Simulating Hair with Cosserat Rods

Master Semester Project

Lucas Strauss

lucas.strauss@epfl.ch

Computer Graphics and Geometry Laboratory
EPFL

Supervisors:
Christopher Brandt
Prof. Mark Pauly

January 16, 2020

1 Introduction

With the popularization of digital simulation methods and the digital revolution [12], more and more video games and movies require simulation in diverse forms. Simulation in general is a compromise between realism and performance. In the domain of hair simulation great progress has been made, be it in video games where a character can have its hair computed in real time or in a movie where it is possible to have a slower but more realistic hairstyle simulation. However, this aspect remains a very challenging domain of digital character creation. In this project we study a new approach of hair simulation using the Cosserat Rod theory in order to obtain a more realistic description of hairs. We implement a user-friendly interface to play with hairs attached to a mesh that can be moved.

2 Related work

Multiple techniques of hair simulation have been developed over the years. One can use volumetric methods [13] and describe the whole set of hair as one big volume of particles, which is related to particle simulation [7]. Another basic yet efficient method is to describe the hair as a linked sequence of points where each pair of connected points is modelled by a small spring [5]. This simple model allows fast and efficient simulation of large amount of hair strands, as shown in [9]. Recently a novel approach has been developed by Bouaziz et al. [3] (which is itself an extension of Position Based Dynamics [10] for general simulation). This method has then been extended to work with Cosserat Rods by Soler et al. [14] and allows representing a hair strands as a serie of points and orientations, allowing the easy handling of twisting, shearing, bending and stretching behavior. This is the approach that will be studied in this project. Similarities can be found with the one of Bertails et al. [2] who use Super-Helices to describe hairs.

To handle collisions McAdams et al. [8] uses Lagrangian methods to obtain a highly realistic behavior of hairs allowing self-collision and stickiness but at the cost of several minutes of computation per frame. Using the volumetric approach can lead to pretty good and efficient results as shown by Bando et al. [1].

3 Theory

Before we dive into the implementation details, here are a some important theory concepts used throughout this project.

3.1 Cosserat Rods

Continuous Cosserat Rods are described by an arc-length parametrization $\mathbf{r}(s)$ and with each point is associated an orientation. This orientation is represented in our case by a quaternion $u(s)$. A quaternion is an element of $\mathbb{H}$ and generally represented in the form $w + x \cdot \mathrm{i} + y \cdot \mathrm{j} + z \cdot \mathrm{k}$, where $\mathrm{i}, \mathrm{j}, \mathrm{k}$ are the fundamental quaternion units. Here we will represent it as $u = \{w, x, y, z\} \in \mathbb{R}^4$. We denote $\circ$ as the quaternion product and $\overline{u}$ as the conjugated quaternion of u.

This quaternion describes a frame of orthonormal vectors $\{\mathbf{d}_1(s), \mathbf{d}_2(s), \mathbf{d}_3(s)\}$ that can be retrieved with $\mathbf{d}_k = R(u)\mathbf{e}_k = u \cdot \mathbf{e}_k \cdot \overline{u}$, with $R(u)$ being the rotation matrix of the quaternion u and $\mathbf{e}_k$ being one of the 3 basis vector.
This description is discretized by uniformly sampling the rod to obtain a curve with N points. Each element of the rod is defined by two points $\{\mathbf{x}_n, \mathbf{x}_{n+1}\}$ and one quaternion u_n. Hence the discrete rod consist of $N - 1$ quaternions.

To be as clear and consistent as possible we will use the bold notation $\mathbf{x}$ to describe a vector and the non-bold notation u to describe a quaternion.

