Real-Time Fiber-Level Cloth Rendering

Paper Overview

In Literature

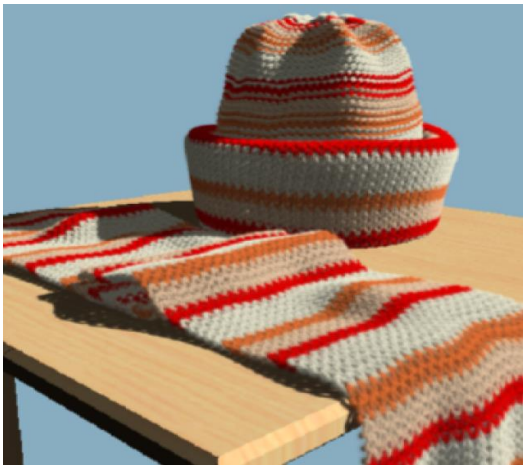
Fiber Level cloth rendering can be computationally very expensive, as one piece of cloth can contain thousands and thousands of segments to render and curves to calculate, not accounting for the unique details relative to each type of fabric, the shadows calculations, and the number of parameters we need to store.

In fact, fabric appearance has been a subject of interest for researchers long enough to have a lot of different methods to render cloth, which separate for example to **Surface-based Approaches**, where the cloth yarn model is approximated into a surface and a texture is then applied (Meseth et al. 2003, Adabala et al. 2003, Kang et al. 2010, Yuen et al. 2012) and **Volumetric approaches**, which consist of generating volume data by looking at the fiber distribution at a specific section of the yarn curve and then replicating it (Groller et al. 1995, Xu et al. 2001, Jakob et al. 2010, Zhao et al. 2013).

Both have advantages, but also present some disadvantages. Concisely these are:

- For Surface-based Approaches:
  - They can be processed in Real-Time, i.e., on the fly on the GPU.
  - They lack realistic details and realistic look of a fabric.

- For Volumetric Approaches:
  - They are a lot more realistic and depict the real surface topology of a cloth.
  - Rendered images take hours to be processed.
  - Needs very large amounts of data to render.



In the paper

The paper presents an alternative to these methodologies, while combining their advantages. The method it describes is done in real-time, handled by the GPU and the overall look of the fabric is realistic. It is not as fast as a sole surface-based method nor as detailed as a volumetric based method but is very practical to use for a quick realistic rendering of cloth for a video game for example. It renders fiber-level cloth in real-time, hence the title of the paper.

The idea is based on a procedural model of fibers depicted in the paper **Fitting procedural yarn models for realistic cloth rendering by Z HAO , S., L UAN , F., AND B ALA , K. 2016,** which makes it possible to draw a fiber curve by following mathematical formulae and doing the calculations with relation to some pre-fitted parameters (for example the angles and the radiuses of the fiber curves).
The input being the control points of the curve formed by the yarns, the yarn curve is then calculated (cubic Bezier or Catmull-Rom curves), followed by the different ply positions with reference to it, which then gives access to the computation of the fiber curves along the plies.
By changing the parameters of the equations, some type of fiber or another can be rendered (hair fiber, migration fiber or loop fiber) to further render the model more realistically.

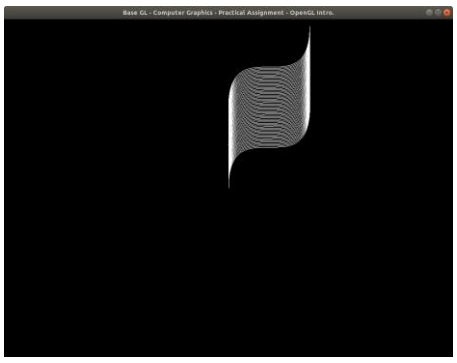
Rendering curves on the GPU makes use of a recent change in the rendering pipeline (2010) where tessellation and geometry stages became programmable. Points are tessellated from the Bezier/Catmull-Rom spline and then position adjusted through the calculations made on the ply and the fiber curves, thus permitting the real-time, on the fly rendering of fibers!

