Mechanics-Aware Deformation of Yarn Pattern Geometry

Supplementary Document

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

Contents

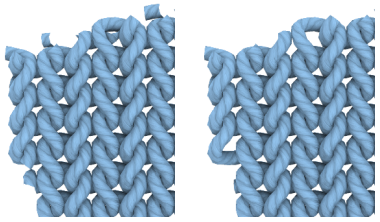

Fig. S1. Tiled yarn geometry before (left) and after (right) removing short yarn fragments.

S1 REMOVAL OF SHORT YARN FRAGMENTS

After tiling the periodic yarn pattern over the mesh as in Section 4, yarns near mesh boundaries may be very short. This may generate tiny floating yarn fragments as illustrated in Figure S1, which we want to remove.

The yarn patterns used in the optimization of Section 3 also store rest lengths $\bar{l}_i$ of edges i. During tiling, we copy these rest lengths along with the other pattern data. After tiling, we can now traverse connected yarns and accumulate these rest lengths to compute a parameter

$$a_0 = 0, \qquad a_i = \sum_{j=1}^{i} \bar{l}_{j-1}. \tag{S1}$$

Here, we use the final value of a on a yarn to remove yarns that are smaller than some user-specified minimum length. We also use a during rendering of ply twists, since it provides a parametrization that is independent of local stretching (and so stretching a yarn segment would not change the amount of twisting). Note that in general the rest lengths are different from the edge lengths in the pattern geometry corresponding to $\mathbf{I} = \mathrm{Id}$, $\mathbf{II} = \mathbf{0}$, where the yarns may locally still exhibit strains, e.g. tension balancing out collisions.

S2 RENDERING DETAILS

In this section, we provide detail on world-space mapping of edge normals, the tessellation of yarn centerlines into cylindrical geometry including yarn twists, our volume-preserving radius-scaling heuristic, and on the twistable ply- and fiber-level detail texture maps.

S2.1 Transformation of Edge Normals

Yarn twists are defined with a twist variable θ and reference directors $\underline{\boldsymbol{d}}_1$ (edge normals) per edge. In Section 4, we displace quantities from a material-space configuration $\boldsymbol{Q}$ to a deformed material-space configuration $\hat{\boldsymbol{Q}}$ and then map that to world-space values $\boldsymbol{q}$ along with the mesh. The twist variables transform trivially (simply copying values), since the edge normals take care of the actual orientation of the twisting frame. In principle, this means that we would have to first transform $\underline{\boldsymbol{d}}_1$ along with the vertex-displacements ($\hat{\boldsymbol{X}} = \boldsymbol{X} + \Delta\boldsymbol{X}$) and then along with the world-space mapping $\boldsymbol{x}$ from (9), which may be non-linear (due to Phong deformation [James 2020]). Similar to how normals transform with normal matrices, $\underline{\boldsymbol{d}}_1$ get transformed by the inverse transpose of the Jacobian $\boldsymbol{J}^{-\top}$ of the total mapping $\boldsymbol{X} \to \boldsymbol{x}$. We efficiently approximate the transformation with

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{F}, & \boldsymbol{n} \end{bmatrix}, \tag{S2}$$

where $\boldsymbol{F}$ is the underlying triangle's 3×2 deformation gradient and $\boldsymbol{n}$ is the interpolated mesh-vertex normal. We apply this transformation to the initial values of $\underline{\boldsymbol{d}}_1$ from the pattern tiling. This approximation is cheap compared to the full Jacobian including Phong Deformation and local displacements. During rendering, we simply reorthonormalize the transformed edge normals with respect to yarn centerline tangents.

S2.2 Geometry Tessellation

We use a geometry shader to tessellate cylindrical yarn meshes. The inputs are a global list of yarn vertices and index lists for each yarn. Each yarn vertex has the following data: a position $\boldsymbol{x}$, the twist θ of it's outgoing incident edge, a transformed normal $\boldsymbol{n} = \boldsymbol{J}^{-\top}\underline{\boldsymbol{d}}_1$ of the same edge using (S2), the cumulative rest length parameter a up until that vertex, as well as a copy of the original undeformed material-space coordinates for cloth-level texturing. Since the material space of the mesh is encoded as uv-coordinates and the yarn pattern is tiled over that space, this is a natural way of texturing or coloring a garment.