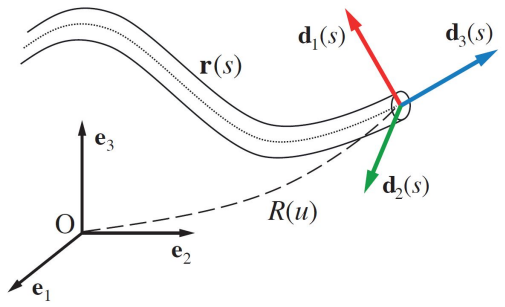

Figure 1: A Cosserat Rod and its orientation

3.2 Projective Dynamics

The Projective Dynamics framework is based on the concept of constraint projection. It tries to solve the Newton's equation described with implicit Euler integration of the form

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \cdot \mathbf{v}_{n+1}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h \cdot \mathbf{M}^{-1}(\mathbf{f}_{int}(\mathbf{x}_{n+1}) + \mathbf{f}_{ext})$$

Where $\mathbf{x}, \mathbf{v} \in \mathbb{R}^{N \times 3}$ are respectively the positions and the velocities, $M \in \mathbb{R}^{3N \times 3N}$ is the mass matrix and h is the time step. $\mathbf{f}_{ext}$ are the external forces of the system and $\mathbf{f}_{int}$ are the internal forces, it is this last term that will be represented as constraints. The algorithm tries to find a trade-off between the constraints and the momentum term in two steps: a local step

and a global step. The local step computes the projected positions for each constraints and the global step tries to find an agreement between all these constraints in order to update the positions.

However, as Cosserat Rods are being used we have to take into account the orientations associated with each position. This extension is far from trivial and is described in [14]. Equivalently to the velocity $\mathbf{v}$ and the mass matrix $\mathbf{M}$ for the positions are the angular velocities ω and the inertia matrix $\mathbf{J} \in \mathbb{R}^{4(N-1)\times 4(N-1)}$ for the quaternions. In addition to those comes the torque variable τ. This can be seen as a rotating external force applied on each quaternion.
The inertia matrix is defined as the concatenation of $\mathbf{J}_n = l_n \cdot \rho \cdot diag(0, J_1, J_2, J_3)$ for each orientation index n and segment length l_n with mass density ρ. $J_1 = J_2 = \frac{\pi r^4}{4}, J_3 = J_1 + J_2 = \frac{\pi r^4}{2}$ are the moment of intertia of each basis axis.

Putting everything together we need to define the flat vector $\mathbf{q}$ as the concatenation of the positions and the orientations $\mathbf{q} = \left[\mathbf{x}_1^\top, \ldots, \mathbf{x}_N^\top, u_1, \ldots, u_{N-1}\right]^\top$ as well as $\mathbf{M}^* = \begin{pmatrix} \mathbf{M} & \\ & \mathbf{J} \end{pmatrix}$ the concatenation of the mass matrix and and the inertia matrix.

4 Implementation

A difference between the Projective Dynamics for Cosserat Rods and the original one is that we mainly use flat vectors ($\mathbb{R}^{3N}$) instead of arrays of vectors ($\mathbb{R}^{N\times 3}$)

4.1 Algorithm

Regarding the local step (line 9) the details are in the Constraints section 4.3.
For the global step (line 11) the function returns the least squares solution of the following system:

$$\left(\frac{\mathbf{M}^*}{h^2} + \sum_i w_i \mathbf{S}_i^\top \mathbf{A}_i^\top \mathbf{A}_i^\top \mathbf{S}_i\right) \mathbf{q}^{t+1} = \frac{\mathbf{M}^*}{h^2}\mathbf{s}^t + \sum_i w_i \mathbf{S}_i^\top \mathbf{A}_i^\top \mathbf{B}_i \mathbf{p}_i$$

Here w_i are the weights of each individual constraints, they model the contribution of that constraint to the overall system. The $\mathbf{S}_i$ matrix is the selection matrix, which identifies the variables of $\mathbf{q}$ involved in the constraint. The $\mathbf{A}_i$ and $\mathbf{B}_i$ matrices are specific to each constraint type and described more precisely in section 4.3.

Note that the left hand side of the system does not depend on the projections and can therefore be pre-factorized when we initialize our system.

4.2 Additional functions

The *Eigen* library [6] was used as much as possible for data types and matrix operations because of its optimized implementation.

