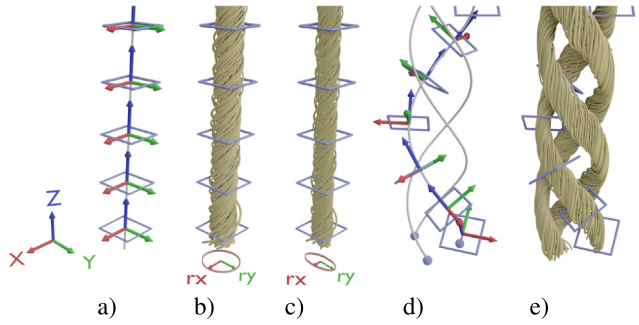

Fig. 1. Hierarchical twisting process. (a) Level 1 corresponds to a straight polygonal line. (b) Twisted fibers from the first level form a ply on the second level. (c) Before twisting plies into a yarn, the x-axis of each ply is downscaled to create an elliptical cross-section. (d) Multiple initial positions (blue) are sampled, and a helix curve with the specified properties is created at each. These curves, denoted as *center lines*, represent the paths of the different plies. (e) Deformed copies of the initial input follow each helix curve, resulting in the yarn on the third level and forming the input for the next step.

3. Generation of synthetic training data

Our learning-based approach to infer yarn parameters from images relies on the availability of a database of images of yarns with respective annotations. However, to the best of our knowledge, such a database has not been presented so far. Whereas performing exact measurements of the parameters of a real yarn is a complex and time-consuming task that requires experts as well as additional hardware such as a CT scanner, we overcome this problem by leveraging modeling and rendering tools from the field of computer graphics to create images of synthetic yarns with known parameters that can be directly used for learning applications.

To enable robust parameter inference from photographs of real yarns, the synthetic yarn images used for training the underlying neural model have to be highly realistic, i.e. they have to accurately model the yarn structure with its underlying arrangement of individual fibers. We utilize a fiber-based model rather than a volumetric one to gain more control over the generation and achieve a higher quality.

We mimic the actual manufacturing process by introducing a hierarchical approach. Multiple fibers are twisted together to form a ply, and, in turn, multiple plies are twisted together to form a yarn. If necessary, multiple thinner yarns can be twisted into a thicker yarn, denoted as *cord*, which is sometimes the case in knitwear manufacturing. We denote the yarn resulting from this hierarchical procedure as *raw yarn*. This specific terminology is widely accepted within the graphics community [1,54,55]. However, it is worth noting that within the field of textile research, the terms may have different interpretations. Here, *fibers*, *ply*, and *yarn* respectively refer to *fibers* (or *filaments*), *single yarn*, and *doubled/plied yarn* (or *folded yarn*) [56].

In addition to capturing the characteristics of the fiber arrangement of the yarn structure, we also have to consider that some of the fibers, referred to as *flyaways*, may deviate from their intended arrangement within yarns and run outside the thread. These deviations are caused by friction, aging or errors in the manufacturing process and play an essential role in the overall appearance of yarns and the fabrics made from them.

Therefore, our yarn model is controlled by a number of parameters, which belong to two types: raw yarn parameters and flyaway parameters. During generation, these parameters are stored along with the respectively resulting images, and later serve as training labels for the network training.

The raw yarn is recursively built from multiple hierarchical levels (see Fig. 1 and Algorithm 1). In the next step, flyaways are added (Algorithm 2) and detailed fiber parameters such as material and cross-section are defined. Then the yarn can be rendered accordingly. We

generate and render the synthetic yarn images using *Blender*, which offers advanced modeling capabilities that can be fully controlled by Python scripts, making it suitable for procedural modeling. Especially, the high-level mesh modifiers allow for relatively compact scripts. Additionally, it also contains a path tracer capable of rendering photo-realistic images, which allows us to build an all-in-one pipeline.

3.1. Hierarchical yarn model

During the first step, yarns, plies and fibers are represented as polygonal lines, i.e. a tuple (V, E) that stores the vertex positions $V = \{v_i \in \mathbb{R}^3 | i \in \mathbb{N}\}$ and their edges $E = \{(i,j) | i,j \in \mathbb{N}\}$. Note that our generation process allows for an arbitrary number of levels. However, in the rest of the paper, we will demonstrate the concept using the three levels given in terms of fibers, plies, and yarns. Algorithm 1 presents an overview of our recursive hierarchical generation of the raw yarn.

Algorithm 1 Recursive hierarchical generation of raw yarn

Require: level of raw yarn *level*
procedure BUILDLEVEL(*level*)
  if *level* = 0 **then**
    create straight polygonal line *line*
    return *line*
  else
    template ← BUILDLEVEL(*level* − 1)
  end if
  P ← create N instance positions using Eq. (2) or Eq. (3)
  output ← ∅
  for all $p \in P$ **do**
    I ← *copy*(*template*)
    I ← scale x coordinate of I with e for elliptical cross-section
    I ← rotate I using Eq. (4)
    center I at position p
    generate helix at position p using Eqs. (5), (6) and (7) and let I follow the helix
    output ← *output* ∪ I
  end for
  return *output*
end procedure

The input of the first level, the fiber level, is a simple straight polygonal line that has to be chosen sufficiently large to allow for the required resolution. The vertices v_i of the line are given by

$$v_i = \begin{pmatrix} 0 & 0 & i\alpha_f \end{pmatrix}^T \tag{1}$$

Here, α_f denotes the distance between two consecutive vertices of a fiber.

In each of the higher levels, we start by creating a set of N 2D instance start positions p_i. We define two variations of this procedure, one for small amounts of instances (~ 7), and one for larger numbers of instances (in practice up to 200). Both are illustrated in Fig. 2. In both cases, we add some jitter j_{xy} to the sample positions. For small numbers N of instances, we generate a regular pattern on a circle with radius r:

$$p_i = r \begin{pmatrix} \sin\theta_i \\ \cos\theta_i \end{pmatrix} + j_{xy} \begin{pmatrix} R_1 \\ R_2 \end{pmatrix}, \quad \theta_i = 2\pi \frac{i}{N} \tag{2}$$

Here and in the following, R_1 and R_2 as well as the later used R are zero-mean, unit-variance normally distributed random variables that are redrawn for each occurrence.

For larger numbers of instances, we sample the whole area of a disc. We distribute fewer samples towards the center, as instances in the middle are mostly occluded by the outer ones.

$$p_i = r_i \begin{pmatrix} \sin\theta_i \\ \cos\theta_i \end{pmatrix} + j_{xy} \begin{pmatrix} R_1 \\ R_2 \end{pmatrix}, \quad r_i = r\frac{i^{0.3}}{N^{0.3}}, \quad \theta_i = 2\pi \cdot 0.137 \cdot i \tag{3}$$

Fig. 2. Ply and fiber distribution for the process explained in Fig. 1. For each level, multiple instances of the previous level are created and placed at initial positions according to a specified distribution. We use more randomness (middle) and jitter (right) on the fiber level and more structure on the ply level (left).

The heuristically chosen constants create a slightly pseudo-random distribution that is enhanced by the added jitter.

Next, for each sample point, we copy the instance template from the previous level, and then each instance is transformed as follows: Since the sampling patterns are roughly circular, we downscale the template along the x-axis, transforming its cross-section into an ellipse (see Fig. 1, c). The rotation ensures that the smaller radius of the ellipse is oriented towards the center, which simulates the squeezing of the individual fibers for dense packing. As a last step, the template is translated to the position of the sampling point.

