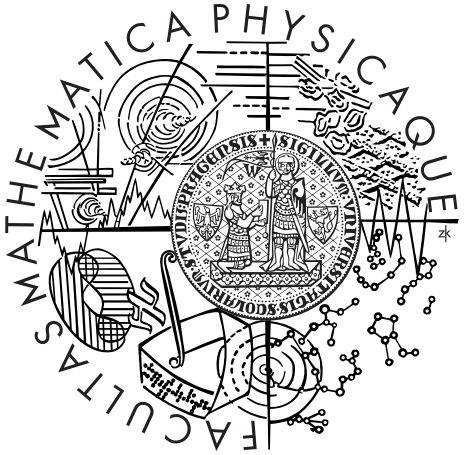
Charles University in Prague
Faculty of Mathematics and Physics

DOCTORAL THESIS



Petr Kmoch

Natural GPU-friendly dynamic hair animation

Department of Software and Computer Science Education

Supervisor of the doctoral thesis: RNDr. Josef Pelikán
Study programme: Computer Science
Specialization: Software Systems

Prague 2015

I would like to thank my supervisor RNDr. Josef Pelikán for his valuable guidance and support throughout my work on this thesis.

I also want to express thanks to Ugo Bonanni, Ph.D. for our fruitful collaboration and for all the ideas we've discussed and shared, and to Prof. Nadia Magnenat-Thalmann for giving me the opportunity to work at MIRALab; time spent there was most beneficial to my work, and a great source of inspiration.

I would further like to thank Jan Beneš for his extensive proofreading and numerous suggestions for improvements in this text. My thanks also go to Petr Kadleček for additional proofreading, as well as to all other colleagues from the Computer Graphics Group at the Faculty of Mathematics and Physics of Charles University for the friendly atmosphere always present in the office, in which no question is considered bothersome.

I also want to thank Mike Flemming for his helpful proofreading.

Last but not least, I would like to thank my parents and my family for their lasting support in this work and in all other aspects of my life.

Parts of this research were supported by GAUK (Grant Agency of the Charles University) grants nr. 50107 and 100209.

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date ..... signature of the author

Název práce: Přirozená dynamická animace vlasů vhodná pro GPU

Autor: Petr Kmoch

Katedra: Katedra Software a Výuky Informatiky

Vedoucí disertační práce: RNDr. Josef Pelikán, Katedra Software a Výuky Informatiky

Abstrakt: Přirozený vzhled vlasů je jedním z klíčových aspektů realističnosti virtuálních lidských postav, protože obličej a hlava přirozeně přitahují lidský pohled. V pohyblivých scénách je realistické chování vlasů důležité stejně jako vzhled. Pro animaci vlasů se často používají fyzikální principy a dynamická simulace, protože jiné tradiční metody animace—jako animace pomocí kostry nebo snímání pohybu—se na vlasy aplikují obtížně. Dynamická animace vlasů je otevřený problém bez známého nejlepšího řešení. Důvodem jsou velmi specifické mechanické vlastnosti vlasů spolu s tím, jak velké množství vlasů se na hlavě nachází. Realistická a přitom rychlá animace vlasů je proto náročný úkol.

V této práci se zaměřujeme na metody dynamické animace vlasů, které mohou pracovat v reálém čase nebo alespoň interaktivně a při tom si zachovat fyzikální věrnost. Na základě výzkumu a analýzy vlastností vlasů z oblasti kosmetického průmyslu jsme navrhli novou metodu dynamické animace vlasů, která poskytuje realističtější výsledky než obdobné existující metody a přitom nabízí lepší výkon i stabilitu. Naši metodu jsme aplikovali na dva různé přístupy k animaci vlasů, abychom dokázali její nezávislost na konkrétní reprezentaci vlasů. U jednoho z těchto přístupů nám naše metoda umožňuje nahradit iterativní minimalizaci funkce přímým výpočtem, čímž se tento krok simulace o řád zrychlil a zároveň se zvýšila jeho stabilita.

Na základě pozorování pohybových charakteristik skutečných vlasů jsme dále navrhli novou metodu uspořádání simulovaných vlasů, která umožňuje reprezentovat větší množství vlasů pomocí menšího počtu simulovaných primitiv, bez nutnosti umělé interpolace.

Dále jsme analyzovali chování vlasů, které se dotýkají, a na základě této analýzy jsme navrhli efektivní metodu řešení kolizí mezi vlasy.

Celá naše animační metoda je navržena s ohledem na možnost implementace na současných vysoce paralelních výpočetních architekturách jako jsou grafické karty (GPU). Pro ověření tohoto návrhu jsme jádro našeho animačního systému naimplementovali také na GPU.

Klíčová slova: vlasy, dynamická animace, GPU

Title: Natural GPU-friendly dynamic hair animation

Author: Petr Kmoch

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Josef Pelikán, Department of Software and Computer Science Education

Abstract: Natural-looking hair is a key component for presenting believable virtual humans, because the head and face form natural focal points of the human figure. In non-static scenes, hair behaviour is just as important as its looks. Principles of physics and dynamic simulation are often used for animating hair, because other traditional animation approaches—such as skeletal animation or motion capture—are difficult to apply to hair. Dynamic animation of hair is still an open problem without a known best solution, because hair has quite specific mechanical properties which, combined with the high number of hairs typically comprising a hairstyle, make realistic and efficient simulation challenging.

In this work, we focus on dynamic hair animation methods capable of providing real-time or interactive performance while staying physically plausible. Basing on research and analysis of hair properties from the cosmetic industry, we have devised a novel hair animation method which provides more realistic results than existing comparable methods while at the same time offering better performance and stability. We have applied this method to two different approaches to hair animation in order to prove its independence on any particular representation of hair. In one of these approaches, our method allows us to replace an iterative function minimisation with a direct computation, giving a speed-up of about an order of magnitude in this one stage of the simulation, while at the same time increasing its robustness.

Based on further observations natural behaviour of real hair, we have also proposed a novel organisation of simulated hair. This allows representing a larger number of hair strands by simulating fewer primitives without introducing artificial interpolation.

Additionally, we have analysed behaviour of real hair when in mutual contact and based on this analysis, proposed an efficient model of collision response for collisions between hair strands.

Our entire method is designed to be easily usable with current massively parallel computation architectures such as GPUs. To validate this design decision, we have also created a proof-of-concept implementation of the core parts of our simulation system on the GPU.

Keywords: hair, dynamic animation, GPU

Contents

List of Figures

Chapter 1

Introduction

The topic of this thesis is dynamic animation of human hair. Our primary focus is on utilising properties of real hair to improve realism and efficiency of hair animation methods. As a secondary goal, we strive to provide a solution capable of producing results fast by utilising parallel computation on commonly available hardware such as Graphics Processing Units (GPUs).

1.1 Scope and Motivation

Dynamically animating human hair is a broad topic. Different animation methods can focus on different aspects such as speed, physical realism of the generated animation or even artistic control over the result. This largely depends on the the intended use of the method. Cinematic animation requires extensive artistic control over the result while still maintaining physical believability. Computation time is not really an issue, as long as at least a scaled-down interactive preview is available. For computer games, on the other hand, speed is key and everything else is secondary.

In this thesis, we focus on approaches to hair animation which offer physically plausible results at interactive rates. That is, we want to find a method which is firmly physics-based, but we're willing to abstract or simplify some details in order to gain speed.

On a human or human-like character, the face and head form a natural focal point—when first looking at such a character, our eyes are instinctively drawn to them (Cerf et al. 2008). This high *saliency* of the human face is well studied in the fields of face recognition and cognitive psychology (Sharma et al. 2009). For this reason, realistic depiction of the face and head plays a key role in the perceived realism of the observed character. Another important factor is familiarity. We are all extremely familiar with what a human looks like. This means we're automatically more demanding on the realism of computer-generated human characters. There are errors in the realism of the behaviour of a human character's hair which will be identified when the same errors in the animation of a lion's mane or a horse's tail would go unnoticed.

Another point which must be considered when dealing with realistic depiction of virtual humans is the so-called *uncanny valley* (Mori 2012). This term applies to a phenomenon of human psychology and perception when viewing an imitation of a living being, particularly a human. When the imitation, such as an animated

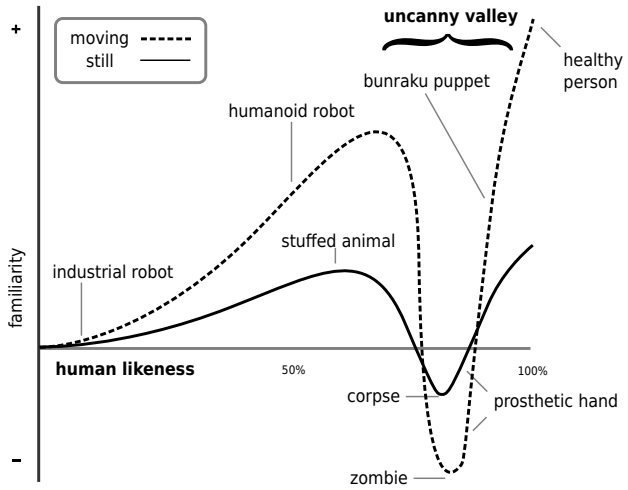


Figure 1.1: "Uncanny valley" phenomenon—high, but still imperfect levels of realism induce a negative response in viewers[1].

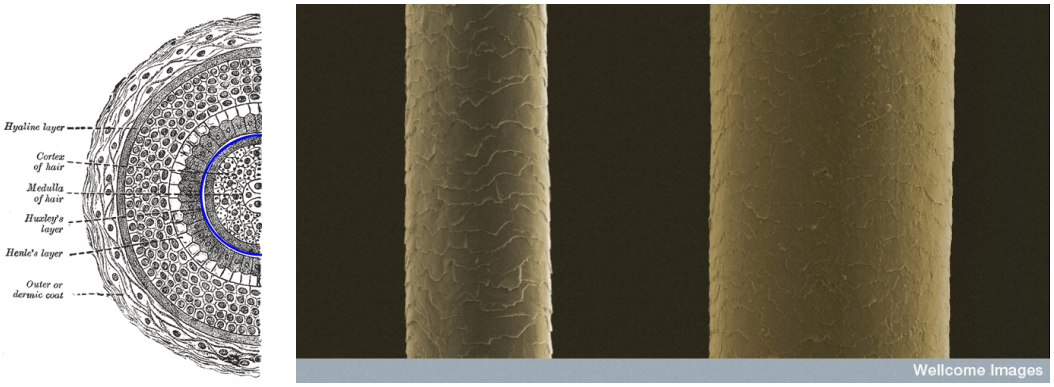
virtual character, is obviously artificial, improving the realism of its appearance or behaviour is appreciated by viewers and perceived as better. Up to a certain point, this perception of representation quality continues to grow with increasing realism of the presentation. However, when the virtual entity comes close to reality but is still perceptibly distinct, a negative perception or revulsion is induced in viewers. Only when the presentation realism increases further to match reality extremely well does the trend reverse again and invoke positive reactions an empathy from observers. Refer to Figure 1.1 for a graphical representation. This phenomenon is studied in several fields dealing with presentation of artificial humans such as digital entertainment (Tinwell 2014) and especially robotics (Ho et al. 2008). The existence of this phenomenon is further motivation for realism in hair animation.

Unfortunately, realistically animating hair is no easy task. Human hair exhibits several specific physical characteristics. It is practically unstretchable and unshearable. At the same time it bends and twists easily, but resumes its rest shape when external load is removed. Additionally, hair strands have a naturally anisotropic character; the length of a typical strand is several orders of magnitude larger than its diameter. This can of course pose problems for accuracy of numerical computations involving hair. These properties, combined with the fact that a typical human has over 100,000 individual hair strands, make accurate and fast physical simulation very difficult.

1.2 Human Hair

Human hair is the subject of extensive study in fields ranging from cosmetics to biology to forensic science. A detailed analysis of hair physiochemical properties, behaviour and morphology was given by Robbins (2002).

[1] Graph image by Smurrayinchester, licensed under Creative Commons BY-SA-3.0, taken from `http://en.wikipedia.org/wiki/File:Mori_Uncanny_Valley.svg`.



(a) Hair follicle (b) Cuticle scales

Figure 1.2: Cross section of a hair follicle, with the hair shaft delimited by a blue line (a). Cuticle scales on the surface of hair strands (b)[2].

A typical human hairstyle consists of between 100,000 and 150,000 hair strands. Hair grows from follicles found in the skin of the scalp.

The mean cross-section diameter of human hair varies with ethnicity and is generally in the range of 80–100 μm (Swift 1995). An important property of hair strands is that they are elliptical in cross section, with eccentricity mainly between 1.2 and 1.7, again dependent on ethnicity (Vernall 1961). This eccentricity is important to our model, and we will refer to it in Section 3.1.2.

Internally, a hair strand is composed of three main parts: the cuticle, cortex, and medulla (see Figure 1.2). At the core of the hair shaft is the medulla, composed of loosely connected keratin cells. The middle layer is the cortex which contains melanin and is therefore responsible for hair colour. It also determines tensile properties of the strand, and is co-responsible for bending and twisting elastics, along with the outer cuticle layer composed of overlapping translucent keratin scales. Overall, keratin accounts for 65–95% of hair composition. It gives hair its mechanical properties, the most relevant of these being elasticity in bending and twisting along with extremely high tensile strength and resistance to shear.

A comprehensive overview of relevant physiochemical properties of hair is given by Bonanni (2010).

1.3 Related Areas

While our work is in the domain of physically-based hair animation, other aspects of virtual hair are also interesting subjects of extensive study. Most of the properties we have described in the previous section which make hair animation a difficult task have a similarly complicating effect on the related areas of hair capture, modelling, and rendering. We describe these areas briefly in this section, but as they are not central to our work, we refer the reader to existing overview

[2] (a) Original image is a faithful reproduction from (Gray 1918), in public domain.
(b) Image by Anne Weston, LRI, CRUK, Wellcome Images, licensed under Creative Commons BY-NC-ND-2.0, taken from `https://www.flickr.com/photos/wellcomeimages/5814146681/`.

literature (Hadap et al. 2007; Yuksel and Tariq 2010) for more information on these topics.

1.3.1 Acquisition and Modelling

The sheer number of hairs on typical human head makes modelling a virtual hairstyle a non-trivial task. Methods of creating virtual hair can be divided into two broad groups: those based on reconstruction from real-world data, and modelling approaches.

Compared to solid object reconstruction, methods capturing hair have to deal with hair's complex scattering properties and tiny cross-section dimensions. Another complication of hairstyle reconstruction is the fact that not only the surface, but the entire internal structure of the hairstyle must be captured; without it, subsequent re-lighting, shadow computation, or animation cannot be realistic. Most methods of reconstructing hair shape from visual images therefore require carefully controlled lighting and environment set-up and/or a complex capture apparatus, which generally limits them to use in laboratory conditions. Specific techniques applied to this problem include utilising depth-of-field information (Jakob et al. 2009), thermal imaging (Herrera et al. 2012), or reconstruction guided by a pre-computed database of examples (Hu et al. 2014).

Still, traditional 3D modelling remains by far the most common methodology of obtaining a virtual hairstyle. 3D modelling packages such as *Maya* or *3D Studio Max* generally incorporate their own tools for modelling hair, which combine curve-based approaches, continuum simulation and strand dynamics. Yet these tools are usually a world unto themselves and their interaction with other modelling tools from the package can be limited. For 3D artists most familiar with polygonal meshes, this makes such tools rather slow and impractical to use, the end result being several hours are normally required to model a virtual hairstyle. An interesting advancement in this field is the method of Yuksel et al. (2009), who proposed an effective way of representing hair with a polygonal mesh for modelling. This allows artists to design hairstyles using tools they are most familiar with.

A specific approach to modelling hair is virtual hairstyling, that is, simulating the tools used by real-world hairdressers (Magnenat-Thalmann et al. 2006; Ward et al. 2006). Interaction which seeks to simulate reality requires the hair to respond to the user's actions in a physically correct way, which means a dynamic animation scheme must be employed. This makes styling-based approaches the modelling most related to our topic, and we will discuss it more in Section 3.4.2. Such approaches can be very effective due to their intuitiveness, especially if combined with means of 3D input such as a haptic device. On the other hand, this also requires new interaction metaphors which will allow artists to make the best use of all interaction modes (keyboard, traditional 2D mouse/stylus input, 3D/haptics) available. We have explored design of such metaphors for virtual hair styling in (Bonanni et al. 2009b).

1.3.2 Rendering

The high number of strands, each of which typically has sub-pixel thickness under normal zoom, makes rendering hair a challenging task. Matters are complicated further by the fact that hair is partially translucent, which gives rise to complex scattering phenomena within the hair volume and makes correct attenuation and shadow rendering crucial for perceived realism of rendered hairstyles.

Due to the small size of its cross section, hair does not lend itself too well to lighting using traditional models such as the illumination model of Phong (1975). An empirical hair-specific lighting model was developed by Kajiya and Kay (1989) and quickly became the de-facto standard way of shading hair. They proposed a diffuse component obtained by integrating a Lambertian model over a perfect half-cylinder, combined with an ad-hoc specular component similar to that used in Phong shading. An interesting property of their model is that due to hair's small cross-section size, the tangent vector is used in place of the normal for computing incident angles.

In 2003, Marschner et al. performed an extensive set of measurements of light interacting with human hair fibres and designed a new hair lighting model based on these measurements. They found that hair lighting has practically no diffuse component and the colour we normally perceive in hair is caused by a secondary tinted, but specular reflection of hair which has travelled through the strand once and reflect towards the observer from the far side of the strand. Its offset from the primary specular highlight is caused by the tilt of cuticle scales on the strand surface. A drawback of this model is its high computational intensity, although faster approximations have been proposed (Scheuermann 2004; Zinke et al. 2008).

The most intuitive approach to hair rendering is to render each individual strand. Hair strands can be rendered as poly-lines, generalised cylinders, or spline curves. Rendering individual hairs generally suffers from sever aliasing due to the small cross-section size of strands. Due to this, and the high number of strands which need to be rendered for a believable hairstyle, many simplifications have been proposed. These include rendering primitives which represent multiple hairs such as textured triangle strips (Ward and Lin 2003; Koster et al. 2004) or cylinders (Ward et al. 2003), as well as volumetric approaches based on billboard splatting (Bando et al. 2003). Realism of results produced by such simplification methods can be greatly enhanced by using tangent mapping (similar to normal mapping) or another way of per-pixel tangent variation with along a suitable lighting model.

Hair is partially translucent, which makes light attenuation and self-shadowing inside the hairstyle volume a vital component of visual believability. Deep shadow maps were devised by Lokovic and Veach (2000) for computing hair shadowing. A deep shadow map is an extension of a plain shadow buffer; each element (pixel) in the map stores the attenuation function along a ray cast through that element, encoded as a polyline. Deep shadow maps provide very realistic results, at the cost of high computation and memory demands. A very efficient approximation for deep shadow maps was introduced by Yuksel and Keyser (2008). This method, called deep opacity maps, packs all its data in an ordinary 4-channel texture, one per light source. Similar to a shadow buffer, one channel stores a depth map from the point of view of the light source. Each of the remaining channels stores one

layer of attenuation—layer width is fixed, but in each texel layers start from the depth of the illuminated surface point. This makes the layers curved, following the shape of the hairstyle, allowing them to utilise available texture storage space very efficiently.

1.4 Dynamic Animation

Dynamic animation is animation based on physical simulation. The object to be animated is represented by a physical system which describes physical effects acting on the object, and how the object reacts to them. For simulating a solid object such as hair, these effects will typically be forces and torques. The system's description is given as a function of time. This function is then evaluated in time points which correspond to the animation frames. The system's state is computed by the function and used to render the animation frame. This function represents the *equations of motion* of the system.

1.4.1 Notation

Before proceeding further, we'll establish some notation used throughout the text. Because we're dealing with dynamic simulation and a system's evolution over time, nearly all quantities are functions of time. We want to make it clear which ones are and which are not, but explicitly spelling out every quantity as a function of t would lead to clutter and hinder formula readability. We therefore adopt the convention that if a quantity is *not* a function of time, it is underlined: $\mathbf{g}$ depends on time while $\underline{\beta}$ does not. Exceptions are well-known constants such as π, and variables used for indexing.

As is common in related literature, we use the dot accent to denote differentiation by time. For a time-dependent value f:

$$\dot{f} = \frac{\mathrm{d}f}{\mathrm{d}t} \tag{1.1}$$

Another element of notation we apply consistently is using a bold upright typeface for vectors, and normal font weight for scalar values. For example, $\mathbf{v} \cdot \mathbf{w} = m$ states that the dot product of vectors $\mathbf{v}$ and $\mathbf{w}$ is m.

To conserve space and keep formulae readable, we also introduce a notation for the gradient of a function by a vector:

$$\nabla_{\mathbf{z}} f = \frac{\mathrm{d}f}{\mathrm{d}\mathbf{z}} \tag{1.2}$$

We occasionally need to refer to a particular component of a vector. In such case, we index the components starting from 0 and use double bracket notation. If more than one coordinate is listed, the result is a vector:

$$\begin{aligned} \begin{pmatrix} a \\ b \\ c \end{pmatrix} [\![1]\!] &= b \\ \begin{pmatrix} a \\ b \\ c \end{pmatrix} [\![0, 2]\!] &= \begin{pmatrix} a \\ c \end{pmatrix} \end{aligned} \tag{1.3}$$

1.4.2 Equations of Motion

Equations of motion are a set of equations which describe how the simulated system evolves in time. This means they will normally be differential equations with time as the independent variable.

As we've covered above, the primary component of dynamic animation are forces acting on the system. Since by Newton's second law force equals mass times acceleration, use of forces leads to equations of motion being second-order differential equations (recall that acceleration is the second derivative of position with respect to time). If we take vector $\mathbf{g}$ as describing the state of our system, the most general form of such equations of motion is this:

$$\ddot{\mathbf{g}} = \mathrm{F}\left(\mathbf{g}, \dot{\mathbf{g}}\right) \tag{1.4}$$

The exact form of the equations of motion depends on the system being simulated and on the method used to represent/discretise it. We will now present the method of Lagrangian mechanics, used for both physical models presented in this thesis.

Lagrangian mechanics is a method of describing the dynamic characteristics of a system and deriving equations of motion from this description. We present it here briefly to give foundation to our description of hair simulation methods in Chapter 3. For further details and proofs, we refer the reader to existing literature on the subject, such as (Morin 2008).

In Lagrangian mechanics, we describe the system we want to simulate using *generalised coordinates.* We denote the vector of generalised coordinates $\mathbf{g}$, and the corresponding vector of generalised velocities $\dot{\mathbf{g}}$. If the number of generalised coordinates is γ, then $\mathbf{g}, \dot{\mathbf{g}} \in \mathbb{R}^{\gamma}$.

Generalised coordinates parametrise the system in a way which is "natural" for the system or easy to express. For example, when simulating a body moving along a fixed trajectory (such as a roller-coaster), we can parametrise the trajectory as a curve and represent the position of the body using one generalised coordinate, the curve parameter value at its current position.

Even more important than the reduction of dimensionality (one scalar parameter instead of a 3-dimensional Cartesian coordinate vector) is the implicit handling of constraints. When using a Cartesian model, we would have to explicitly model the constraints of the motion (staying on the tracks) and account for them using forces. Lagrangian mechanics allows us to express such constraints implicitly by our choice of generalised coordinates and by formulating the system's internal energy based on these coordinates.

It is of course perfectly possible to select a suitable set of Cartesian coordinates as the generalised coordinates. When simulating a mass-spring system, for example, the Cartesian coordinates of the mass points make for obvious generalised coordinates.

The core component of Lagrangian mechanics is the *Lagrangian* $\mathcal{L}\left(\mathbf{g}, \dot{\mathbf{g}}\right)$, a function which describes the entire dynamics of the system by capturing its kinetic energy T and potential energy U:

$$\mathcal{L}\left(\mathbf{g}, \dot{\mathbf{g}}\right) = T\left(\mathbf{g}, \dot{\mathbf{g}}\right) - U\left(\mathbf{g}, \dot{\mathbf{g}}\right) \tag{1.5}$$

Lagrangian mechanics describes how equations of motion are derived from the

Lagrangian. These, also called the Euler-Lagrange equations, or Lagrange's equations of the second kind, take the following form:

$$\left(\forall i \in \left\{1, \ldots, \underline{\gamma}\right\}\right) \; \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{g}_i}\left(\mathbf{g}, \dot{\mathbf{g}}\right)\right) - \frac{\partial \mathcal{L}}{\partial g_i}\left(\mathbf{g}, \dot{\mathbf{g}}\right) = 0 \tag{1.6}$$

g_i and $\dot{g}_i$ stands for the i-th component of vector $\mathbf{g}$ or $\dot{\mathbf{g}}$, respectively.

The form of the equations presented in equation 1.6 assumes potential forces only. When modelling certain systems, the internal forces can be both potential and viscous. In such case, we have to add a dissipation term to the equations of motion. The standard formulation for this dissipation term D was proposed by Lord Rayleigh (Goldstein 1980) and takes the following form:

$$D\left(\dot{\mathbf{g}}\right) = \frac{1}{2} \sum_{i=1}^{\underline{\gamma}} \sum_{j=1}^{\underline{\gamma}} \underline{\gamma}_{ij} \dot{g}_i \dot{g}_j \tag{1.7}$$

The constants $\underline{\gamma}_{ij}$ are related to damping coefficients of the system being modelled. The dissipation energy is then added to the equations of motion:

$$\left(\forall i \in \left\{1, \ldots, \underline{\gamma}\right\}\right) \; \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{g}_i}\left(\mathbf{g}, \dot{\mathbf{g}}\right)\right) - \frac{\partial \mathcal{L}}{\partial g_i}\left(\mathbf{g}, \dot{\mathbf{g}}\right) + \frac{\partial D}{\partial \dot{g}_i}\left(\dot{\mathbf{g}}\right) = 0 \tag{1.8}$$

Equations 1.6 and 1.8 represent the equations of motion when there is no external input to the system, and only its kinetic and potential energy (and possibly viscous dissipation) affects its dynamic state. In most simulations, we actually want external influences to be present, such as gravity, air/wind effects, interaction with other objects, or explicit user interaction. Together, these *external forces* $\mathbf{F}$ can be included in the equations of motion as the right-hand side:

$$\left(\forall i \in \left\{1, \ldots, \underline{\gamma}\right\}\right) \; \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{g}_i}\left(\mathbf{g}, \dot{\mathbf{g}}\right)\right) - \frac{\partial \mathcal{L}}{\partial g_i}\left(\mathbf{g}, \dot{\mathbf{g}}\right) = \mathbf{F}\left(\mathbf{g}, \dot{\mathbf{g}}\right) \tag{1.9}$$

If the system's internal forces are viscous, a dissipation term can be added to equation 1.9 analogously to equation 1.8.

In addition to the implicit constraints embedded in the choice of generalised coordinates, Lagrangian mechanics can also be used to model systems with additional, explicit constraints, if they can be expressed in the following form:

$$\mathbf{C}\left(\mathbf{g}\right) = \mathbf{0} \tag{1.10}$$

We assume there are ζ constraints, meaning $\mathbf{C}\left(\mathbf{g}\right) \in \mathbb{R}^{\zeta}$.

For each constraint, an additional variable, called the *Lagrange multiplier,* is introduced. These are represented as a vector $\boldsymbol{\lambda} \in \mathbb{R}^{\zeta}$. The Lagrangian is then modified as follows:

$$\tilde{\mathcal{L}}\left(\mathbf{g}, \dot{\mathbf{g}}\right) = \mathcal{L} + \boldsymbol{\lambda} \cdot \mathbf{C}\left(\mathbf{g}\right) \tag{1.11}$$

The equations of motion are obtained from equation 1.11 by the same process as equation 1.6 from equation 1.5:

$$\left(\forall i \in \left\{1, \ldots, \underline{\gamma}\right\}\right) \quad \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{g}_i}\left(\mathbf{g}, \dot{\mathbf{g}}\right)\right) - \frac{\partial \mathcal{L}}{\partial g_i}\left(\mathbf{g}, \dot{\mathbf{g}}\right) + \sum_{j=1}^{\zeta} \lambda_j \frac{\partial C_j}{\partial g_i}\left(\mathbf{g}\right) = 0 \tag{1.12}$$

$$\left(\forall j \in \{1, \ldots, \zeta\}\right) \qquad C_j\left(\mathbf{g}\right) = 0 \tag{1.13}$$

The equations of motion 1.12 are called Lagrange's equations of the first kind. Along with the constraint equations 1.13, they form a system of $\underline{\gamma} + \zeta$ equations with an equal number of unknowns.

