Real-time Fiber-Level Cloth Rendering in WebGL using React Three Fiber and GLSL

Daniel Faller

April 2025

Abstract

This project investigates the real-time rendering of fiber-level cloth structures using **React Three Fiber (R3F)**, **Three.js**, and **WebGL**. Inspired by the approach proposed by Wu and Yuksel (2017), which procedurally generates millions of fibers with Level of Detail (LoD) and core fiber optimization on modern GPUs, I adapted such methodology to the constraints of browser-based rendering.

Due to the absence of tessellation shaders and limited draw call batching in WebGL, I implemented a per-`ply` fiber instancing strategy that splits the geometry into smaller reusable components, enabling consistent visual fidelity while keeping GPU memory usage manageable. I differentiate fiber types — *migration*, *loop*, and *hair* — and simulate radius variations using GLSL vertex shaders. The `plyCenter` and `radialOffset` values are passed as per-vertex attributes, allowing dynamic fiber displacement and twist control entirely on the GPU. This emulates the core principles of procedural fiber formation in real time.

Due to the architecture chosen, my implementation sacrifices the efficiency of creating vertex on the GPU, however I achieve similiar fidelity through per-vertex displacement and runtime parameter tuning. Visual comparisons and parametric controls are provided to allow fiber appearance exploration. The application presents high interactive defining multiple render parameters that can alter the rendering in the browser with acceptable performance.

Keywords: fiber-level cloth rendering, procedural geometry, WebGL, Three.js, GLSL, real-time rendering, React Three Fiber

1 Introduction

Cloth rendering traditionally relies on simplified surface representations and texture-based shading models. However, realistic textile simulation demands consideration of the underlying fiber geometry and its influence on light scattering. The work of Wu and Yuksel (2017) demonstrated that fiber-level geometry can be generated procedurally and rendered efficiently in real-time on modern GPUs, using tessellation and specialized shading techniques.

This project reinterprets that approach within the constraints of WebGL and React Three Fiber (R3F), targeting interactive rendering of fiber-level cloth structures directly in the browser. The implementation simulates fiber properties such as twisting, radius oscillation, and type classification (migration, loop, and hair), while allowing runtime control over visual parameters. By leveraging GLSL shaders and texture-free geometry encoding, this solution balances visual fidelity with platform limitations.

2 Implementation Details

2.1 Fiber types

In the original method proposed by Wu and Yuksel (2017), all fibers are initially modeled as hair-type structures. Their system dynamically slices fiber segments into small periods, each of which can stochastically adopt hair-like behavior based on predefined noise functions or probability thresholds. This fine-grained approach enables rich local variations, particularly suited for close-up inspection of fuzz and broken ends.

In contrast, I implemented fiber type variation through controlled randomness based on the ply index. Each fiber is marked as migration, loop, or hair, affecting its radius function and endpoint behavior. The migration type oscillates smoothly between a minimum and maximum radius; the loop type uses an exaggerated maximum; the hair type has truncated ends to simulate fuzziness.

2.2 Calculating Plies positions

To calculate `aplyCenter` defines the local axis of rotation for each fiber (visualize 2 for reference), while `radialOffset` varies its displacement from the centerline. These are passed to the shaders per vertex, enabling elliptical offsets and phase shifts (represented in 3). I also included dynamic parameters for fiber count, resolution, twist frequency, and wiggle strength, controlled through a GUI panel using the Leva library.

2.3 Simulating tube geometries

A second divergence is how I handled tube shapes:For this case, an additional rendering alternative was explored through the implementation of a fiber representation based on a tube-like geometry. My method generates radials sections using a configurable number of segments (radialSegments). Each segment contributes to the illusion of a tubular strand by forming a ring of vertices at each sample point along the ply path.

Pseudo-Tessellation on the CPU

Since WebGL does not support native tessellation shaders, I implemented a pseudo-tessellation mechanism entirely on the CPU. For each fiber, I interpolate points along the ply curve using the `resolution` parameter and then manually generate the necessary vertex data to simulate curvature and displacement. This process includes computing local tangent vectors, normals, and generating a transformed vertex ring (for tubes) or displaced line (for strand-like fibers).

The resulting geometry is stored in a `BufferGeometry` and passed directly to the GPU, which leads to increased memory transfer compared to GPU-side tessellation (1). This makes the method less efficient, as geometry generation must be redone whenever parameters such as twist or resolution change, increasing CPU workload and memory bandwidth usage.

2.4 Render

Rendering is performed using custom GLSL shaders and Three.js `ShaderMaterial`. Fragment shading options include visualizing plies by index, fiber type. Although volumetric self-shadowing and physically based BRDFs were not included due to performance constraints, the visual richness achieved demonstrates the viability of this fiber-level approach in the browser.