To simulate the twisting that occurs during the production of real yarns, we create a helix in the z-direction at each sample point $p = (p_x, p_y)^T$ and transform the template to follow it accordingly (Fig. 1). The helix is given by:

$$\theta_i = \frac{i}{H} 2\pi + \arctan2\left(p_y, p_x\right) \tag{4}$$

$$r_h = \sqrt{p_x^2 + p_y^2} \tag{5}$$

$$s_i = 1 + \max\left(0, R_h\right) \cdot \cos\left(2R_j + \frac{i}{H} 2\pi R_k\right) \tag{6}$$

$$v_i = \begin{pmatrix} r_h s_i \sin\theta_i & r_h s_i \cos\theta_i & \frac{i}{H}\alpha_h + j_z R_i \end{pmatrix}^T \tag{7}$$

Here, α_h is the height of each complete turn, denoted as the *pitch* of a helix. H is the helix resolution, i.e. the number of vertices per turn. The number of turns for the helix depends on the desired total length of the generated yarn. Since the helix is always curved around the center line, its radius r_h is determined by the position of the sample point p. The angle θ_i has an offset that ensures that the first vertex coincides with p. The random variables R_j, R_k, R_i and R_h are drawn once per vertex and once per helix, respectively. s_i is the fiber migration value, i.e. the modulation of the helix radius that varies along the vertical axis. It is realized by scaling the radius with a height-dependent cosine function with random amplitude, offset and phase speed. j_z is a zero-mean, normally distributed random variable with a standard deviation of 0.02 that represents slight jitter in the z-direction.

Note that each template point is transformed to a local coordinate frame given by the helix at the corresponding height. We do not perform an actual physical simulation for the twisting process, as this would require a complex numerical simulation and thus increase computation time drastically.

For our recursive hierarchical raw yarn generation, we used generic variable names, such as N, r and α_h. The parameters of each level for the three-level fiber–ply–yarn model, as used in the following, are summarized in Table 1.

3.2. Flyaway generation

After creating the raw yarn structure according to the previous section, we now model the flyaways. Flyaways are fibers that got displaced from their original position within the yarn. Following previous work [1,27], we distinguish between two different categories of flyaways. *Hair flyaways* are fibers where one side is completely outside the yarn, whereas *loop flyaways* are fibers where both ends of the fiber

Table 1
Parameters of our procedural Blender yarn model that are to be predicted from images. Top: Fiber parameters, Middle: Ply parameters, Bottom: Flyaway parameters. Although fiber distribution and migration are technically not among the flyaway parameters, we consider them as such for our parameter prediction due to their probabilistic nature.

Parameter type	Parameter name	Explained
Fiber amount	m	Number of fibers in each ply
Fiber ellipse	t_x, t_y	Radii of fiber ellipse
Fiber twist	α	Pitch of the ply helix
Number of plies	n	Number of plies in the yarn
Ply ellipse	r_x, r_y	Radii of ply ellipse
Ply twist	α_{ply}, R_{ply}	Pitch and radius of the yarn helix
Fiber migration	R_h	Jitter of fibers in radial direction
Fiber distribution	j_{xy}	Jitter of fibers in xy plane direction
Flyaway amount	g	Number of flyaways
Loop probability	p	Probability for loop type flyaway
Hair flyaways	β, l_{hair}, s	Angle, hair length, fuzziness
Loop flyaways	l_{loop}, d_{mean}, d_{std}	Loop length, Mean and std of distance from ply center

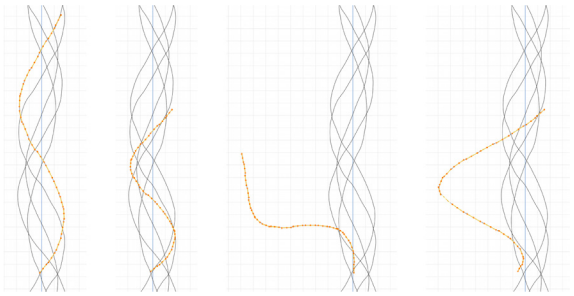

Fig. 3. Generation of flyaways. (a) A random vertex strip is selected and duplicated to become the new flyaway. (b) The flyaway is scaled along its up-axis to exaggerate details. (c) Hair flyaway: The flyaway from (b) is rotated along its lowest point. (d) Loop flyaway: The flyaway from (b), where the vertices are moved radially according to a sine function, except for the first and last vertices, which remain at their previous locations, while the middle vertex is offset the most to simulate a loop.

are still inside the yarn, but some of the in-between part is outside the main yarn. Both types of flyaways and the key steps of their creation are shown in Fig. 3.

The generation of flyaways is summarized in Algorithm 2. Flyaways are created by copying and transforming parts of the yarn. First, we determine whether the new flyaway will be a loop or a hair flyaway by drawing a uniformly distributed random number in $[0, 1]$ and determining whether it is greater or less than the loop probability p_l.

In both cases, the flyaway length is determined from a given mean and a fixed standard deviation. Note that typical means are of the same order of magnitude as the standard deviations used. To find a fiber segment for the new flyaway, a random vertex is selected and the chain of connected vertices is followed in a random direction. If this chain ends before the desired length is reached, the process is repeated with a different starting vertex (rejection sampling). Once a suitable segment is found, it is copied and transformed according to its type in the next step. Copying a segment from the original yarn, rather than creating a new vertex line, preserves the deformation from the overlapping helixes from different levels, thereby improving the degree of realism.

Loop flyaways are created by overlaying the segment with a sine wave by adding an offset to each vertex:

$$o_i = d \sin\left(\frac{i\pi}{j}\right) \begin{pmatrix} v_x & v_y & 0 \end{pmatrix}^T, \quad d = d_{mean} + d_{std} R \tag{8}$$

The sine wave moves the vertex in a radial direction, keeping its vertical coordinate untouched. j is the total number of vertices in the segment, so exactly half a period of the sine wave is used, ensuring

Algorithm 2 Flyaway generation

Require: flyaway parameters g, p_l, β, l_{hair}, s, l_{loop}, d_{mean}, d_{std}
procedure ADDFLYAWAYS(g, p_l, β, l_{hair}, s, l_{loop}, d_{mean}, d_{std})
  for $k \in [1, g]$ **do**
    $fly \leftarrow loop$ with probability p_l, or $fly \leftarrow hair$ else
    if $fly = loop$ **then**
      $length \leftarrow l_{loop} + 0.01R$ (R as explained in Section 3.1)
    else
      $length \leftarrow l_{hair} + 0.05R$
    end if
    find fiber segment S of length $length$ via rejection sampling
    if $fly = loop$ **then**
      create loop flyaway using Eq. (8) on S
    else
      scale z-coordinates of S and rotate by β to create hair flyaway (Fig. 3)
    end if
  end for
end procedure

that the first and last vertices remain at their original positions, thus creating the loop shape. The amplitude d is chosen per flyaway, not per vertex.

Hair flyaways are created by rotating the segment by the angle β (see Fig. 3). Prior to rotation, they are scaled along the vertical axis by a value of s to amplify their shape.

Once all levels and flyaways are created, the bevel parameter is set to control the thickness and ellipticity of each fiber, giving the object a proper volume. All learnable parameters for the yarn and the flyaways are summarized in Table 1.

3.3. Further implementation details

To increase the realism of the resulting yarn appearance, we apply a reflectance model to the individual fibers, which describes their view- and illumination-dependent appearance. This allow us to obtain synthetic images of yarns by placing the yarn in a pre-built scene that resembles our measurement environment in the lab where we took the photos of real yarns.

We implemented the yarn generation as a Python script inside the 3D modeling suite Blender, since it not only provides many of the operations needed during the generation, but also has powerful rendering capabilities. For the rendering of the fibers, we apply Blender's *principled hair BSDF shader* (based on Chiang et al. [57]) to represent their material. While this shader's underlying assumption of circular cross-sections slightly violates the scenario encountered for fibers with elliptical cross-sections, our experiments revealed that different types of cross-sections are still detected well in both synthetic and real test cases. Therefore we conclude that the introduced error is small enough to not strictly require a new BSDF shader for elliptical cross-sections. The scene is then rendered using the cycles path tracer, which is capable of rendering photo-realistic images with full global illumination, to generate images depicting the synthesized yarns according to the conditions we expect to occur in photographs of real yarns.