1.4.3 Numerical Integration

As we've shown in the preceding section, the dynamics of a system are described by differential equations of motion. Computing the system's evolution over time requires solving (integrating) these equations: given the state of the system at time t_n and all preceding states, compute the state of the system at time $t_{n+1} = t_n + \Delta t$. Only for extremely specific systems is a closed-form solution for these equations available; such luxury is normally reserved for textbook examples. When simulating a real-life system, we need to apply numerical solution methods.

Numerically solving a differential equation means computing a solution which will approximate the true solution to a given degree of accuracy. There are many methods of solving differential equations numerically; they generally belong into one of the following categories:

Runge-Kutta methods These methods compute the value in step $t_{n+1} = t_n + \Delta t$ by utilising only the current value at time t_n. Depending on the exact method, intermediary values (such as value at time $t_n + \frac{\Delta t}{2}$) can be computed and used to better approximate the solution at time t_{n+1}, but they are not retained once the computation finishes.

Multistep methods In contrast, the basic principle of multistep methods is to utilise data from more than one previous point in time when computing the next timestep. The Runge-Kutta methods presented above always start the computation of each timestep t_{n+1} from the value and derivative from a *single* past time point t_n. Higher-order Runge-Kutta methods can compute intermediary values such as half-steps from this before computing the final step, but all these inbetween values are discarded after the next timestep is computed. In contrast to this, multistep methods use two or more past timesteps $t_n, t_{n-1}, \ldots$.

There are also other methods of numerical integration which do not conform to the above grouping. However, as numerical integration is a tool for us rather than a core topic of our work, we refer the reader to existing literature (Bradie 2005; Butcher 2003; Hairer et al. 2006) for further details. We will likewise omit multistep methods from further discussion, as we do not use them.

The simplest Runge-Kutta method is the explicit Euler method. Let us assume we have a system of differential equations of the following form:

$$\dot{\mathbf{y}}(t) = F\left(t, \mathbf{y}(t)\right) \tag{1.14}$$

The explicit Euler method then computes the value in the next time step using a finite difference approximation:

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + \Delta t F\left(t_n, \mathbf{y}(t_n)\right) \tag{1.15}$$

Explicit Euler is a first-order method, that is, the error of computing one time step is $\mathcal{O}\left((\Delta t)^1\right)$. Higher-order methods exist, their error bounded by a higher power of the time step. A widely used higher-order method is the 4th order Runge-Kutta method (sometimes called simply *the* Runge-Kutta method). This

method uses a sequence of computed intermediary points to achieve order in $\mathcal{O}\left((\Delta t)^4\right)$:

$$\begin{aligned} k_1 &= F\left(t_n, \mathbf{y}\left(t_n\right)\right) \\ k_2 &= F\left(t_n + \frac{\Delta t}{2}, \mathbf{y}\left(t_n\right) + \frac{\Delta t}{2} k_1\right) \\ k_3 &= F\left(t_n + \frac{\Delta t}{2}, \mathbf{y}\left(t_n\right) + \frac{\Delta t}{2} k_2\right) \\ k_4 &= F\left(t_n + \Delta t, \mathbf{y}\left(t_n\right) + \Delta t k_3\right) \\ \mathbf{y}\left(t_{n+1}\right) &= \mathbf{y}\left(t_m\right) + \frac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \end{aligned} \tag{1.16}$$

The final value in the next time step is computed using a weighted average of values in between time steps. This method offers better stability and more precise result, at the cost of having to evaluate F and $\mathbf{y}$ four times for each time step computed. In the domain of dynamic animation, evaluating $\mathbf{y}$ means updating object positions, rotations, and other dynamic state; evaluating F amounts to computing forces, torques and any other interactions between simulated objects. If collision response is part of the dynamics (such as penalty forces), collisions must be detected and handled in each intermediate step as well.

Both methods we've shown above are *explicit. Implicit* numerical integration methods also exist. They offer excellent stability at the price of being computationally intensive. An example is the implicit (also called backwards) Euler method:

$$\mathbf{y}\left(t_{n+1}\right) = \mathbf{y}\left(t_n\right) + \Delta t F\left(t_{n+1}, \mathbf{y}\left(t_{n+1}\right)\right) \tag{1.17}$$

Notice that while the explicit Euler method given in equation 1.15 can be computed directly, the implicit equation 1.17 requires solving a dense system of equations of the unknown $\mathbf{y}\left(t_{n+1}\right)$. As a consequence, implicit methods are less commonly used in real-time animation systems due to their high computation cost.

All methods we've discussed so far are designed to solve first-order differential equations; that is, $\dot{\mathbf{y}}$ is present in the equations, but higher order derivatives like *včy* are not (the order of a differential equation is not to be confused with the order of the solution method; the latter refers to the method's error bounds, as we've outlined above). However, note that most systems used for dynamic animation (including Lagrangian mechanics we've covered in Section 1.4.2) work with forces and acceleration, which is a second derivative of position. In general, the equations of motion have the following form:

$$\ddot{\mathbf{y}}\left(t\right) = F\left(t, \mathbf{y}\left(t\right), \dot{\mathbf{y}}\left(t\right)\right) \tag{1.18}$$

The standard approach is to transfer this into a system of twice the number of variables by introducing new variables $\mathbf{z}$ to stand for $\dot{\mathbf{y}}$:

$$\begin{aligned} \mathbf{z}\left(t\right) &= \dot{\mathbf{y}}\left(t\right) \\ \dot{\mathbf{z}}\left(t\right) &= F\left(t, \mathbf{y}\left(t\right), \mathbf{z}\left(t\right)\right) \end{aligned} \tag{1.19}$$

The second order equations in $\mathbf{y}$ and $\dot{\mathbf{y}}$ have thus been transformed to first-order equations in $\mathbf{y}$ and $\mathbf{z}$. Another way of viewing this transformation is treating $\dot{\mathbf{y}}$ as an independent variable rather than as $\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t}$.

This is a largely theoretical construct. In practical implementation, the equation $\mathbf{z}(t) = \dot{\mathbf{y}}(t)$ is considered implicitly true and the computation simply uses the values of position ($\mathbf{y}$), velocity ($\dot{\mathbf{y}}$), and acceleration ($\ddot{\mathbf{y}}$). All equations of motion which we present in Chapter 3 assume this principle is applied.

1.5 Contribution of this Thesis

In order to present our work in a logical fashion, we found it necessary to interleave it with description of existing methods throughout the thesis. These methods generally provide the context necessary for proper explanation of our work. To prevent any confusion regarding which parts of this thesis present original research, our contributions are summarised here, with references to the relevant sections where they are presented and explained in detail.

- Building on research from the hair cosmetics industry which has—to the best of our knowledge—never been applied in the area of computer graphics or animation, we have devised a new approach to modelling hair bending behaviour in dynamic simulation. In Sections 3.4 and 3.6, we apply this approach to two existing hair animation methods and show how it improves both realism and performance of the simulation.

- Based on observations of real-world hair behaviour, we have proposed a specific representation of hair volume which captures a broad range of real-world hairstyles and lends itself both to efficient simulation and collision detection. We couple this with a collision response scheme modelling different interactions observed in real hair. These topics are covered in Section 3.6.3 and Chapter 4.

- Our hair animation algorithm is designed to be GPU-friendly. In Section 3.7, we present a proof-of-concept implementation which offloads the core part of the simulation to the GPU.

Finally, Section 5.1 presents our hair animation algorithms in compact form suitable for global overview.

Chapter 2

History and State of the Art

This chapter presents an overview of existing work on the subject of dynamic hair animation. We go rather further back in history in this overview, to provide a solid foundation and put the field of dynamic hair animation in context. Also, we have found that some ideas presented in older works and not usually considered in more recent ones can actually be applied to modern animation methods with good effect. In fact, several of this thesis' contributions were partly inspired by such ideas, and in this chapter we concentrate on the works which introduced them.

The methods presented are grouped according to the fundamental model or mechanism they use to represent the physical behaviour of hair. The main division is between methods which model hair behaviour in some explicit way, presented in Section 2.1, and approaches which use an alternative principle such as fluid dynamics, which are covered in Section 2.2. An essential point in the evolution of explicit methods was the introduction of the Kirchhoff and Cosserat theory of elastic rods to hair animation. Because of this model's importance, we present methods using it in separate Section 2.3 instead of grouping them with other explicit approaches.

2.1 Explicit Hair Representation

Most dynamic animation methods use some form of explicit representation of hair. This means that the dynamically simulated entities correspond to hair strands. This need not be a one-to-one correspondence—several methods simulate entire clumps of many strands as a single dynamic entity—but the core point is that the dynamic entity is intended to capture behaviour of a well-defined set of concrete strands.

Most methods representing hair explicitly use the notion of a rigid multi-body chain. A rigid multi-body chain is a sequence of rigid bodies, usually thin and cylindrical, connected by mechanical joints. The exact shape and properties of the bodies and joints vary between methods. Overall, approaches based on rigid-body chains tend to provide more accurate simulation results, at the cost of high computational intensity.

An alternative to rigid bodies is a mass-spring system. Such models are based on a set of objects with mass, usually point masses, connected by massless springs. Again, the exact configuration of springs used to represent hair varies between

approaches. Mass-spring methods generally offer fast computation and lend themselves to simpler integration schemes, as long the springs are not too stiff. Results they produce tend to be less realistic; recall from Section 1.2 that human hair is practically inextensible, an effect hard to represent correctly with springs. Rigid body and spring approaches are sometimes combined to mitigate each other's drawbacks.

One common feature of explicit representation methods is use of *guide strands.* To keep the size of the simulated system manageable, only a small number of strands (or other primitives such as wisps) are actually simulated using a full dynamic simulation. Typically, the number of simulated entities will be on the order of hundreds for a full hairstyle. Compared to the 100,000–150,000 strands comprising a typical human hairstyle, the guide strands alone could not produce believable visual results. More strands are then interpolated from the guide strands solely for purposes of rendering. This enables visual representation of a full hairstyle while keeping the computational requirements of the simulation at a manageable level. Being a simplification, the method is not without drawbacks, the chief of them being uniformity of the resulting hairstyle. Different authors use different approaches to mitigate this effect of interpolation, and we will mention them where applicable.

One representative of rigid multi-body chains is the method of Chang et al. (2002). They use a model of point masses connected by rigid segments to represent sparse guide strands. The joints themselves are modelled as a cascade of two separate one-dimensional joints with a fixed rotation axis. Rotations around the hair's longitudinal axis are prohibited, resulting in just two degrees of freedom per joint. Such neglecting of twist deformations is commonly found in other explicit approaches to hair animation as well; however, as we show in Section 3.1.2, not considering twist prevents the simulation from correctly capturing important behaviour of real hair, especially in regards to curliness.

To account for hair-hair interaction, the method introduces two auxiliary structures. The first one are static links, acting on selected pairs of hair vertices (joints). When the relative position of the two linked vertices change, spring forces are applied to draw the vertices back to their original relative configuration. As these links represent static charges, cosmetics, curly intertwining, and similar effects which can be broken by excessive force, there is a threshold set for each link and when the vertices separate by more than this threshold, the link is broken permanently.

The other structure employed are triangle strips connecting guide strands with nearby roots. These triangle strips simulate dense hair between the sparse guide strands and when a strand intersects with a triangle, damped spring force is applied to the strand to simulate hair-hair collision. The triangles are not used for rendering purposes; instead, dense hair is interpolated from the guide strands for rendering. Parts of our method, described in Section 3.6.3 and Chapter 4, employ features partly inspired by these structures.

Hair–body collisions are not solved using forces. Instead, if a hair vertex comes too close to the body, its velocity is set to that of the body. Further acceleration deeper into the body is prohibited; the vertex can still move away or slide along the body, but a frictional force is applied. Penetrating collisions with other parts of the scene are solved by relocating the penetrating vertices

away from the penetrated object and propagating this offset along the length of the strand. Hair-body and hair-object collisions are solved for all hair strands, not just the guide strands. At the time of its publishing, the method worked offline, requiring about 20 seconds to simulate one animation frame using 200 guide strands of 15 segments each.

A slight variation on the multi-body chain is used by Choe et al. (2005). In their method, rigid segments are connected with a linear and an angular spring rather than with joints. The presence of the angular spring allows the method to capture torsional effects.

The method does not simulate individual strands: the segment chains are used as skeletons of cylindrical wisps. Individual strands are created for rendering purposes only, as slightly varied copies of the wisp skeleton.

To account for hair-hair interaction within a wisp, its cross-section diameter increases in proportion to its speed. Wisp–wisp collisions are simulated by applying penalty forces and viscous drag to interpenetrating wisps. Wisp–body collisions are detected by predicting the next simulation state and when a collision occurs, corrective forces are applied to prevent the actual collision. This solves more than 99% of all collisions. The rest is simply dealt with at rendering time by slightly altering the wisp shape. This method also worked in offline simulation times when published, requiring 1–5 seconds to compute an animation frame when simulating 100–300 wisps with a 10 ms time step.

Rigid segments connected by angular springs are also used by Koh and Huang (2001) for simulating strips of hair. The segments are connected by resistance-free joints; bending stiffness is provided by two angular springs at each joint, and torsion is not considered. This dynamic model is applied to control points of a spline surface. The surface is used to represent hair dynamically, and tesselated into a mesh for rendering. Hair–head collisions are solved by approximating the head with an ellipsoid and applying penalty forces to penetrating segments. Springs between strips are used for hair–hair collision avoidance.

A hierarchical approach to dynamic hair simulation was presented by Bertails et al. (2003). They simulate dynamic clustering of hair using a pre-computed tree of cluster hierarchy. Each hair wisp has such a tree associated with its skeleton. When the wisp moves with sufficient acceleration, nodes close to the tip are split, allowing the hairs in the wisp to spread. When acceleration decreases and the individual strands come together, they can be merged again. This process saves from having to compute detailed animation of those parts of the hairstyle where natural clustering causes the strands to behave uniformly.

The dynamic model itself varies based on hair character. Straight hair is modelled by a rigid multi-body chain, whereas curly hair is represented by point masses connected by springs. The reason is the dynamic skeleton controls an entire wisp of hair and not individual strands. The choice of springs for curly hair then enables these wisps to stretch or compress (the curls of the wisp coming farther apart or closer together), while the individual strands retain their length; such behaviour is indeed observed in real curly hair.

For wisp–wisp collision detection, wisps are treated as cylinders. Collision response depends on the mutual orientation of the colliding wisps. If they are

well aligned, they can penetrate each other, but a frictional force is applied. If the colliding wisps are perpendicular, repulsive forces are applied instead.

For hair-body collisions, the head model has several bounding spheres, which are tested for intersection with the wisp cylinders. When penetration occurs, the colliding node is moved out of the body and a frictional force is applied. The method offers considerable acceleration compared to simulating the finest level of detail, but it did not reach interactive frame rates when published; the performance was about 1 s to compute 4 integration time steps of 10 ms each.

A similar hierarchy is used by Ward and Lin (2003) when extending the simulation model of Ward et al. (2003). The method uses three levels of abstraction for hair: individual strands, clusters represented as generalised cylinders, and flat strips modelled as subdivision surfaces. For simulation, the underlying skeleton is the same in all three levels: a chain of nodes represented as point masses and connected by rigid segments, with angular springs at the nodes modelling bending stiffness (the model does not consider torsion). Rendering and collision detection are handled differently for each abstraction level. Subdivision forms the basis of rendering, with curves used for strands and surfaces used for clusters (representing the generalised cylinder's surface) and strips (rendering the strip directly). Collision detection uses sphere-swept volumes around the appropriate representation (polyline, cylinder, surface), with bounding volume hierarchies used for the higher-level abstractions as well as for the scene. Collisions are handled by explicit relocation and velocity changes applied to skeletons of interpenetrating hair.

The different abstraction levels are used as a level-of-detail (LOD) scheme. Ward and Lin (2003) extend this LOD scheme with a precomputed hierarchy of subdivisions for each abstraction level, to make transitions between LODs smoother and faster to evaluate. In addition, they introduce a notion of classifying hair–hair collisions based on the mutual orientation of colliding strands. This dynamic LOD scheme is further extended by Ward et al. (2006) for use in a virtual hair-dressing scenario using a 3D input device. Based on a regular voxel grid, simulation is selectively disabled for hair which with which the user is not currently interacting.

A complex mass-spring system for hair animation was presented by Selle et al. (2008). The stated objective of their method is to simulate an entire hairstyle of 100,000–150,000 strands explicitly, without using any interpolated strands. To enable simulation of such a large number of strands, the method uses a mass-spring system, as these are generally faster to compute. An inherent property of spring-based systems is difficulty in modelling twist. Selle et al. mitigate this by introducing additional torsional springs in a tetrahedral structure. This is possible directly for curly hair, but adding virtual particles is necessary for straight hair.

To be able to use stiff springs to capture hair inextensibility, the authors propose a novel discretisation of linear springs which makes the springs' elastic forces linear in position. Strain limiting is used to prevent excessive stretching during violent head motions. Hair sticking together due to styling products, friction etc. is also modelled by additional springs. Hair–hair collisions are handled as edge/edge repulsion, and for performance reasons, collision detection is not

performed between strand segments connected by a spring which simulates them sticking together. For hair–body collisions, the body is represented using level sets interpolated from motion capture data.

While the method did not reach the full number of strands found in a real hairstyle, explicit simulation of up to 10,000 (1,000,000 particles) strands was presented. The method uses a short time step (typically $6 \cdot 10^{-4}$ s, although the exact length is adaptive), and at time of publishing, it took 4–40 minutes to simulate one such frame on four quad-core Opteron machines.

Lin et al. (2011) use a combination of rigid-body chains and particles to simulate wet hair. Hair strands, simulated as a rigid-body chains, interact with fluid particles simulation water and can absorb them. This absorption leads to an increase in clustering, an effect readily observed in real hair.

The method of Chai et al. (2014) combines dynamic animation of a reduced model with data-driven corrections utilising a detailed pre-computed database of simulated motion. Collision handling is decoupled from the simulation and applied as a detail-preserving correction on top of the reduced simulated model. Skinning is used when interpolating from guide strands.

2.2 Volumetric Approaches

Volumetric methods are a class of dynamic hair animation methods which primarily seek to capture the large-scale behaviour of an entire hairstyle rather than the detailed motion of individual hair strands. Every hair simulation method has to make some compromises and simplifications when simulating an entire hairstyle; fully simulating the 100,000–150,000 strands found in a typical human hairstyle is beyond current capabilities (Selle et al. 2008). Explicit methods do this simulating only a subset of hair (guide strands) or by simulating larger formations of hair as a single entity (wisps). Various techniques are then used to restore the impression of volume to the hairstyle; we've described several in the previous section.

Volumetric methods take the opposite approach. They sacrifice the fine details of individual strands' motion and instead approach hair animation as the task of simulating the behaviour of the hairstyle as a whole. The tool employed to this effect is usually fluid dynamics. For this reason, such approaches are usually well-suited to hairstyles with a certain level of uniformity. Highly specific local features or hair with complex styling can usually not be replicated faithfully by volumetric methods.

The idea of dynamically simulating hair using fluid dynamics was introduced by Hadap and Magnenat-Thalmann (2001). They represent individual strands as rigid multi-body chains of segments connected by 3-DOF joints are thus able to simulate torsion. The dynamics of the chain is expressed using spatial algebra (Featherstone 1987).

Unlike other multi-body chain approaches, Hadap and Magnenat-Thalmann use no explicit mechanism for interactions of the simulated strands with each other or with the surrounding world. Instead, the entire hairstyle is treated as

a volume of abstract "hair matter" continuum, which is simulated as a fluid using smoothed particle hydrodynamics (Desbrun and Gascuel 1996). The fluid continuum is represented by particles. These particles are glued to the segments of individual hair strands. All forces acting on hair (hair–hair interaction, gravity, air drag etc.) are calculated in the particle system and then applied to the multi-body chains, which are used for rendering. In effect, the rigid segments and continuum particles form a coupled system, exerting forces on each other. The continuum is only used for the simulation, rendering is done on the explicit hair strands only.

Hair collisions with solid objects (including hair–body collisions) are handled by boundary particles placed along the boundary of the solid object. These particles are not affected by forces coming from the hair particle system, but they exert repelling forces on approaching hair particles. This method gave offline performance when published.

Bando et al. (2003) take the approach of treating hair as a continuum even further. Geometric representation of individual strands is dropped altogether and hair is simulated entirely using smoothed particles, which thus bear no connection to any concrete strands.

Hair–body collisions are solved by applying repulsive forces to particles colliding with the body. The force is chosen so that it dissipates the relative velocity of the particle normal to the body. For such particles, friction is also applied.

Since there is no notion of individual strands usable for rendering, hair is rendered using texture splatting. The method is limited to simple hairstyles, but it provided interactive frame rates when published, simulating and rendering 2,000 particles at around 7 frames per second.

Further performance improvement is obtained by Volino and Magnenat-Thalmann (2006). In their method, hair animation is reduced to free-form deformation using a three-dimensional lattice. Nodes of the lattice are treated as particles and simulated as a fluid particle system using the Conjugate Gradient method (Press et al. 1992). Hair segments are attached to the lattice using viscoelastic springs called lattice stiffeners. Air, gravity and similar effects are expressed as lattice stiffeners acting on all lattice nodes.

Only guide strands are simulated using the lattice, with more hair strands interpolated from these using small random offsets to prevent unnatural uniformity of the hair. Hair–hair interactions are captured in the lattice computation and thus don't have to be modelled explicitly. Hair–body collisions are solved by approximating the surface of the body using metaballs (Bloomenthal and Bajaj 1997). These exert forces on the lattice nodes penetrating them. To save up computation time, few metaballs are used and they are parametrised adaptively for different lattice nodes. The method provided interactive to real-time simulation rendering when published. Gupta and Magnenat-Thalmann (2005) improved the method by computing self-shadows from hair density in the lattice while still offering interactive frame rate.

2.3 Rod-based approaches

This section describes approaches based on simulating hair using the Kirchhoff and Cosserat theory of the physics of elastic rods. Technically, such methods would be classified as explicit in the taxonomy we've established, but we feel the introduction of rod theory to hair representation was sufficiently novel to warrant separate discussion.s

A rod is defined as a deformable body such that one of its dimensions—its length—is significantly larger than the remaining two dimensions, which make up the rod's cross section. Rod theory has been the subject of many works from the domain of physics and structural engineering (e.g. Kirchhoff 1859; Love 1906; Dill 1992). It was first introduced into computer graphics by Pai (2002) for simulation of sutures used during laparoscopic surgery.

Our own approach is based on the Cosserat formulation of Kirchhoff rod theory as well. As such, we give an extended overview of the rod physics model and some methods using it in Chapter 3. We present it here briefly to provide context for other methods which we do not describe to such depth, but please refer to Section 3.1.1 for a detailed discussion of the rod theory.

An elastic rod is defined by the position of its centreline in 3-D space and by the shape and orientation of its cross section at each point of the centreline. A fully general rod can undergo arbitrary deformations: bending, twisting, extension, compression, and shear. The Cosserat and Kirchhoff theory provides formulae for the rod's internal energy and reaction to external loads; we present these in Section 3.1.1. Methods using it for dynamic hair animation generally differ in ways in which they discretise the model and simulate it.

Rod theory was first used for solving hair statics by Bertails et al. (2005b), and later applied to dynamic hair animation by Bertails et al. (2006). They discretise the rod into an implicit representation as a piecewise helical curve. We build a part of our approach on this method, so we discuss it in-depth in Section 3.3.

Sobottka et al. (2008) use a different rod discretisation for hair animation. They treat the simulation as a two-point boundary value problem instead of the usual initial value postulation, and integrate the resulting equations of motion using the generalised α-method, an implicit integration scheme from the domain of structural dynamics (Chung and Hulbert 1993; Goyal et al. 2003). When published, the method reached interactive rates around 8 frames per second for simulating a single wisp of hair. It should be noted that the method requires boundary conditions to be specified for both ends of the simulated rod.

Chapter 3

Simulating Hair Dynamics

We have investigated several different approaches to modelling physical behaviour of hair in Chapter 2. This chapter will present the specific method we have chosen to model hair in our simulation.

Terminology Occasionally, we need to refer to hair follicle/root positions or configurations on the scalp. We borrow the geography terms *latitude* and *longitude* for such descriptions. For us, the latitudinal direction goes from the top of the head vertically downwards, when in an upright position; it could also be called the cranial–caudal direction. Longitude then refers to going around the head horizontally. Refer to Figure 3.1 for an illustration.

3.1 Modelling Individual Hair

We first define an explicit model for individual hair strands. We start from properties of real hair as outlined in Chapter 1 and introduce several simplifications to arrive at a manageable model.

[1] Head image by Robin Fredman, arrows and labels added by Petr Kmoch, licensed under Creative Commons BY-SA-3.0, taken from `http://undeadstawa.deviantart.com/art/Human-Head-back-view-195491403`.

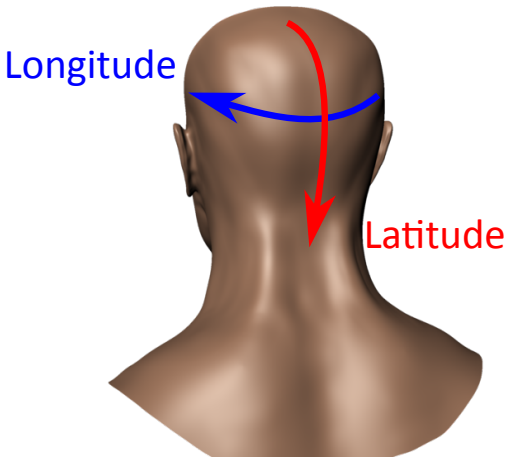


Figure 3.1: Latitudinal and longitudinal directions on the scalp[1].

First, we neglect internal structure, treating the entire strand as homogeneous. This is a fairly safe simplification, as most data on material properties of hair is actually available for entire strands rather than separately for the cuticle, cortex, and medulla (Swift 1995; Wortmann and Schwan-Jonczyk 2006).

For the purpose of strand dynamics, we also neglect the fact that real hair cuticle consists of overlapping tilted scales, and treat the hair surface as smooth. We do model the effects of cuticle scales on interactions between two strands touching each other, though. These are covered by our use of wisps (described in Section 3.6.3) and by tangling effects in of hair–hair collisions, discussed in Chapter 4.

Finally, we treat the cross section as a perfect ellipse, with eccentricity constant along the strand. Variations in cross section size (scaling) along the strand length are possible, and eccentricity can of course vary between strands. Therefore, the final model is in effect a generalised cylinder of elliptical cross section.

Once we define a hair strand this way, we model its behaviour using the dynamics of an *elastic rod.* A rod is a deformable body whose one dimension (length) is significantly larger than the other two (cross section). This perfectly matches properties of human hair strands, where the difference between length and diameter is 4 orders of magnitude.

The idea of using rod theory for animation is not new—it was first introduced into computer graphics by Pai (2002), and used for hair animation by Bertails et al. (2006) and Sobottka et al. (2008). Rod dynamics are also used in computer graphics outside of hair animation, for example by Bergou et al. (2008) and Spillmann and Teschner (2007). In the next section, we formulate the rod theory as it is used by many animation methods, before introducing our own approach in subsequent sections.

3.1.1 Rod Mechanics

The theory of elastic rods has been laid down by Kirchhoff (1859) and built upon by many later works (such as Love 1906; Dill 1992). The Kirchhoff theory applies to rods whose deformation is dominated by bending and twisting while shear and extension are small and slowly varying. As we've stated in Chapter 1, human hair is practically inextensible and unshearable. Therefore, we can safely ignore these deformation modes altogether.

In the rest of this section, we will present the Kirchhoff analysis of rods, simplified to deal with inextensible and unshearable rods only. The configuration of such a rod is given by its centreline and the shape and orientation of its cross section. We formalise these components as functions defined over the rod's length $\underline{L}$:

- Position of the rod's centreline in 3D space:

$$\mathbf{x}(s) : [0, \underline{L}] \to \mathbb{R}^3 \tag{3.1}$$

- Unit vector of the major axis of the rod's cross section:

$$\mathbf{m}_1(s) : [0, \underline{L}] \to \mathbb{R}^3 \tag{3.2}$$

- Unit vector of the minor axis of the rod's cross section:

$$\mathbf{m}_2(s) : [0, \underline{L}] \to \mathbb{R}^3 \tag{3.3}$$

- Scaling factor of the rod's cross section:

$$\underline{S}(s) : [0, \underline{L}] \to (0, +\infty) \tag{3.4}$$

  Remember that we have limited ourselves to rods with elliptical cross sections of constant eccentricity, so a scaling factor is all that is necessary to describe the cross-section shape. We also define $\underline{a}_1$ and $\underline{a}_2$ to be the rod's reference major and minor axis length, respectively. The major and minor axis radii of the rod at point s are then equal to $\underline{S}(s)\,\underline{a}_1$ and $\underline{S}(s)\,\underline{a}_2$, respectively. See Figure 3.2 for an illustration of these concepts.