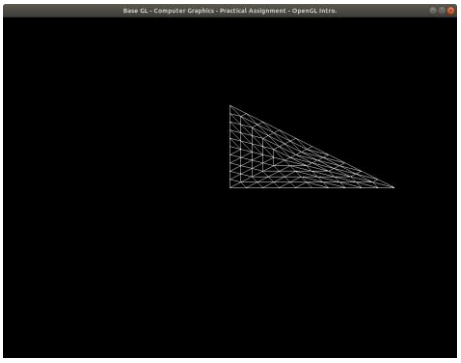
Additionally to this interesting method, the paper also includes optimization functionalities, such as the use of core fibers so that tessellated curves can be used in a better way (a limit of 64 tessllated curves is imposed in current-gen GPUs), a level-of-detail strategy to reduce the number of fibers rendered with relation to the position of the fiber in screen space and its length with relation to the screen, self-shadows calculations as well as ambient occlusion for fiber-like structures.

In conclusion, the paper builds upon an existing procedural model (2016) and modifies its use in a way it can be profited from in real-time applications and with much less parameters and computational costs. With power and capacity of GPUs growing, the methodology described in this paper can become the one used by default to render very realistic looking cloth on the fly with low computational costs.

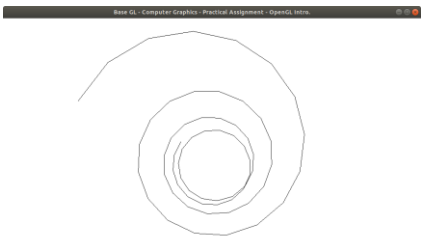
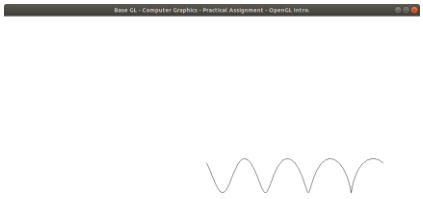
Main focus in the project

Throughout the project, I focused thoroughly on fiber generation and did not go through the optimization processes.
To be able to understand the underlying process described in the paper, one should have a great understanding of how tessellation stage shaders work.
This was the starting point for me, as I started gaining knowledge and searching for examples to try and reproduce. My first few renderings with tessellation stages were the following for example:

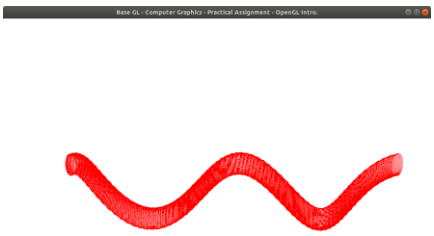
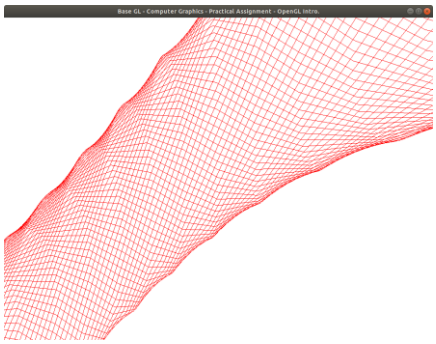
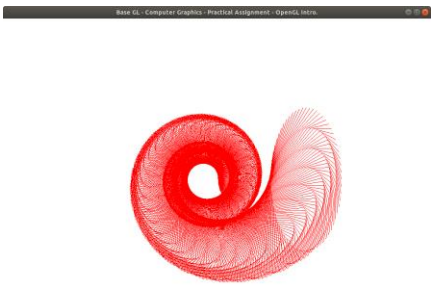




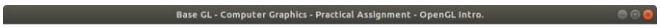
Plotting a Bezier curve and tessellating it into many curves was a big progress, and I started trying to draw some other shapes, mainly periodic functions, which play a big role in this paper, and replicate the ply curves and the fiber curves.
First drawings were simple use cases, where the yarn curve was a straight line along the x-axis, and formulas and vectors were hardcoded.