3.4. Extensions to state-of-the-art yarn generator

Whereas Zhao et al. [1] focused on woven cloth made of cotton, silk, rayon and polyester yarns, we observed that in addition to these fiber types, knitwear is often made of various types of natural wool (cashmere, virgin wool, etc.) and acrylic a as wool substitute, as they offer exceptional warming properties and knitwear is mainly worn or used in the colder months. These and most other fiber types have longer flyaways, and their fibers exhibit elliptical cross-sections rather

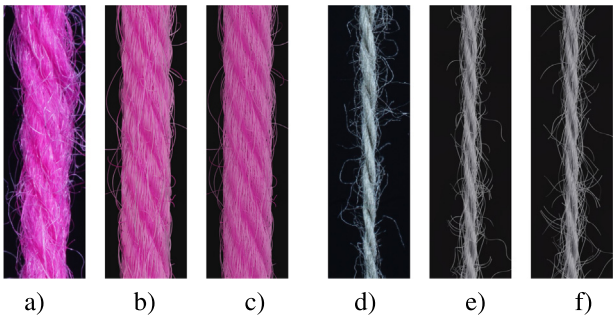

Fig. 4. Left: Comparison of fiber cross section. (a) Photograph of a wool yarn with elliptical cross section. (b) Virtual yarn generated with elliptical fiber cross-section. (c) Virtual yarn generated with circular fiber cross-section. The changes in geometry are hard to spot when zoomed out, however the shading and in particular the strength of the specular highlights is clearly affected by the cross-section shape. Right: Effect of the squeeze parameter s. (d) Reference, (e) With squeeze, (f) Without squeeze.

than circular ones, as assumed by Zhao et al. [1]. These observations inspired us to make the following extensions to the current state-of-the-art models [1,27,47]:

- Hair flyaways: Instead of implementing hair flyaways in terms of adding hair arcs [1] or using fibers of one migration period [47], we simulate them similarly to loop flyaways in terms of being pulled out of the plies. Hence, the twist characteristics are preserved (see Fig. 3, (c)). Furthermore, we leverage hair squeezing to simulate the effect that when flyaways are released from the twist, they are less stretched and contract slightly (see Fig. 3, b). These two steps make even the longer flyaways look realistic (see Fig. 4, d–f), which is extremely important for training with synthetic data, since these long flyaways highly contribute to the realism of synthetic images.
- Elliptical fiber cross-sections: We implement the ellipticity of the cross-section of fibers, which is particularly prominent in natural hair fibers such as wool. Although the geometric changes are too small to be seen directly, the shape of the cross-section affects the shading during the rendering (e.g. the prominence of specular highlights see Fig. 4, a–c), as discussed by Khungurn et al. [42].
- Local coordinate frame transformation for helix mapping: In the approach by Zhao et al. [1], plies were twisted by sliding individual vertices orthogonally to the global vertical axis. Instead, we apply a local coordinate system transformation, which leads to more plausible results similar to Wu et al.'s approach [47]. We illustrate the difference between both transformations in the supplemental material.
- Hierarchical generation: Sometimes, when multiple thinner threads are twisted into a thicker thread, yarns with more than three levels occur. Our hierarchical generator allows for any number of levels.

Evaluation of performance. Although, in its current implementation, the yarn generation process is more optimized for clarity and ease of use rather than efficiency, the time for generating all fiber and flyaway curves (about 6 to 12 s per image) is significantly less than the rendering time (about 1 to 4 min). This makes it suitable for our purpose of generating a database of yarns, but further optimization of the generation process may be an aspect for future developments.

3.5. Yarn dataset

To represent the variations in color and reflective characteristics encountered in real yarns in our synthesized yarn dataset, we sample different of these parameter configurations of the *Direct Coloring* parametrization of Blender Hair BSDF shader uniformly within available intervals and then rendering the resulting yarns in different conditions that we expect to occur when considering photos of real yarns. Furthermore, we sample different configurations of the parameters from Table 1 directly or with the help of auxiliary variables, in order to get geometric variability of plausible looking yarns. We provide details of our guided parameter sampling procedure in the supplementary material. All yarns in our database consist of two to six plies. Note that our yarn generator allows the generation of yarns with more plies, but our observations indicate that three, four and five plies are the most common scenarios in the case of knitting yarns. In the supplemental we depict some of the yarns from the database. In total, we sampled 4000 parameter configurations for the synthetic training set and 345 parameter configurations for the synthetic validation set, resulting in 4000 images with a resolution of 2000 × 600 pixels for training and 345 images with a resolution of 2000 × 600 pixels for validation. Whereas we noticed a significant performance improvement for training on 4000 examples in comparison to training on 3000 examples, a further increase of the training dataset to 5000 examples did not lead to further visual improvements. For more information, we refer to the supplemental material.

Although our yarn generator can generate many levels of hierarchy, for proof-of-concept purposes, in this paper, we focused on yarns made up of plies and did not investigate learning the next level, where multiple thinner yarns are twisted into a thicker yarn. Therefore, our database does not include such yarns. Furthermore, by rendering the yarn in different scenes, including various indoor and outdoor settings, training data for *in-the-wild* yarn parameter estimation could be generated.

4. Inference of yarn characteristics from input images

We formulate the prediction of the parameters for our procedural yarn model from single images as a regression problem. Here an encoder f maps an input image to a latent code which then becomes the input to a regression head h (Fig. 5) which performs the parameter regression. This regression path within our model is trained to minimize an L_1 loss between the prediction obtained on synthetic yarn images $x^{(i)}$ and their ground truth parameter $y^{(i)}$ used in the generation model.

$$\mathcal{L}_{\text{regress}} = \mathbb{E}\left[\left\| h(f(x^{(i)})) - \hat{y}^{(i)} \right\|_1\right] \tag{9}$$

We will refer to this network simply as *Reg*.

Although the synthetic training data was carefully generated to match the appearance of real yarns in photographs as accurately as possible, a domain gap between the synthetic and real images cannot be ruled out. To address the domain gap, we investigated the impact of adding some non-annotated real images to the training and utilized a semi-supervised training process which alternates synthetic and real images in the training process to improve the extrapolation from synthetic images with known yarn parameters to real photographs. For this purpose, we extended our aforementioned regression model into an autoencoder with an additional regression by adding a decoder model d. The autoencoder of the path is trained to minimize a simple image reconstruction loss both on synthetic and real images:

$$\mathcal{L}_{\text{recon}} = \mathbb{E}\left[\| d(f(x^{(i)})) - x^{(i)} \|_F^2\right]. \tag{10}$$

This unsupervised training process enables our encoder to be trained to map synthetic and real images into the same latent space from which the regression head predicts the yarn parameters for synthetic data points. During inference, only the encoder and regression head are required to predict inputs for the parametric yarn model. In an ideal case, the encoder maps the synthetic and real images of similar yarns to vectors that are within a close proximity in its latent space. However, such a behavior is not guaranteed by the reconstruction loss. On the contrary, the encoder might learn to distinguish between the synthetic and real images so that their latent vectors form two distinctive clusters. We propose an additional regularization term which explicitly penalizes

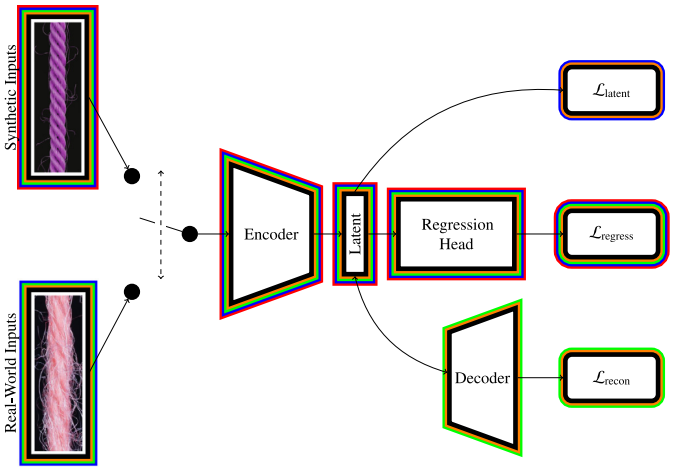

Fig. 5. The depicted architecture combines all the different networks we investigated: The networks Reg and Reg_{latent} consist of the encoder and the regression head, while the networks Reg^{ae} and Reg^{ae}_{latent} additionally include the decoder.

the distance between latent codes of synthetic images and the average latent code of real images in the encoder's latent space:

$$\mathcal{L}_{\text{latent}} = \mathbb{E}\left[\|f(x^{(i)}) - \text{sg}(\mu_{\text{SMA}})\|_F^2\right] \tag{11}$$

where sg denotes the stop gradient function (i.e. μ_{SMA} is considered to be a constant in the backward step) and μ_{SMA} is a simple moving average of latent codes of real images for the last 5 batches. With this additional regularization term the average latent codes of synthetic and real images are close to each other, and distinctive clusters become energetically less optimal However, it is more restrictive by forcing the latent code to be roughly normal distributed which is not necessary in this application.