Algorithm 1 Extended Projective Dynamics for Cosserat Rods

1: **function Solve**$(\mathbf{q}^t, \mathbf{v}^t, \omega^t)$
2: $\quad {\mathbf{s}_\mathbf{x}}^t = \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{M}^{-1} \cdot \mathbf{f}_{ext}$
3: $\quad \mathbf{s}_\omega^t = \boldsymbol{\omega}^t + h\mathbf{J}^{-1}\left[\tau - \boldsymbol{\omega}^t \times (\mathbf{J}\boldsymbol{\omega}^t)\right]$
4: $\quad s_u^t = u^t + \frac{1}{2}h(u^t \circ s_\omega^t)$
5: $\quad \mathbf{s}^t = \left[{\mathbf{s}_\mathbf{x}}^t, s_u^t\right]^\top$
6: $\quad \mathbf{q}^{t+1} = \mathbf{s}^t$
7: $\quad$ **for** $i = 1$ **to** numIterations **do**
8: $\quad\quad$ **for all constraints** c_i **do**
9: $\quad\quad\quad \mathbf{p}_i = c_i.\mathit{ProjectOnConstraintSet}(\mathbf{q}^{t+1})$
10: $\quad\quad$ **end for**
11: $\quad\quad \mathbf{q}^{t+1} = \mathit{SolveLinearSystem}(\mathbf{s}^t, \mathbf{p}_1, \mathbf{p}_2, \ldots)$
12: $\quad$ **end for**
13: $\quad \mathbf{v}^{t+1} = \frac{1}{h}(\mathbf{x}^{t+1} - \mathbf{x}^t)$
14: $\quad \omega^{t+1} = \frac{2}{h}(\overline{u}^t \circ u^{t+1})$
15: $\quad$ **return** $\mathbf{q}^{t+1}, \mathbf{v}^{t+1}, \omega^{t+1}$
16: **end function**

As shown in line 14, operations are performed sometimes on the orientations as usual quaternions instead of as a vector like at line 3. This back and forth move in the representation requires us to have efficient and stable functions that performs these conversions. Going from a vector of quaternions ($\mathbb{R}^{4(N-1)}$) to a vector of positions ($\mathbb{R}^{(N-1)\times 3}$) is achieved by simply taking the imaginary part of the normalized quaternions (as described in [14]). For the other way around we need to reconstruct the quaternion using the fact that it is normalized (i.e. $\sqrt{w^2 + x^2 + y^2 + z^2} = 1$).

Some operations that are not supported by the *Eigen* library are the component-wise multiplication of quaternions (as required on line 4 and 14) and cross product between flat vectors (line 3). These operations are easily parallelizable and should not impact the performance of the algorithm.

4.3 Constraints

Here we describe the implemented constraints. This correspond to the line 9 in the above algorithm.

- **Bending and Twisting**: The goal of this constraint is to minimize the difference of rotation of two consecutive quaternions. The mathematical derivation can be found in [14]. The involved variables are retrieved with $\mathbf{S}_i\mathbf{q} = [u_n, u_{n+1}]^\top$ and reformulated as $\mathbf{p}_i = \{u_n^*, u_{n+1}^*\}$ with $u_n^* = u_n \circ \frac{\boldsymbol{\Omega}_n}{2}$, $u_{n+1}^* = u_{n+1} \circ \frac{\overline{\boldsymbol{\Omega}}_n}{2}$ and $\boldsymbol{\Omega}_n = Im(\overline{u}_n \circ u_{n+1})$. Here $\circ$

is the quaternion multiplication and $Im(u)$ is the imaginary part of the quaternion. In addition to this the matrices $\mathbf{A}_i = \mathbf{B}_i = \mathbf{I}_8$, where $\mathbf{I}_k$ is the $k \times k$ identity matrix.

- **Stretch and Shear**: Here we want to keep the rod from extending and shearing too much. This constraint is also originally stated in [14]. To achieve this we retrieve the variables $[\mathbf{x}_{n+1}, \mathbf{x}_n, u_n]^\top$ from $\mathbf{S}_i\mathbf{q}$. The projection variables are $\mathbf{p}_i = \{\mathbf{x}_f^*, u_n^*\}$ and computed as follows: $\mathbf{x}_f^* = \mathbf{d}_3$ and $u_n^* = u_n \circ \partial u_n$, where $\mathbf{d}_3 = R(u)\mathbf{e}_3$ is the normal of the rod's cross section, ∂u_n is the quaternion representing the differential rotation between $\mathbf{d}_3$ and $\frac{1}{l}(\mathbf{x}_{n+1} - \mathbf{x}_n)$.