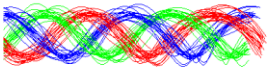
Progressively and after many iterations, I succeeded in plotting one ply curve and eventually from the ply curve to calculate the fiber curves for the ply and tessellate them.

As you can see, I now had the fiber structure of one ply, and it was not hard doing this for three plies later on. But this said, you can also conclude that the taken radiuses of the fibers in the pictures above are all the same since no fiber is drawn inside the circumference of the ply. In order to remediate to this, the method employed in the paper was the usage of a 1D structure to store the radius and the initial angle of a fiber. To avoid having to set up a big dataset, what I did instead was dividing the circumference of the ply by 21 (21 tesselated fiber for each ply) for the angles and using a random function on the CPU side (returning a random float between 0 and 1) to set up the radius of each fiber by multiplying its result by a common chosen radius to obtain quasi-random radiuses for the fibers. The results were the following:



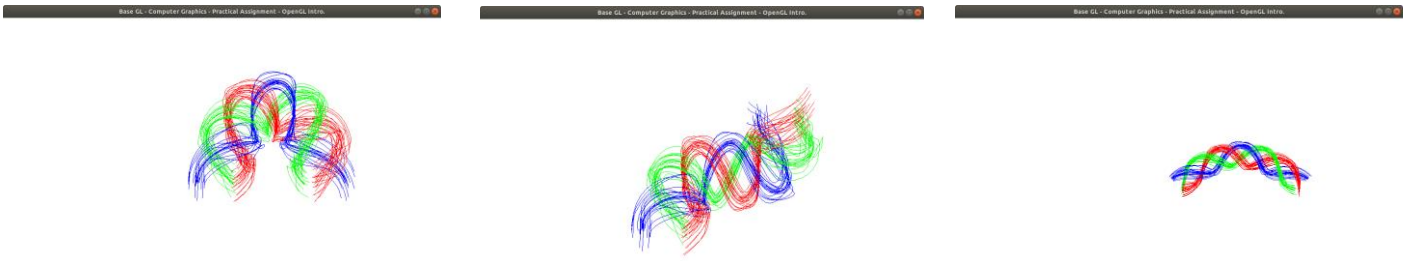




Next stage was switching to any Bezier curve, and be able to correctly render the fibers independently of its orientation. This was a difficult step since the normal vectors of the Bezier curve are much harder to calculate than in the previous examples where the yarn curve was a simple line and its normals could be hard-coded.
After thorough research, there were two methods to compute the orientation of the Bezier curve:

- Calculating the Frenet normals: this method yielded complete change of orientation for normals at inflexion points of the curve and was thus discarded.
- Implementing Rotation Minimising Frames: this method calculates reliable normals for the curves and remediates to the miscalculations induced by the first method.