Figure 3.3 shows a rod with material frame orientation depicted.

Notation When working with these (and other) functions defined over the rod's length, we will sometimes need to differentiate them with respect to the arc length parameter s. We introduce the prime accent as notation for such differentiation; for any function $f(s, \ldots)$ defined along the rod:

$$f'(s, \ldots) = \frac{\mathrm{d}f}{\mathrm{d}s}(s, \ldots) \tag{3.5}$$

Because the cross section is an ellipse, $\mathbf{m}_1$ and $\mathbf{m}_2$ are orthonormal vectors in the cross-section plane. Additionally consider the tangent of the rod's centreline:

$$\mathbf{t}(s) = \frac{\mathbf{x}'(s)}{\left\|\mathbf{x}'(s)\right\|} \tag{3.6}$$

The tangent is of course perpendicular to the cross section. Therefore, $\mathbf{t}$, $\mathbf{m}_1$, and $\mathbf{m}_2$ together form an orthonormal frame adapted to the rod—its *material frame,* denoted $\mathcal{M}$. The rate of change of the tangent then corresponds to the rod's *curvature*. We denote the curvature vector $\boldsymbol{\kappa}$:

$$\boldsymbol{\kappa}(s) = \mathbf{t}'(s) \tag{3.7}$$

$$\kappa(s) = \left\|\boldsymbol{\kappa}(s)\right\| \tag{3.8}$$

The curvature is collinear with the rod's *normal vector* $\mathbf{n}$:

$$\mathbf{n}(s) = \frac{\boldsymbol{\kappa}(s)}{\kappa(s)} \tag{3.9}$$

We also define the rod's (unit) *binormal vector* $\mathbf{b}$:

$$\mathbf{b}(s) = \mathbf{t}(s) \times \mathbf{n}(s) \tag{3.10}$$

The vector $\kappa\mathbf{b}$ is called the rod's *curvature binormal* and can also be defined like this:

$$\kappa\mathbf{b}(s) = \mathbf{t}(s) \times \boldsymbol{\kappa}(s) \tag{3.11}$$

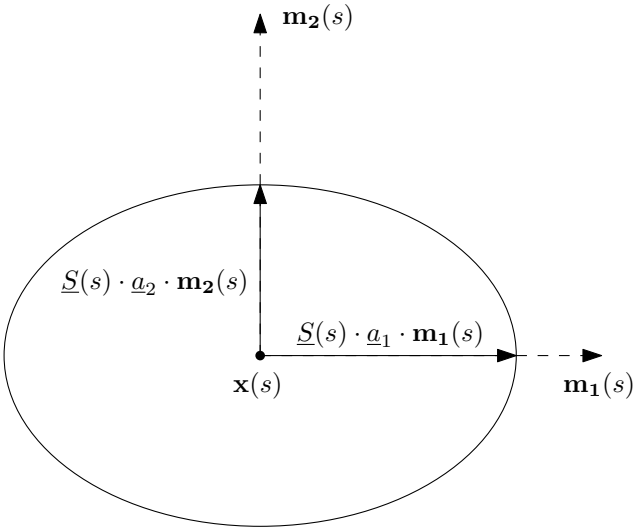


Figure 3.2: Unit-length material axes $\mathbf{m}_1(s)$ and $\mathbf{m}_2(s)$ determine the orientation of the cross section. The cross-section size is computed from the per-strand reference lengths $\underline{a}_{1,2}$ which also specify eccentricity, and from $\underline{S}(s)$ which represents size variation along the strand.

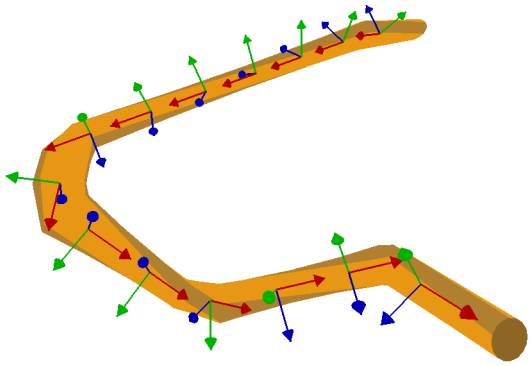


Figure 3.3: Deformed rod with elliptical cross section. Material frame orientation along the rod is shown: red arrows depict the tangent, green and blue arrows are material axes of the cross section.

To be able to use a rod as a dynamic model for our hair simulation, we need its equations of motion (see Section 1.4.2). To formulate these, we need to express elastic energy, or internal energy, of the rod. For this in turn, we'll need functions which describe *strain* on the rod, and a formulation for its *stiffness.*

Strain functions can be formulated as rate of change of the material frame expressed in the material frame coordinates:

$$\begin{aligned} \omega_1(s) &= \mathbf{t}'(s) \cdot \mathbf{m}_1(s) \\ \omega_2(s) &= \mathbf{t}'(s) \cdot \mathbf{m}_2(s) \\ \tau(s) &= \mathbf{m}_1'(s) \cdot \mathbf{m}_2(s) \end{aligned} \tag{3.12}$$

We know from equation 3.7 that the rate of change of the tangent is the rod's curvature. ω_1 and ω_2 therefore represent the curvature vector expressed in the rod's material frame. This measures the rod's *bending* in the direction of the respective material frame axis. That is, ω_1 measures bending in the direction of the major axis, which can alternatively be phrased as bending over the minor axis. Analogously for ω_2. We will sometimes refer to these quantities collectively as components of the *bending vector* $\boldsymbol{\omega}$:

$$\boldsymbol{\omega}(s) = \begin{pmatrix} \omega_1(s) \\ \omega_2(s) \end{pmatrix} \tag{3.13}$$

To be able to express the rod's internal energy, we also need formulae for its stiffness–that is, its resistance to deformation. Stiffness values of the rod depend on its material and its cross-section shape. We use the following notation for them:

- Bending stiffness for bending along the major axis $\mathbf{m}_1$ and minor axis $\mathbf{m}_2$, respectively:
$$\underline{\alpha}_{1,2}(s) : [0, \underline{L}] \to (0, +\infty) \tag{3.14}$$
- Twisting stiffness.
$$\underline{\beta}(s) : [0, \underline{L}] \to (0, +\infty) \tag{3.15}$$

Notice that we present the stiffness values as invariant in time. That is because we assume the material properties of the rod's material do not change throughout the simulation. If we were to simulate application of cosmetic substances, for example, this assumption would not necessarily hold. The cross-section shape and area does not change either, because we have limited ourselves to inextensible and unshearable rods. This means all constituent components of the stiffness values are time-independent, and so are the stiffness values themselves.

The material property which affects bending stiffness is Young's modulus $\underline{E}$. The cross-section shape affects the stiffness by its second moment of area $\underline{I}(s)$. Knowing that the cross section is an ellipse, and using standard engineering formulae for the second moment of area of an ellipse (Beer 2010), we can express bending stiffness as follows:

$$\begin{aligned} \underline{\alpha}_1(s) &= \underline{EI}_1(s) = \frac{\pi}{4}\underline{E}\left(\underline{S}(s)\right)^4 \underline{a}_1\underline{a}_2^3 \\ \underline{\alpha}_2(s) &= \underline{EI}_2(s) = \frac{\pi}{4}\underline{E}\left(\underline{S}(s)\right)^4 \underline{a}_1^3\underline{a}_2 \end{aligned} \tag{3.16}$$

Analogously, twisting stiffness is affected by shear modulus $\underline{\mu}$ and the cross-section moment of inertia $\underline{J}(s)$. The moment of inertia for an elliptical cross section can be found in engineering literature (Irgens 2008), leading to the following formula for twist stiffness:

$$\underline{\beta}(s) = \underline{\mu}\,\underline{J}(s) = \pi\underline{\mu}\left(\underline{S}(s)\right)^4 \frac{\underline{a}_1^3\underline{a}_2^3}{\underline{a}_1^2 + \underline{a}_2^2} \tag{3.17}$$

Because we treat the rod as a perfectly elastic deformable body, it has exactly one "natural" configuration—one which the rod would take if not subjected to any external forces. It is the configuration which minimises the rod's internal energy. We refer to this as the *natural* or *rest* configuration. The rest shape is an intrinsic property of the rod and can in principle involve arbitrary bending and torsion. As an example, the rest shape of a typical spring is a helix.

We use a hat accent to denote the value a quantity has for the rod's rest configuration. For example, $\boldsymbol{\omega}(s)$ is the rod's bending vector at point s in time t, while $\hat{\underline{\boldsymbol{\omega}}}(s)$ is the rod's natural bending in the rest configuration.

We use Γ to denote the entire definition of the simulated rod:

$$\Gamma = \left\{\mathbf{x}, \mathbf{m}_1, \mathbf{m}_2, \underline{S}, \underline{\alpha}_1, \underline{\alpha}_2, \underline{\beta}, \hat{\underline{\boldsymbol{\omega}}}, \hat{\underline{\tau}}\right\} \tag{3.18}$$

With the strains and stiffness values in place, it is possible to express the internal elastic energy of a rod. This has been formulated by Audoly and Pomeau (2010):

$$U(\Gamma) = U_{\text{twist}}(\Gamma) + U_{\text{bend}}(\Gamma) \tag{3.19}$$

$$U_{\text{twist}}(\Gamma) = \frac{1}{2}\int_0^{\underline{L}} \underline{\beta}(s)\left(\tau(s) - \hat{\underline{\tau}}(s)\right)^2 \mathrm{d}s \tag{3.20}$$

$$U_{\text{bend}}(\Gamma) = \frac{1}{2}\int_0^{\underline{L}} \sum_{i=1}^{2} \underline{\alpha}_i(s)\left(\omega_i(s) - \underline{\hat{\omega}_i}(s)\right)^2 \mathrm{d}s \tag{3.21}$$

Alternatively, equation 3.21 can also be formulated using bending vectors:

$$U_{\text{bend}}(\Gamma) = \frac{1}{2}\int_0^{\underline{L}} \left(\boldsymbol{\omega}(s) - \hat{\underline{\boldsymbol{\omega}}}(s)\right)^{\mathrm{T}} \underline{\mathrm{B}}(s)\left(\boldsymbol{\omega}(s) - \hat{\underline{\boldsymbol{\omega}}}(s)\right) \mathrm{d}s \tag{3.22}$$

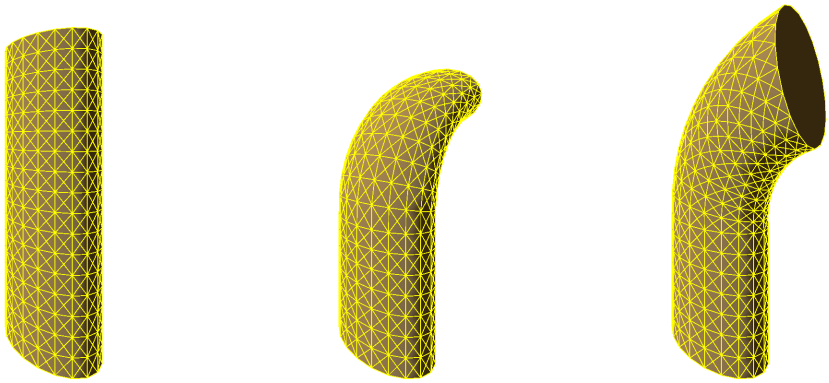
$\underline{\mathrm{B}}$ is the bending stiffness matrix:

$$\underline{\mathrm{B}}(s) = \begin{pmatrix} \underline{\alpha}_1(s) & 0 \\ 0 & \underline{\alpha}_2(s) \end{pmatrix} \tag{3.23}$$

The fact that off-diagonal elements of $\underline{\mathrm{B}}$ are zero follows from the strain analysis presented by Love (1906).

Notice that the above formulation is based on the Kirchhoff theory of rod dynamics which only holds when the rod's extension and shear deformation is small. We therefore rely on hair's high resistance to these deformation modes (as discussed in Section 1.2) for validity of its representation as a Kirchhoff rod.

We now have all the components necessary for building the equations of motion: centreline position $\mathbf{x}(s)$ specified in equation 3.1, strain functions given by equation 3.12, stiffness values (equations 3.16 and 3.17), and internal energy formulation (equations 3.19–3.22). All that is left to do is come up with a suitable set of generalised coordinates—in other words, a suitable discretisation of the continuous rod model presented above. But before moving to that, we first present an analysis of properties specific to hair.



(a) Eccentric rod

(b) Bend over major axis

(c) Bend over minor axis

Figure 3.4: Rod with elliptical cross section of eccentricity 1.67. Notice that while bending over the major axis (b) leads to similar deformations of the rod's material, bending over the minor axis (c) would require extreme compression of the material on one side and extension on the other.

3.1.2 Hair Specifics

So far, we have simplified away some properties of hair strands in order to be able to model them as elastic rods. This allows us to describe their dynamics using well-known physics mechanisms outlined above. Now that we have a physics description, we shall look at whether the fact that we're modelling *hair* can help us either improve or simplify the model.

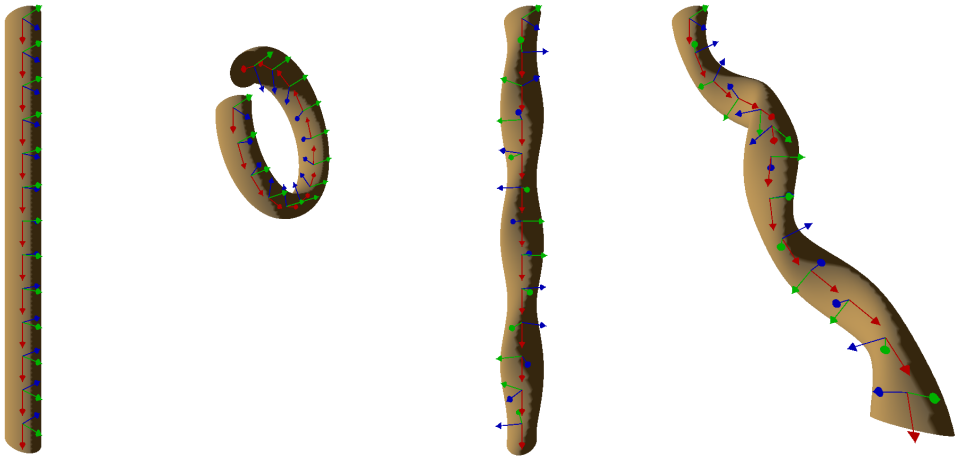
We have already used hair's inextensibility and unshearability to limit the physical model we've just presented to bending and twisting. A generic rod can twist and bend in arbitrary ways, meaning that no further simplifications are possible. In this form, the model forms the basis of several state-of-the-art approaches to hair animation (Bertails et al. 2006; Sobottka et al. 2008).

However, a very important observation about human hair has been made by Swift (1995):

Due to its elliptic cross section, hair tends to preferably bend over its major axis.

See Figure 3.4 for an illustration.

Importance of this observation is easily demonstrated on the fact that this behaviour is one of the two main components giving rise to curliness in real human hair. The other component is intrinsic twist. A naturally untwisted hair strand will be straight or wavy when at rest. A strand with intrinsic twist will form a curl. The reason is that in a naturally twisted hair strand, the direction of the cross-sectional major axis $\mathbf{m}_1$ gradually changes. Since bending favours the major axis, the shape will naturally tend to a helical shape, i.e. a curl. This is demonstrated in Figure 3.5. With no intrinsic twist, the direction of the major axis stays constant along the strand and bending thus leads to a circular shape (Figure 3.5b), or to a wavy shape if the bending orientation changes. When the strand is intrinsically twisted, however, the direction of the major axis along the strand changes and so bending over it will produce a curl, as in Figure 3.5d. This



(a) Zero twist (b) Zero twist: bends into circle (c) Twisted (d) Twisted: bends into curl

Figure 3.5: A naturally untwisted rod (a) forms a circle when bent over its major axis (b). A rod with non-zero natural twist (c) will form a curl instead (d). Note that the total amount of bending, as well as the amount of bending in each point of the rod, is exactly the same in both scenarios (b) and (d).

is how real hair bends naturally—when made to bend in a certain direction, it will actually twist so that the bending happens over the major axis of its cross section.

Computer simulations are of course capable of accurately capturing and modelling hair curliness. However, to the best of our knowledge, no existing method of simulating hair dynamics uses this property of major-axis preference *explicitly*.

For us, it is a key property and forms the cornerstone of our model of hair dynamics. We primarily use it to reduce the number of degrees of freedom of the model, constraining bending so that it happens over the major axis. This is useful in and of itself by reducing the size of the problem we have to solve. More importantly, however, it can allow us to employ specialised solutions to some parts of the problem. Later in this chapter, we will show two existing approaches to discretisation of the Kirchhoff rod model (Sections 3.3 and 3.5) and demonstrate how exactly each of them can be simplified by application of this principle, in Sections 3.4 and 3.6, respectively.

3.2 Collective Hair Properties

In the previous section, we've established the model we'll be using for simulating individual hair strands. However, our goal is to simulate an entire hairstyle, that is, a full hair volume. A typical human has between 100,000 and 150,000 hairs. Since our goal is to approach real-time simulation on hardware plausible for a desktop workstation, simulating all strands in a full hairstyle individually is not an option (Selle et al. 2008).

Fortunately, simulating each individual strand is not necessary for a plausible, or even realistic, result. In effect, the sheer number and proximity of hair strands

causes them to behave in a collective fashion. Features of this collective behaviour can be studied and applied to a simulation system. As we've shown in Chapter 2, elements of individual strand behaviour and collective behaviour can be mixed and meshed in many different ways. Recall that some methods go so far as to treat the hairstyle as an actual volume of abstract "hair matter" whose behaviour is governed by fluid dynamics (Hadap and Magnenat-Thalmann 2001; Bando et al. 2003; Volino and Magnenat-Thalmann 2006).

Analysing properties of an individual hair strand quickly leads to the physics model of a rod as the best way to represent it. Collective behaviour of an entire hairstyle doesn't offer a similarly clear-cut model for its representation. In the following sections, we will analyse different aspects of collective hair behaviour and see what features we can extract from it to add to our simulation model.

3.2.1 Hair Volume

Individual hair strands in a volume of hair are constantly interacting in complex ways. These include:

Mechanical friction The effect of friction in hair is generally larger and more erratic than might be expected. This is caused by the surface structure of hair; recall from Section 1.2 that hair has overlapping tilted scales on its surface. This makes friction response highly dependent on mutual orientation of touching strands.

Electrostatic effects Hair is near the very top of the triboelectric series, which means it easily generates static charge via the triboelectric effect (AlphaLab Inc. 2015). Given hair strands' minuscule mass, such static charge can have significant impact on their behaviour. It can make hair stretch out and "stand on end," or cause strands to stick to negatively charged objects.

Lipid cohesion Sebum (oil) is transferred to hair from the scalp sebaceous glands and hair follicles, giving strands a shiny appearance and causing them to clump together more.

Cosmetics Cosmetic products can affect hair behaviour in a vast range of ways, both increasing and decreasing elasticity, rigidity, and tendency to cluster (Robbins 2002).

More often than not, the above effects make hair strands stick together. It is therefore not surprising that in most hairstyles, hair tends to form clumps or wisps.

In these structures, hair strands close to each other will generally behave very consistently. Many methods use this fact to simplify their model by fully simulating the dynamics of only a small number of guide strands. More strands are then interpolated from these guide strands for rendering, sometimes participating in collision detection as well. Examples of these methods include (Daldegan et al. 1993; Chang et al. 2002; Kim and Neumann 2002; Bertails et al. 2005a).

We want to take advantage of this hair behaviour too. We observe that for a large class of hairstyles, not only does hair form wisps, the wisps are of a specific structure. They tend to be "flat," somewhat similar to ribbons—hair with roots

(a) (b) (c)

Figure 3.6: Illustration of flat wisp formation from strands with roots of similar latitude (a); latitudinal distance exaggerated for illustration purposes. (b) and (c) show different hair styles where flat wisps are visible[2].

of similar latitude often cling together, forming a wisp which can be about 1–2 cm wide, but is typically only a few millimetres thick (see Figure 3.6).

We use these flat wisps as another basic principle of our hair animation method. Where the major-axis bending directly shapes the dynamics of individual strands, the flat-wisp paradigm controls a strand's behaviour through its interaction with other strands. We also use it to guide distribution of simulated hair on the scalp. Details of applying this principle are presented in Section 3.6.3 and Chapter 4.

Together, the observations of preferential bending over the major axis and formation of flat wisps form the basis of our approach to hair animation. However, it is important to realise that these two principles are independent. The observation that hair bends over the major axis only is valid even for hairstyles where flat wisps do not form. To demonstrate this, we apply the bending principle to one hair simulation method, without considering flat wisps. This is described in Section 3.4.

Nevertheless, utilising the tendency of hair to form flat wisps can give significant savings in computing power while maintaining realistic behaviour. We show this by applying both principles, i.e. the entirety of our approach, to a different rod discretisation in Section 3.6. There, we show how these two principles can be utilised together to improve both performance and realism.

Due to the constant contact of strands in a hairstyle, any hair simulation method based on explicit representation of strands must be able to handle hair–hair collisions. In our approach, a significant part of this is solved by the flat

[2] (b) Image by Boss Tweed, licensed under Creative Commons, BY-2.0, taken from `https://commons.wikimedia.org/wiki/File:Maria_Sharapova_at_the_2007_US_Open_(Cropped).jpg`.
(c) Image by avilasal, licensed under Creative Commons, BY-2.0, taken from `https://www.flickr.com/photos/7608209@N08/4310706981/`.

wisp paradigm described above. Naturally, collision handling is still necessary; we describe our method in Chapter 4.

3.2.2 Hair–World Interactions

In any realistic setting, hair will always be interacting with other bodies in the scene. Even in a very minimalistic set-up, there is always at least one other object constantly interacting with the hair: the head (and, depending on the hair's length, shoulders and torso as well). This means that any hair simulation method must be able to handle hair–world interactions.

An important observation can be made: hair is much lighter than almost anything else it can come in contact with. This leads to very lopsided collision scenarios, where other objects are hardly affected by coming into contact with hair (unless they are small enough to become entangled). This is advantageous for simulation in a number of ways.

When considering collisions of hair with a heavy object (such as the head), it is a feasible simplification to have only the hair react to the collision, the other object not being affected at all. One immediate advantage is that the object's reaction does not need to be computed. But it can also simplify the collision response of hair itself: it is valid to assume that the object's behaviour will remain unaffected by whatever response the hair takes, which generally leads to simpler computation.

We limit ourselves to simulating such collisions in our method. Our way of handling them is described in Section 3.6.2.

3.3 Helix-based Hair

In this section, we will present one way of discretising the Kirchhoff rod model. This discretisation scheme, called Super-Helices, was introduced by Bertails et al. (2006). We first repeat their model here as originally presented, before we apply our bending-based simplifications to it in Section 3.4. Note that while the principles are taken from (Bertails et al. 2006), we use our notation for consistency.

3.3.1 Super-Helix Discretisation

We choose a set of $(N+1)$ points S_i in the strand's arc $[0, \underline{L}]$ such that $0 = S_1 < S_2 < \ldots < S_{N+1} = L$. The length of the strand, is then divided into N intervals $\mathbb{S}_Q = \left[S_Q, S_{Q+1}\right]$, for $Q = 1 \ldots N$. These segments need not have the same length. Where typical discretisation might approximate the centreline with straight line segments, the Super-Helix segments are *helices*. A helix is a curve with constant curvature and torsion. That is, twist τ_Q and bending $\omega_{1,Q}$, $\omega_{2,Q}$ are constant, but not necessarily 0, on segment $\mathbb{S}_Q$. A Super-Helix is therefore a *piecewise helical* curve, as illustrated in Figure 3.7.

Let us denote the constant values of twist and bending on each segment thus:

$$\begin{aligned} q_{0,Q} &= \tau_Q \\ q_{1,Q} &= \omega_{1,Q} \\ q_{2,Q} &= \omega_{2,Q} \end{aligned} \tag{3.24}$$

These values form the *generalised coordinates* used by the Super-Helix model. They are collected into a vector $\mathbf{q}$ of $3N$ components. This vector $\mathbf{q}$ fully describes the configuration of a strand discretised using the Super-Helix model.

From these generalised coordinates, the twist and bending of the entire strand can then be reconstructed like this:

$$\begin{aligned} \tau(s) &= \sum_{Q=1}^{N} q_{0,Q} \mathbf{1}_Q(s) \\ \omega_1(s) &= \sum_{Q=1}^{N} q_{1,Q} \mathbf{1}_Q(s) \\ \omega_2(s) &= \sum_{Q=1}^{N} q_{2,Q} \mathbf{1}_Q(s) \end{aligned} \tag{3.25}$$

where $\mathbf{1}_Q(s)$ is the characteristic function of the segment $\mathbb{S}_Q$:

$$\mathbf{1}_Q(s) = \begin{cases} 1 & s \in \mathbb{S}_Q \\ 0 & s \notin \mathbb{S}_Q \end{cases}$$

We follow to express equations of motion of the strand in these generalised coordinates.

3.3.2 Super-Helix Equations of Motion

In their Super-Helix model, Bertails et al. (2006) use Lagrangian mechanics with the dissipation term included (see equation 1.8) to derive the equations of motion for the strand:

$$\begin{aligned} &\left(\forall i \in \{0,1,2\}\right) \ \left(\forall Q \in \{1,\dots,N\}\right) \\ &\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{q}_{i,Q}}\right)(\mathbf{q},\dot{\mathbf{q}}) \ - \ \frac{\partial T}{\partial q_{i,Q}}(\mathbf{q},\dot{\mathbf{q}}) \\ &\quad + \ \frac{\partial U}{\partial q_{i,Q}}(\mathbf{q},\dot{\mathbf{q}}) \ + \ \frac{\partial D}{\partial \dot{q}_{i,Q}}(\mathbf{q},\dot{\mathbf{q}}) = \\ &\qquad\qquad = f_{i,Q}(\mathbf{q},\dot{\mathbf{q}}) \end{aligned} \tag{3.26}$$

The terms on the left-hand side describe the physics of the system: $T(\mathbf{q},\dot{\mathbf{q}})$ is the kinetic energy of the strand, $U(\mathbf{q},\dot{\mathbf{q}})$ is its internal elastic energy, and finally $D(\mathbf{q},\dot{\mathbf{q}})$ is the dissipation potential capturing visco-elastic effects.

The right-hand side represents external forces: $f_{i,Q}(\mathbf{q},\dot{\mathbf{q}})$ is the generalised external force acting on generalised coordinate $q_{i,Q}$. It can be computed from

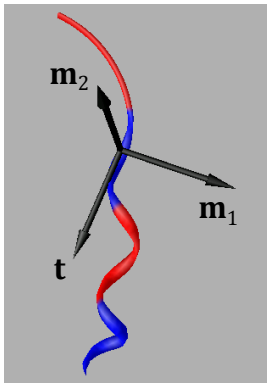


Figure 3.7: Rod discretised as a Super-Helix of 4 segments shown in alternating colours. Material frame at one point of the curve is also depicted.

cartesian force density $\mathbf{F}(s)$ of external forces acting on the strand using the following formula:

$$f_{i,Q}(\mathbf{q}, \dot{\mathbf{q}}) = \int_0^{\underline{L}} \mathrm{J}_{i,Q}(s, \mathbf{q}) \cdot \mathbf{F}(s)\, \mathrm{d}s \tag{3.27}$$

where $\mathrm{J}_{i,Q}(s, \mathbf{q})$ is the Jacobian matrix:

$$\mathrm{J}_{i,Q}(s, \mathbf{q}) = \frac{\partial \mathbf{x}}{\partial q_{i,Q}}(s, \mathbf{q}) \tag{3.28}$$

The external forces $\mathbf{F}(s)$ come from the simulated environment, and typically include phenomena like gravity, air drag and interaction with other objects.

Next, we will present formulae for the individual energy terms which make up the *internal* forces appearing in equation equation 3.26.