$$\text{Here } \mathbf{A}_i = \begin{bmatrix} \frac{1}{l}\mathbf{I}_3 & -\frac{1}{l}\mathbf{I}_3 & \mathbf{O}_{3,4} \\ \mathbf{O}_{4,3} & \mathbf{O}_{4,3} & \mathbf{I}_4 \end{bmatrix}, \mathbf{B}_i = \begin{bmatrix} \mathbf{I}_3 & \mathbf{O}_{3,4} \\ \mathbf{O}_{4,3} & \mathbf{I}_4 \end{bmatrix}$$

  where $\mathbf{O}_{k,m}$ is a $k \times m$ zero matrix. Note that is the only implemented constraint where the matrices $\mathbf{A}_i$ and $\mathbf{B}_i$ are not identity.

- **Moving Position**: For the simulation to be interactive we need to be able to attach the root of the hair strands to a mesh that can be moved by the user. This constraint models this behavior by first requiring a pointer to the list of the mesh points and the index of a point that will be the projected position for each time step. Formally we retrieve the constrained point with $\mathbf{S}_i\mathbf{q} = [\mathbf{x}_n]^\top$ then fetch the corresponding point $\mathbf{x}'_n$ of the mesh using our pointer and the projected variable become $\mathbf{p}_i = \mathbf{x}'_n$. This use of pointers make the implementation very efficient.
  The matrices $\mathbf{A}_i$ and $\mathbf{B}_i$ are simply $\mathbf{I}_3$.

- **Normal**: This simple constraint aims at forcing the root of the rod (the first element) to keep its original orientation. This is particularly useful when combined with the Moving Position constraint because it prevents the hair strand to have a vertical resting position which adds a touch of realism to the simulation. The required variable is just $\mathbf{S}_i\mathbf{q} = [u_n]^\top$ and requires an initial objective quaternion u'_n. The projection is trivially given by $\mathbf{p}_i = u'_n$ and $\mathbf{A}_i = \mathbf{B}_i = \mathbf{I}_4$.

- **Sphere Collision**: Because we use a sphere mesh in our implementation, it is more efficient to handle collisions in an implicit manner. Instead of checking the collision condition of an arbitrary object, we use the implicit sphere equation and project the point on the closest point lying on the sphere if $||\mathbf{x}_n - \mathbf{x}_0|| < r$, with $\mathbf{S}_i\mathbf{q} = \mathbf{x}_n$, $\mathbf{x}_0$ the center of the sphere and r its radius. This constraint can also handle the sphere moving if we provide it the array of mesh points and a "witness" index point. Because we assume that the sphere keeps the same shape, we can use this point to check at every time step its position and compare it to its initial position to find the updated sphere center. The projection becomes: $\mathbf{p}_i = \begin{cases} \mathbf{x}_0 + \frac{\mathbf{x}_0 - \mathbf{x}_n}{||\mathbf{x}_0 - \mathbf{x}_n||} \cdot r & \text{if } ||\mathbf{x}_0 - \mathbf{x}_n|| \leq r \\ \mathbf{x}_n & \text{otherwise} \end{cases}$

Again $\mathbf{A}_i = \mathbf{B}_i = \mathbf{I}_3$.
The current implementation can add a *force factor* to the constraint. This allows the weight to be smaller by projecting the point further away from the real value in order to give him more importance.

- **Self Collision**: Handling collisions with the rods is not an easy task. One of the simpler method is to describe the interaction with a spring model. Each position (obtained with $\mathbf{S}_i\mathbf{q} = \mathbf{x}_n$ as usual) is constrained to a set of other points (to avoid an exponential number of constraints) that can be neighboring rod points. We then compute the projection by first initializing $\mathbf{p}_i = \mathbf{x}_n$ and then for each point $\mathbf{x}_m$ of the constraint set, if $||\mathbf{x}_n - \mathbf{x}_m|| < \textit{min distance}$ add $\frac{\mathbf{x}_n - \mathbf{x}_m}{||\mathbf{x}_n - \mathbf{x}_m||} \cdot (\textit{min distance} - ||\mathbf{x}_n - \mathbf{x}_m||)$ to $\mathbf{p}_i$.
The $\mathbf{A}_i$ and $\mathbf{B}_i$ matrices are again $\mathbf{I}_3$.