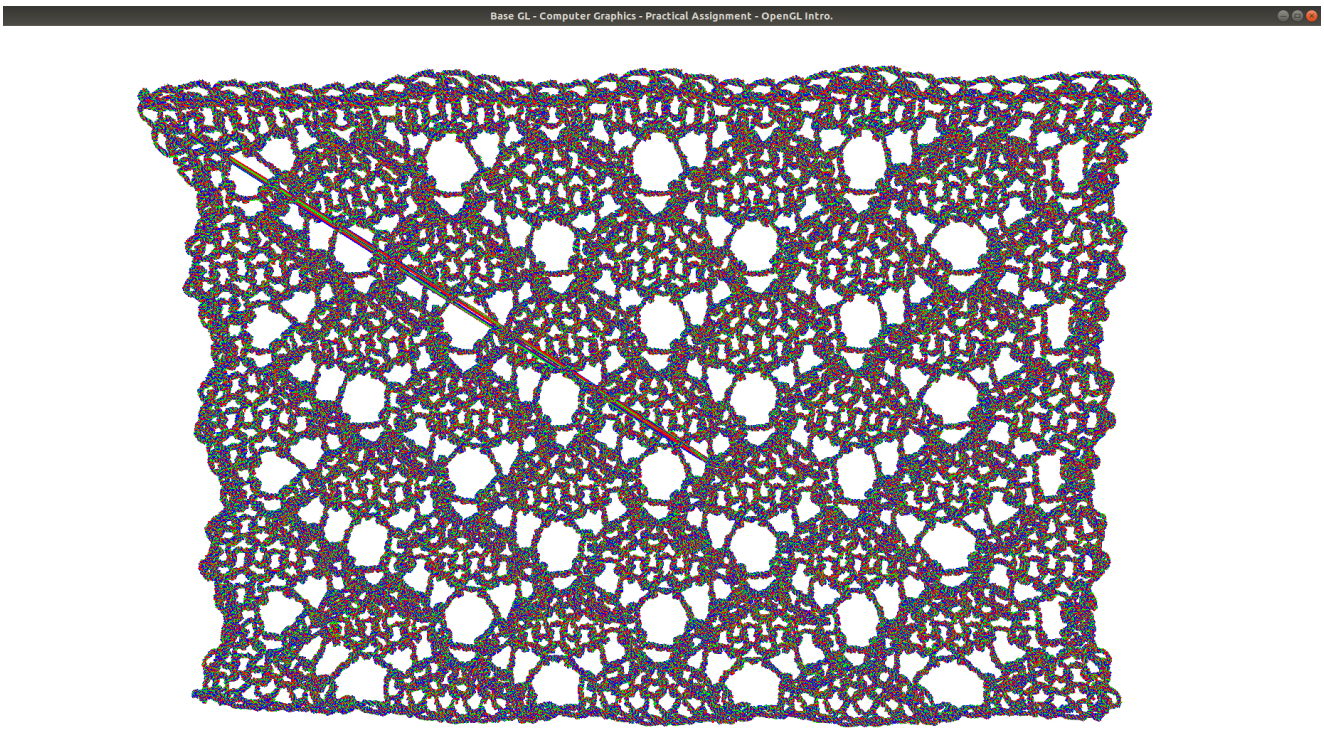
Results were the following:

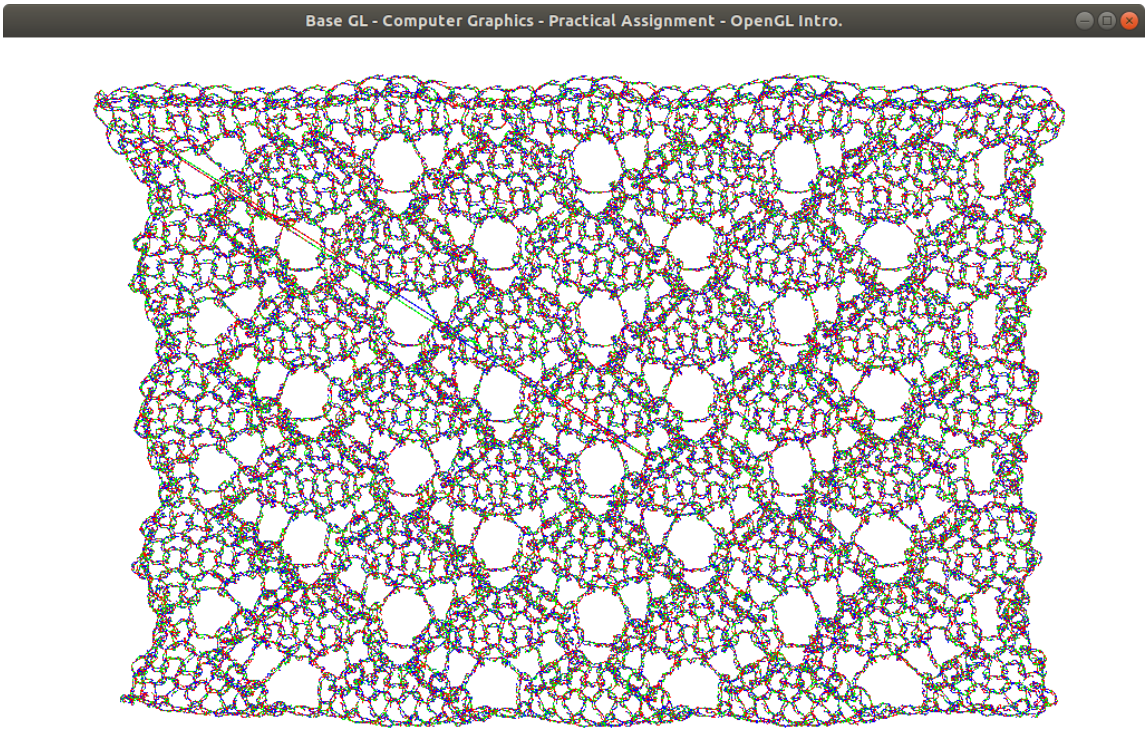


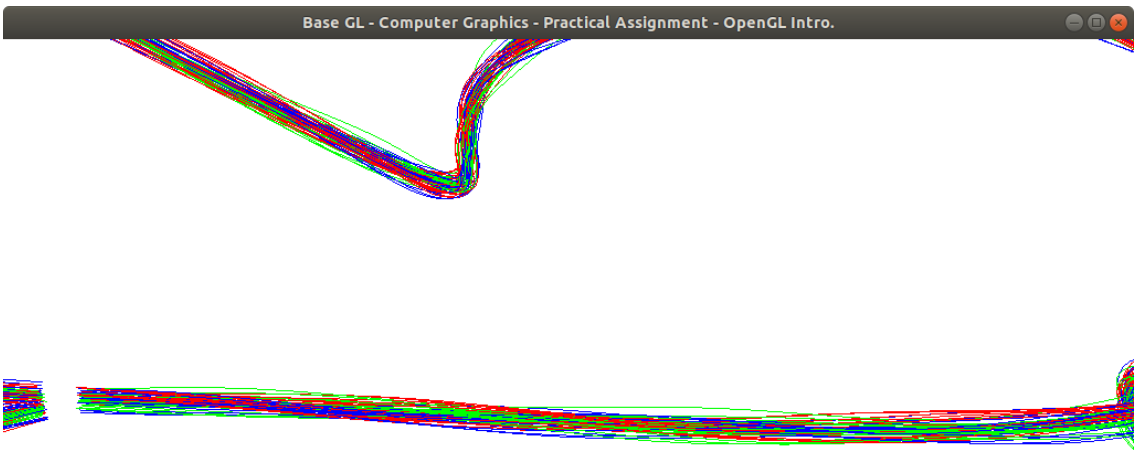
The fiber-level model was now ready to be tested on bigger sets of control points, i.e. some of the models present on http://www.cemyuksel.com/research/yarnmodels/ . Results are shown in the next section.

Results

Here are some results of applying the fiber level rendering to an imported yarn model, for different parameters. To compare, you can view the mesh in the Models folder with MeshLab for instance.




Base GL - Computer Graphics - Practical Assignment - OpenGL Intro.


Base GL - Computer Graphics - Practical Assignment - OpenGL Intro.

Some limitations:

- A line of fibers coming from the top-left corner of the mesh to the center can be seen. I was unable to replicate this bug with the simple models, and so was not able to resolve the problem in time.
- The continuity of the fibers still has to be addressed in the code because discontinuities can be seen in the fiber structure of the model in the results.
- I was not able to import the other models because they did not include the normals for the vertices (for the ones I have tried) which are crucial for the Rotation Minimizing Frames methods.
- The .obj file I have used coded quads which is twice as much control points as needed.

Finally, for the performance, it was not necessary to plot charts because optimization methods were not implemented, and thus comparisons could not be made. A point worth noting is that my computer slowed down a lot and was lagging during rendering. For the tests, I was using an Nvidia GeForce GTX 1050, which is not a decent GPU, but is not bad either. This may suggest that an unoptimized real-time fiber-level rendering of cloth may still not be ready for common applications and usage.

A few precisions:

- Papers or documents used to conduct this project are referenced in code next to the lines of code they were used in.
- While texting, you can use the mouse controls to:
    - Rotate the camera (Left mouse button)
    - Move the camera (Right mouse button)
    - Zoom (Scroll button- you have to press it and pan)