The vertices are passed to the geometry shader stage as primitives of four consecutive yarn vertices, from which we consistently tessellate the middle edge such that the adjacent primitives seamlessly connect, as illustrated in Figure S2. To do so, we push the edge-based quantities onto the vertices with rest-length-weighted averaging. The rest lengths can be computed from a or stored explicitly per edge. We average vertex tangents $\boldsymbol{t}_{v_i}$, vertex normals $\boldsymbol{n}_{v_i}$, and vertex twists θ_{v_i} for the inner two vertices $vi = v_1$ and $v_i = v_2$ of the four-vertex primitive. We orthogonalize the $\boldsymbol{n}_{v_i}$ with respect to $\boldsymbol{t}_{v_i}$,

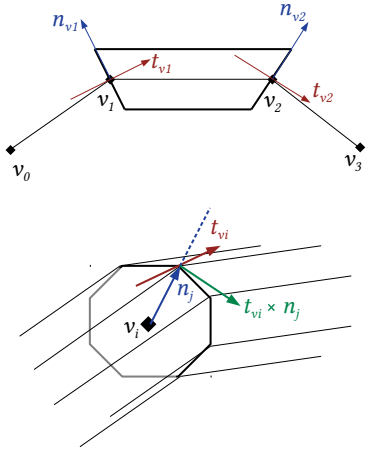

Fig. S2. Top: Four-vertex input primitive for the geometry shader. A yarn segment is tessellated for the middle edge, using averaged per-vertex tangents $\boldsymbol{t}_{v_i}$ and normals $\boldsymbol{n}_{v_i}$. Bottom: Cylinders are generated from tessellated cross-sections per centerline vertex vi, with tangential frames $\boldsymbol{M}_j = [\boldsymbol{t}_{v_i} \times \boldsymbol{n}_j, \boldsymbol{t}_{v_i}, \boldsymbol{n}_j]$.

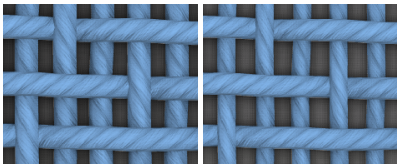

Fig. S3. Tessellated yarn geometry without (left) and with (right) volume-preserving radius scaling. In our experiments, we found the effect to be small such as shown here.

normalize both vectors and compute vertex binormals b_{v_i} as the cross-product

$$\boldsymbol{b}_{v_i} = \boldsymbol{t}_{v_i} \times \boldsymbol{n}_{v_i}. \tag{S3}$$

Volume-Preserving Heuristic. At this point, we also compute local yarn radii such that the deforming yarns preserve volume compared to their rest shape. This is a fast approximation that appears to work well as shown in Figure S3. It would be possible to instead integrate the method of Montazeri et al. [2019] for a data-driven solution to adapting yarn cross-sections to stretching, or to use a different thickness heuristic.

For each of the three edges e_i in the geometry primitive, we compute volume-preserving radii r_{e_i} from their current length l_{e_i}, their rest lengths $\bar{l}_{e_i}$, and the base radius R:

$$r_{e_i} = R\sqrt{\bar{l}_{e_i}/l_{e_i}} \tag{S4}$$

We additionally clamp the ratio of the modified vs. original radii such that their ratio to the original yarn radius is between 0.25 and 2.0, although this almost never takes effect for "reasonable" deformation within the realm of the strains sampled for our precomputed physical

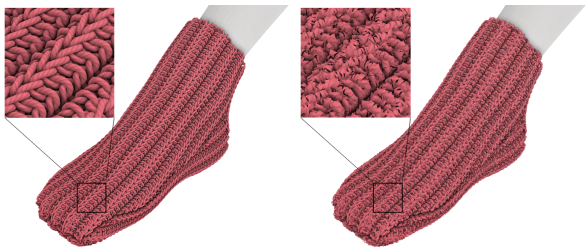

Fig. S4. Adding random displacements to tessellated yarn cylinder vertices is an efficient way of generating a fuzzy look.

yarn data. We compute vertex radii r_{v_i} using the weighted average discussed above.

Tessellation. Finally, with the rotational frames ($\boldsymbol{n}_{v_i}$ $\boldsymbol{b}_{v_i}$) and the twists θ_{v_i} at the two vertices of the middle segment, we can generate the cylindrical mesh by generating a discretized circle at each vertex and connecting them with triangular faces.