4.4 Graphical User Interface

The interface uses the *nanogui* library [11] to display the mesh and the rods using *OpenGL*.

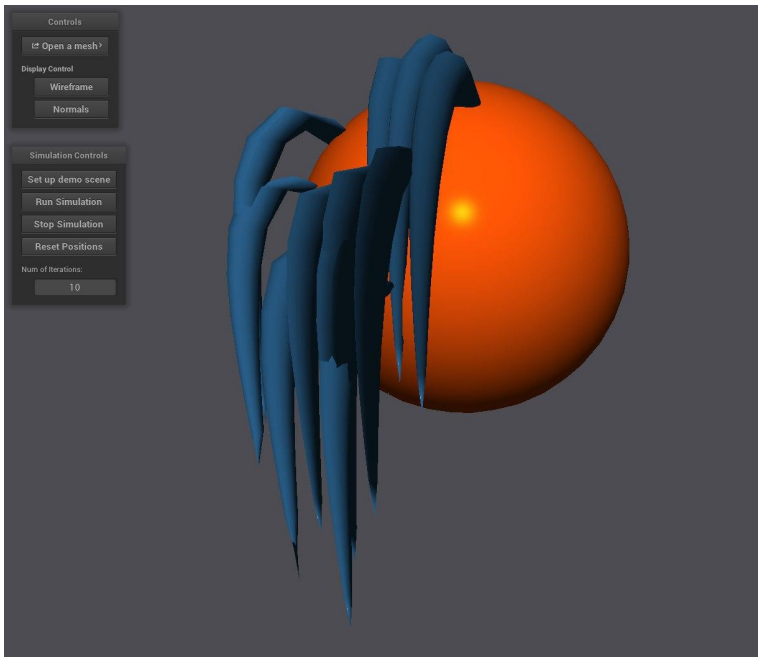

Figure 2: The user interface of the simulation applet with 12 rods of 10 segments each

A single *OpenGL* shader containing each rod is recomputed at each frame from the data of the algorithm using the positions, normals and tangents of each rod segment. From this we can draw the 3D hair strand by offsetting the vertices around each position and connecting them correctly to create faces.