In three different networks Reg_{latent}, Reg^{ae} and Reg^{ae}_{latent} we investigated the following three combinations of those losses:

- Network Reg_{latent}: $\mathcal{L}_{\text{reglat}} = \mathcal{L}_{\text{regress}} + \lambda_{\text{latent1}}\mathcal{L}_{\text{latent}}$.
- Network Reg^{ae}: $\mathcal{L}_{\text{regrec}} = \mathcal{L}_{\text{regress}} + \lambda_{\text{recon1}}\mathcal{L}_{\text{recon}}$.
- Network Reg^{ae}_{latent}: The combination of both previous variants, i.e.: $\mathcal{L}_{\text{combined}} = \mathcal{L}_{\text{regress}} + \lambda_{\text{recon}}\mathcal{L}_{\text{recon}} + \lambda_{\text{latent}}\mathcal{L}_{\text{latent}}$.

where $\lambda_{\text{recon}}, \lambda_{\text{recon1}}, \lambda_{\text{latent}}, \lambda_{\text{latent1}}$ are hyper-parameters of the models. The combined networks are provided in Fig. 5.

Network architecture. The encoder architecture is a pure CNN model based on ResNet [58] were the average pooling has been moved to the regression head, i.e. the latent codes are the tensors which result from the convolution stack. We explore both the ResNet18 and ResNet34 configurations with the standard ResNet block as proposed by He et al. [58] as well as the more recently proposed convnext blocks which also replaces the batch normalization with layer normalization [59]. If the encoder uses a ResNet18 or ResNet34 architecture, the regression head h is a two-layer MLP with a hidden dimension of 512 and an Exponential Linear Unit activation function after the first layer. Otherwise, the regression head is a linear projection of the 512-dimensional input onto the required output dimension.

The decoder g consists of four transposed convolutions with kernel sizes $k_1 = 2$, $k_2 = 2$, $k_3 = 2$, $k_4 = 2$, the stride $s_i = k_i/2$ and output kernel sizes $256, 128, 64, 3$. The first three layers use ReLU activations, while the last layer uses the Tanh function to ensure that the output values are within the range of the pixel values.

4.1. Inference of yarn parameters

In the following, we motivate our choice of a suitable training procedure that is capable of handling the challenging nature of the underlying problem. We considered the saliency maps [60] of the network Reg trained twice on the synthetic dataset to predict the set of the raw yarn's parameters and the set of the flyaway parameters with two independent models. An entry $m_{i,j}$ in a saliency map for a model f that has been trained on a subset of P parameters is the maximum derivative of the average value of the predicted parameters with respect to a pixel $x_{i,j,c}$ in the input image over the color channels c, i.e.

$$m_{i,j} = \max_c \left| \frac{\partial}{\partial x_{i,j,c}} \left(\frac{1}{P} \sum_p f_p(x) \right) \right|. \tag{12}$$

In contrast to saliency maps that have been proposed within the context of classification networks, we consider the derivative of the mean of the predicted parameters because we need to investigate the effect of a pixel on the entire subset on which the network was trained.

As shown in Fig. 6, the saliency maps for the neural network trained to extrapolate the raw yarn parameters show an increased sensitivity to changes in the central region of the yarn image. In contrast, the saliency maps corresponding to the network trained to predict flyaway

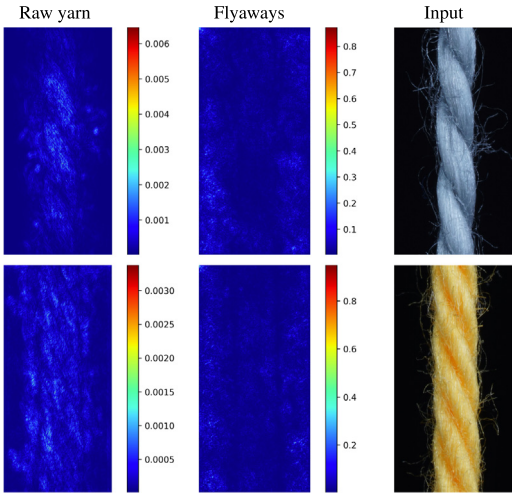

Fig. 6. Saliency maps computed for networks that were trained either to predict raw yarn or flyaway parameters of the yarn model. The color temperature in a saliency map indicates an input pixel's influence on the predicated parameter. Warmer colors correspond to a stronger influence.

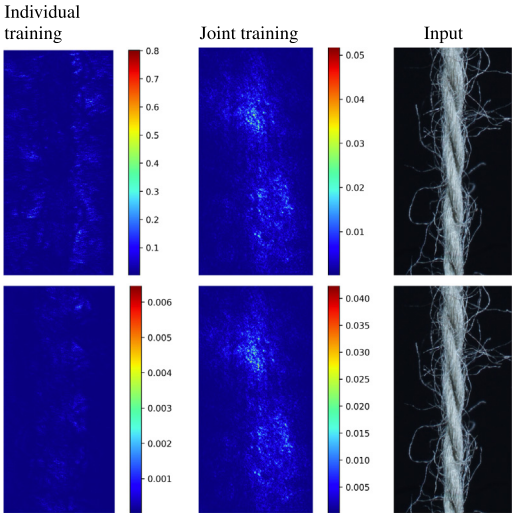

Fig. 7. Saliency maps for the yarn twist pitch α_{ply} (top row) and the yarn radius R_{ply} (bottom row), obtained from the individual training of Reg for each parameter (left) vs. the joint training of Reg (middle) .

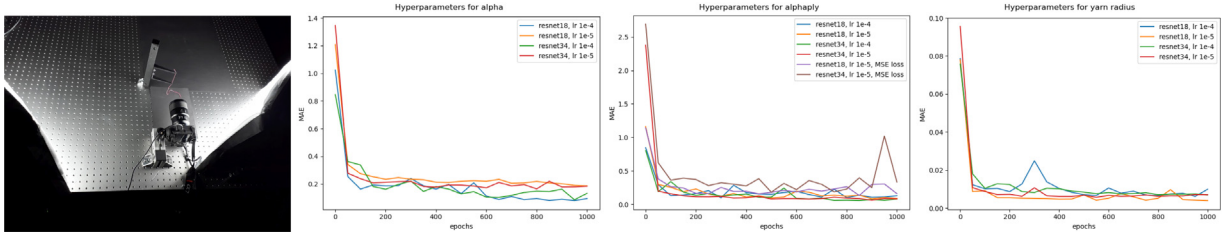

Fig. 8. (a) Our setup for capturing the test yarns. (b)-(d) Examples of validation loss comparisons for hyperparameter determination for parameters α (b), α_{ply} (c) and R_{ply} (d). Based on the loss values we chose the model of ResNet18, learning rate = $1e^{-4}$ and epoch 850 for α, ResNet34, learning rate = $1e^{-4}$ and epoch 850 for α_{ply} and ResNet18, learning rate = $1e^{-5}$ and epoch 1000 for R_{ply}.

parameters show an increased sensitivity of this network to the peripheral regions within the input images. The insights gained from these saliency maps led us to conclude that training individual models for raw yarn parameters and flyaway parameters would be a more effective approach.