We compute the position $\boldsymbol{x}_j$ of the j-th circle vertex at primitive vertex vi as

$$s = \sin(\frac{2\pi j}{N-1} + \theta_{v_i}), \tag{S5a}$$

$$c = \sin(\frac{2\pi j}{N-1} + \theta_{v_i}), \tag{S5b}$$

$$\boldsymbol{n}_j = c\,\boldsymbol{n}_{v_i} + s\,\boldsymbol{b}_{v_i}, \tag{S5c}$$

$$\boldsymbol{x}_j = \boldsymbol{x}_{v_i} + r_{v_i}\,\boldsymbol{n}_j, \tag{S5d}$$

where N is the number of vertices per circle. $\boldsymbol{n}_j$ is the normal to the cylinder at cylinder-vertex j and different from the edge normal $\boldsymbol{n}_{v_i}$ of the centerline vertex v_i. We implement the elastic rods yarn twist here as an offset to the arguments of s and c. Note that this basically corresponds to the combined computation and tessellation of "material frames" from [Bergou et al. 2010]. It is possible to perform a naive level-of-detail optimization at this stage by reducing N depending on the depth value of vertex v_i. For normal mapping, we also emit a tangent-frame matrix $\boldsymbol{M}_j$ per vertex

$$\boldsymbol{M}_j = \left[\boldsymbol{t}_{v_i} \times \boldsymbol{n}_j, \quad \boldsymbol{t}_{v_i}, \quad \boldsymbol{n}_j\right], \tag{S6}$$

We render the tessellated geometry with "matcaps" (spherical environment maps) and screen-space ambient occlusion. In Figure S4, we briefly experimented adding fuzz detail by displacing vertices similar to [Zhong et al. 2001].

S2.3 Ply Texture Mapping

We add ply- and fiber-level detail onto the tessellated yarn geometry using *twistable* normal maps and ambient occlusion maps. The twists discussed here are separate from the discrete elastic rods twist θ, which we already incorporate into the cylinder tessellation. Instead, the twists in this section represent the helical geometry of yarn plies. We propose texture maps that can be tiled and twisted along the yarn in a parametric fashion. Notably, this twisting and tiling has an influence on the resulting normals.

The process with which we construct parametrized normals is as follows: we define ambient occlusion and normal textures for a single ply half-cylinder (Figure S5 top left), we stretch, tile and shear

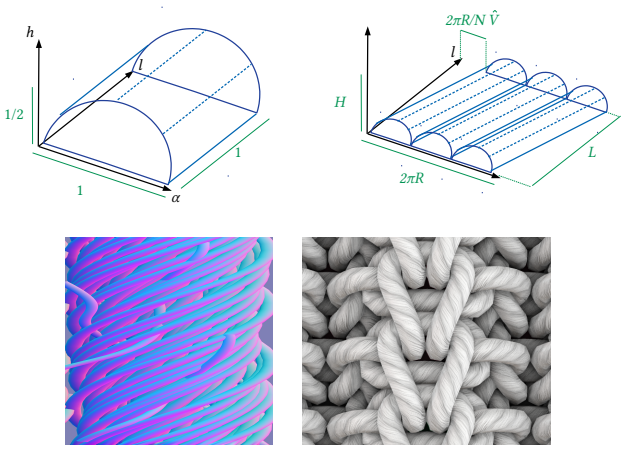

Fig. S5. Top left: The normalized single-ply space from which we bake geometry (indicated by the blue half-cylinder) into normal and ambient occlusion textures. Top right: Transformation of single-ply into stretched/tiled/twisted ply space. Bottom left: Our baked combined normal and ambient occlusion texture with fiber geometry for a single ply. Bottom right: Yarn geometry rendered with our twistable texture maps.

these textures into an intermediate space (Figure S5 top right), and lastly with "standard" normal mapping we map the normal into the tangent frame of our textured yarn geometry (Figure S2).

The projection of geometry in single-ply space can be described using a height field h

$$\boldsymbol{p}_{\text{ply}}(\hat{\alpha}, \hat{l}) = \begin{pmatrix} \hat{\alpha} \\ \hat{l} \\ h(\hat{\alpha}, \hat{l}) \end{pmatrix} \tag{S7}$$

with the "across-ply" parameter $\hat{\alpha}$ and the "along-ply" $\hat{l}$ in $[0, 1]$. This single-ply geometry is what we bake into a combined normal and occlusion texture (Figure S5 bottom left). Specifically, we bake $\left(\frac{n_x+1}{2}, \frac{n_y+1}{2}, \text{AO}\right)$ into the RGB channels, where n_x and n_y are the components of the normals to h, and AO is the ambient occlusion value.