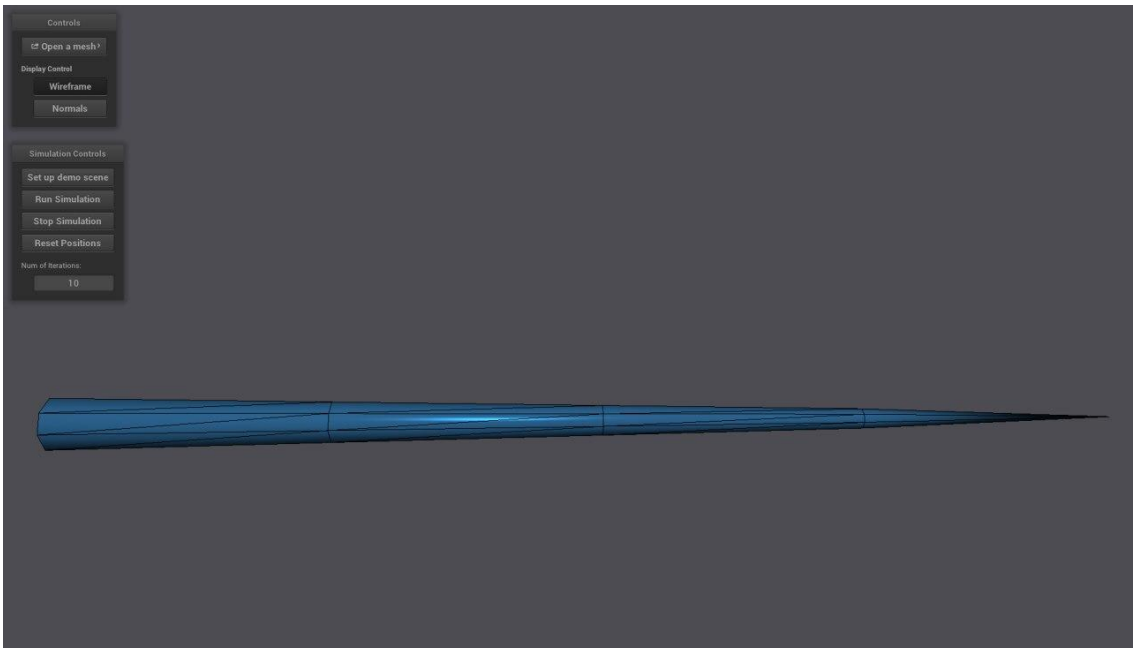

Figure 3: A single rod in wireframe mode

5 Limitations

In the project's current state the simulations present some damping due to the different collision constraints. Having realistic collision without impacting performance too much is a real challenge because of the way projections work. If we want robust collisions we need to assign large weights to the collision constraints but if the point is not in a colliding position, constraint will project it on itself with a large weight which will make the simulation behave like in a viscous fluid. No benchmarks or performance measures have been made but the simulation runs in real time on a consumer desktop (i7-4770S 3.1GHz and 8GB RAM) with a dozen hairs of 10 elements each.

6 Conclusion and future work

The basic simulation has been implemented and works well. The performance of the implementation should be evaluated to find bottlenecks, to see how the algorithm scales as the number of rods increases and measure the computational impact of each constraint.

Now that the simulation framework is implemented, an interesting topic would be to find an efficient way of handling collisions. Several techniques could be explored such as volumetric methods or by finding a way of only adding the collision constraints when needed without having to re-factorize the left hand side matrix because it is computationally heavy.

One interesting approach could be to derive the Cosserat Rods theory for Hyper Reduced Projective Dynamics [4] for an even faster simulation.

References

[1] Yosuke Bando, Bing-Yu Chen, and Tomoyuki Nishita. "Animating hair with loosely connected particles". In: *Computer Graphics Forum*. Vol. 22. 3. Wiley Online Library. 2003, pp. 411–418.

[2] Florence Bertails et al. "Super-helices for predicting the dynamics of natural hair". In: *ACM Transactions on Graphics (TOG)*. Vol. 25. 3. ACM. 2006, pp. 1180–1187.

[3] Sofien Bouaziz et al. "Projective dynamics: fusing constraint projections for fast simulation". In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 154.

[4] Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. "Hyper-reduced projective dynamics". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), p. 80.

[5] Byoungwon Choe, Min Gyu Choi, and Hyeong-Seok Ko. "Simulating complex hair with robust collision handling". In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM. 2005, pp. 153–160.

[6] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[7] Sunil Hadap and Nadia Magnenat-Thalmann. "Modeling dynamic hair as a continuum". In: *Computer Graphics Forum*. Vol. 20. 3. Wiley Online Library. 2001, pp. 329–338.

[8] Aleka McAdams et al. "Detail preserving continuum simulation of straight hair". In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 3. ACM. 2009, p. 62.

[9] Matthias Müller, Tae-Yong Kim, and Nuttapong Chentanez. "Fast Simulation of Inextensible Hair and Fur." In: *VRIPHYS* 12 (2012), pp. 39–44.

[10] Matthias Müller et al. "Position based dynamics". In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118.

[11] *NanoGUI Minimalistic GUI library for OpenGL*. `https://nanogui.readthedocs.io/en/latest/`. Accessed: 2020-01-16.

[12] Alejandro Pardo. "Hollywood and the Digital Revolution: New Consumers, New Markets, New Business Models". In: *Mise au point. Cahiers de l'association française des enseignants et chercheurs en cinéma et audiovisuel* 4 (2012).

[13] Lena Petrovic, Mark Henne, and John Anderson. "Volumetric methods for simulation and rendering of hair". In: *Pixar Animation Studios* 2.4 (2005).

[14] Carlota Soler, Tobias Martin, and Olga Sorkine-Hornung. "Cosserat Rods with Projective Dynamics". In: *Computer Graphics Forum*. Vol. 37. 8. Wiley Online Library. 2018, pp. 137–147.