Minor variations in the parameters associated with the flyaway characteristics do not have a significant effect on the resulting yarn, since achieving plausible flyaways primarily requires matching the distribution of these flyaway characteristics. On the contrary, small changes in the parameters associated with the basic yarn structure, i.e. the yarn without flyaways, already have a significant visual influence on the resulting yarn. This is clearly illustrated in Fig. 13, which shows different yarn results based on small variations in the α_{ply} parameter. Based on these observations, we conducted a comparative examination of saliency maps generated by models based on the network *Reg* that were individually trained on the synthetic dataset for each raw yarn parameter (e.g., Fig. 7 shows the maps for the yarn radius and α_{ply} parameters). Our analysis revealed significant variations in these saliency maps. Guided by the results obtained from these individual saliency maps, we then explored the potential of using a set of individual networks for the isolated prediction of each raw yarn parameter. We will denote these individual parameter networks as Reg_{params}. The architecture components of Reg_{params} correspond to the components of *Reg* in Fig. 5. A similar separation of models has already been used by Nishida et al. [61]. We compared the previously described approach of using two separate networks to predict the raw yarn parameters and the flyaway characteristics with the approach of using Reg_{params} to predict each raw yarn parameter separately along with a network to predict the flyaway characteristics.

In this context, we leveraged further priors for some of the parameters to exploit their underlying nature. For the parameter *number of plies*, we changed the last layer from the identity function as used for regression to a softmax function, thereby framing the prediction of this discrete parameter as a classification problem. The underlying motivation is that most knitting yarns have 2 to 6 plies and the estimation of the number of plies based a classification might be easier than based on a regression. Furthermore, we distinguish fibers according to their elliptic cross-section characteristics into *thin fibers* ($t_x = 0.01$, $t_y = 0.007$) and *thick fibers* ($t_x = 0.018$, $t_y = 0.01$), which we also frame as a classification problem since considering all intermediate states seems tricky and there seems to be no such significant perceptual difference for these intermediate states.

5. Experiments

Training, validation and test data. We use 4000 synthetic yarns for training and 345 synthetic yarns for validation as mentioned in Section 3.5. For training of the networks Reg_{latent}, Reg^{ae} and Reg^{ae}_{latent} we additionally used 56 real yarns. To get insights on the performance of our method for parameter inference for real yarns depicted in photographs, we tested our approach on different knitting yarns, which were not included in the training set and were made either of one type of fiber such as wool, acrylic, cotton, polyamide, etc. or of a mix of different types of fibers (Fig. 10, top row, second yarn). We took the corresponding photos of the yarns under a simple lab setup (see Fig. 8 a) with a Sony α7RIII camera using the makrolens Makro G OSS with FE 90 mm F2.8. Then we cropped the photos to the size of 600 × 2000 pixels, ensuring that the yarn roughly runs through the center of the image. These cropped photos served as an input for the parameter inference.

Fig. 9. 1st and 3rd rows: images of a real knitted cloth (made with yarns from the top row of Fig. 10) for the pattern consisting of knit (1st row) and purl (3rd row) stitches. 2nd and 4th rows: rendering of the same stitch pattern with the inferred yarn with default material settings.

Details of the training process. To improve the robustness of the trained models, we increase the variety of the training data by randomly cropping the 4000 images of a size of 2000 × 600 pixels to the network input size of 1200 × 584 pixels during each epoch. Other than cropping, we did not perform any transformation of the inputs. Then we ran the training for 1000 epochs with a batch size of 32 and a learning rate of 0.0001 based on the Adam optimizer [62]. For this purpose, we used three Nvidia Titan XP GPUs, each having 12 GB of RAM. Based on this hardware, the training for the flyaway network and the full regression network took approx. 11 h each. When training only for one parameter, the training for the ResNet34 took ca. 4 h, while for the ResNet18 it was 2.5 h.

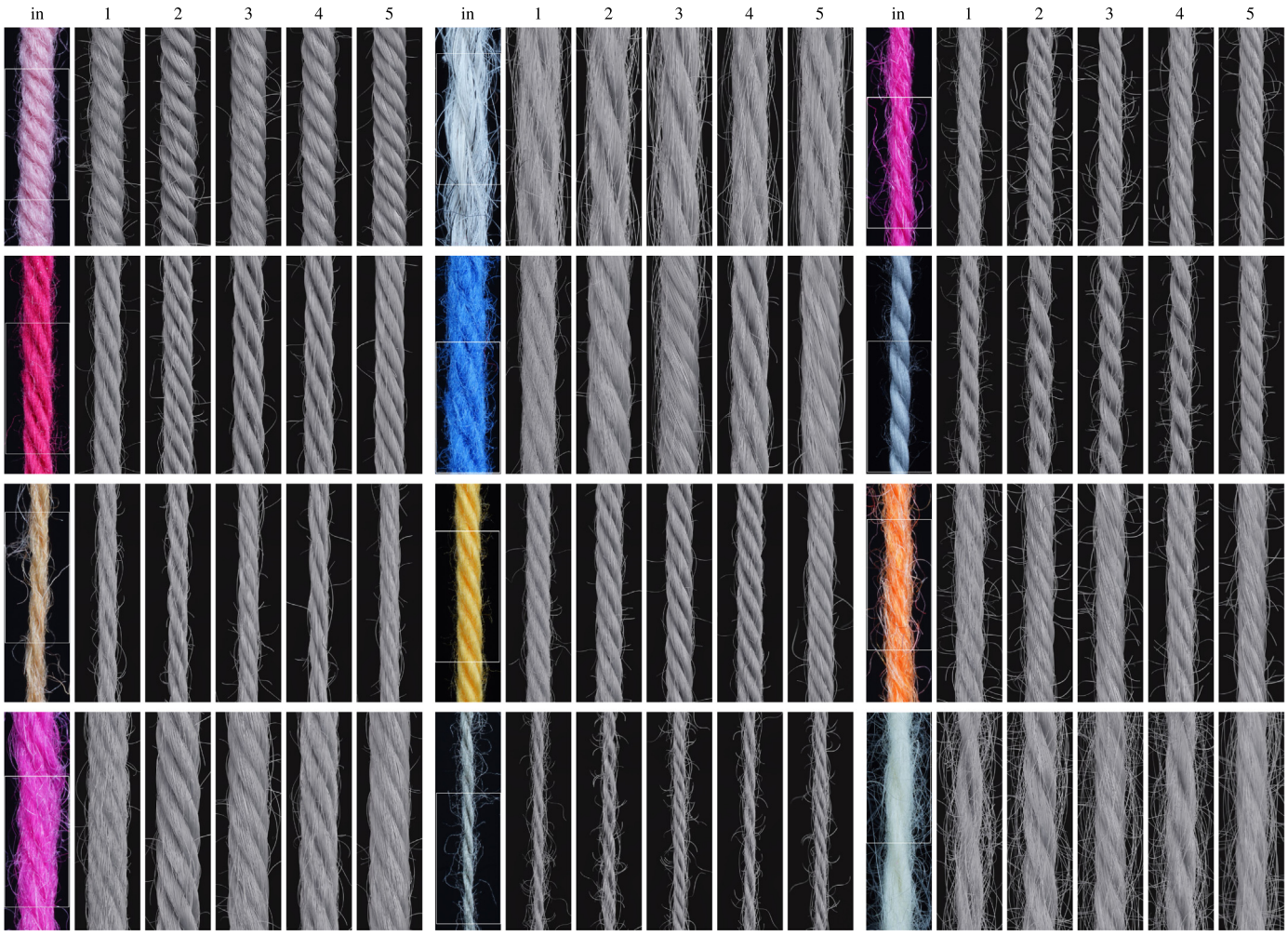

Fig. 10. in = input image, 1 = reconstruction image from Reg_{params}, 2 = Reg, 3 = Reg_{latent}, 4 = Reg^{ae}, 5 = Reg^{ae}_{latent}. The rectangle region shows the input image, which was randomly cropped from the whole image.