Kinetic energy T is defined in the classical way, mass times square of velocity:

$$T(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2} \int_0^{\underline{L}} \rho A(s) \left(\dot{\mathbf{x}}(s, \mathbf{q})\right)^2 \mathrm{d}s \tag{3.29}$$

ρ is material density of the hair matter and $A(s)$ is the area of the cross section. The Super-Helix generalised coordinates are very abstract and do not directly correspond to the hairs' position. For this reason, it is not possible to simply use $\dot{\mathbf{q}}$ when computing kinetic energy; the actual velocity of the strand has to be used, which is why equation 3.29 above still contains an integral. $\mathbf{x}$ and $\dot{\mathbf{x}}$ depend on $\mathbf{q}$ in a highly complex, non-linear fashion; we discuss this in more detail in Section 3.3.3.

The formula for internal energy U is obtained by applying the Super-Helix discretisation to equation 3.19:

$$\begin{aligned} U_{\text{twist}}(\mathbf{q}, \dot{\mathbf{q}}) &= \frac{1}{2} \underline{\beta} \sum_{Q=1}^{N} \left(q_{0,Q} - \underline{\hat{\tau}}_Q\right)^2 \\ U_{\text{bend}}(\mathbf{q}, \dot{\mathbf{q}}) &= \frac{1}{2} \sum_{Q=1}^{N} \sum_{i=1}^{2} \underline{\alpha}_i \left(q_{i,Q} - \underline{\hat{\omega}}_{i,Q}\right)^2 \end{aligned} \tag{3.30}$$

Here, $\underline{\hat{\tau}}_Q$ and $\underline{\hat{\boldsymbol{\omega}}}_Q$ refer to the rest-state values of twist and bending on segment $\mathbb{S}_Q$, respectively. Per the generalised coordinate definition (equation 3.24), these are effectively the rest-state values of $q_{i,Q}$.

The final component, the dissipation potential D, uses the standard formula of the Rayleigh dissipation function:

$$D\left(\mathbf{q}, \dot{\mathbf{q}}\right) = \frac{1}{2}\underline{\gamma}\sum_{Q=1}^{N}\sum_{i=0}^{2}\dot{q}_{i,Q}^2 \tag{3.31}$$

$\underline{\gamma}$ is an empirical internal friction coefficient. Bertails et al. (2006) determine its value in a calibration process by matching the behaviour of the simulation with real hair in a vertical oscillatory motion.

3.3.3 Recovering the Hair Shape

Due to the extremely implicit nature of the generalised coordinates used by the Super-Helix model, it is not trivial to reconstruct the actual shape of the hair strand. This reconstruction is required for computing the velocity $\dot{\mathbf{x}}$ and Jacobian $\mathrm{J}_{i,Q}\left(s, \mathbf{q}\right)$ in the equations of motion (equations 3.29 and 3.27, respectively), as well as for purposes such as rendering or collision detection.

The reconstruction is based on using the Darboux vector $\boldsymbol{\Omega}_{\mathrm{M}}$ of the strand's material frame, for which these equations hold:

$$\begin{aligned}
\mathbf{t}'\left(s\right) &= \boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \mathbf{t}\left(s\right) \\
\mathbf{m}_1'\left(s\right) &= \boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \mathbf{m}_1\left(s\right) \\
\mathbf{m}_2'\left(s\right) &= \boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \mathbf{m}_2\left(s\right)
\end{aligned} \tag{3.32}$$

By substituting these equations into equation 3.12, we find that the strain of the strand represents the coordinates of the Darboux vector in the material frame:

$$\boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) = \tau\left(s\right)\mathbf{t}\left(s\right) + \omega_1\left(s\right)\mathbf{m}_1\left(s\right) + \omega_2\left(s\right)\mathbf{m}_2\left(s\right) \tag{3.33}$$

We compute $\boldsymbol{\Omega}_{\mathrm{M}}'\left(s\right)$:

$$\begin{aligned}
\boldsymbol{\Omega}_{\mathrm{M}}'\left(s\right) &= \tau'\left(s\right)\mathbf{t}\left(s\right) + \omega_1'\left(s\right)\mathbf{m}_1\left(s\right) + \omega_2'\left(s\right)\mathbf{m}_2\left(s\right) \\
&\quad + \tau\left(s\right)\mathbf{t}'\left(s\right) + \omega_1\left(s\right)\mathbf{m}_1'\left(s\right) + \omega_2\left(s\right)\mathbf{m}_2'\left(s\right) \\
&= \tau'\left(s\right)\mathbf{t}\left(s\right) + \omega_1'\left(s\right)\mathbf{m}_1\left(s\right) + \omega_2'\left(s\right)\mathbf{m}_2\left(s\right) \\
&\quad + \tau\left(s\right)\left(\boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \mathbf{t}\left(s\right)\right) && \text{from eq. 3.32} \\
&\quad + \omega_1\left(s\right)\left(\boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \mathbf{m}_1\left(s\right)\right) \\
&\quad + \omega_2\left(s\right)\left(\boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \mathbf{m}_2\left(s\right)\right) \\
&= \tau'\left(s\right)\mathbf{t}\left(s\right) + \omega_1'\left(s\right)\mathbf{m}_1\left(s\right) + \omega_2'\left(s\right)\mathbf{m}_2\left(s\right) \\
&\quad + \boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \left(\tau\left(s\right)\mathbf{t}\left(s\right)\right) \\
&\quad + \boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \left(\omega_1\left(s\right)\mathbf{m}_1\left(s\right)\right) \\
&\quad + \boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \left(\omega_2\left(s\right)\mathbf{m}_2\left(s\right)\right) \\
&= \tau'\left(s\right)\mathbf{t}\left(s\right) + \omega_1'\left(s\right)\mathbf{m}_1\left(s\right) + \omega_2'\left(s\right)\mathbf{m}_2\left(s\right) \\
&\quad \boldsymbol{\Omega}_{\mathrm{M}}\left(s\right) \times \left(\tau\left(s\right)\mathbf{t}\left(s\right) + \omega_1\left(s\right)\mathbf{m}_1\left(s\right) + \omega_2\left(s\right)\mathbf{m}_2\left(s\right)\right)
\end{aligned} \tag{3.34}$$

$$
\begin{aligned}
&= \tau'(s)\,\mathbf{t}(s) + \omega_1'(s)\,\mathbf{m}_1(s) + \omega_2'(s)\,\mathbf{m}_2(s) \\
&\quad \boldsymbol{\Omega}_{\mathrm{M}}(s) \times \boldsymbol{\Omega}_{\mathrm{M}}(s) && \text{from eq. 3.33} \\
&= \tau'(s)\,\mathbf{t}(s) + \omega_1'(s)\,\mathbf{m}_1(s) + \omega_2'(s)\,\mathbf{m}_2(s)
\end{aligned}
$$

Notice that for each segment $\mathbb{S}_Q$, twist and bending are constant (their derivatives are 0). It follows that $\boldsymbol{\Omega}_{\mathrm{M}}'(s) = \mathbf{0}$ for $s \in \mathbb{S}_Q$, which means that the Darboux vector is constant on each segment. We will use this property to reconstruct the material frame and from it, the centreline, by integrating the appropriate equations.

Notation We start by introducing notation for the length of the Darboux vector on segment $\mathbb{S}_Q$ and a unit vector collinear with it:

$$
\begin{aligned}
\Omega_Q &= \left\|\boldsymbol{\Omega}_{\mathrm{M}}(s)\right\| \\
\vec{\boldsymbol{\Omega}}_Q &= \frac{\boldsymbol{\Omega}_{\mathrm{M}}(s)}{\Omega_Q} \qquad \text{for } s \in \mathbb{S}_Q
\end{aligned}
\tag{3.35}
$$

We also define superscripts $\parallel$ and $\perp$ for the parallel and perpendicular components of the projection of an arbitrary vector $\mathbf{v}$ onto the axis $\vec{\boldsymbol{\Omega}}_Q$:

$$
\begin{aligned}
\mathbf{v}^{\parallel} &= \left(\mathbf{v} \cdot \vec{\boldsymbol{\Omega}}_Q\right) \vec{\boldsymbol{\Omega}}_Q \\
\mathbf{v}^{\perp} &= \mathbf{v} - \mathbf{v}^{\parallel}
\end{aligned}
\tag{3.36}
$$

With these tools in hand, we seek to reconstruct the material frame at any point s on the strand.

We do this integrating equation 3.32 over each segment. Because the Darboux vector constant on the segment, such integration amounts to rotating the material frame around $\vec{\boldsymbol{\Omega}}_Q$ with rotation speed Ω_Q per unit of arc length along the centreline. Let us assume we have already reconstructed the tangent vector on the root-end of segment $\mathbb{S}_Q$: $\mathbf{t}_Q = \mathbf{t}\left(S_Q\right)$. From this, we obtain the tangent at any point $s \in \mathbb{S}_Q$ by rotating around $\vec{\boldsymbol{\Omega}}_Q$ by angle $\Omega_Q\left(s - S_Q\right)$:

$$
\mathbf{t}(s) = \mathbf{t}_Q^{\parallel} + \mathbf{t}_Q^{\perp} \cos\left(\Omega_Q\left(s - S_Q\right)\right) + \vec{\boldsymbol{\Omega}}_Q \times \mathbf{t}_Q^{\perp} \sin\left(\Omega_Q\left(s - S_Q\right)\right) \tag{3.37}
$$

Replacing $\mathbf{t}$ with $\mathbf{m}_1$ and $\mathbf{m}_2$ in the above equation finds the major and minor axis, giving us the complete material frame at each point in the segment. By processing segments sequentially starting at the strand's root, we obtain the material frame of the entire strand.

To reconstruct the centreline on a segment, let us again assume we already have $\mathbf{x}_Q = \mathbf{x}\left(S_Q\right)$. We integrate equation 3.6 to obtain the following:

$$
\begin{aligned}
\mathbf{x}(s) = \quad & \mathbf{x}_Q + \left(s - S_Q\right)\mathbf{t}_Q^{\parallel} \\
& + \frac{\sin\left(\Omega_Q\left(s - S_Q\right)\right)}{\Omega_Q}\mathbf{t}_Q^{\perp} \\
& + \frac{1 - \cos\left(\Omega_Q\left(s - S_Q\right)\right)}{\Omega_Q}\vec{\boldsymbol{\Omega}}_Q \times \mathbf{t}_Q^{\perp}
\end{aligned}
\tag{3.38}
$$

The centreline of the entire strand is again obtained by processing segments sequentially starting from the root.

The equations above are applicable when the segment is a non-degenerate helix, that is, both its twist and bending are non-zero. If $\tau_Q = 0$ and $\boldsymbol{\omega}_Q \neq \mathbf{0}$, the segment is an arc of circle; if $\boldsymbol{\omega}_Q = \mathbf{0}$, the segment is a straight line. In both of these cases, reconstruction amounts to trivial application of formulae for these geometric shapes.

3.4 Adapting the Super-Helix Model

We will now describe how our hair animation principles can be applied to the Super-Helix discretisation of Bertails et al. (2006) which we have presented in the previous section.

Recall from Section 3.1.2 that the cornerstone of our approach is the fact that hair bends over its cross-section major axis only. Due to the fact that generalised coordinates of the Super-Helix discretisation are based on the strand's curvature, this observation can be applied directly. It simply means setting $\omega_1 = 0$ and thus removing it as a free variable.

Our motivation for doing this is twofold. First, thanks to the fact that the application of our approach to the Super-Helix model is so direct, we can easily measure the effect on computation speed, simulation stability, and perceived realism with minimum potential for distortion. In this way, application to Super-Helices can serve as validation of our bending model. Second, applying it to multiple rod discretisation approaches (we will build on an entirely different model in Section 3.6) proves that our approach applies directly to the theoretical rod model and is independent of any particular discretisation.

At the same time, the Super-Helix model is quite specific in its highly implicit nature, and other hair-specific observations we have made in Section 3.2 would not transfer to it nearly as easily. We therefore only apply the basic bending model to Super-Helices; the other components of our model are only applied to the explicit discretisation introduced in Section 3.5.

3.4.1 Simplified Model

Our model builds on the fact that hair tends to bend over its major axis only. The Super-Helix model uses bending over the material frame axes directly as generalised coordinates, so we can simply eliminate the one corresponding to bending over the minor axis, and treat it consistently as 0 in all formulae. We have first presented this modification in (Bonanni and Kmoch 2008).

Recall from equation 3.12 that the bending values ω_1 and ω_2 correspond to the rod's curvature vector projected onto the major and minor axis, respectively. Since the curvature vector points in the direction of the centreline bending, its projection onto a cross-section frame axis gives the amount of bending *along* that axis. This is equivalent to bending *over* the other axis. The variable we want to remove is therefore ω_1.

To emphasise the fact that bending over the minor axis is unconditionally 0 in our model and therefore only one bending value can be nonzero, we omit the

index from ω_2 and $\underline{\alpha}_2$, spelling them just ω and $\underline{\alpha}$ in this section. We renumber the generalised coordinates accordingly:

$$\begin{aligned} q_{0,Q} &= \tau_Q \\ q_{1,Q} &= \omega_Q \end{aligned} \tag{3.39}$$

The size of the generalised coordinate vector decreases by 33% from $3N$ to $2N$. The equations of motion, given by equation 3.26 for the full model, become:

$$\begin{aligned} &\left(\forall i \in \{0,1\}\right)\ \left(\forall Q \in \{1,\ldots,N\}\right) \\ &\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{q}_{i,Q}}\right)(\mathbf{q},\dot{\mathbf{q}}) \ - \ \frac{\partial T}{\partial q_{i,Q}}(\mathbf{q},\dot{\mathbf{q}}) \\ &\quad + \ \frac{\partial U}{\partial q_{i,Q}}(\mathbf{q},\dot{\mathbf{q}}) \ + \ \frac{\partial D}{\partial \dot{q}_{i,Q}}(\mathbf{q},\dot{\mathbf{q}}) = \\ &\qquad\qquad = f_{i,Q}(\mathbf{q},\dot{\mathbf{q}}) \end{aligned} \tag{3.40}$$

We next apply the bending simplification to the individual terms of these equations. Where practical, we present both the energy formulation and its required derivative.

Internal energy, introduced in equation 3.30, is simplified as follows:

$$\begin{aligned} U(\mathbf{q}) &= \frac{1}{2}\sum_{Q=1}^{N}\left(\underline{\beta}\left(\tau_Q - \hat{\underline{\tau}}_Q\right)^2 + \underline{\alpha}\left(\omega_Q - \hat{\underline{\omega}}_Q\right)^2\right) \\ \frac{\partial U}{\partial \tau_Q}(\mathbf{q}) &= \underline{\beta}\left(\tau_Q - \hat{\underline{\tau}}_Q\right) \\ \frac{\partial U}{\partial \omega_Q}(\mathbf{q}) &= \underline{\alpha}\left(\omega_Q - \hat{\underline{\omega}}_Q\right) \end{aligned} \tag{3.41}$$

The dissipation potential (equation 3.31 becomes:

$$\begin{aligned} D(\dot{\mathbf{q}}) &= \frac{1}{2}\underline{\gamma}\sum_{Q=1}^{N}\left(\dot{\tau}_Q^2 + \dot{\omega}_Q^2\right) \\ \frac{\partial U}{\partial \dot{\tau}_Q}(\dot{\mathbf{q}}) &= \underline{\gamma}\dot{\tau}_Q \\ \frac{\partial U}{\partial \dot{\omega}_Q}(\dot{\mathbf{q}}) &= \underline{\gamma}\dot{\omega}_Q \end{aligned} \tag{3.42}$$

The kinetic energy formulation, as presented in equation 3.29, does not change with the coordinate reduction. It still depends on the non-trivial hair shape reconstruction necessary to go from the abstract generalised coordinates to the 3D shape. The reconstruction itself works as presented in Section 3.3.3, except that the Darboux vector of the material frame is orthogonal to the minor axis:

$$\begin{aligned} \mathbf{\Omega}_{\mathrm{M}}(s) &= \tau(s)\,\mathbf{t}(s) + \omega(s)\,\mathbf{m}_1(s) \\ \mathbf{\Omega}'_{\mathrm{M}}(s) &= \tau'(s)\,\mathbf{t}(s) + \omega'(s)\,\mathbf{m}_1(s) \end{aligned} \tag{3.43}$$

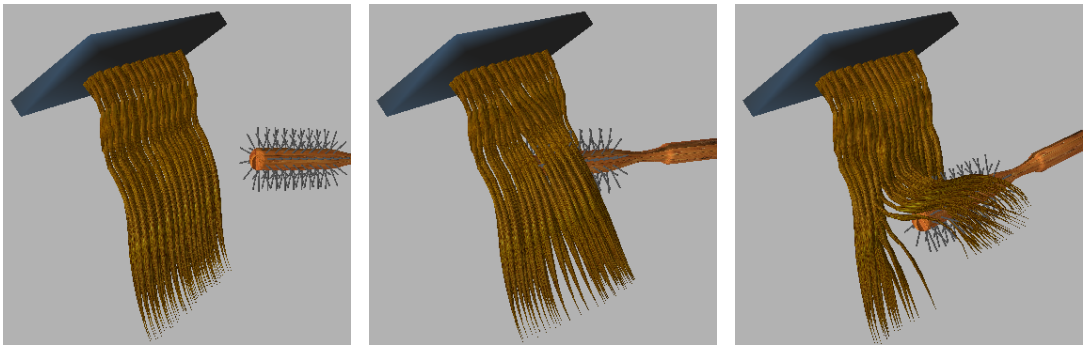


Figure 3.8: Screenshots from haptic interaction with hair simulated by our simplified Super-Helix model.

3.4.2 Integrating with Haptics

In (Bonanni and Kmoch 2008), we have combined our simplified Super-Helix animation method described above with haptic interaction. Our goal was to provide a proof-of-concept implementation of haptic interaction with hair focused on hairstyling operations. In our set-up, the user operates a tool (represented by a brush model, see Figure 3.8) to interact with the hair.

Collisions are detected between the tool and a mesh representation of the hair obtained from the reconstruction process (Section 3.3.3). The first tool–hair collision constrains the tool to stick to the hair until it has *combed* the hair over its whole length or is explicitly detached. If we denote the tool velocity $\dot{\mathbf{h}}$, interaction forces can be computed in the following way:

$$\begin{aligned} \mathbf{F}_{\text{hap}} &= \upsilon \dot{\mathbf{h}}^{\perp} + \chi\left(\underline{\gamma}\right) \dot{\mathbf{h}}^{\parallel}, \quad \text{where} \\ \dot{\mathbf{h}}^{\parallel} &= \left(\dot{\mathbf{h}} \cdot \mathbf{t}\right) \mathbf{t} \\ \dot{\mathbf{h}}^{\perp} &= \dot{\mathbf{h}} - \dot{\mathbf{h}}^{\parallel} \end{aligned} \tag{3.44}$$

Here, $\parallel$ and $\perp$ denote projection parallel/perpendicular to the hair's tangent at point of contact.

The perpendicular component $\upsilon \dot{\mathbf{h}}^{\perp}$ captures the tool moving against the strand and thus bending it. υ is a scaling factor applied to the hair bending stiffness to account for the force range of the haptic device and the desired perceived rigidity.

The parallel component $\chi\left(\underline{\gamma}\right) \dot{\mathbf{h}}^{\parallel}$ applies when the tool moves along the strand, brushing it. The factor χ captures the resistance felt when combing, and it is a function of the hair's dissipation coefficient $\underline{\gamma}$.

We tested the described system with a Force Dimension Omega haptic device. We implemented the haptic rendering based on the library CHAI3D by Conti et al. (2005). This allows the implementation to support most commercial haptic devices based on impedance control.

The haptic model was further developed by Bonanni et al. (2009b) and Bonanni (2010).

Figure 3.9: Screenshots from hair reacting to wind, animated by our simplified Super-Helix model.

3.4.3 Evaluation

We have successfully applied the simplified model to a hair animation system. Figure 3.9 shows several screenshots from an animation of hair in wind, simulated using our simplified Super-Helices.

Our motivation for this was two-fold. First, to validate our idea of simplifying hair animation methods by explicitly prohibiting bending over the hair's minor axis. The Super-Helix model fit this purpose perfectly, as bending over cross-section axes forms an explicit degree of freedom of the model and bending over the minor axis is therefore straightforward to eliminate. Our second motivation was to demonstrate that the principle can be applied independent of our other observations (such as flat wisp formation).

We consider both of these goals achieved, as the simplified model was successfully used for virtual hairstyling, providing sufficient update rate for haptic interaction (Bonanni et al. 2009b; Bonanni et al. 2009a). However, the implementation also exposed certain shortcomings of an implicitly-parametrised model such as Super-Helices. The need for an expensive reconstruction process to recover the shape of the simulated hair is chief among them. It poses a challenge especially for collision handling, which would be greatly facilitated by a more direct correspondence between the dynamic variables and 3D representation of the simulated hair. An obvious way in which an explicit representation would be useful is collision detection; with an implicit model, reconstruction has to be performed to detect collisions even if simulation runs in a faster loop than rendering. However, this is largely an issue of efficiency.

The aspect of collision handling which is actually hindered more by the implicit representation is collision response. Because of the complex and indirect relationship between simulation variables and hair shape, the only way in which collisions can realistically affect hair behaviour is through forces. The original method of Bertails et al. (2006) also handles collision response exclusively through forces: penalty and friction forces for hair–head collisions and anisotropic friction forces for hair–hair collisions. However, force-based methods are not always the best collision response approach, as they can introduce instability issues.

The ability to explore a wider range of options for collision response in hair is one of the reasons we chose an explicit representation as the next base model to which to apply our approach. We describe the base model and our method built on top of it in the remainder of this work.

3.5 Explicit Rod Model

As we've stated in Section 3.3.3, the Super-Helix discretisation uses an *implicit* representation of hair, capturing the entire configuration of the strand with relatively few generalised coordinates. Non-trivial effort is required to reconstruct the actual shape of the strand. While this allows the dynamic equations solved to be small (as the number of variables is limited), it poses a problem for other parts of the simulation which must process the 3D shape of the strand. The most important of these is collision detection.

Other rod-based simulation methods exist which take a more explicit approach. We will describe one such method introduced by Bergou et al. (2008), called *Discrete Elastic Rods*. We start by presenting the method in this section as formulated by its authors; further on, we will apply our hair modelling principles to it, yielding a method with better performance and realism.

The Discrete Elastic Rod method rests on two core principles. The first of these is special representation and handling of twist; we will elaborate on this in sections 3.5.1 and 3.5.3.

The second principle is that major sources of stiffness, such as the rod's inextensibility, are expressed as constraints and removed from the equations of motion. A separate constraint-enforcing step then follows each integration step. This improves stability of the simulation while allowing use of a fast-to-compute explicit integration method. We will present details of this set-up in Section 3.5.4.

We chose to base our work on this method because both of these principles align well with our approach. The method's representation and handling of twist lends itself extremely well to simplification based on our observations of real hair behaviour (strong preference of major axis bending). Low equation stiffness allows for stable simulation even when using single-precision arithmetic, which is important for an efficient GPU implementation. Furthermore, it turns out most of the computations involved in integration and computing forces are easily parallelisable.

3.5.1 Reduced-coordinate Material Frame Representation

Recall from Section 3.1.1 that in the general case, for each point s on the strand, we have 9 variables which vary with time: the components of the 3-dimensional vectors $\mathbf{x}(s)$, $\mathbf{m}_1(s)$, and $\mathbf{m}_2(s)$. However, these variables are not independent: the vectors $\mathbf{m}_1(s)$ and $\mathbf{m}_2(s)$ are orthonormal, so their components are closely coupled. Bergou et al. (2008) seek to expose this coupling by reducing the number of variables. To this end, they turn to differential geometry and the concept of the *Bishop frame*.

The Bishop frame $\mathcal{F}(s) = \{\mathbf{t}(s), \mathbf{u}_1(s), \mathbf{u}_2(s)\}$ is an adapted orthonormal frame with no twist. That is, the following holds:

$$(\forall s) \qquad \mathbf{u}_1'(s) \cdot \mathbf{u}_2(s) = -\mathbf{u}_2'(s) \cdot \mathbf{u}_1(s) = 0 \tag{3.45}$$

Because of this, specifying $\mathbf{u}_1(s_0)$ and $\mathbf{u}_2(s_0)$ for any one point s_0 is enough to define the entire frame for every point s on the rod. The most convenient place is to specify it at the root ($s = 0$), where it can be made identical to the material frame. We will now show how the frame is defined at the entire length of the rod, based on this initial specification.

Let us use $\boldsymbol{\Omega}_{\mathrm{B}}$ to denote the Darboux vector of the Bishop frame; it has the following properties:

$$\begin{aligned} \mathbf{t}'(s) &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \times \mathbf{t}(s) \\ \mathbf{u}_1'(s) &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \times \mathbf{u}_1(s) \\ \mathbf{u}_2'(s) &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \times \mathbf{u}_2(s) \end{aligned} \tag{3.46}$$

We will prove that $\boldsymbol{\Omega}_{\mathrm{B}}(s)$ is orthogonal to $\mathbf{t}(s)$. From equation 3.46:

$$\begin{aligned} \mathbf{u}_1'(s) &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \times \mathbf{u}_1(s) \\ \mathbf{u}_1'(s) \cdot \mathbf{u}_2(s) &= \big(\boldsymbol{\Omega}_{\mathrm{B}}(s) \times \mathbf{u}_1(s)\big) \cdot \mathbf{u}_2(s) && \text{multiplied by } \mathbf{u}_2 \\ 0 &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \cdot \big(\mathbf{u}_1(s) \times \mathbf{u}_2(s)\big) && \text{from equation 3.45} \end{aligned} \tag{3.47}$$

The vector $\big(\mathbf{u}_1(s) \times \mathbf{u}_2(s)\big)$ is collinear with $\mathbf{t}(s)$, which means $\boldsymbol{\Omega}_{\mathrm{B}}(s)$ is orthogonal to $\mathbf{t}(s)$.

From equation 3.46, $\boldsymbol{\Omega}_{\mathrm{B}}(s)$ is also orthogonal to $\mathbf{t}'(s) = \mathbf{n}(s)$, making it collinear with the centreline's curvature binormal $\kappa\mathbf{b}(s)$. In fact, we can show that $\boldsymbol{\Omega}_{\mathrm{B}}(s) = \kappa\mathbf{b}(s)$. We substitute equation 3.7 into equation 3.46 and proceed as follows:

$$\begin{aligned} \boldsymbol{\kappa}(s) &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \times \mathbf{t}(s) \\ \boldsymbol{\kappa}(s) \cdot \boldsymbol{\kappa}(s) &= \boldsymbol{\kappa}(s) \cdot \big(\boldsymbol{\Omega}_{\mathrm{B}}(s) \times \mathbf{t}(s)\big) && \text{multiplied by } \boldsymbol{\kappa}(s) \\ \big(\kappa(s)\big)^2 &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \cdot \big(\mathbf{t}(s) \times \boldsymbol{\kappa}(s)\big) && \text{by triple product rules} \\ \big(\kappa(s)\big)^2 &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \cdot \kappa\mathbf{b}(s) && \text{from equation 3.11} \\ \kappa(s) &= \boldsymbol{\Omega}_{\mathrm{B}}(s) \cdot \mathbf{b}(s) && \text{divided by } \kappa(s) \\ \kappa\mathbf{b}(s) &= \boldsymbol{\Omega}_{\mathrm{B}}(s) && \text{multiplied by } \mathbf{b}(s) \end{aligned} \tag{3.48}$$

If the curvature κ is 0, the rod is unbent and therefore the Bishop frame will not vary. Hence the Bishop frame derivatives are $\mathbf{0}$ and so again $\boldsymbol{\Omega}_{\mathrm{B}} = \mathbf{0} = \kappa\mathbf{b}$.