Next we stretch, tile, and shear the single-ply geometry into an intermediate space (corresponding to a flattening of the tessellated yarn cylinder)

$$\boldsymbol{p}_{\text{int}}(\hat{\alpha}, a) = \begin{pmatrix} 2\pi R\hat{\alpha} \\ a \\ 2H\, h(N\hat{\alpha} - a/L\, \hat{V}, a/L) \end{pmatrix} \tag{S8}$$

with cumulative rest length a along the yarn, N the number of tiled plies, R the yarn radius (not the volume-preserving one), H the transformed height, L the transformed length, and $\hat{V}$ a twist factor.

We want a nice expression for the normal $\boldsymbol{n}_{\text{int}}$ of $\boldsymbol{p}_{\text{int}}$ using the baked single-ply texture values n_x and n_y. We redefine $L = \hat{L}R/N$ and $H = \hat{H}R$. Then with texture coordinates

$$\boldsymbol{w} = \begin{pmatrix} N\hat{\alpha} - a\,\hat{V}/(R\hat{L}) \\ a/(R\hat{L}) \end{pmatrix}, \tag{S9}$$

the normal of p_{int} is proportional to

$$\boldsymbol{n}_{\text{int}}(\hat{\alpha}, a) \propto \begin{pmatrix} \frac{\hat{H}N n_x}{\pi}(\boldsymbol{w}) \\ \frac{2\hat{H}N}{\hat{L}}\left(n_y(\boldsymbol{w}) - \hat{V} n_x(\boldsymbol{w})\right) \\ \sqrt{1 - n_x^2(\boldsymbol{w}) - n_y^2(\boldsymbol{w})} \end{pmatrix}. \tag{S10}$$

To apply our transformed normal maps, during tessellation we compute $\hat{\alpha} = \frac{2\pi j}{N-1}$ from (S5), and store $\boldsymbol{w}$ as well as $\boldsymbol{M}_j$ from (S6) per cylinder vertex. These quantities are interpolated in the fragment shader, where we then compute $\boldsymbol{n}_{\text{int}}$ from (S10) and map it into the tangent frame as $\boldsymbol{n}_{\text{final}} = \boldsymbol{M}\, \boldsymbol{n}_{\text{int}}$.

S3 OTHER IMPLEMENTATION DETAILS

Preloading Animation Data. We experiment with cloth mesh animations stored as obj-file sequences. For meshes of moderate size, loading meshes each frame can become a bottleneck. Therefore, we preload mesh animations into memory before adding and animating yarn detail. Additionally, we convert and serialize several mesh animations into a binary format for faster loading.

GPU Buffers. As stated in the main paper, we do all per-yarn-vertex computation on the gpu, and perform cloth-mesh computations on the cpu. With our results based on non-remeshing cloth simulations, we can precompute material-space mesh quantities (such as shape matrices and topology-dependent interpolation weights) once on the cpu without becoming a bottleneck. For more performance and remeshing cloth simulations, we suggest to implement all mesh processing on the gpu.

For yarn vertices, we mainly use two gpu buffers: one for material-space data and one for world-space data. The material-space data basically corresponds to the initial flat tiling and triangle assignment, which we do once on the cpu. We then buffer this data onto the gpu, and allocate space for the world-space buffer. Afterwards, the yarn data remains on the gpu, getting passed directly into shaders for rendering, and thus avoiding the need for expensive data-passing between the cpu and gpu.

Mesh Interpolation Weights. We use modified Shepard weights [James 2020] for strain interpolation from mesh faces to mesh vertices. We also use the same weights to compute vertex normals. To compute these weights, we use the mesh-topology in material space. Specifically, we have two sets of faces for material-space and world-space topology respectively. This way we can prohibit interpolation over uv-seams, i.e. faces that are neighbors in world space but not in material space do not count as neighbors for face-to-vertex interpolation of strains and normals.

REFERENCES

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

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

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

Hua Zhong, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. 2001. Realistic and efficient rendering of free-form knitwear. *The Journal of Visualization and Computer Animation* 12, 1 (2001), 13–22.