Table 2
Validation loss of different networks. Note that especially the important yarn twist parameters α, α_{ply} and R_{ply} are better learned with parameter specific networks Reg_{params}.

	Reg_{params}	Reg	Reg^{ae}	Reg_{latent}	Reg^{ae}_{latent}
r_x	0.0080	0.0082	0.0080	0.0097	0.0087
r_y	0.0066	0.0066	0.0074	0.0083	0.0079
m	12	12	13	14	14
α	0.0807	0.2493	0.2587	0.3230	0.3026
α_{ply}	0.0614	0.1953	0.2101	0.2400	0.2433
R_{ply}	0.00444	0.0082	0.0092	0.0092	0.0095

5.1. Parameter inference on real data

Validation of training process. First, we validated how the proposed networks perform on synthetic data. For this purpose, we compared the inference of yarn parameters on validation data through Reg_{params} (a set of different models) against Reg, Reg_{latent}, Reg^{ae} and Reg^{ae}_{latent}, each representing one model for all parameters. For this comparison, we have chosen the best hyperparameters and the best epoch based on the validation loss computed on the synthetic validation set. Fig. 8 illustrates the validation losses of Reg_{params} for the twisting parameters α, α_{ply} and yarn radius R_{ply}.

Table 2 shows the comparison of the best models of every case. We can see that the loss over each parameter is larger when training one model for all parameters of the raw yarn instead of training specific models with different hyperparameters for each parameter separately.

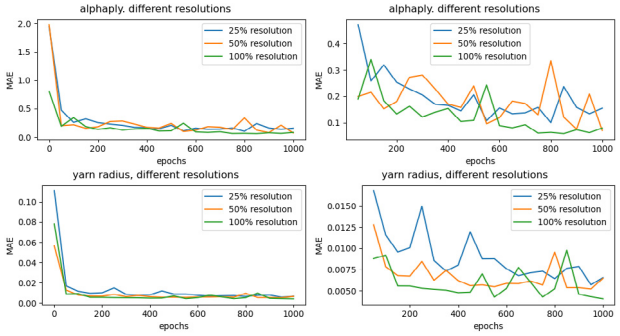

Fig. 11. Validation loss comparisons for trainings with different resolution of input images. Left: full loss curves, right: loss curves without the first element.

While this indicates a better capability to infer yarn parameters on synthetic data, we did not yet analyze the generalization to images depicting real yarns, which will follow with the experiments regarding performance analysis on real data.

Performance evaluation. We now provide an evaluation of the performance of the different approaches discussed in the previous paragraph on real yarns. For the prediction of the flyaway parameters, we use the same flyaway model for all these approaches. In our experiments, the flyaway model with the lowest validation loss was the ResNet18

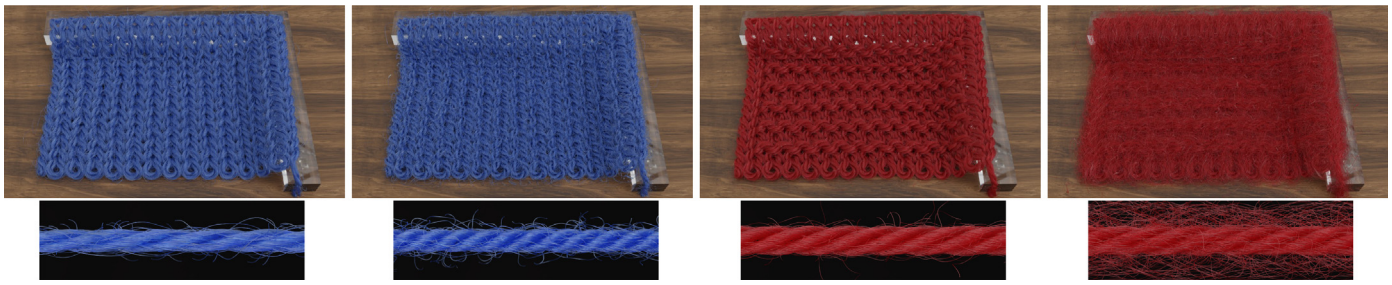

Fig. 12. Two examples of editing operations for yarns with original inferred parameters and the edited ones together with corresponding renderings of knitted patches. Reflectance parameters were not part of the inference but chosen arbitrarily for demonstration. 1st column: golden yarn from Fig. 10 in the 3rd row, left. 2nd column: the same yarn but with both pitch parameters α and α_{ply} divided by 2. 3rd column: yellow yarn from Fig. 10 in the 3rd row. 4th column: the same yarn but with parameters for flyaway amount and length multiplied by 2.

model trained with a learning rate of 0.001 and MAE loss, which we therefore use for the subsequent experiments. Exemplary results of our experiments including a comparison between the investigated approaches can be observed in Fig. 10. The corresponding inferred parameters are presented in the supplemental material. The renderings of the parameters inferred from parameter-specific networks Reg_{params} for each parameter of the raw yarn look more similar to the input image, than the renderings from the other approaches. Since we do not have the ground truth parameters for our real-world yarns, we can only compare the geometry appearance of the yarns. Based on the appearance comparison to the input image, we conclude that the approach of the parameter-specific networks is most suitable for the given task.

We tested how our inferred yarns will look in renderings of knitting samples. For this we created in Blender knitting patterns with curves similar to previous work [17] and followed the curve with the corresponding yarn that was inferred from an image of a straight yarn. In Fig. 9, we show the renderings of knitting samples, made with the three yarns of the top row from Fig. 10 with the parameters inferred from the parameter-specific networks and the flyaway network.

We observe that yarns with different geometry lead to entirely different appearances of the same pattern. Furthermore, we can see that if the inferred yarn looks similar to the yarn in the image, the pattern rendered with the inferred yarn will also look similar to the pattern knitted with the real yarn.

Once the parameters are inferred, we can use them also for editing and for the creation of new yarns. Some examples for modification of the twist parameters α and α_{ply} as well as some flyaways parameters are depicted in Fig. 12.

Ablation study regarding effect of resolution. To get insights on the effect of the resolution of the input images, we trained the networks for the different yarn parameters on images of significantly lower resolution. We experimented with the reduction to 50% and 25% of the original resolution of 1200 × 584 pixels. Fig. 11 shows the validation loss plots for the α_{ply} and yarn radius parameters for different resolutions. Fig. 13 shows some visual comparisons of reconstructed yarns with the corresponding parameters for α_{ply}. As can be observed, the achieved accuracy decreases with decreasing image resolution. We expect this to be a result of the lower quality of the depiction of the individual fiber arrangements that can be seen in terms of a blurring of the yarn structure.

In order to demonstrate the robustness of our approach to different exposure times we made exposure series of the input yarns and tested images with different exposure times. The results show that as long as the images are not too dark or over-exposed, the inferred parameters vary only insignificantly and the reconstructions are very similar.

5.2. Limitations

In addition to the dependence on the quality of the depicted yarns (as shown in the previous section), our approach depends on having the variations to be expected in the test data included in the training data. Note that our dataset includes only yarns with a normal (helix-like) fiber twisting. However, other types of fiber twisting could also occur as shown with the example in Fig. 14. The depiction shows a reconstruction that exhibits a high similarity to the input yarn. The thickness on both ply- and yarn-level as well as the number of twists closely follow the original structure. Since we did not consider this type of ply-twist in our yarn generator, there is also some deviation. We expect that such deviations might be handled by further extending the dataset regarding further types of yarn variations.