Equation 3.48 allows us to express the rod's bending (equation 3.13) in terms of the curvature binormal:

$$\begin{aligned} \boldsymbol{\omega}(s) &= \begin{pmatrix} \mathbf{t}'(s) \cdot \mathbf{m}_1(s) \\ \mathbf{t}'(s) \cdot \mathbf{m}_2(s) \end{pmatrix} \\ &= \begin{pmatrix} \big(\kappa\mathbf{b}(s) \times \mathbf{t}(s)\big) \cdot \mathbf{m}_1(s) \\ \big(\kappa\mathbf{b}(s) \times \mathbf{t}(s)\big) \cdot \mathbf{m}_2(s) \end{pmatrix} && \text{by eq. 3.46 \& 3.48} \\ &= \begin{pmatrix} \kappa\mathbf{b}(s) \cdot \big(\mathbf{t}(s) \times \mathbf{m}_1(s)\big) \\ \kappa\mathbf{b}(s) \cdot \big(\mathbf{t}(s) \times \mathbf{m}_2(s)\big) \end{pmatrix} && \text{by triple product rules} \\ &= \begin{pmatrix} \kappa\mathbf{b}(s) \cdot \mathbf{m}_2(s) \\ -\kappa\mathbf{b}(s) \cdot \mathbf{m}_1(s) \end{pmatrix} && \text{right-handed frame} \end{aligned} \tag{3.49}$$

Next, we follow Bergou et al. (2008) in using $\boldsymbol{\Omega}_{\mathrm{B}}$ to define *parallel transport* as the process of transporting a vector $\mathbf{z}$ from one point on the centreline to another by integrating the following equation:

$$\mathbf{z}'(s) = \boldsymbol{\Omega}_{\mathrm{B}}(s) \times \mathbf{z}(s) = \kappa\mathbf{b}(s) \times \mathbf{z}(s) \tag{3.50}$$

Notice that parallel transport is essentially a rotation about the binormal (or curvature binormal). Because it is based on the Bishop frame, applying parallel transport to a vector causes it to proceed along the centreline in a twist-free manner. Equation 3.46 shows that the axes of the Bishop frame evolve using parallel transport. Parallel transport is therefore the process which defines the entire Bishop frame based on its assignment at the root.

Bergou et al. (2008) note that having such a uniquely-defined frame allows for a very efficient parametrisation of the material frame (Langer and Singer 1996). At every point on the rod, we can express the material frame using a single scalar $\theta(s)$: the angle between the material frame and the Bishop frame, understood as rotation around the tangent.

$$\mathbf{m}_1(s) = \cos\theta(s)\ \mathbf{u}_1(s) + \sin\theta(s)\ \mathbf{u}_2(s) \tag{3.51}$$
$$\mathbf{m}_2(s) = -\sin\theta(s)\ \mathbf{u}_1(s) + \cos\theta(s)\ \mathbf{u}_2(s) \tag{3.52}$$

Bergou et al. (2008) make a key observation that the rod's twist can be expressed using θ like this:

$$\tau(s) = \theta'(s) \tag{3.53}$$

We proceed to prove this claim. To keep the notation brief, we omit the dependency on s.

$$\begin{aligned}\mathbf{m}_1' &= (\cos\theta\mathbf{u}_1 + \sin\theta\mathbf{u}_2)' \\ &= -\theta'\sin\theta\mathbf{u}_1 + \cos\theta\mathbf{u}_1' + \theta'\cos\theta\mathbf{u}_2 + \sin\theta\mathbf{u}_2'\end{aligned} \tag{3.54}$$

By substituting equation 3.54 and equation 3.52 into the definition of twist in equation 3.12, we get:

$$\begin{aligned}\tau &= \mathbf{m}_1' \cdot \mathbf{m}_2 \\ &= \left(-\theta'\sin\theta\mathbf{u}_1 + \cos\theta\mathbf{u}_1' + \theta'\cos\theta\mathbf{u}_2 + \sin\theta\mathbf{u}_2'\right) \cdot (-\sin\theta\mathbf{u}_1 + \cos\theta\mathbf{u}_2) \\ &= \theta'\sin^2\theta\mathbf{u}_1\cdot\mathbf{u}_1 - \sin\theta\cos\theta\mathbf{u}_1'\cdot\mathbf{u}_2 - \theta'\sin\theta\cos\theta\mathbf{u}_1\cdot\mathbf{u}_2 - \sin^2\theta\mathbf{u}_1\cdot\mathbf{u}_2' - \\ &\quad \theta'\sin\theta\cos\theta\mathbf{u}_1\cdot\mathbf{u}_2 + \cos^2\theta\mathbf{u}_1'\cdot\mathbf{u}_2 + \theta'\cos^2\theta\mathbf{u}_2\cdot\mathbf{u}_2 + \sin\theta\cos\theta\mathbf{u}_2\cdot\mathbf{u}_2'\end{aligned} \tag{3.55}$$

Equation 3.46 tells us that $\mathbf{u}_1'$ is orthogonal to $\mathbf{u}_1$, and similarly $\mathbf{u}_2'$ is orthogonal to $\mathbf{u}_2$. $\mathbf{u}_1$ and $\mathbf{u}_2$ are orthogonal as well, and each of them is a unit vector. Using these properties and equation 3.45, we can simplify equation 3.55 as follows:

$$\tau = \theta'\sin^2\theta + \theta'\cos^2\theta = \theta' \tag{3.56}$$

Which proves equation 3.53

This way, the problem has been re-stated using only 4 time-dependent variables: the components of the 3-dimensional vector $\mathbf{x}(s)$ and the material frame angle $\theta(s)$. The twisting component of the rod's internal energy, originally given in equation 3.20, can therefore equivalently be expressed as follows:

$$U_{\text{twist}}(\Gamma) = \frac{1}{2}\int_0^{\underline{L}} \underline{\beta}(s)\left(\theta'(s)\right)^2 \mathrm{d}s \tag{3.57}$$

3.5.2 Node+Edge Discretisation

We now follow Bergou et al. (2008) in discretising the rod model using concepts introduced in Section 3.5.1. We choose a set of $(N+2)$ points S_i in the rod's arc $[0, \underline{L}]$ such that $0 = S_0 < S_1 < \ldots < S_{N+1} = L$. The centreline is then discretised into a set of $(N+2)$ *vertices* $\mathbf{x}_i = \mathbf{x}(S_i)$. These vertices are connected by $(N+1)$ line segment *edges* $\mathbf{e}^0, \mathbf{e}^1, \ldots, \mathbf{e}^N$, where $\mathbf{e}^i = \mathbf{x}_{i+1} - \mathbf{x}_i$. Notice the use of different index placement for vertices and edges. This notation is used consistently, using lower indices for vertex-related quantities and upper indices for edge-related ones.

We need to assign a material frame to this discretised curve. Notice that there is a clear definition of the tangent of an edge: a vector collinear with the edge itself. On the other hand, trying to assign a tangent direction to a vertex would be ambiguous. An orthonormal material frame $\mathcal{M}^i = \left\{\mathbf{t}^i, \mathbf{m}_1^i, \mathbf{m}_2^i\right\}$ is therefore assigned to each edge $\mathbf{e}^i$. The frame is adapted to the curve:

$$\mathbf{t}^i = \frac{\mathbf{e}^i}{\|\mathbf{e}^i\|} \tag{3.58}$$

The vectors $\mathbf{m}_1^i$ and $\mathbf{m}_2^i$ represent the major and minor axis of the cross section, respectively.

Recall from Section 3.5.1 that we can represent the material frame using scalar rotation of the Bishop frame. We can do this in the discrete setting as well by introducing a discrete version of the Bishop frame: $\mathcal{F}^i = \left\{\mathbf{t}^i, \mathbf{u}_1^i, \mathbf{u}_2^i\right\}$. Analogously to the continuous case, we then define θ^i as the angle between the material frame and the Bishop frame on edge i:

$$\mathbf{m}_1^i = \cos\theta^i\, \mathbf{u}_1^i + \sin\theta^i\, \mathbf{u}_2^i \tag{3.59}$$

$$\mathbf{m}_2^i = -\sin\theta^i\, \mathbf{u}_1^i + \cos\theta^i\, \mathbf{u}_2^i \tag{3.60}$$

Just like in the continuous case, we want to use parallel transport to define the Bishop frame on edges $1, 2, \ldots N$ based on its assignment on edge 0. As stated previously, parallel transport corresponds to a rotation about the binormal/curvature binormal. In the continuous setting, the curvature binormal at a point is simply an appropriately scaled binormal. In the discrete setting, this is no longer the case. The reason is that the "ordinary" binormal, $\mathbf{b}^i$, is naturally assigned to edge i, while the curvature binormal $(\kappa\mathbf{b})_j$ describes the rod's bending and is therefore better assigned to node j. As we want to use parallel transport to transform a frame assigned to edge $(j-1)$ to one assigned to edge j, it makes sense to assign parallel transport to the node j between these edges. A vector assigned to node j is therefore ideal as an axis of this transformation, more precisely rotation. With this in mind, we will consider discrete parallel transport as a rotation around the curvature binormal.

When $\mathbf{t}^{j-1}$ and $\mathbf{t}^j$ are not collinear (i.e. $(\kappa\mathbf{b})_j$ is nonzero), they span the osculating plane of the centreline at node j. The curvature binormal is then orthogonal to this osculating plane. As such, it must be collinear with $\mathbf{t}^{j-1} \times \mathbf{t}^j$. We therefore define discrete parallel transport as a set of rotation matrices $\{\mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_N\}$ such that:

$$\mathrm{P}_i \left(\mathbf{t}^{j-1} \times \mathbf{t}^j\right) = \mathbf{t}^{j-1} \times \mathbf{t}^j \tag{3.61}$$

$$\mathrm{P}_i\, \mathbf{t}^{i-1} = \mathbf{t}^i \tag{3.62}$$

Equation 3.61 defines the axis of rotation, while equation 3.62 defines the angle. This definition cannot be used when $\mathbf{t}^{i-1}$ and $\mathbf{t}^i$ are collinear; in such case, we define P_i as the identity matrix when $\mathbf{t}^i = \mathbf{t}^{i-1}$, and consider it undefined when $\mathbf{t}^i = -\mathbf{t}^{i-1}$. Notice the latter case implies a 180° turn in one point of the centreline, something which the underlying physical simulation is not supposed to allow.

With discrete parallel transport in place, we can define the discrete Bishop frame for all edges iteratively:

$$\begin{aligned} \mathbf{u}_1^i &= \mathrm{P}_i\, \mathbf{u}_1^{i-1} \\ \mathbf{u}_2^i &= \mathbf{t}^i \times \mathbf{u}_1^i \end{aligned} \tag{3.63}$$

$\mathbf{u}_1^0$ can be defined arbitrarily; we generally set it equal to $\mathbf{m}_1^0$ at the start of the simulation.

Pointwise & Integrated Quantities

In the continuous setting, quantities are represented as functions defined over the interval $[0, \underline{L}]$, that is, the rod's length. Energy is then expressed by integrating the appropriate quantities along that interval.

In the discrete case, quantities are defined at discrete points. Some of these simply represent the value of the continuous function at that point (such as $\mathbf{x}_i = \mathbf{x}(S_i)$). However, other discrete quantities already represent an integrated value. We call such quantities *integrated,* as opposed to the former, *pointwise* ones.

An integrated quantity assigns a value to a domain $\mathbb{D} \subseteq [0, \underline{L}]$, representing an integral of a function over that domain. An integrated quantity can be converted to a pointwise one by dividing its value by $|\mathbb{D}|$.

In the discrete case, for an integrated quantity occurring at node $\mathbf{x}_i$, the domain $\mathbb{D}_i$ consists of the closer halves of edges $\mathbf{e}^{i-1}$ and $\mathbf{e}^i$:

$$\mathbb{D}_i = \left[\frac{1}{2}\left(S_{i-1} + S_i\right), \frac{1}{2}\left(S_i + S_{i+1}\right)\right] \tag{3.64}$$

If we define $l_i = \left|\mathbf{e}^{i-1}\right| + \left|\mathbf{e}^i\right|$, then the length of this domain is $|\mathbb{D}_i| = l_i/2$.

3.5.3 Explicit Equations of Motion

Our goal for dynamic animation is to express the equations of motion, which capture how the system reacts to internal and external forces. Recall that we describe the system using $3(N+2) + (N+1)$ time-varying quantities: the first $N+2$ are 3-dimensional vectors $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N+2}$ describing the centreline position, the remaining $N+1$ are scalars $\theta^0, \theta^1, \ldots, \theta^N$ describing the orientation of the material frame.

Again, Lagrangian mechanics are used to obtain the equations of motion. We use only the node positions $\mathbf{x}$ and velocities $\dot{\mathbf{x}}$ as generalised coordinates. θ is not included; we treat it as a function of $\mathbf{x}$ instead.

We start from equation 1.9; by substituting our generalised coordinates, we obtain the following as the equation of motion for one node $\mathbf{x}_i$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}T}{\mathrm{d}\dot{\mathbf{x}}_i}(\mathbf{x}, \dot{\mathbf{x}}, \theta) - \frac{\mathrm{d}U}{\mathrm{d}\dot{\mathbf{x}}_i}(\mathbf{x}, \dot{\mathbf{x}}, \theta)\right) - \frac{\mathrm{d}T}{\mathrm{d}\mathbf{x}_i}(\mathbf{x}, \dot{\mathbf{x}}, \theta) + \frac{\mathrm{d}U}{\mathrm{d}\mathbf{x}_i}(\mathbf{x}, \dot{\mathbf{x}}, \theta) = \mathbf{F}_i \tag{3.65}$$

$\mathbf{F}_i$ is the external force acting on node i; external forces include effects such as gravity and air drag, or interaction with objects in the scene. Kinetic energy is defined in the classical way, mass times square of velocity:

$$T\left(\mathbf{x}, \dot{\mathbf{x}}, \theta\right) = \frac{1}{2}\dot{\mathbf{x}}^{\mathrm{T}}\, \underline{\mathrm{M}}\, \dot{\mathbf{x}} \tag{3.66}$$

Notice that it does not depend on $\mathbf{x}$. From studying equations 3.20 and 3.21, it seems that internal energy U does not depend on $\dot{\mathbf{x}}$. We will prove this later when we show the exact formula for U, but let us simply assume it for now. By substituting equation 3.66 into equation 3.65 and applying this assumption, we simplify the equations of motion considerably:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}}{\mathrm{d}\dot{\mathbf{x}}_i}\left(\frac{1}{2}\dot{\mathbf{x}}^{\mathrm{T}}\, \underline{\mathrm{M}}\, \dot{\mathbf{x}}\right) &= -\frac{\mathrm{d}U}{\mathrm{d}\mathbf{x}_i}\left(\mathbf{x}, \theta\right) + \mathbf{F}_i \\ \underline{M}_i\ddot{\mathbf{x}}_i &= -\frac{\partial U}{\partial \mathbf{x}_i}\left(\mathbf{x}, \theta\right) - \sum_{j=0}^{N}\frac{\partial U}{\partial \theta^j}\left(\mathbf{x}, \theta\right)\frac{\partial \theta^j}{\partial \mathbf{x}_i} + \mathbf{F}_i \end{aligned} \tag{3.67}$$

Notice that the total derivative $\frac{\mathrm{d}U}{\mathrm{d}\mathbf{x}_i}$ captures both explicit and implicit (through θ) dependence on the centreline position, while the partial derivative $\frac{\partial U}{\partial \mathbf{x}_i}$ only reflects the explicit one.

Notation We introduce the following notation to simplify equations where a gradient of a quantity z with respect to the rod parameters is involved:

$$\begin{aligned} \nabla_i z &= \frac{\partial z}{\partial \mathbf{x}_i} \\ \nabla^j z &= \frac{\partial z}{\partial \theta^j} \end{aligned} \tag{3.68}$$

We rewrite the equations of motion in this more compact form:

$$\underline{M}_i\ddot{\mathbf{x}}_i = -\nabla_i U\left(\mathbf{x}, \theta\right) - \sum_{j=0}^{N}\nabla^j U\left(\mathbf{x}, \theta\right)\nabla_i\theta^j + \mathbf{F}_i \tag{3.69}$$

We will now proceed to express the constituent terms of these equations.

Discrete Curvature Binormal

The first step will be to find the formula for the curvature binormal in the discrete model. Bergou et al. (2008) turn to differential geometry and the concept of *holonomy* for it. When parallel transporting a frame around a closed loop, holonomy of the loop measures the difference in the frame's orientation at the start and end. It is therefore a scalar quantity—the angle of rotation (around the tangent) between the initial frame and the same frame parallel transported around the loop once.

In this section, we consider how the quantities we're interested in change with slight variation of the centreline. We therefore view them as functions of

one variation-controlling parameter, where an argument value of 0 represents the original, unvaried state. We take this variation to be arbitrary, but small. We will later use the formulations discovered here for computing derivatives.

We take two consecutive edge tangent vectors $\mathbf{t}^{i-1}(0)$ and $\mathbf{t}^{i}(0)$ and their corresponding varied state $\mathbf{t}^{i-1}(\varepsilon)$ and $\mathbf{t}^{i}(\varepsilon)$. Parallel transport $\mathrm{P}_i(0)$ transports a frame from unvaried edge $i-1$ to edge i, and analogously $\mathrm{P}_i(\varepsilon)$ transports along the varied edges. Finally, there is also parallel transport *along the variation*—a rotation $\tilde{\mathrm{P}}^{i}(\varepsilon)$ such that:

$$\tilde{\mathrm{P}}^{i}(\varepsilon)\ \mathbf{t}^{i}(0) = \mathbf{t}^{i}(\varepsilon) \tag{3.70}$$

$$\tilde{\mathrm{P}}^{i}(\varepsilon)\ \Big(\mathbf{t}^{i}(0) \times \mathbf{t}^{i}(\varepsilon)\Big) = \mathbf{t}^{i}(0) \times \mathbf{t}^{i}(\varepsilon) \tag{3.71}$$

Next, we will form a closed loop by following this sequence of transformations: $\mathbf{t}^{i-1}(0) \to \mathbf{t}^{i}(0) \rightsquigarrow \mathbf{t}^{i}(\varepsilon) \to \mathbf{t}^{i-1}(\varepsilon) \rightsquigarrow \mathbf{t}^{i-1}(0)$. Straight arrows correspond to moving along the rod, while wavy arrows represent variation. The whole sequence can be expressed in matrix form like this:

$$\mathrm{R}^{i-1}(\varepsilon) = \Big(\tilde{\mathrm{P}}^{i-1}(\varepsilon)\Big)^{\mathrm{T}}\ \Big(\mathrm{P}_i(\varepsilon)\Big)^{\mathrm{T}}\ \tilde{\mathrm{P}}^{i}(\varepsilon)\ \mathrm{P}_i(0) \tag{3.72}$$

From the definitions of the constituent matrices, it follows that $\mathrm{R}^{i-1}(\varepsilon)\ \mathbf{t}^{i-1}(0) = \mathbf{t}^{i-1}(0)$, so $\mathrm{R}^{i-1}(\varepsilon)$ is just a rotation around the axis $\mathbf{t}^{i-1}(0)$. We denote the angle of this rotation $\psi_i(\varepsilon)$. This angle is in fact the holonomy of the closed loop of parallel transports which make up $\mathrm{R}^{i-1}(\varepsilon)$.

In the equations of motion, we will need the gradient of ψ_i for all $i = 1, \ldots, N$. Building on the work of Vries (2005), Bergou et al. (2008) arrive at the following formula for the variation of ψ_i for a centreline variation $\delta\mathbf{x}$:

$$\delta\psi_i = \frac{-2\mathbf{t}^{i-1} \times \mathbf{t}^{i}}{1 + \mathbf{t}^{i-1} \cdot \mathbf{t}^{i}} \left(\frac{1}{2}\frac{\delta\mathbf{x}_i - \delta\mathbf{x}_{i-1}}{\|\mathbf{e}^{i-1}\|} + \frac{1}{2}\frac{\delta\mathbf{x}_{i+1} - \delta\mathbf{x}_i}{\|\mathbf{e}^{i}\|}\right) \tag{3.73}$$

The continuous equivalent is as follows:

$$\delta\left(\int_0^{\underline{L}} \psi(s)\,\mathrm{d}s\right) = -\int_0^{\underline{L}} \kappa\mathbf{b}(s) \cdot \frac{\partial\delta\mathbf{x}}{\partial s}(s)\,\mathrm{d}s \tag{3.74}$$

Comparing equation 3.73 and equation 3.74 shows general symmetry between them—the second factor in equation 3.73 is a finite-difference approximation of the corresponding factor in equation 3.74. It is therefore natural to use the first factor as the discrete equivalent of the integrated curvature binormal. Notice that this makes discrete curvature binormal an integrated quantity.

We can re-write the formula using edge vectors instead of tangent vectors like this:

$$(\kappa\mathbf{b})_i = \frac{2\mathbf{e}^{i-1} \times \mathbf{e}^{i}}{\|\mathbf{e}^{i-1}\|\|\mathbf{e}^{i}\| + \mathbf{e}^{i-1} \cdot \mathbf{e}^{i}} \tag{3.75}$$

Bishop Frame Evolution

Our next goal is to describe the variation of the Bishop frame with respect to changes of the centreline. By the same reasoning as above, the change will take

form of a rotation of the frame around the tangent. ψ_i is the angle needed to align the result of parallel transport $\tilde{\mathrm{P}}^i(\varepsilon)\,\mathrm{P}_i(0)$ to the result of parallel transport $\mathrm{P}_i(\varepsilon)\,\tilde{\mathrm{P}}^{i-1}(\varepsilon)$. Since the Bishop frame evolves under parallel transport, it is also the angle between $\mathrm{P}_i(\varepsilon)\,\tilde{\mathrm{P}}^{i-1}(\varepsilon)\,\mathcal{F}^{i-1}(0)$ and $\tilde{\mathrm{P}}^i(\varepsilon)\,\mathcal{F}^i(0)$ (which is $\tilde{\mathrm{P}}^i(\varepsilon)\,\mathrm{P}_i(0)\,\mathcal{F}^{i-1}(0)$).

Let us denote Ψ^i the angle between the result of $\mathrm{P}_i(\varepsilon)\cdots\mathrm{P}_1(\varepsilon)\,\tilde{\mathrm{P}}^0(\varepsilon)\,\mathcal{F}^0(0) = \mathcal{F}^i(\varepsilon)$ and $\tilde{\mathrm{P}}^i(\varepsilon)\,\mathrm{P}_i(0)\cdots\mathrm{P}_1(0)\,\mathcal{F}^0(0) = \tilde{\mathrm{P}}^i(\varepsilon)\,\mathcal{F}^i(0)$. Iterative application of the above rule leads to this formula:

$$\Psi^i = \sum_{j=1}^{i} \psi_j \tag{3.76}$$

What we need for computing forces in the equations of motion is the gradient with respect to change in the centreline positions. For ψ_i, we substitute equation 3.75 into equation 3.73, perform differentiation and obtain the following:

$$\begin{aligned}
\nabla_{i-1}\psi_i &= \frac{(\kappa\mathbf{b})_i}{2\|\mathbf{e}^{i-1}\|} \\
\nabla_{i+1}\psi_i &= \frac{(\kappa\mathbf{b})_i}{2\|\mathbf{e}^{i}\|} \\
\nabla_i\psi_i &= -\nabla_{i-1}\psi_i - \nabla_{i+1}\psi_i \\
\nabla_j\psi_i &= 0 \quad \text{for } j \notin \{i-1, i, i+1\}
\end{aligned} \tag{3.77}$$

Differentiating equation 3.76 yields:

$$\nabla_i\Psi^j = \sum_{k=1}^{j} \nabla_i\psi_k \tag{3.78}$$

Equation 3.77 guarantees that this sum has no more than three non-zero terms.

Post-integration Update The Bishop frame forms our reference for the material frame orientation θ. We must therefore make sure that the Bishop frame is kept consistent. When a new time step is computed, it is possible that the new tangent $\mathbf{t}^0(t_{n+1})$ is not orthogonal to the Bishop axis $\mathbf{u}_1^0(t_n)$. We must therefore update the Bishop frame at edge 0.

The correct way updating the Bishop frame is via parallel transport. What we need is parallel transport *in time,* which will update $\mathbf{u}_1^0(t_n)$ to $\mathbf{u}_1^0(t_{n+1})$. As we've shown above, parallel transport is a rotation. When $\mathbf{t}^0(t_{n+1}) \neq \mathbf{t}^0(t_n)$, we compute the rotation T required to align these two vectors:

$$\begin{aligned}
\mathrm{axis}(\mathrm{T}) &= \frac{\mathbf{t}^0(t_n) \times \mathbf{t}^0(t_{n+1})}{\left\|\mathbf{t}^0(t_n) \times \mathbf{t}^0(t_{n+1})\right\|} \\
\mathrm{angle}(\mathrm{T}) &= \arccos\left(\mathbf{t}^0(t_n) \cdot \mathbf{t}^0(t_{n+1})\right)
\end{aligned} \tag{3.79}$$

We then apply this rotation to $\mathbf{u}_1^0(t_n)$ and compute the updated Bishop frame of the root segment:

$$\begin{aligned}
\mathbf{u}_1^0(t_{n+1}) &= \mathrm{T}\,\mathbf{u}_1^0(t_n) \\
\mathbf{u}_2^0(t_{n+1}) &= \mathbf{t}^0(t_{n+1}) \times \mathbf{u}_1^0(t_{n+1})
\end{aligned} \tag{3.80}$$

Discrete Internal Energy

With the above apparatus, we can now express equation 3.19 using the discrete model.

We will start with the bending energy U_{bend} and discretise it per node:

$$\begin{aligned} U_{\text{bend}}(\Gamma) &= \frac{1}{2}\int_0^{\underline{L}} \left(\boldsymbol{\omega}(s) - \underline{\hat{\boldsymbol{\omega}}}(s)\right)^{\mathrm{T}} \underline{\mathrm{B}}(s) \left(\boldsymbol{\omega}(s) - \underline{\hat{\boldsymbol{\omega}}}(s)\right) \mathrm{d}s \\ &= \frac{1}{2}\sum_{i=1}^{N} \frac{l_i}{2} \left(U_{\text{bend}}\right)_i \end{aligned} \tag{3.81}$$

$\left(U_{\text{bend}}\right)_i$ is a pointwise quantity representing the bending energy coming from node $\mathbf{x}_i$. Recall from equation 3.49 that the bending vector $\boldsymbol{\omega}$ is the curvature binormal expressed in the material frame (rotated by $\pi/2$ around the tangent). The curvature binormal $\left(\kappa\mathbf{b}\right)_i$ is an integrated quantity over the domain $\mathbb{D}_i$, spanning half each of edge $\mathbf{e}^{i-1}$ and $\mathbf{e}^i$. Let us denote $\boldsymbol{\omega}_i^j$ the bending at node i expressed in the material frame of edge $j \in \{i-1, i\}$:

$$\boldsymbol{\omega}_i^j = \begin{pmatrix} \left(\kappa\mathbf{b}\right)_i \cdot \mathbf{m}_2^j \\ -\left(\kappa\mathbf{b}\right)_i \cdot \mathbf{m}_1^j \end{pmatrix} \tag{3.82}$$

The integrated bending at node i is then:

$$\frac{1}{2}\sum_{j=i-1}^{i} \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right) \tag{3.83}$$

We convert it to a pointwise quantity by dividing each occurrence of $\left(\kappa\mathbf{b}\right)_i$ with $l_i/2$:

$$\left(U_{\text{bend}}\right)_i = \frac{2}{\left(l_i\right)^2}\sum_{j=i-1}^{i} \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right) \tag{3.84}$$

Substituting equation 3.84 into equation 3.81 gives us the final formula for discrete bending energy:

$$U_{\text{bend}}(\Gamma) = \frac{1}{2}\sum_{i=1}^{N} \frac{1}{l_i} \sum_{j=i-1}^{i} \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right) \tag{3.85}$$

For twisting energy U_{twist}, we again start with a discretisation using pointwise per-node twisting energy $\left(U_{\text{twist}}\right)_i$:

$$\begin{aligned} U_{\text{twist}}(\Gamma) &= \frac{1}{2}\int_0^{\underline{L}} \underline{\beta}(s) \left(\tau(s)\right)^2 \mathrm{d}s \\ &= \frac{1}{2}\sum_{i=1}^{N} \frac{l_i}{2} \left(U_{\text{twist}}\right)_i \end{aligned} \tag{3.86}$$

Bergou et al. (2008) do not consider rods with a natural twist, hence there is no $\underline{\hat{\tau}}(s)$ term in the equation. Recall from equation 3.53 that twist $\tau(s) = \theta'(s)$. The discrete equivalent is integrated discrete twist:

$$\tau_i = \theta^i - \theta^{i-1} \tag{3.87}$$

We convert this to a pointwise quantity and substitute into equation 3.81 to obtain the final form of discrete twist energy:

$$U_{\text{twist}}(\Gamma) = \sum_{i=1}^{N} \underline{\beta}_i \frac{(\tau_i)^2}{l_i} = \sum_{i=1}^{N} \underline{\beta}_i \frac{\left(\theta^i - \theta^{i-1}\right)^2}{l_i} \tag{3.88}$$

The discrete form of equation 3.19 is then as follows:

$$\begin{aligned} U(\mathbf{x}, \theta, t) &= \sum_{i=1}^{N} \underline{\beta}_i \frac{\left(\theta^i - \theta^{i-1}\right)^2}{l_i} + \frac{1}{2} \sum_{i=1}^{N} \frac{1}{l_i} \sum_{j=i-1}^{i} \left(\boldsymbol{\omega}_i^j - \hat{\underline{\boldsymbol{\omega}}}_i^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \hat{\underline{\boldsymbol{\omega}}}_i^j\right) \\ &= \sum_{i=1}^{N} \frac{1}{l_i} \left(\underline{\beta}_i \left(\theta^i - \theta^{i-1}\right)^2 + \frac{1}{2} \sum_{j=i-1}^{i} \left(\boldsymbol{\omega}_i^j - \hat{\underline{\boldsymbol{\omega}}}_i^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \hat{\underline{\boldsymbol{\omega}}}_i^j\right) \right) \end{aligned} \tag{3.89}$$

Twist Evolution

Since we've limited ourselves to inextensible and unshearable rods, the equations of motion of the rod describe its bending and twisting. Bergou et al. (2008) note that these two deformation modes propagate through the rod at vastly different time scales.