3 Limitations and Future Improvements

While this implementation offers a compelling approximation of fiber-level cloth rendering in WebGL, there are still important components missing for a truly convincing visual result. Notably, shadow maps and Level of Detail (LoD) strategies were not implemented. Both techniques are essential for accurately portraying fabric self-occlusion and efficiently managing GPU workload across various zoom levels.

The current rendering method for line-based fibers already incorporates a resolution factor for automatic subdivision of control points. This significantly enhances the smoothness of the fiber curves and adds geometric detail with minimal manual effort. In future iterations, this approach could be extended by exploring other interpolation schemes, such as B-splines or Bézier-based smoothing, to offer more stylistic or physically-informed representations.

These potential improvements would increase both the aesthetic realism and performance scalability of the system, especially in scenes involving multiple garments or animated characters.

Appendix: Images

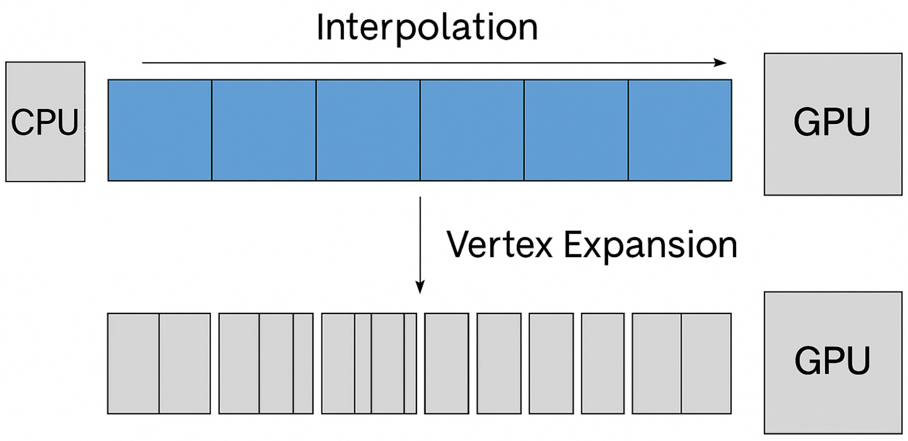

Figure 1: Memory layout of pseudo-tessellation in the CPU: each segment is interpolated and expanded into a full vertex set before being transferred to the GPU.

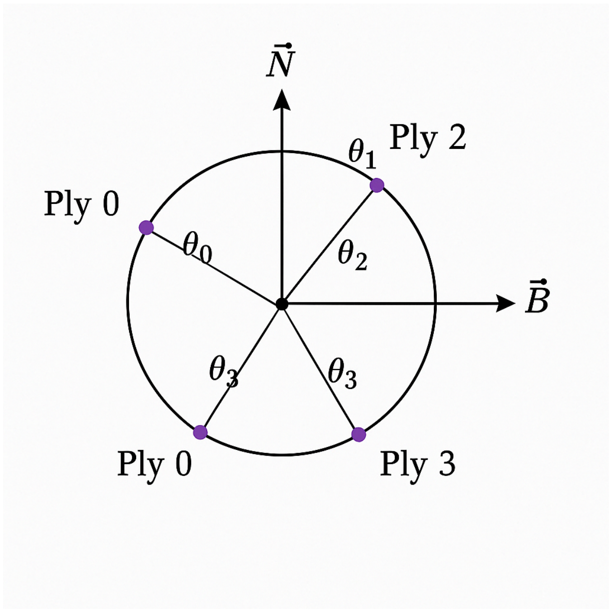

Figure 2: Visualization of `plyCenter` each ply receives a index that can be used to calculate its initial position around the curve

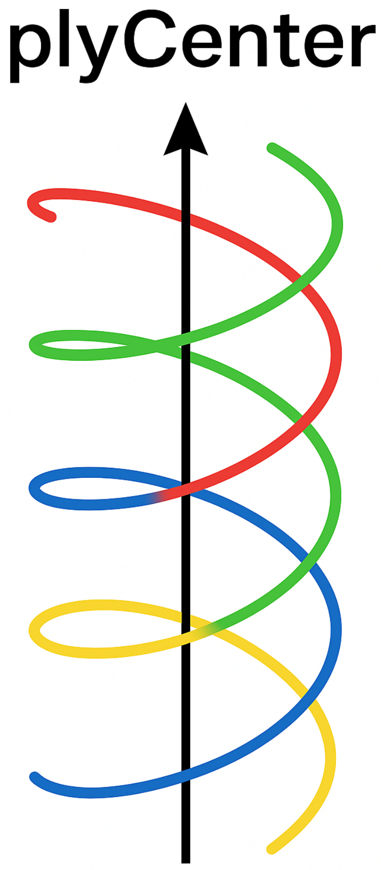

Figure 3: Example of `Plies` as the reference axis around which fiber positions are offset. Each ply defines a local coordinate frame for fiber generation.