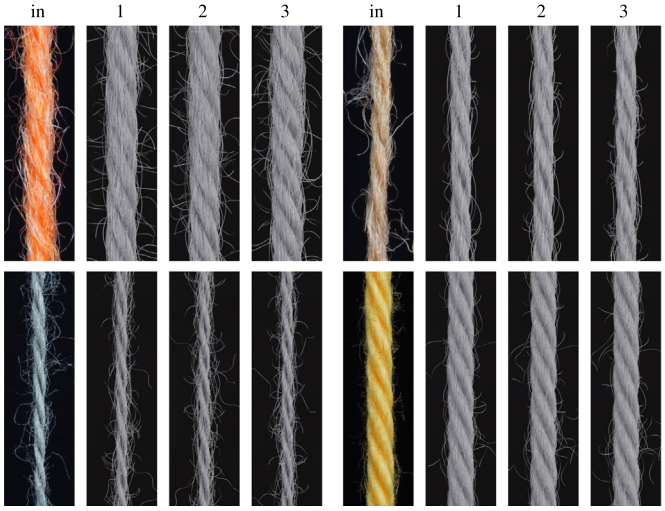

Fig. 13. in = input image, 1 = reconstruction image from Reg_{params}, where the α_{ply} parameter was trained on images with full resolution, 2 = α_{ply} parameter was trained on images with 50% of the full resolution, 3 = α_{ply} parameter was trained on images with 25% of the full resolution.

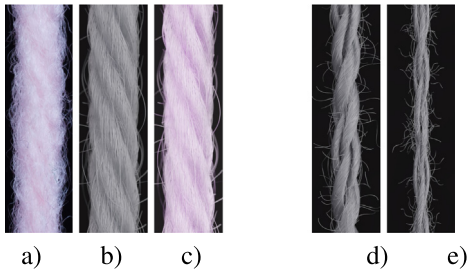

Fig. 14. (a) Input image of a yarn made by unusual (non-helical) fiber twisting procedure. (b) Rendering of a yarn with inferred parameters with default material. (c) Rendering with color. (d) and (e) Examples of yarns of fourth level, where two thinner yarns are twisted into one to make it thicker and better suitable for knitting: (d) Two yarns of the thin gray yarn from Fig. 10, fourth row, (e) Two yarns of the thick gray yarn from second row of Fig. 10.

Furthermore, despite the fact that our yarn generator also supports the fourth hierarchical level (i.e., the level where thinner yarns are twisted into thicker yarns (cords), see Fig. 14 (d)-(e)), we only included yarns consisting of the first three levels, which limits our approach to the prediction of the characteristics up to the third level. However, the extension to the fourth level is straightforward and we leave it for future work.

6. Conclusions

We presented an investigation of different neural inverse procedural modeling methods with different architectures and loss formulations to infer procedural yarn parameters from a single yarn image. The key to our approach was the accurate hierarchical parametric modeling of yarns, enhanced by handling elliptic fiber cross-sections, as occurring in many types of natural hair fibers, as well as more accurately handling flyaway characteristics and the twisting axis and the respective generation of synthetic yarns that are realistic enough so that the trained model can extrapolate to the real yarn inputs. Our experiments indicate that the complexity of yarn structures in terms of twisting and migration characteristics of the involved fibers can be better encountered in terms of ensembles of networks that focus on individual characteristics than in terms of a single neural network that estimates all parameters. In addition, we demonstrated that carefully designed parametric, procedural yarn models in combination with respective neural architectures and respective loss functions even allow robust parameter inference based on models trained from synthetic data only. In the scope of this paper, we focused solely on the geometric fiber arrangement including migration characteristics (i.e. flyaways) and left the prediction of the reflectance characteristics of knitting yarns for future work. Further developments may also consider a further hierarchical level of yarns, i.e. thinner yarns twisted to thicker yarns. Whereas we did not focus on inferring parameters for this kind of yarns, our yarn generator would be able to produce the respective characteristics and might allow enriching the dataset accordingly in future work.

CRediT authorship contribution statement

Elena Trunz: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Validation, Software, Visualization, Writing – original draft, Writing – review & editing. **Jonathan Klein:** Software, Writing – review & editing. **Jan Müller:** Validation, Writing – review & editing. **Lukas Bode:** Visualization. **Ralf Sarlette:** Investigation. **Michael Weinmann:** Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Reinhard Klein:** Conceptualization, Funding acquisition, Methodology, Resources, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Code/Data will be made available on acceptance.

Acknowledgments

The research leading to these results was partially funded by the DFG grant NFDI4Culture - Consortium for research data on material and immaterial cultural heritage and by the DFG project KL 1142/11-2 (DFG Research Unit FOR 2535 Anticipating Human Behavior).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2023.12.013.

References

[1] Zhao S, Luan F, Bala K. Fitting procedural yarn models for realistic cloth rendering. ACM Trans Graph 2016;35(4):1–11.
[2] Schröder K, Zhao S, Zinke A. Recent advances in physically-based appearance modeling of cloth. In: SIGGRAPH Asia 2012 courses. SA '12, New York, NY, USA: Association for Computing Machinery; 2012, http://dx.doi.org/10.1145/2407783.2407795.
[3] Castillo C, Aliaga C, López-Moreno J. Challenges in appearance capture and predictive modeling of textile materials. In: Proceedings of the workshop on material appearance modeling. 2017, p. 21–4.
[4] Castillo C, López-Moreno J, Aliaga C. Recent advances in fabric appearance reproduction. Comput Graph 2019;84:103–21.
[5] Amor N, Noman MT, Petru M. Classification of textile polymer composites: Recent trends and challenges. Polymers 2021;13(16):2592.
[6] Pagán EA, Salvatella MdMG, Pitarch MD, Muñoz AL, Toledo MdMM, Ruiz JM, et al. From silk to digital technologies: A gateway to new opportunities for creative industries, traditional crafts and designers. The SILKNOW case. Sustainability 2020;12(19):8279.
[7] Noor A, Saeed MA, Ullah T, Uddin Z, Ullah Khan RMW. A review of artificial intelligence applications in apparel industry. J Text Inst 2021;1–10.
[8] Mohammadi SO, Kalhor A. Smart fashion: A review of AI applications in the fashion & apparel industry. 2021, arXiv preprint arXiv:2111.00905.
[9] Wang J, Zhao S, Tong X, Snyder J, Guo B. Modeling anisotropic surface reflectance with example-based microfacet synthesis. In: ACM SIGGRAPH 2008 Papers. 2008, p. 1–9.
[10] Dong Y, Wang J, Tong X, Snyder J, Lan Y, Ben-Ezra M, et al. Manifold bootstrapping for SVBRDF capture. ACM Trans Graph 2010;29(4):1–10.
[11] Dana KJ, Nayar SK, Van Ginneken B, Koenderink JJ. Reflectance and texture of real-world surfaces authors. In: Proceedings of IEEE computer society conference on computer vision and pattern recognition. IEEE Computer Society; 1997, p. 151.
[12] Weinmann M, Gall J, Klein R. Material classification based on training data synthesized using a BTF database. In: European conference on computer vision. Springer; 2014, p. 156–71.
[13] Filip J, Kolafová M, Havlíček M, Vávra R, Haindl M, H. R. Evaluating physical and rendered material appearance. In: The visual computer (computer graphics international 2018). Springer; 2018.
[14] Kaldor JM, James DL, Marschner S. Simulating knitted cloth at the yarn level. In: ACM SIGGRAPH 2008 papers. 2008, p. 1–9.
[15] Cirio G, Lopez-Moreno J, Miraut D, Otaduy MA. Yarn-level simulation of woven cloth. ACM Trans Graph 2014;33(6):1–11.
[16] Martín-Garrido A, Miguel E, Otaduy MÁ. Toward estimation of yarn-level cloth simulation models. In: CEIG. 2018, p. 21–4.
[17] Yuksel C, Kaldor JM, James DL, Marschner S. Stitch meshes for modeling knitted clothing with yarn-level detail. ACM Trans Graph 2012;31(4):1–12.
[18] Drago F, Chiba N. Painting canvas synthesis. Vis Comput 2004;20(5):314–28.
[19] Irawan P, Marschner S. Specular reflection from woven cloth. ACM Trans Graph 2012;31(1):1–20.
[20] Sadeghi I, Bisker O, De Deken J, Jensen HW. A practical microcylinder appearance model for cloth rendering. ACM Trans Graph 2013;32(2):1–12.
[21] Guarnera GC, Hall P, Chesnais A, Glencross M. Woven fabric model creation from a single image. ACM Trans Graph 2017;36(5):1–13.
[22] Perlin K. An image synthesizer. ACM Siggraph Comput Graph 1985;19(3):287–96.
[23] Dobashi Y, Iwasaki K, Okabe M, Ijiri T, Todo H. Inverse appearance modeling of interwoven cloth. Vis Comput 2019;35(2):175–90.
[24] Jin W, Wang B, Hasan M, Guo Y, Marschner S, Yan L-Q. Woven fabric capture from a single photo. In: SIGGRAPH Asia 2022 conference papers. 2022, p. 1–8.
[25] Schroder K, Klein R, Zinke A. A volumetric approach to predictive rendering of fabrics. Comput Graph Forum 2011;30(4):1277–86.
[26] Montazeri Z, Gammelmark S, Zhao S, Jensen HW. Systems and methods to compute the appearance of woven and knitted textiles at the ply-level. 2021, Google Patents US Patent 11, 049, 291.
[27] Schröder K, Zinke A, Klein R. Image-based reverse engineering and visual prototyping of woven cloth. IEEE Trans Vis Comput Graphics 2015;21(2):188–200. http://dx.doi.org/10.1109/TVCG.2014.2339831.
[28] Trunz E, Merzbach S, Klein J, Schulze T, Weinmann M, Klein R. Inverse procedural modeling of knitwear. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019, p. 8630–9.
[29] Kaspar A, Oh T-H, Makatura L, Kellnhofer P, Matusik W. Neural inverse knitting: From images to manufacturing instructions. In: Chaudhuri K, Salakhutdinov R, editors. Proceedings of the 36th international conference on machine learning. Proceedings of machine learning research, vol. 97, Long Beach, California, USA: PMLR; 2019, p. 3272–81, URL: http://proceedings.mlr.press/v97/kaspar19a.html.