Twist waves propagate faster in rods with smaller rotational inertia of the cross section. In the limit, as the cross section inertia vanishes, twist propagates instantly—the rod's twist would always depend on its centreline shape only, minimising internal elastic energy (equation 3.19).

The speed of bending wave propagation, on the other hand, is proportional to $\underline{a}_1/\lambda$, where $\underline{a}_1$ represents the cross-section size and λ is the wavelength. Notice that for the motions we want to simulate, $\lambda \gg \underline{a}_1$, so bending waves propagate much slower than twist ones.

We are interested in the dynamics and temporal evolution of the rod's shape, i.e. its bending. Twist waves propagate so fast that unless our simulation time step is extremely short, we can consider their propagation instantaneous and simply remove them from the equations of motion. We therefore assume that the material frame is always oriented such that the rod's internal energy U is minimal for the given centreline positions, with the following implication:

$$\nabla^i U(\mathbf{x}, \theta) = 0 \tag{3.90}$$

Equation 3.90 applies to all segments i on which the rod can twist freely. However, the simulation model allows some segments to be *clamped*. The material frame orientation on a clamped segment j is then an external constraint:

$$(\forall t)\ \ \theta^j = \theta^j_{\text{clamp}} \tag{3.91}$$

Because we're dealing with hair, we generally constrain segment 0, where the hair strand is attached to the scalp.

During simulation, the material frame orientation is updated in a quasistatic fashion in each simulation step, before computing forces, by applying equation 3.90 to all unclamped segments j. Bergou et al. (2008) use Newton minimisation of $U(\mathbf{x}, \theta, t)$ with respect to θ. This requires expressing the gradient and the Hessian of the internal energy.

We start by differentiating equation 3.85. We introduce the following shorthand notation for computing $\nabla^j U_{\text{bend}}$:

$$W_i = \frac{1}{l_i}\left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right) \tag{3.92}$$

This allows us to write:

$$\nabla^j U_{\text{bend}}(\mathbf{x}, \theta) = \frac{1}{2}\left(\nabla^j W_j + \nabla^j W_{j+1}\right) \tag{3.93}$$

Differentiating W_i yields:

$$\nabla^j W_i = \frac{2}{l_i}\left(\nabla^j \boldsymbol{\omega}_i^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right) \tag{3.94}$$

Differentiating equations 3.51 and 3.52 gives us the following:

$$\begin{aligned} \nabla^j \mathbf{m}_1^j &= \mathbf{m}_2^j \\ \nabla^j \mathbf{m}_2^j &= -\mathbf{m}_1^j, \qquad \text{therefore} \\ \nabla^j \boldsymbol{\omega}_i^j &= \underline{\mathrm{J}}\, \boldsymbol{\omega}_i^j \end{aligned} \tag{3.95}$$

where $\underline{\mathrm{J}}$ is counter-clockwise rotation by $\pi/2$:

$$\underline{\mathrm{J}} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{3.96}$$

$$\underline{\mathrm{J}}^{-1} = \underline{\mathrm{J}}^{\mathrm{T}} = -\underline{\mathrm{J}} \tag{3.97}$$

Next, we differentiate equation 3.88, which is straightforward:

$$\nabla^j U_{\text{twist}}(\mathbf{x}, \theta) = 2\left(\underline{\beta}_j \frac{\tau_j}{l_j} - \underline{\beta}_{j+1} \frac{\tau_{j+1}}{l_{j+1}}\right) \tag{3.98}$$

By combining these results, we get the following gradient of internal energy:

$$\begin{aligned} \nabla^j U(\mathbf{x}, \theta) = \quad & \frac{1}{l_j}\left(\boldsymbol{\omega}_j^j\right)^{\mathrm{T}} \underline{\mathrm{J}}\, \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_j^j - \underline{\hat{\boldsymbol{\omega}}}_j^j\right) \\ & + \frac{1}{l_{j+1}}\left(\boldsymbol{\omega}_{j+1}^j\right)^{\mathrm{T}} \underline{\mathrm{J}}\, \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_{j+1}^j - \underline{\hat{\boldsymbol{\omega}}}_{j+1}^j\right) \\ & + 2\left(\underline{\beta}_j \frac{\tau_j}{l_j} - \underline{\beta}_{j+1} \frac{\tau_{j+1}}{l_{j+1}}\right) \end{aligned} \tag{3.99}$$

The next step is to compute the Hessian by differentiating equation 3.99. Notice that $\dfrac{\partial^2 W_i}{\partial \theta^j \partial \theta^k} = 0$ for $j \neq k$. So we differentiate just equation 3.94:

$$\begin{aligned} \frac{\partial^2 W_i}{\partial {\theta^j}^2} &= \frac{2}{l_i}\left(-\underline{\mathrm{J}}\, \boldsymbol{\omega}_i^j\right)^{\mathrm{T}} \underline{\mathrm{J}}\, \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right) + \frac{2}{l_i}\left(\boldsymbol{\omega}_i^j\right)^{\mathrm{T}} \underline{\mathrm{J}}\, \underline{\mathrm{B}}^j \left(-\underline{\mathrm{J}}\, \boldsymbol{\omega}_i^j\right) \\ &= -\frac{2}{l_i}\left(\boldsymbol{\omega}_i^j\right)^{\mathrm{T}} \underline{\mathrm{J}}^{\mathrm{T}}\, \underline{\mathrm{J}}\, \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right) - \frac{2}{l_i}\left(\boldsymbol{\omega}_i^j\right)^{\mathrm{T}} \underline{\mathrm{J}}\, \underline{\mathrm{B}}^j\, \underline{\mathrm{J}}\, \boldsymbol{\omega}_i^j \\ &= \frac{2}{l_i}\left(\boldsymbol{\omega}_i^j\right)^{\mathrm{T}} \underline{\mathrm{J}}^{\mathrm{T}}\, \underline{\mathrm{B}}^j\, \underline{\mathrm{J}}\, \boldsymbol{\omega}_i^j - \frac{2}{l_i}\left(\boldsymbol{\omega}_i^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_i^j - \underline{\hat{\boldsymbol{\omega}}}_i^j\right) \end{aligned} \tag{3.100}$$

The last step follows from appropriate application of equation 3.97.

Again, differentiating equation 3.98 is straightforward. By combining these results, we get the Hessian:

$$\begin{aligned}
\frac{\partial^2 U}{\partial \theta^j \partial \theta^{j-1}} &= -\frac{2\underline{\beta}_j}{l_j} \\
\frac{\partial^2 U}{\partial \theta^j \partial \theta^{j+1}} &= -\frac{2\underline{\beta}_{j+1}}{l_{j+1}} \\
\frac{\partial^2 U}{\partial {\theta^j}^2} &= \quad \frac{1}{l_j}\left(\boldsymbol{\omega}_j^j\right)^{\mathrm{T}} \underline{\mathrm{J}}^{\mathrm{T}}\, \underline{\mathrm{B}}^j\, \underline{\mathrm{J}}\, \boldsymbol{\omega}_j^j - \frac{1}{l_j}\left(\boldsymbol{\omega}_j^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_j^j - \underline{\hat{\boldsymbol{\omega}}}_j^j\right) \\
&\quad + \frac{1}{l_{j+1}}\left(\boldsymbol{\omega}_{j+1}^j\right)^{\mathrm{T}} \underline{\mathrm{J}}^{\mathrm{T}}\, \underline{\mathrm{B}}^j\, \underline{\mathrm{J}}\, \boldsymbol{\omega}_{j+1}^j - \frac{1}{l_{j+1}}\left(\boldsymbol{\omega}_{j+1}^j\right)^{\mathrm{T}} \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_{j+1}^j - \underline{\hat{\boldsymbol{\omega}}}_{j+1}^j\right) \\
&\quad + \frac{2\underline{\beta}_j}{l_j} + \frac{2\underline{\beta}_{j+1}}{l_{j+1}} \\
\frac{\partial^2 U}{\partial \theta^j \partial \theta^k} &= 0 \quad \text{for } k \notin \{j-1, j, j+1\}
\end{aligned} \tag{3.101}$$

Material Frame Gradient

Our eventual goal is to express all components of equation 3.69. One of these is the gradient of θ^j with respect to $\mathbf{x}_i$. Recall what θ^j actually represents: the rotational angle between the material frame and the Bishop frame at edge j. We've established above that varying the centreline by ε rotates the Bishop frame at edge j by Ψ^j; refer to equation 3.76 and the discussion preceding it. Therefore, the same material frame orientation is obtained by subtracting Ψ^j from the varied θ^j. In other words:

$$\nabla_i \theta^j = -\nabla_i \Psi^j \tag{3.102}$$

Internal Energy Gradient

The final part missing from equation 3.69 is the gradient of internal energy U (equations 3.88 and 3.85) with respect to node positions. Notice that equation 3.88 only depends on θ and not on $\mathbf{x}$, which means that:

$$\nabla_i U = \nabla_i U_{\text{bend}} \tag{3.103}$$

Straightforward differentiation of equation 3.85 gives us:

$$\nabla_i U_{\text{bend}}\left(\mathbf{x}, \theta\right) = \sum_{k=1}^{N} \frac{1}{l_k} \sum_{j=k-1}^{k} \nabla_i \boldsymbol{\omega}_k^j\, \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_k^j - \underline{\hat{\boldsymbol{\omega}}}_k^j\right) \tag{3.104}$$

By differentiating $\boldsymbol{\omega}_k^j$ and applying equation 3.102, we obtain the following:

$$\nabla_i \boldsymbol{\omega}_k^j = \begin{pmatrix} \left(\mathbf{m}_2^j\right)^{\mathrm{T}} \\ -\left(\mathbf{m}_1^j\right)^{\mathrm{T}} \end{pmatrix} \nabla_i \left(\kappa \mathbf{b}\right)_k - \underline{\mathrm{J}}\, \boldsymbol{\omega}_k^j \left(\nabla_i \Psi^j\right)^{\mathrm{T}} \tag{3.105}$$

The only missing bit is the gradient of the curvature binormal. We introduce a shorthand notation and re-write equation 3.75:

$$\mathcal{E}^i = \left\|\mathbf{e}^{i-1}\right\|\left\|\mathbf{e}^i\right\| + \mathbf{e}^{i-1}\cdot\mathbf{e}^i \tag{3.106}$$

$$(\kappa\mathbf{b})_i = \frac{2\mathbf{e}^{i-1}\times\mathbf{e}^i}{\mathcal{E}^i} \tag{3.107}$$

We proceed to differentiate equation 3.107:

$$\nabla_{i-1}(\kappa\mathbf{b})_i = \frac{2\nabla_{i-1}\left(\mathbf{e}^{i-1}\times\mathbf{e}^i\right)\mathcal{E}^i}{\left(\mathcal{E}^i\right)^2} - \frac{2\left(\mathbf{e}^{i-1}\times\mathbf{e}^i\right)\nabla_{i-1}\mathcal{E}^i}{\left(\mathcal{E}^i\right)^2} \tag{3.108}$$

$\mathbf{e}^i$ does not depend on $\mathbf{x}_{i-1}$:

$$\nabla_{i-1}(\kappa\mathbf{b})_i = \frac{2\left[\mathbf{e}^i\right]}{\mathcal{E}^i} - \frac{2\left(\mathbf{e}^{i-1}\times\mathbf{e}^i\right)\nabla_{i-1}\mathcal{E}^i}{\left(\mathcal{E}^i\right)^2} \tag{3.109}$$

The notation $[\mathbf{v}]$ denotes a 3×3 matrix such that $[\mathbf{v}]\ \mathbf{z} = \mathbf{v}\times\mathbf{z}$. To compute $\nabla_{i-1}\mathcal{E}^i$, we employ the premise that the rod is inextensible and thus $\left\|\mathbf{e}^{i-1}\right\|$ does not depend on $\mathbf{x}_{i-1}$. This, along with equation 3.75, allows us to reach the following form of the gradient:

$$\begin{aligned}\nabla_{i-1}(\kappa\mathbf{b})_i &= \frac{2\left[\mathbf{e}^i\right]}{\mathcal{E}^i} - \frac{2\left(\mathbf{e}^{i-1}\times\mathbf{e}^i\right)}{\mathcal{E}^i}\frac{\nabla_{i-1}\mathcal{E}^i}{\mathcal{E}^i}\\ &= \frac{2\left[\mathbf{e}^i\right]}{\mathcal{E}^i} + (\kappa\mathbf{b})_i\frac{\left(\mathbf{e}^i\right)^{\mathrm{T}}}{\mathcal{E}^i}\\ &= \frac{2\left[\mathbf{e}^i\right] + (\kappa\mathbf{b})_i\left(\mathbf{e}^i\right)^{\mathrm{T}}}{\|\mathbf{e}^{i-1}\|\|\mathbf{e}^i\| + \mathbf{e}^{i-1}\cdot\mathbf{e}^i}\end{aligned} \tag{3.110}$$

By following analogous steps, we obtain:

$$\nabla_{i+1}(\kappa\mathbf{b})_i = \frac{2\left[\mathbf{e}^{i-1}\right] + (\kappa\mathbf{b})_i\left(\mathbf{e}^{i-1}\right)^{\mathrm{T}}}{\|\mathbf{e}^{i-1}\|\|\mathbf{e}^i\| + \mathbf{e}^{i-1}\cdot\mathbf{e}^i} \tag{3.111}$$

$$\nabla_i(\kappa\mathbf{b})_i = -\nabla_{i-1}(\kappa\mathbf{b})_i - \nabla_{i+1}(\kappa\mathbf{b})_i \tag{3.112}$$

$$\nabla_k(\kappa\mathbf{b})_i = 0 \quad \text{for } k\notin\{i-1,i,i+1\} \tag{3.113}$$

Integration

We now have all the components of the equations of motion (equation 3.69) in place: equations 3.75, 3.77, 3.78, 3.82, 3.89, 3.99, 3.102, 3.103, 3.104, and 3.105. This allows us to apply an integration scheme and evolve the system in time. Following Bergou et al. (2008), we use the symplectic Euler integration method (Hairer et al. 2006).

The symplectic (or semi-implicit) Euler method requires the integrated differential equation to be expressible as a pair of equations in the following form:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = \mathrm{f}(t,b) \tag{3.114}$$

$$\frac{\mathrm{d}b}{\mathrm{d}t} = \mathrm{g}(t,a) \tag{3.115}$$

The integration is then computed as follows:

$$b_{n+1} = b_n + \mathrm{g}\left(t_n, a_n\right) \Delta t \tag{3.116}$$

$$a_{n+1} = a_n + \mathrm{f}\left(t_n, b_{n+1}\right) \Delta t \tag{3.117}$$

We use the classic set-up for integrating acceleration-based equations of motion using a first-order scheme which we've described in Section 1.4.3:

$$\frac{\mathrm{d}\dot{\mathbf{x}}}{\mathrm{d}t} = -\underline{\mathrm{M}}^{-1} \frac{\mathrm{d}U}{\mathrm{d}\mathbf{x}}\left(\mathbf{x}, \theta\right) + \mathbf{F}\left(\mathbf{x}\right) \tag{3.118}$$

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \dot{\mathbf{x}} \tag{3.119}$$

From this formulation, it is clear that our equations conform to the preconditions of the symplectic Euler method, with

$$a = \dot{\mathbf{x}}$$

$$b = \mathbf{x}$$

Recall that thanks to quasistatic treatment of twist, the gradient of internal energy with respect to θ^j is zero for all unclamped edges j. The actual equations of motion for $i = 1, \ldots, N+1$ are therefore as follows:

$$\ddot{\mathbf{x}}_i = -\frac{1}{\underline{M}_i} \nabla_i U_{\text{bend}} + \frac{1}{\underline{M}_i} \sum_{j:\text{clamped}} \nabla^j U \nabla_i \Psi^j \tag{3.120}$$

The energy derivatives $\nabla_i U_{\text{bend}}$ and $\nabla^j U$ are given by equations 3.104 and 3.99, respectively.

3.5.4 Constraint Enforcement

As we've outlined in the introduction to this method, the equations of motion presented above do *not* enforce the rod's inextensibility.

In contrast to the Super-Helix method, inextensibility is not used in the definition of the generalised coordinates we use, so it must be expressed as an explicit *constraint*. Other constraints are also possible: coupling the position of a node and the material frame orientation on the adjoining edge to the position and rotation of a rigid body, which is simulated separately.

Introducing explicit constraints means that we must base our system on equation 1.12. We could proceed by adopting it as our equations of motion and applying normal integration to evolve our system in time. However, hard constraints lead to stiff equations, which must be solved with costly integration methods and/or very small time steps. An alternative approach is to split one step of evolving the dynamic system into two sub-steps: unconstrained integration followed by constraint enforcement.

Bergou et al. (2008) take the two-step approach. We will first present it as a generic mechanism in Lagrangian mechanics, before applying it to our rods.

Lagrangian Mechanics with Explicit Constraint Enforcement

For this generic presentation, we turn back to the notation of Section 1.4.2: our generalised coordinates are the vector $\mathbf{g} \in \mathbb{R}^{\underline{\gamma}}$. We will also assume a direct form of kinetic energy in the generalised coordinates:

$$T(\mathbf{g}, \dot{\mathbf{g}}) = \frac{1}{2} \dot{\mathbf{g}}^{\mathrm{T}} \, \mathrm{N} \, \dot{\mathbf{g}} \tag{3.121}$$

N is a generalised mass matrix.

The first step is to perform integration of the unconstrained equations of motion (equation 1.6). After this integration step, we have the unconstrained coordinates $\breve{\mathbf{g}}$ at time $t_n = t_0 + n\Delta t$, where Δt is the time step of the integration. We apply a constraint enforcement step which seeks to find a configuration $\mathbf{g}$ which will be "similar" to the unconstrained one, but satisfy the constraints. There are multiple ways of enforcing constraints after integration; Bergou et al. (2008) use a manifold projection method of Goldenthal et al. (2007).

Manifold projection is a family of methods (Hairer et al. 2006) which work by projecting the unconstrained solution $\breve{\mathbf{g}}$ onto the constraint manifold $\mathcal{C}$ defined as follows:

$$\mathcal{C} = \left\{ \mathbf{z} : \mathbf{C}(\mathbf{z}) = \mathbf{0} \right\} \tag{3.122}$$

The projection is defined as finding a vector $\delta\mathbf{g}$ such that $\breve{\mathbf{g}} + \delta\mathbf{g} = \mathbf{g} \in \mathcal{C}$. The projection should try to find a point $\mathbf{g}$ on the manifold which is "close" to the unconstrained point $\breve{\mathbf{g}}$. For this, we need to establish a measure of distance. The metric adopted by Goldenthal et al. (2007) is generalised kinetic energy:

$$\mathcal{T}(\delta\mathbf{g}) = (\delta\mathbf{g})^{\mathrm{T}} \; \mathrm{N} \; (\delta\mathbf{g}) \tag{3.123}$$

In the case where generalised coordinates represent 3-dimensional coordinates of discrete nodes (as is the case in our rod simulation), this metric represents a displacement of the individual nodes weighted by the mass of each node.

The projection is then defined as minimisation of the following objective function:

$$\mathcal{W}(\delta\mathbf{g}, \boldsymbol{\lambda}) = \frac{1}{2(\Delta t)^2} (\delta\mathbf{g})^{\mathrm{T}} \; \mathrm{N} \; (\delta\mathbf{g}) + \mathbf{C}(\mathbf{g}) \cdot \boldsymbol{\lambda} \tag{3.124}$$

The closest point on $\mathcal{C}$ can be found e.g. by using Newton minimisation on $\mathcal{W}$: find increasingly better approximations of the "shortest" (in terms of $\mathcal{T}$) valid step $\delta\mathbf{g}$. Due to this iterative approach, this method of solution is known as step-and-project. Other approaches are possible (Goldenthal et al. 2007; Bergou et al. 2008), but as we use Newton minimisation in our hair simulation, we will not discuss them here.

After the constraint enforcement step finishes, $\dot{\mathbf{g}}$ must be updated to reflect the actual change in $\mathbf{g}$ from time t_{n-1} to t_n; the one computed by integration only reflects the change $\mathbf{g}(t_{n-1}) \to \breve{\mathbf{g}}(t_n)$. This update can be expressed as follows:

$$\dot{\mathbf{g}} = \breve{\dot{\mathbf{g}}} \frac{1}{\Delta t} (\mathbf{g} - \breve{\mathbf{g}}) \tag{3.125}$$

Constraints for Rods

We proceed to introduce constraints as they are used in the Discrete Elastic Rod method. Its authors employ two types of constraints: inextensibility and rigid body coupling.

Inextensibility An inextensibility constraint is introduced for each edge of the rod:

$$\mathrm{CI}^j = \mathbf{e}^j \cdot \mathbf{e}^j - \underline{\hat{\mathbf{e}}}^j \cdot \underline{\hat{\mathbf{e}}}^j \tag{3.126}$$

Rigid Body Coupling Any edge of the rod can be attached to a rigid body. The position of the edge's endpoints and the orientation of its material frame have to correspond to the position and rotation of the attached rigid body. Let us assume R rigid bodies $1, \ldots, R$ are attached to the rod at edges $a_1, \ldots, a_R$. $\mathbf{r}_i \in \mathbb{R}^3$ denotes the position of body i. Its rotation is represented by the unit quaternion $\mathbf{q}_i \in \mathbb{H}$. The coupling is then expressed using these constraints:

$$\mathrm{CU}_i = \mathbf{q}_i \cdot \mathbf{q}_i - 1 \tag{3.127}$$

$$\mathrm{CB}_i^1 = \mathbf{q}_i \underline{\hat{\mathbf{x}}}_{a_i} \mathbf{q}_i^* + \mathbf{r}_i - \mathbf{x}_{a_i} \tag{3.128}$$

$$\mathrm{CB}_i^2 = \mathbf{q}_i \underline{\hat{\mathbf{x}}}_{a_{i+1}} \mathbf{q}_i^* + \mathbf{r}_i - \mathbf{x}_{a_{i+1}} \tag{3.129}$$

$\mathbf{q}^*$ is the conjugate of $\mathbf{q}$. Equation 3.127 ensures unit length of $\mathbf{q}_i$, so that $\mathbf{q}_i \underline{\hat{\mathbf{x}}}_{a_i} \mathbf{q}_i^*$ is a rotation. The other two equations ensure that both ends of the rod's edge stay in sync with the rigid body. Material frame orientation is maintained by clamping, as discussed in Section 3.5.3.

The variables $\mathbf{r}$ and $\mathbf{q}$ do not enter the rod equations of motion, as the body is simulated separately. However, they do form part of the constraint enforcement step. The following generalised coordinates $\mathbf{g}$, generalised velocity $\dot{\mathbf{g}}$, and generalised mass matrix N, are therefore used for the constraint enforcement step:

$$\mathbf{g} = \left(\mathbf{q}_1, \ldots, \mathbf{q}_R, \mathbf{r}_1, \ldots, \mathbf{r}_R, \mathbf{x}_0, \ldots, \mathbf{x}_{N+1}\right)^{\mathrm{T}} \tag{3.130}$$

$$\dot{\mathbf{g}} = \left(\mathbf{q}_1^{-1}\dot{\mathbf{q}}_1, \ldots, \mathbf{q}_R^{-1}\dot{\mathbf{q}}_R, \dot{\mathbf{r}}_1, \ldots, \dot{\mathbf{r}}_R, \dot{\mathbf{x}}_0, \ldots, \dot{\mathbf{x}}_{N+1}\right)^{\mathrm{T}} \tag{3.131}$$

$$\mathrm{N} = \begin{pmatrix} 4\underline{\mathrm{I}}_1 & & & & & & & \\ & \ddots & & & & & & \\ & & 4\underline{\mathrm{I}}_R & & & & & \\ & & & \underline{w}_1 [1]_{3\times3} & & & & \\ & & & & \ddots & & & \\ & & & & & \underline{w}_R [1]_{3\times3} & & \\ & & & & & & \underline{\mathrm{M}} \end{pmatrix} \tag{3.132}$$

Here, $[1]_{3\times3}$ represents a 3×3 identity matrix. $\underline{w}_i$ is the scalar mass of rigid body i, and $\underline{\mathrm{I}}_i$ is its moment of inertia tensor, expressed as a 3×3 matrix in reference coordinates.

After constraint enforcement, velocities have to be updated according to equation 3.125:

$$\begin{pmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{r}} \\ \dot{\mathbf{x}} \end{pmatrix} = \begin{pmatrix} \breve{\dot{\mathbf{q}}} \\ \breve{\dot{\mathbf{r}}} \\ \breve{\dot{\mathbf{x}}} \end{pmatrix} - \frac{1}{\Delta t} \begin{pmatrix} 2\breve{\mathbf{q}}\mathbf{q}^{-1} \\ \breve{\mathbf{r}} - \mathbf{r} \\ \breve{\mathbf{x}} - \mathbf{x} \end{pmatrix} \tag{3.133}$$

3.6 Simulating Hair as Explicit Rods

We build on the Discrete Elastic Rod method of Bergou et al. (2008) as laid out in the previous section, applying the core principles of our hair animation approach to it.

We adopt their basic premise of discretising the rod which represents the hair strand into nodes connected by straight edges. We also use the coordinate reduction representing material frame orientation as deviation from the Bishop frame. We introduce our own algorithm for computing twist, which is significantly better for hair than the Newton minimisation used by the generic method. This algorithm is presented in Section 3.6.1.

Equations of motion presented by Bergou et al. (2008) (equation 3.120) reflect the rod's internal energy only. We need to extend them with external forces acting on the strand:

$$\frac{\mathrm{d}\dot{\mathbf{x}}_i}{\mathrm{d}t} = -\frac{1}{\underline{M}_i}\frac{\partial U_{\text{bend}}}{\partial \mathbf{x}_i} + \frac{1}{\underline{M}_i}\sum_{j:\text{clamped}}\frac{\partial U}{\partial \theta^j}\frac{\partial \Psi^j}{\partial \mathbf{x}_i} + \frac{1}{\underline{M}_i}\mathbf{F}_i \tag{3.134}$$

The basic external force present in our simulation is gravity:

$$\mathbf{F}_i^{\text{gravity}} = \underline{M}_i\mathbf{g} \tag{3.135}$$

Other external forces are added to this term as applicable; we describe them later in the text.

In the constraint-enforcement step, we use the same inextensibility constraints as the original method. We take a different approach to rigid body coupling, however.

The only rigid body normally coupled to a hair strand is the head. Due to the immense difference in mass and inertia between hair and the head, we do not model this coupling with constraints. Instead, we neglect any dynamic effect hair might have on the head and simply treat the node $\mathbf{x}_0$ (which corresponds to the strand's root) as immovably attached to one point on the scalp. We generally operate in the coordinate system attached to the head, which makes the root simply stationary.

We also introduce additional constraints for hair–head interaction, as detailed in Section 3.6.2. Finally, we model hair configurations commonly found in real-world hairstyles by adding extra elements to the simulation, described in Section 3.6.3.

Hair–hair collisions are the topic of Chapter 4.

3.6.1 Hair Twisting Model

The cornerstone of our approach to hair animation is the fact that hair prefers to bend over its major axis. This is in line with the observation Bergou et al. (2008) make about the speed of twist wave propagation (see Twist evolution in Section 3.5.3): when a hair bends, it will actually twist so that the bending happens over the major axis. The implication is that the current shape (bending) of the centreline is the chief factor determining the strand's twist.