[30] Morris P, Merkin J, Rennell R. Modelling of yarn properties from fibre properties. J Texti Inst 1 Fibre Sci Text Technol 1999;90(3):322–35.
[31] Tao X. Mechanical properties of a migrating fiber. Text Res J 1996;66(12):754–62.
[32] Keefe M. Solid modeling applied to fibrous assemblies. Part I: Twisted yarns. J Text Inst 1994;85(3):338–49.
[33] Shinohara T, Takayama J-y, Ohyama S, Kobayashi A. Extraction of yarn positional information from a three-dimensional CT image of textile fabric using yarn tracing with a filament model for structure analysis. Text Res J 2010;80(7):623–30.
[34] Xu Y-Q, Chen Y, Lin S, Zhong H, Wu E, Guo B, et al. Photorealistic rendering of knitwear using the lumislice. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques. 2001, p. 391–8.
[35] Jakob W, Arbree A, Moon JT, Bala K, Marschner S. A radiative transfer framework for rendering materials with anisotropic structure. In: ACM SIGGRAPH 2010 papers. 2010, p. 1–13.
[36] Zhao S, Jakob W, Marschner S, Bala K. Building volumetric appearance models of fabric using micro CT imaging. ACM Trans Graph 2011;30(4):1–10.
[37] Zhao S, Jakob W, Marschner S, Bala K. Structure-aware synthesis for predictive woven fabric appearance. ACM Trans Graph 2012;31(4):1–10.
[38] Zhao S, Hašan M, Ramamoorthi R, Bala K. Modular flux transfer: Efficient rendering of high-resolution volumes with repeated structures. ACM Trans Graph 2013;32(4):1–12.
[39] Khungurn P, Schroeder D, Zhao S, Bala K, Marschner S. Matching real fabrics with micro-appearance models. ACM Trans Graph 2015;35(1):1.
[40] Voborova J, Garg A, Neckar B, Ibrahim S. Yarn properties measurement: An optical approach. In: 2nd international textile, clothing and design conference. 2004.
[41] Aliaga C, Castillo C, Gutierrez D, Otaduy MA, Lopez-Moreno J, Jarabo A. An appearance model for textile fibers. Comput Graph Forum 2017;36(4):35–45.
[42] Khungurn P, Marschner S. Azimuthal scattering from elliptical hair fibers. ACM Trans Graph 2017;36(2):13:1–23. http://dx.doi.org/10.1145/2998578.
[43] Montazeri Z, Gammelmark SB, Zhao S, Jensen HW. A practical ply-based appearance model of woven fabrics. ACM Trans Graph 2020;39(6):251:1–251:13.
[44] Montazeri Z, Gammelmark SB, Jensen HW, Zhao S. A practical ply-based appearance modeling for knitted fabrics. 2021, CoRR abs/2105.02475.
[45] Saalfeld A, Reibold F, Dachsbacher C. Image-based fitting of procedural yarn models. In: Klein R, Rushmeier H, editors. Workshop on material appearance modeling. The Eurographics Association; 2018, http://dx.doi.org/10.2312/mam.20181194.
[46] Wu H-y, Chen X-w, Zhang C-x, Zhou B, Zhao Q-p. Modeling yarn-level geometry from a single micro-image. Front Inf Technol Electron Eng 2019;20(9):1165–74.
[47] Wu K, Yuksel C. Real-time cloth rendering with fiber-level detail. IEEE Trans Vis Comput Graphics 2017;PP(99):1. http://dx.doi.org/10.1109/TVCG.2017.2731949.
[48] Bouman KL, Xiao B, Battaglia P, Freeman WT. Estimating the material properties of fabric from video. In: Proceedings of the IEEE international conference on computer vision. 2013, p. 1984–91.
[49] Rasheed AH, Romero V, Bertails-Descoubes F, Wuhrer S, Franco J-S, Lazarus A. Learning to measure the static friction coefficient in cloth contact. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020, p. 9912–21.
[50] Runia TF, Gavrilyuk K, Snoek CG, Smeulders AW. Cloth in the wind: A case study of physical measurement through simulation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020, p. 10498–507.
[51] Liang J, Lin M, Koltun V. Differentiable cloth simulation for inverse problems. Adv Neural Inf Process Syst 2019;32.
[52] Li Y, Du T, Wu K, Xu J, Matusik W. DiffCloth: Differentiable cloth simulation with dry frictional contact. ACM Trans Graph 2022;42(1):1–20.
[53] Gong D, Zhu Z, Bulpitt AJ, Wang H. Fine-grained differentiable physics: A yarn-level model for fabrics. 2022, arXiv preprint arXiv:2202.00504.
[54] Kaldor JM. Simulating yarn-based cloth [Ph.D. thesis], Cornell University; 2011.
[55] Zhao S. Modeling and rendering fabrics at micron-resolution [Ph.D. thesis], Cornell University; 2014.
[56] Shinohara T, Takayama J-y, Ohyama S, Kobayashi A. Extraction of yarn positional information from a three-dimensional CT image of textile fabric using yarn tracing with a filament model for structure analysis. Textle Res J - TEXT RES J 2010;80:623–30. http://dx.doi.org/10.1177/0040517509342320.
[57] Chiang MJ-Y, Bitterli B, Tappan C, Burley B. A practical and controllable hair and fur model for production path tracing. Comput Graph Forum 2016;35(2):275–83.
[58] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. 2015, CoRR abs/1512.03385, URL: http://arxiv.org/abs/1512.03385.
[59] Liu Z, Mao H, Wu C, Feichtenhofer C, Darrell T, Xie S. A ConvNet for the 2020s. 2022, CoRR abs/2201.03545, URL: https://arxiv.org/abs/2201.03545.
[60] Simonyan K, Vedaldi A, Zisserman A. Deep inside convolutional networks: Visualising image classification models and saliency maps. 2013, arXiv preprint arXiv:1312.6034.
[61] Nishida G, Bousseau A, Aliaga D. Procedural modeling of a building from a single image. Comput Graph Forum 2018;37:415–29. http://dx.doi.org/10.1111/cgf.13372.
[62] Kingma DP, Ba J. Adam: A method for stochastic optimization. In: Bengio Y, LeCun Y, editors. Proceedings of 3rd international conference on learning representations. 2015.