Recall that the Discrete Elastic Rod method (Bergou et al. 2008) uses Newton minimisation to find the twist configuration which minimises the rod's internal energy, given a fixed shape of the centreline. In theory, the same approach could be used for hair as well. Bending stiffness over the minor axis is significantly larger than that over the major one, so the minimisation would find the proper twist. However, this approach is not optimal.

First, human hair is a remarkably stiff material. The stiffness values appropriate for a hair strand are 2–3 orders of magnitude larger than those used in the examples presented by Bergou et al. (2008). Such high stiffness makes the Newton minimisation numerically unstable.

Second, Newton minimisation is an iterative process. Each iteration requires the solution of a system of linear equations, making the whole process computationally expensive.

Fortunately, a better solution exists. Since we know that hair will twist so that it bends over the major axis, we do not need to compute twist iteratively—given the bent centreline shape, we can compute the appropriate twist in a *direct, explicit* fashion.

In the continuous setting, the strand could always twist so that it bends exclusively over the major axis. In the discrete case, this is not fully possible. Bending exclusively over the major axis is reached when the curvature binormal is collinear with the major axis. However, recall that the discrete curvature binormal $(\kappa\mathbf{b})_i$ is defined for each node i, while material frames are assigned to segments. Since $(\kappa\mathbf{b})_i$ and $(\kappa\mathbf{b})_{i+1}$ are unlikely to be collinear, an orientation of the major axis $\mathbf{m}_1^i$ such that no bending over the minor axis occurs is principally impossible. Instead, we present an algorithm to compute twist such that it merely minimises bending over the minor axis, not eliminates it.

We have first introduced our hair-specific algorithm for computing twist in (Kmoch et al. 2009). In the following text, we present an improved version of the algorithm which handles edge cases better.

Hair Twisting Algorithm

The algorithm makes use of the notion that bending over the minor axis is to be avoided as much as possible. It assumes that any decrease of bending over the minor axis leads to lower internal energy, regardless of the amount of extra twist introduced. In other words, the assumption is that effective bending stiffness over the minor axis is much larger than effective twisting stiffness:

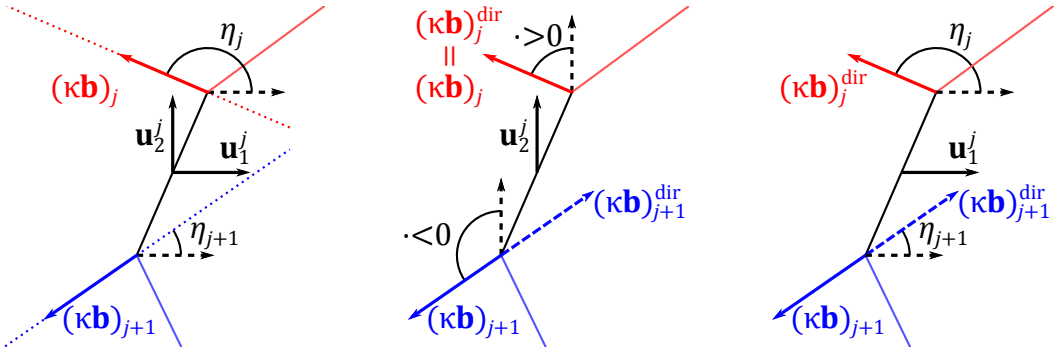
$$\begin{aligned} \left|\frac{\mathrm{d}U}{\mathrm{d}\omega_2}\right| &\gg \left|\frac{\mathrm{d}U}{\mathrm{d}\tau}\right| \\ \left|\frac{\mathrm{d}U}{\mathrm{d}\omega_2}\right| &\gg \left|\frac{\mathrm{d}U}{\mathrm{d}\omega_1}\right| \end{aligned} \tag{3.136}$$

Note that both of these hold for real hair, and form the physical basis of the observations of Swift (1995).

Our goal is to find the material frame orientation, that is, the value of the angles θ^j for all edges j. The algorithm itself consists of two sequential steps:

1. For each edge j, find the *unoriented* direction of the major axis $\mathbf{m}_1^j$ which will minimise bending over the minor axis at nodes $\mathbf{x}_j$ and $\mathbf{x}_{j+1}$. This establishes the value of θ^j up to a whole multiple of π: $\theta^j = \theta^j_{\mathrm{dir}} + h_j\pi$, where $h_j \in \mathbb{Z}$ represents the unknown multiple.

2. Within the direction obtained in step 1, find the orientation which will minimise elastic energy. This determines θ^j completely by fixing h_j in the above equation.



(a) Goal (b) Determining $(\kappa\mathbf{b})_k^{\text{dir}}$ (c) Computing η_k

Figure 3.10: Computing angle $\eta_k, k \in \{j, j+1\}$ between $\mathbf{u}_1^j$ and $(\kappa\mathbf{b})_k^{\text{dir}}$. Red and blue are used for the root-facing and tip-facing nodes and segments, respectively. Figure (a) shows the angles η_j and η_{j+1} we want to compute. (b) demonstrates finding $(\kappa\mathbf{b})_k^{\text{dir}}$ by computing $(\kappa\mathbf{b})_k \cdot \mathbf{u}_2^j$, and (c) shows how η_k is found from $\mathbf{u}_1^j \cdot (\kappa\mathbf{b})_k^{\text{dir}}$.

Step 1 is illustrated in Figures 3.10 and 3.11. For each (unclamped) edge j we first find the angles η_j and η_{j+1} between the Bishop axis $\mathbf{u}_1^j$ and the unoriented direction of respective curvature binormals $(\kappa\mathbf{b})_j$ and $(\kappa\mathbf{b})_{j+1}$ at adjoining nodes. Note that the curvature binormal at a node is orthogonal to the adjoining edges, which means that all nonzero vectors involved are coplanar. The exact formula for $\eta_k, k \in \{j, j+1\}$ is as follows:

$$(\kappa\mathbf{b})_k^{\text{dir}} = \begin{cases} (\kappa\mathbf{b})_k & \text{if } (\kappa\mathbf{b})_k \cdot \mathbf{u}_2^j \geq 0 \\ -(\kappa\mathbf{b})_k & \text{if } (\kappa\mathbf{b})_k \cdot \mathbf{u}_2^j < 0 \end{cases} \qquad \text{(Figure 3.10b)} \tag{3.137}$$

$$\eta_k = \begin{cases} \arccos\left(\frac{1}{\left\|(\kappa\mathbf{b})_k^{\text{dir}}\right\|}\mathbf{u}_1^j \cdot (\kappa\mathbf{b})_k^{\text{dir}}\right) & \text{if } (\kappa\mathbf{b})_k^{\text{dir}} \neq \mathbf{0} \\ \text{not defined} & \text{if } (\kappa\mathbf{b})_k^{\text{dir}} = \mathbf{0} \end{cases} \qquad \text{(Figure 3.10c)} \tag{3.138}$$

For each edge j, there are three possible scenarios:

One bent node When η_k is defined for only one $k \in \{j, j+1\}$, we simply set $\theta_{\text{dir}}^j = \eta_k$. We call such edge j a *half-bent* edge.

Two bent nodes When both η_j and η_{j+1} are defined, we need to find a direction which will bisect the smaller angle between $(\kappa\mathbf{b})_j^{\text{dir}}$ and $(\kappa\mathbf{b})_{j+1}^{\text{dir}}$ (see Figure 3.11):

$$\theta_{\text{dir}}^j = \begin{cases} \frac{1}{2}\left(\eta_j + \eta_{j+1}\right) & \text{if } \left|\eta_j - \eta_{j+1}\right| \leq \pi/2 \\ \frac{1}{2}\left(\eta_j + \eta_{j+1} + \pi\right) & \text{if } \left|\eta_j - \eta_{j+1}\right| > \pi/2 \end{cases} \tag{3.139}$$

Then, we add or subtract a whole multiple of π so that $0 \leq \theta_{\text{dir}}^j < \pi$. Such an edge j is called a *fully-bent* edge.

No bent nodes If neither η_j nor η_{j+1} is defined, we call such an edge j an *unbent* edge. Unbent edges on one strand form a set of zero or more contiguous sequences. We denote this set $\mathcal{U}$; material frame orientation of unbent edges will be computed in the second step of the algorithm. The important property of unbent edges is that their material frame orientation is not constrained by bending—it can be arbitrary.

With the first step completed, we now have a set $\mathcal{U}$ of unbent edges, and for each edge k not in this set, a major axis direction θ^k_{dir} such that bending over the minor axis is minimised on both adjoining nodes.

We now need to find the actual major axis orientation, θ^j, for each edge j, both bent and unbent. Notice that this cannot affect the bending component of elastic energy (U_{bend}, equation 3.85), as that depends on the (unoriented) directions of the cross-section axes only. We therefore only seek to find material frame orientation which will minimise the twist component of internal energy (U_{twist}, equation 3.88). This is valid thanks to equation 3.136, from which it follows that decreasing twist energy by increasing bending over the minor axis cannot lead to an overall decrease of internal energy.

To find the material frame orientation minimising twist energy, we process the edges sequentially, starting at the root ($j = 0$).

If the edge j is bent (fully or half), finding the orientation amounts to finding h_j, the whole multiple of π to add to θ^j_{dir}; such a situation is depicted in Figure 3.12. Remember that this choice will have no impact on bending energy—that has already been minimised by minimising bending over the minor axis. We therefore simply need to find h_j which will minimise twist deviation:

$$\tau_j - \hat{\underline{\tau}}_j = \theta^j - \theta^{j-1} - \hat{\underline{\tau}}_j = \theta^j_{\text{dir}} + h_j\pi - \theta^{j-1} - \hat{\underline{\tau}}_j \tag{3.140}$$

Because we're processing the strand sequentially, the andle θ^{j-1} has already been found (assume $\theta^{-1} = 0$). The only unknown in equation 3.140 is therefore h_j and the minimisation boils down to a trivial rounding operation:

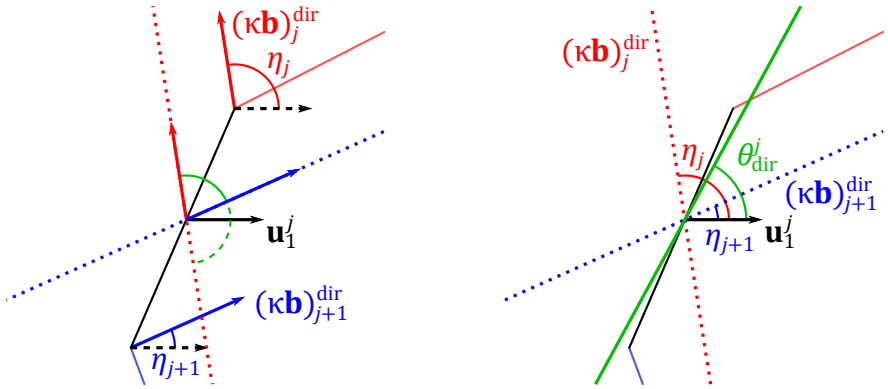
$$h_j = \text{round}\left(\frac{1}{\pi}\left(\hat{\underline{\tau}}_j - \left(\theta^j_{\text{dir}} - \theta^{j-1}\right)\right)\right) \tag{3.141}$$

If, on the other hand, the edge j is unbent, it is actually a start of a contiguous sequence of K unbent edges; even if the next edge $j + 1$ is not unbent, we view edge j as a sequence of length $K = 1$. Which means our situation is thus:

- θ^{j-1} has already been fully determined.
- Either $j + K = N + 1$, or $\theta^{j+K}_{\text{dir}}$ is known (from step 1).
- For all $k \in \{j, \ldots, j + K - 1\}$, θ^k_{dir} is not defined.

If the unbent sequence covers the entire rest of the strand (that is, $j+K = N+1$), the solution is trivial. The strand will simply adopt the rest-state twist on the entire unbent sequence:

$$\left(\forall k \in \{j, \ldots, N\}\right)\ \theta^k = \theta^{j-1} + \sum_{i=j}^{k} \hat{\underline{\tau}}_i \tag{3.142}$$



(a) $|\eta_j - \eta_{j+1}| \leq \pi/2$ (b) $\theta^j_{\text{dir}} = \frac{1}{2}(\eta_j + \eta_{j+1})$

(c) $|\eta_j - \eta_{j+1}| > \pi/2$ (d) $\theta^j_{\text{dir}} = \frac{1}{2}(\eta_j + \eta_{j+1} + \pi)$

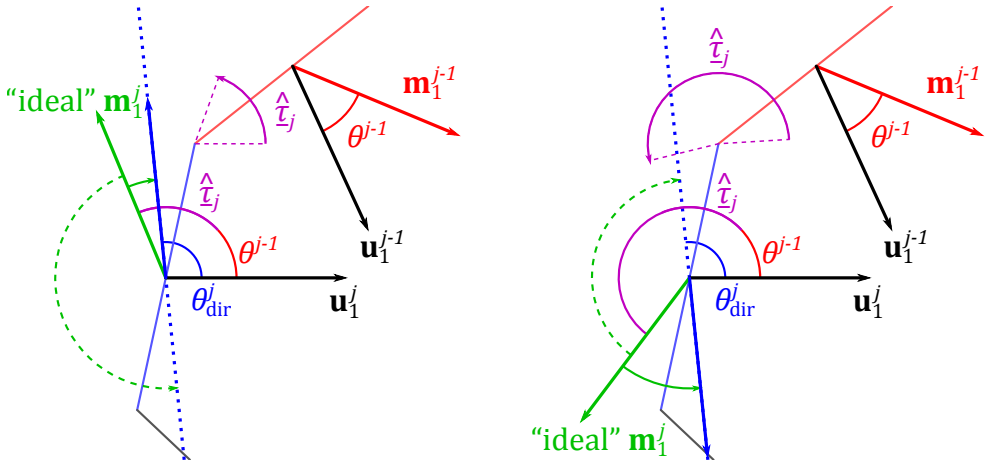
Figure 3.11: Finding θ^j_{dir} for an edge with two bent nodes. We need to choose the smaller from the two angles between the lines collinear with curvature binormals (dotted red and blue). Figure (a) shows the case when the angle between $(\kappa\mathbf{b})_j^{\text{dir}}$ and $(\kappa\mathbf{b})_{j+1}^{\text{dir}}$ (solid green) is smaller; θ^j_{dir} is then between η_j and $\eta j + 1$ (b). The second case, when the smaller angle is between $-(\kappa\mathbf{b})_j^{\text{dir}}$ and $(\kappa\mathbf{b})_{j+1}^{\text{dir}}$, is shown in (c); $\pi/2$ is then added to θ^j_{dir} (d).

When the sequence is followed by a bent edge $j+K$, then $\theta^{j+K}_{\text{dir}}$ is known (from step 1). We start by finding h_{j+K}. To be able to employ equation 3.141, we need θ^{j+K-1}; refer to Figure 3.13 for an illustration of this algorithm. We assign this tentatively:

$$\theta^{j+K-1}_{\text{tentative}} = \theta^{j-1} + \sum_{i=j}^{j+K-1} \hat{\underline{\tau}}_i \tag{3.143}$$

This basically represents the entire unbent sequence twisting so that it assumes rest-state twist. We use this value in equation 3.141 to determine h_{j+K} which will minimise the twist deviation $\left(\theta^{j+K} - \theta^{j+K-1}_{\text{tentative}}\right) - \hat{\underline{\tau}}_{j+K}$. We have thus computed the best possible orientation of the material frame at edge $j+K$, assuming twist of the unbent sequence identical to rest twist. This probably still leaves the twist τ_{j+K} different from the corresponding rest twist. We denote this difference $\tilde{\tau}$:

$$\tilde{\tau} = \tau_{j+K} - \hat{\underline{\tau}}_{j+K} = \left(\theta^{j+K} - \theta^{j+K-1}_{\text{tentative}}\right) - \hat{\underline{\tau}}_{j+K} \tag{3.144}$$



(a) h_j computed as 0 (b) h_j computed as 1

Figure 3.12: Computing θ^j when node j is bent. The green arrow represents $\mathbf{m}_1^j$ which would produce zero twist deviation on segment j, but this cannot be achieved exactly. Only whole multiples of π can be added to θ^j_{dir}. Figure (a) shows an example where the best multiple is $h_j = 0$, while (b) shows a case where adding $h_j = 1$ times π is better for minimising twist deviation. The blue arrow depicts the computed major axis $\mathbf{m}_1^j$ in both cases.

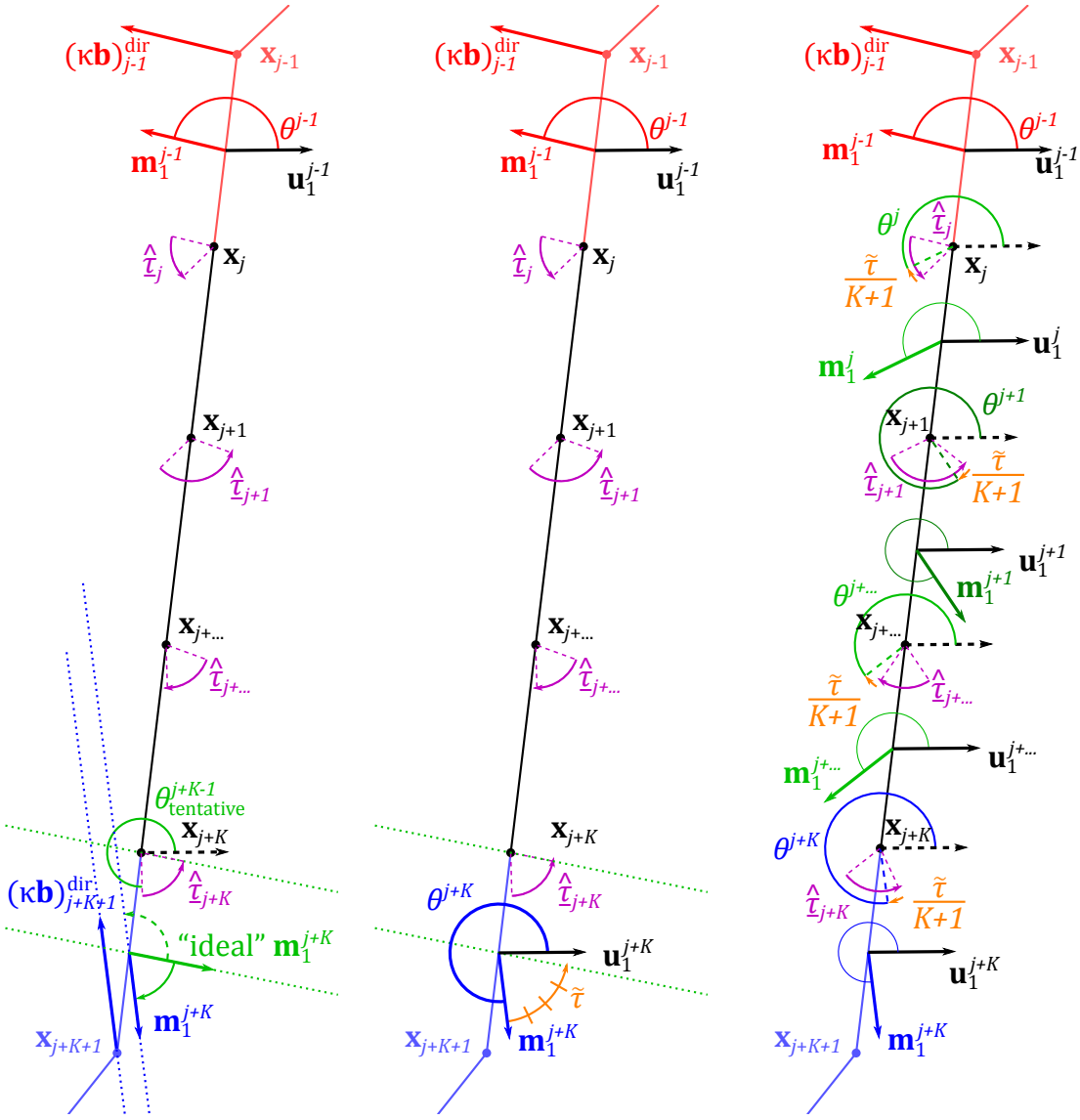
Recall from equation 3.88 that the internal energy is quadratic in deviation from rest twist. We can therefore minimise it by distributing the "error" $\tilde{\tau}$ evenly along the entire unbent sequence:

$$\begin{aligned}\left(\forall k \in \{j, \ldots, j+K-1\}\right)\ \theta^k &= \theta^{j-1} + \sum_{i=j}^{k} \left(\hat{\underline{\tau}}_i + \frac{\tilde{\tau}}{K+1}\right) \\ &= \theta^{j-1} + \sum_{i=j}^{k} \hat{\underline{\tau}}_i + (k-j+1)\,\frac{\tilde{\tau}}{K+1}\end{aligned} \tag{3.145}$$

Notice that all formulae for computing orientation on individual edges in the unbent sequence feature a sum of rest-state twists. We can either compute the sum on the fly, processing the edges of the sequence one after the other, or we can employ a pre-computed lookup table of cumulative twist. The latter solution makes the edges within the sequence independent, allowing them to be processed in parallel.

Twist Algorithm Evaluation

We have implemented both the Newton minimisation as used by Bergou et al. (2008) and our algorithm we've just presented. We compare their performance in Table 3.1; the original implementation of our algorithm as presented in (Kmoch et al. 2009) is also included in the comparison. The basic metric we choose for comparison is absolute time required for the twist computation. You can see from the table that our twist computation clearly outperforms Newton minimisation. The table also shows that the improved version of our algorithm is faster than the original one.



(a) Using $\theta^{j+K-1}_{\text{tentative}}$ to find h_{j+K} and thus $\mathbf{m}_1^{j+K}$ and θ^{j+K}

(b) Using θ^{j+K} to compute twist error $\tilde{\tau}$

(c) Distributing $\tilde{\tau}$ along the entire sequence

Figure 3.13: Computing twist on an unbent sequence. Edges on the sequence are depicted in black (the illustration corresponds to $K = 3$). Figure (a) shows how the major axis orientation on the bent edge immediately following the sequence is found by assuming orientation $\theta^{j+K-1}_{\text{tentative}}$ corresponding to rest-state twist. In the next step, the twist "error" $\tilde{\tau}$ is computed, using equation 3.144 (b); the figure shows the twist error in orange, divided into $K+1$ parts. Finally, material frame orientation θ is computed on the entire sequence by distributing the error evenly among all edges, as per equation 3.145 (c).

# strands	# nodes	Newton	Hair	Better hair
1	30	0.14	0.06	0.02
3	90	0.41	0.18	0.05

Table 3.1: Comparing the performance of updating the material frame after integration using different methods: Newton minimisation, our hair-specific twisting algorithm presented in (Kmoch et al. 2009), and an improved version of this algorithm presented in this thesis. The times are given in milliseconds.

# strands	# nodes	Time step	DER	Hair
1	49	1.0	84%	9%
11	275	1.0	131%	9%
31	2158	0.1	74%	6%

Table 3.2: Time spent updating material frames as a percentage of time used to compute forces. Time steps given in milliseconds. Numbers for DER method were taken from (Bergou et al. 2008, Table 1). Comparison is only approximate due to inability of our system to exactly recreate scenes used by Bergou et al.

To prevent our measurements from potentially being skewed by differences in implementation of Newton minimisation between our work and that of Bergou et al. we also express time required for twist computation relative to the time spent integrating equations of motion, and compare our implementation against the numbers published by Bergou et al. (2008). For this, we have tried to recreate some of the scene set-ups of Bergou et al. Exact replication is not possible because our simulation system is geared towards hair, not arbitrary rods, but we've matched the number of nodes and time-step size where practical. This results of this comparison are presented in Table 3.2. The comparison is not on equal terms, as our hair simulation framework does not allow us to exactly recreate the test scenes of Bergou et al. but even if we allow for a compensating factor of 5 to our disadvantage, it is still clear that our algorithm is very efficient for hair.

3.6.2 Hair–Head Collisions

A significant number of human hair strands is constantly in contact with the head. Any hair simulation method must therefore implement collision detection and response for the head, even if collision with other rigid bodies is not considered. The head is also very specific in that its shape, as well as its position and orientation relative to the strands' roots, does not change. We have used these properties to devise a very efficient yet cheap collision detection & response system for hair–head collisions.

Collision Response Methods

In general, there are many ways to respond to collisions.

Force-based collision response methods compute a repulsion or penalty force

when objects collide. The simplest approach is to detect collisions at the beginning of the time step, and if a collision occurs, add penalty forces which would move the colliding bodies out of collision. One popular mechanism is to use a spring-like penalty force proportional to the bodies' interpenetration. This approach is fast, because the force computation is trivial. The downside is that collisions are not *prevented*—by the time a response force is added to the simulation, the collision has already happened. This may be tolerable for some systems but unacceptable for others. A further disadvantage is that these suddenly appearing forces can cause the simulation to become unstable if they are too large. Tuning the "stiffness" of penalty-based collision response so that it successfully prevents undesired collisions effects without destabilising the integration is not easy.

A more sophisticated force-based approach evaluates collisions at the end of the time step. Collisions are detected and reaction forces computed. The time step is then discarded and re-computed again, with the collision response forces added in right from the start. This approach is much better at actually preventing collisions from happening and is also more numerically stable. The price to pay is performance, as basically the entire simulation has to be computed twice for each time step.

Some methods use a response other than forces, like simply re-positioning colliding bodies into a contact-only configuration. These tend to suffer from energy conservation problems.

Our Hair–Head Collisions

The objective of detecting and handling hair–head collisions is to keep the simulated hair from penetrating into the head model. In other words, the hair is constrained to stay out of the volume of the head. Notice that our simulation method already contains a constraint-enforcement step. We use this and handle hair–head collisions by formulating them as extra constraints to be enforced.

This approach has a number of advantages. Constraint enforcement is already a part of the process, which makes hair–head collision resolution very efficient. There is of course an increase in computation required to actually enforce the head-collision constraints, but there is no additional overhead for facilitating a separate collision handling subsystem. Most importantly, however, it produces an effective and stable response.

Because constraints are enforced as part of the simulation step, this approach actually prevents collisions from happening. There are no stiff forces which could destabilise the integration, and energy conservation is handled by the velocity update step (equation 3.125).

We first presented this approach in (Kmoch et al. 2009). A sphere of centre $\mathbf{h}$ and radius $\underline{H}$ is used to represent the head for collision purposes. Our use of a sphere is motivated by efficiency of computation, but more complex shapes such as an ellipsoid or even metaballs (Bloomenthal and Bajaj 1997) could also be used. The choice of collision representation of the head is a trade-off between representation accuracy and efficiency of computing node–head distance. The only hard requirement on the representation is that the gradient of the node–head distance must be computable, for use in constraint enforcement. Nevertheless, we will use the sphere representation in this text.

After each integration step, all nodes are tested for head penetration, and those which are inside the head are gathered into set $\mathcal{P}$:

$$\mathcal{P} = \left\{ i : (\mathbf{x}_i - \mathbf{h}) \cdot (\mathbf{x}_i - \mathbf{h}) < \underline{H}^2 \right\} \tag{3.146}$$

Nodes in $\mathcal{P}$ are then subjected to the following constraint:

$$(\forall i \in \mathcal{P})\ \mathrm{CH}_i = (\mathbf{x}_i - \mathbf{h}) \cdot (\mathbf{x}_i - \mathbf{h}) - \underline{H}^2 \tag{3.147}$$

This approach worked well, but it had drawbacks:

- Some nodes originally not in $\mathcal{P}$ could penetrate the head as a result of relocating to satisfy the constraints, because the set $\mathcal{P}$ is only computed once in each time step. Given the short time step we use (1–2ms), this does not introduce any noticeable artifacts, but it means the method does not totally prevent collisions in all cases.

- The constraints actually force the node to touch the head instead of merely forcing it out of it. We later discovered this posed convergence problems in complex collision scenarios.

We seek to improve the method to remove these drawbacks.

We achieve this by extending the constraint enforcement step to handle inequality constraints in addition to equality constraints. We will first present this as a theoretical extension of the constraint enforcement mechanism, before detailing its use in our hair animation method.

For this, we again turn back to the notation used in Section 1.4.2: we have a system described using Lagrangian mechanics with generalised coordinates $\mathbf{g}$ and generalised velocities $\dot{\mathbf{g}}$, both $\underline{\gamma}$-dimensional vectors. The system is subject to the following set of constraints:

$$\mathbf{C}(\mathbf{g}) = \mathbf{0} \tag{3.148}$$

$$\mathbf{D}(\mathbf{g}) \geq \mathbf{0} \tag{3.149}$$

$\mathbf{C}(\mathbf{g}) \in \mathbb{R}^{\underline{\zeta}}$ and $\mathbf{D}(\mathbf{g}) \in \mathbb{R}^{\underline{\rho}}$ are vectors of equality and inequality constraints, respectively.

We follow the Discrete Elastic Rod method in integrating the unconstrained equations of motion to obtain a solution $\breve{\mathbf{g}}$ which can violate the constraints. We then perform a constraint enforcement step using a step-and-project method (see Section 3.5.4) to find the final solution $\mathbf{g} = \breve{\mathbf{g}} + \delta\mathbf{g}$. The step-and-project method will find suitable $\delta\mathbf{g}$ while minimising weighted deviation from $\breve{\mathbf{g}}$.

To be able to use Lagrange multipliers for minimisation, we need to transform equation 3.149 into equality constraints. We do this by introducing auxiliary *slack variables* $\mathbf{s} \in \mathbb{R}^{\underline{\rho}}$ (Boyd and Vandenberghe 2004) for these constraints:

$$\mathbf{D}(\mathbf{g}) + \mathbf{s} = \mathbf{0} \tag{3.150}$$

$$\mathbf{s} \geq \mathbf{0} \tag{3.151}$$

It is obvious that when equation 3.150 holds, $D_i(\mathbf{g}) < 0$ if and only if $s_i < 0$. So far, we have transformed each original (general) inequality constraint into one equality constraint and one inequality constraint of a simple form of "variable

is nonnegative." The method of Lagrange multipliers will take care of maintaining the equality constraints, but we still need to ensure the slack variable nonnegativity (equation 3.151).

The function we want to minimise (subject to these constraints) is the generalised kinetic energy (equation 3.123) of the projection (or displacement). To force slack variables to be nonnegative, we extend the objective function as follows:

$$\mathcal{T}(\delta\mathbf{g}, \mathbf{s}) = \frac{1}{2(\Delta t)^2} (\delta\mathbf{g})^{\mathrm{T}} \ \mathrm{N} \ (\delta\mathbf{g}) - \sum_{i=1}^{\rho} \ln s_i \tag{3.152}$$

The crucial point is:

$$\lim_{s_i \to 0+} -\ln s_i = +\infty \tag{3.153}$$

Remember that our goal is to *minimise* the objective function (equation 3.152). If any of the slack variables approaches 0, the term $-\sum_{i=1}^{\rho} \ln s_i$ and with it the entire objective function will grow beyond bounds; this is something the minimisation will prevent, effectively guaranteeing equation 3.151. As we've shown, that is equivalent to equation 3.149, our original inequality constraints.

We have thus defined our projection operator and can apply the step-and-project method. Due to the presence of the slack variable term, we cannot use the fast manifold projection (Goldenthal et al. 2007) originally used by Discrete Elastic Rods. Instead, we use the more general Newton minimisation. We formulate the Lagrangian out of our objective function (equation 3.152) and constraints (equations 3.148 and 3.150):

$$\mathcal{W}(\delta\mathbf{g}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2(\Delta t)^2} (\delta\mathbf{g})^{\mathrm{T}} \ \mathrm{N} \ (\delta\mathbf{g}) - \sum_{i=1}^{\rho} \ln s_i + \mathbf{C}(\breve{\mathbf{g}} + \delta\mathbf{g}) \cdot \boldsymbol{\lambda} + \mathbf{D}(\breve{\mathbf{g}} + \delta\mathbf{g}) \cdot \boldsymbol{\mu} \tag{3.154}$$

Newton minimisation works by solving the following non-linear system:

$$\begin{aligned} \nabla_{\delta\mathbf{g}} \mathcal{W}(\delta\mathbf{g}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \mathbf{0} \\ \nabla_{\mathbf{s}} \mathcal{W}(\delta\mathbf{g}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \mathbf{0} \\ \nabla_{\boldsymbol{\lambda}} \mathcal{W}(\delta\mathbf{g}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \mathbf{0} \\ \nabla_{\boldsymbol{\mu}} \mathcal{W}(\delta\mathbf{g}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \mathbf{0} \end{aligned} \tag{3.155}$$

This gives the stationary points of both the Lagrangian and the objective function. The system is solved using the iterative Newton-Raphson method. This starts from an initial approximate solution $\left(\delta\mathbf{g}^0, \mathbf{s}^0, \boldsymbol{\lambda}^0, \boldsymbol{\mu}^0\right)$ and iteratively computes progressively better solutions:

$$\begin{pmatrix} \delta\mathbf{g}^{n+1} \\ \mathbf{s}^{n+1} \\ \boldsymbol{\lambda}^{n+1} \\ \boldsymbol{\mu}^{n+1} \end{pmatrix} = \begin{pmatrix} \delta\mathbf{g}^{n} \\ \mathbf{s}^{n} \\ \boldsymbol{\lambda}^{n} \\ \boldsymbol{\mu}^{n} \end{pmatrix} + \begin{pmatrix} \Delta\delta\mathbf{g}^{n} \\ \Delta\mathbf{s}^{n} \\ \Delta\boldsymbol{\lambda}^{n} \\ \Delta\boldsymbol{\mu}^{n} \end{pmatrix} \tag{3.156}$$

The process continues until equation 3.155 is solved to within a pre-specified tolerance threshold.

Each step proceeds by using Taylor expansion of $\nabla\mathcal{W}$ to first order:

$$\begin{aligned}\nabla\mathcal{W}\left(\delta\mathbf{g}^{n+1},\mathbf{s}^{n+1},\boldsymbol{\lambda}^{n+1},\boldsymbol{\mu}^{n+1}\right) = \\ \nabla\mathcal{W}\left(\delta\mathbf{g}^{n},\mathbf{s}^{n},\boldsymbol{\lambda}^{n},\boldsymbol{\mu}^{n}\right) + \left(\nabla^2\mathcal{W}\left(\delta\mathbf{g}^{n},\mathbf{s}^{n},\boldsymbol{\lambda}^{n},\boldsymbol{\mu}^{n}\right)\right)\ \left(\Delta\delta\mathbf{g}^{n},\Delta\mathbf{s}^{n},\Delta\boldsymbol{\lambda}^{n},\Delta\boldsymbol{\mu}^{n}\right)^{\mathrm{T}}\end{aligned} \tag{3.157}$$

Notice that this is a system of linear equations in the variables $\Delta\delta\mathbf{g}^n$, $\Delta\mathbf{s}^n$, $\Delta\boldsymbol{\lambda}^n$, $\Delta\boldsymbol{\mu}^n$. We simplify the system by substituting equation 3.154:

$$\begin{pmatrix} \frac{1}{(\Delta t)^2}\mathrm{N} & 0 & \left(\nabla_{\delta\mathbf{g}}\mathbf{C}\right)^{\mathrm{T}} & \left(\nabla_{\delta\mathbf{g}}\mathbf{D}\right)^{\mathrm{T}} \\ 0 & -\left[1\right]\frac{1}{\mathbf{s}^2} & 0 & -\left[1\right] \\ \nabla_{\delta\mathbf{g}}\mathbf{C} & 0 & 0 & 0 \\ \nabla_{\delta\mathbf{g}}\mathbf{D} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta\delta\mathbf{g}^n \\ \Delta\mathbf{s}^n \\ \Delta\boldsymbol{\lambda}^n \\ \Delta\boldsymbol{\mu}^n \end{pmatrix} = \\ \begin{pmatrix} \frac{1}{(\Delta t)^2}\mathrm{N}\left(\delta\mathbf{g}\right) + \left(\nabla_{\delta\mathbf{g}}\mathbf{C}\right)^{\mathrm{T}}\boldsymbol{\lambda} + \left(\nabla_{\delta\mathbf{g}}\mathbf{D}\right)^{\mathrm{T}}\boldsymbol{\mu} \\ -\frac{1}{\mathbf{s}} - 1 \\ \mathbf{C} \\ \mathbf{D} \end{pmatrix} \tag{3.158}$$

This becomes more handy if re-written like this:

$$\begin{pmatrix} \frac{1}{(\Delta t)^2}\mathrm{N} & 0 \\ 0 & -\left[1\right]\frac{1}{\mathbf{s}^2} \end{pmatrix} \begin{pmatrix} \Delta\delta\mathbf{g}^n \\ \Delta\mathbf{s}^n \end{pmatrix} + \begin{pmatrix} \left(\nabla_{\delta\mathbf{g}}\mathbf{C}\right)^{\mathrm{T}} & \left(\nabla_{\delta\mathbf{g}}\mathbf{D}\right)^{\mathrm{T}} \\ 0 & -\left[1\right] \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{\lambda}^n \\ \Delta\boldsymbol{\mu}^n \end{pmatrix} = \\ \begin{pmatrix} \frac{1}{(\Delta t)^2}\mathrm{N}\left(\delta\mathbf{g}\right) + \left(\nabla_{\delta\mathbf{g}}\mathbf{C}\right)^{\mathrm{T}}\boldsymbol{\lambda} + \left(\nabla_{\delta\mathbf{g}}\mathbf{D}\right)^{\mathrm{T}}\boldsymbol{\mu} \\ -\frac{1}{\mathbf{s}} - 1 \end{pmatrix} \tag{3.159}$$

$$\begin{pmatrix} \nabla_{\delta\mathbf{g}}\mathbf{C} \\ \nabla_{\delta\mathbf{g}}\mathbf{D} \end{pmatrix} \left(\Delta\delta\mathbf{g}\right) = \begin{pmatrix} \mathbf{C} \\ \mathbf{D} \end{pmatrix} \tag{3.160}$$

Written this way, it is clear that the way to solve the system is by isolating $(\Delta\delta\mathbf{g}, \Delta\mathbf{s})$ in equation 3.159 and substituting it into equation 3.160, then using the obtained $(\Delta\boldsymbol{\lambda}, \Delta\boldsymbol{\mu})$ to solve equation 3.159.

We now apply this approach to our hair model. For each edge j, an equality constraint CI^j is used to guarantee the hair's inextensibility, as defined in equation 3.126. Inequality constraints are used to prevent hair–head collisions, each constraint controlling one node:

$$\left(\forall i \in \mathcal{P}\right)\ \mathrm{CH}_i = \left(\mathbf{x}_i - \mathbf{h}\right)\cdot\left(\mathbf{x}_i - \mathbf{h}\right) - \underline{H}^2 \tag{3.161}$$

The constraint equations are:

$$\left(\forall j \in 0,\ldots,N\right)\ \mathrm{CI}^j = 0 \tag{3.162}$$

$$\left(\forall i \in \mathcal{P}\right)\ \mathrm{CH}_i \geq 0 \tag{3.163}$$

Thanks to the fact that we can now express inequality in constraints, it is beneficial to activate the hair–head collision constraints when the nodes approach the head, before collision actually occurs. We therefore slightly modify the definition

of $\mathcal{P}$ (originally equation 3.146) by introducing a collision avoidance threshold $\underline{R}$ added to the head radius $\underline{H}$ for this purpose:

$$\mathcal{P} = \left\{ i : (\mathbf{x}_i - \mathbf{h}) \cdot (\mathbf{x}_i - \mathbf{h}) < \left(\underline{H} + \underline{R}\right)^2 \right\} \tag{3.164}$$

Our experiments have shown that setting the value of $\underline{R}$ to around half the average segment length works as the best compromise between computational efficiency and effective hair–head collision avoidance. Notice that $\underline{R}$ is used during the test for inclusion in $\mathcal{P}$ only—the value of the constraint remains the squared distance from the head (equation 3.161).

To actually enforce our extended constraints, we substitute the following into equation 3.154:

$$\mathbf{g} = \mathbf{x} \tag{3.165}$$
$$\mathbf{C} = \mathbf{CI} \tag{3.166}$$
$$\mathbf{D} = \mathbf{CH} \tag{3.167}$$
$$\mathrm{N} = \underline{\mathrm{M}} \tag{3.168}$$

This gives us the following Lagrangian:

$$\begin{aligned} \mathcal{W}(\delta\mathbf{x}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \quad & \frac{1}{2(\Delta t)^2} (\delta\mathbf{x})^{\mathrm{T}} \underline{\mathrm{M}} (\delta\mathbf{x}) - \sum_{i \in \mathcal{P}} \ln s_i \\ & + \mathbf{CI}(\breve{\mathbf{x}} + \delta\mathbf{x}) \cdot \boldsymbol{\lambda} + \mathbf{CH}(\breve{\mathbf{x}} + \delta\mathbf{x}, \mathbf{s}) \cdot \boldsymbol{\mu} \end{aligned} \tag{3.169}$$

To solve equations 3.159 and 3.160, we also need the gradients of constraint. These are obtained using straightforward differentiation:

$$\nabla_i \mathrm{CI}^j = \begin{cases} -2\mathbf{x}_i & \text{if } i = j \\ 2\mathbf{x}_i & \text{if } i = j + 1 \\ 0 & \text{if } i \notin \{j, j+1\} \end{cases} \tag{3.170}$$

$$\nabla_i \mathrm{CH}_j = \begin{cases} 2\mathbf{x}_i & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{3.171}$$

After constraint minimisation finishes, velocities must be updated. In our hair system, the general form of this update (equation 3.133) is simplified as follows:

$$\dot{\mathbf{x}} = \breve{\dot{\mathbf{x}}} - \frac{1}{\Delta t}(\breve{\mathbf{x}} - \mathbf{x}) \tag{3.172}$$

Evaluation

We test the impact of hair–head collision handling by comparing the time required for constraint enforcement with hair–head collisions enabled and disabled. We report the time spent each time step on constraint enforcement in these scenarios:

- Hair–head collisions disabled
- Hair–head collisions enabled, detection margin $\underline{R} = 0.5\,\mathrm{cm}$ (half a segment length)

- Hair–head collisions enabled, detection margin $\underline{R} = 1\,\text{cm}$ (one segment length)

Our findings are given in Table 3.3. At first sight, the overhead shown for straight hair seems surprisingly high. The reason for these high numbers is that straight hair tends to move smoothly and thus does not normally violate its inextensibility constraints too much. For straight hair, hair–head collisions therefore form the primary need to iterate constraint enforcement. This can be seen from the fact that the more complex the hair shape gets (wavy and curly), the smaller the relative overhead of hair–head collisions is. Our observations from full simulations including hair–hair collisions indicate that when hair attains more complex shapes due to mutual interaction, the relative overhead of hair–head collision constrains becomes small.

At the same time, the table shows that the effect of of the collision avoidance threshold $\underline{R}$ on performance is significant. Setting the value too low or not using the threshold at all leads to instances of hair penetrating the head, while unnecessarily high values of the threshold markedly slow down the simulation. A value of half the average segment length has proven a good compromise between effectiveness and efficiency.

In these tests, hair–hair collision constraints (discussed in Section 4.3.1) were not applied in order to not skew the results—handling hair–hair collisions was disabled altogether.

3.6.3 Hair Wisps

In the preceding sections, we have presented how we simulate an isolated hair strand. We will now turn our attention to simulating a full volume of hair.

As we've discussed in Section 3.2, fully simulating all the strands in a typical hairstyle is absolutely beyond the limits of the class of hardware we want to target with our method. To simulate a full hairstyle, we need to introduce some form of simplification. As our approach is based on an explicit simulation model of an individual strand, it makes the most sense to extend it either by using guide strands and interpolation, or by simulating multi-strand wisps as a single entity.

Of these two approaches, we consider explicit wisp simulation the better choice. Methods which use direct interpolation from guide strands tend to pro-

Hair shape	**# strands & nodes**	**No collisions**	$\underline{R} = 0.5\,\text{cm}$	$\underline{R} = 1\,\text{cm}$
Straight	50 : 1500	3.67	6.78 (85%)	8.08 (120%)
Straight	25 : 750	1.85	3.19 (73%)	3.95 (114%)
Wavy	50 : 1500	4.69	6.84 (46%)	8.20 (75%)
Wavy	25 : 750	2.41	3.41 (41%)	3.99 (66%)
Curly	50 : 1500	6.21	7.05 (14%)	8.36 (35%)
Curly	25 : 750	3.14	3.54 (13%)	4.12 (31%)

Table 3.3: Effect of handling hair–head collisions on computation time. Time shown is for the constraint enforcement step only, given in milliseconds.

duce a uniform look which is appropriate for straight hair, but less so for hair which is wavy or curly. The reason is that slight variations in the deformation shape between nearby curled strands tend to mis-align the cuticle scales on the strands' surface, leading to increased friction and cohesion—distinct wisps are far more likely to form in curly hair than in straight hair. Such wisps are something a fully interpolation-based method cannot truly represent.

A method which simulates wisps explicitly clearly does not have this problem. It is however also able to simulate uniform hair. If the individual wisps behave like straight hair, they will have equally little reason to stay too disjoint and will form the uniform hairstyle naturally.

A hybrid approach combining interpolation and wisp-based rendering was introduced by Bertails et al. (2006). In this method, non-simulated strands are generated from guide strands using a linear blend of interpolation and extrapolation. Interpolation from two or more guide strands is used near the root of the strands, progressively being replaced by extrapolation of just one of them towards the tip. Parameters of this progression can be tuned to accommodate hair styles with different tendencies to form disjoint wisps. This approach is very good in providing the best of both worlds, provided the simulation method conforms to two conditions.

First, a wisp must use the same mechanical model as a single strand uses, because with this approach, a wisp is simply a group of strands extrapolated from one guide. From this, the second condition follows as well: all strands must be rendered individually. Wisps are only formed visually, they are not represented as separate objects in the simulation.

This hybrid approach mirrors well how wisps actually form in real hair. However, recall that the reason we have to resort to such simplifications in the first place is to reduce the complexity of simulating all strands individually. In a way, this semi-interpolation method restricts the simplifications we can use to those which still treat all strands as distinct, just not simulated separately.

Rendering individual strands is much cheaper than simulating them, but it is still costly. Bearing in mind that our goal is achieving a fast simulation and display, we want to find a way of representing wisps which will require handling individual strands neither during simulation nor rendering. We want to model and render wisps explicitly as first-class objects, not just as a collection of individual strands.

Our strand simulation model relies heavily on the large eccentricity of the strand's cross section, which makes it ill-suited to simulate an arbitrary wisp. However, recall that in Section 3.2.1 we have shown that in a large class of hairstyles, hair strands tend to naturally form flat, ribbon-like wisps. We seek to capture this effect in our simulation.

Our first intuition was that in many aspects, a flat wisp resembles a strand with eccentricity taken to the extreme: the major axis size would be the wisp width (usually around 1–2 cm), while the minor axis would represent the wisp thickness (a few mm at most). This is similar to the strip representation used by Ward et al. (2003), where a flat subdivision surface is extruded from a dynamically simulated polyline skeleton. However, a quick proof-of-concept implementation found that representing a wisp this way gives highly unrealistic behaviour with our model—the entire wisp does not twist the same way a single strand

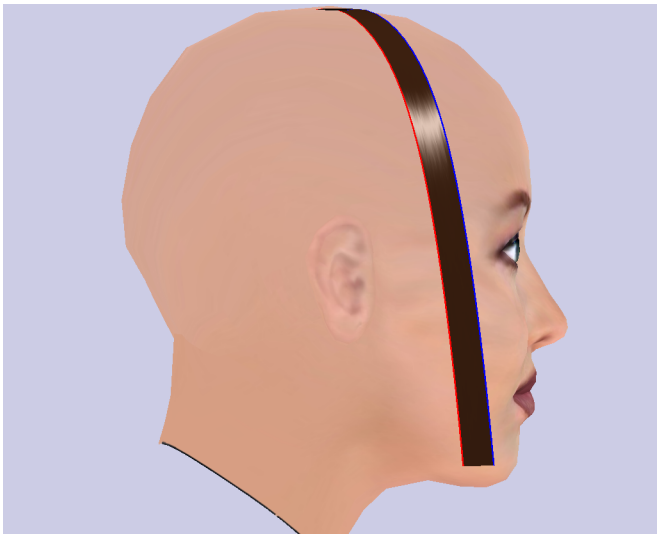


Figure 3.14: Flat wisp represented as a quad strip. Rim strands highlighted in colour.

would. Ward et al. do not account for torsion in their simulation, which is why their representation does not experience these issues. At the same time, as we have shown in Section 3.1.2 and elsewhere in this thesis, twist effects are important for correctly representing all aspects of hair behaviour, especially for non-straight hair. Abandoning torsion to be able to simulate the wisp as a single rod is therefore not an option.

Yet we found that with proper rendering, the visual aspect of such a flat wisp was quite acceptable. Building on these findings, we have devised a proper way of representing flat wisps. We will detail it in the rest of this section.

Representation

Wisp-based methods normally represent a centre or skeleton of the wisp in their dynamic simulation. We take a different approach, representing a wisp with two dynamic strands located at its edges. We call them the *rim strands* of the wisp. In formulae, we use the subscript/superscript L or R to denote quantities belonging to the left and right rim strand, respectively. Left and right are arbitrary identifiers we use to distinguish the strands, and they do not have any specific relation to the strands' position on the scalp.

The wisp itself is then represented as the strip between its rim strands; see Figure 3.14. We define the strands so that they have the same edge lengths. This makes the wisp a strip of quads, although these are not planar in the general case.

By observing real-world hairstyles, we've found that flat wisps tend to form in a certain pattern. On the scalp, the wisp is formed from strands whose roots are arranged in a roughly horizontal line. We mimic this by placing the rim strands' roots at nearly identical "latitude" on the scalp, with about 1–2 cm of "longitudinal" separation. Figure 3.15 shows an example of wisp distribution. The wisp width, as well as latitudinal variation between the two rim strands, are parameters of the hairstyle.

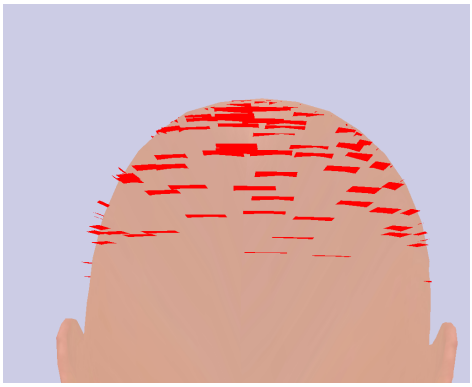


Figure 3.15: Example of flat wisps distributed on the scalp. Each red segment connects the root points of the left and right rim strand of a wisp.

Rendering

We dynamically simulate the rim strands only, but we want to render the entire wisp. One advantage of our wisp representation is that it can be rendered in multiple different ways. We describe them below and provide examples of those we have implemented.

The simplest form is just to render it as a quad strip textured with a hair texture. This is by far the fastest approach, although it leaves much to be desired in terms of visual realism.

A significantly better technique is to actually emulate the individual hair strands being represented by the wisp. There are different ways to effect this, but the core principle is the same: in a fragment shader, compute "virtual geometry" of the hair strands comprising the wisp, and shade accordingly. Figure 3.16 shows an image rendered using a simple tangent map with random perturbations used for representing the strands, combined with a visual shader implementing the hair lighting model of Kajiya and Kay (1989).

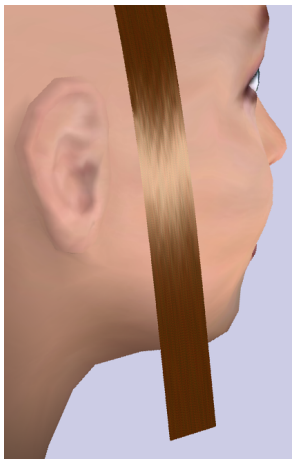
Another simple form of representing the non-simulated strands could be based on the U-shaped strips originally introduced by Liang and Huang (2003) to add volume to strip-based hair representation. Other options for simulating the virtual strands include procedural computation, potentially influenced by the current dynamic state.

An advanced shader-based technique is applying relief mapping (Policarpo et al. 2005; Policarpo and Oliveira 2006) to the wisp to actually compute the full virtual geometry, including occlusions and shadows; see Figure 3.17 for an example. The relief texture can of course be varied by random quantities and/or fed dynamic data from the simulation.

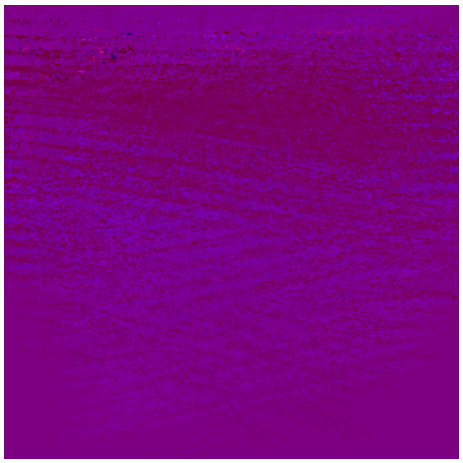
Finally, if we have the resources to spare, nothing stops us from interpolating real strands from the rim guides and rendering them as individual strands.

It is also of course perfectly possible to combine one or more of the techniques mentioned above, such as rendering the wisp using tangent mapping and adding a few interpolated strands to break uniformity.

Combining the techniques can also be used as a level-of-detail scheme. Generic level-of-detail approaches, such as those based on distance, performance, or rendered area, can be applied easily. Since we know the topology of the wisps and



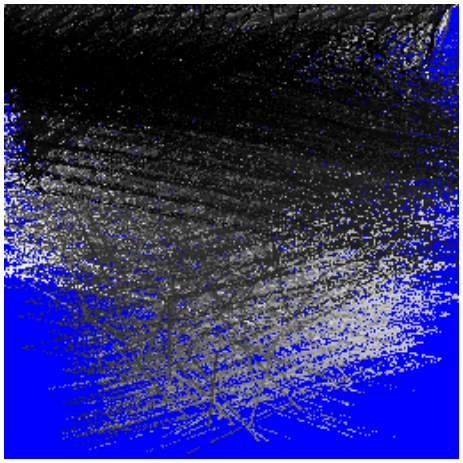
(a) Tangent-mapped wisp



(b) Tangent map used

Figure 3.16: Wisp rendered using a tangent map (a). The Kajiya-Kay hair lighting model uses the tangent instead of the normal, so this is equivalent to normal mapping for polygonal meshes. The texture specifies the tangent vector of the virtual strand in tangent space of the wisp's segment. The tangent map texture is shown in (b). Latitudinal component is encoded in the red channel, normal component in blue. In the real texture, these variations are minor compared to the longitudinal component; the colours have been scaled to increase contrast for visualsation.

(a) Relief-mapped wisp



(b) Relief map used

Figure 3.17: Wisp rendered using relief mapping (a). A depth map is used for modifying the 3D position of each fragment by simple ray casting in the fragment shader. The image was rendered by combining a depth map with the tangent map from Figure 3.16b. In the depth map visualisation (b), grayscale encodes depth, with black being closer to the surface. For this visualisation, empty texels (maximum depth) are rendered in blue instead of white.

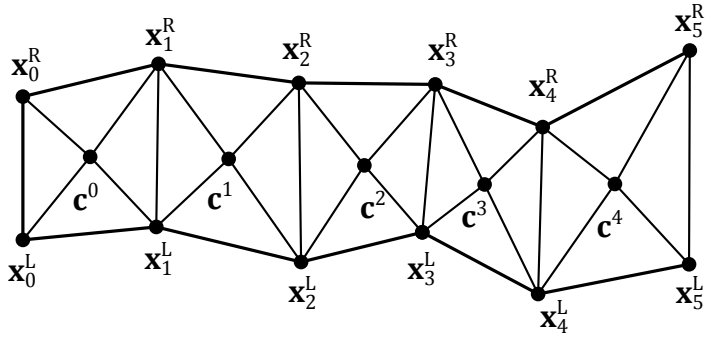


Figure 3.18: Mesh used for rendering a wisp, comprised of triangle fans around centroids of segments.

their layering within the hairstyle, we can also apply worse rendering to wisps—or their parts—which are close to the scalp and likely to be obscured by further hair.

Notice that most of the techniques above treat the wisp as a textured and/or shaded strip of primitives. While the wisp is essentially a strip of quads, these are usually not planar, which is a problem when rendering using a rasterisation library such as OpenGL. We solve this by breaking down each quad into a fan of four triangles by computing a centroid point for it. The centroid is computed as a simple average of the four nodes comprising the quad, see Figure 3.18. This centroid & triangle fan structure is used not only for rendering, but also for hair–hair collision detection, as we will describe in Chapter 4.

Dynamics

Simply rendering the wisp between the rim strands without modifying their behaviour in any way would quickly lead to unsatisfactory results, as the strands could drift apart or even cross over, which real wisps normally do not do. We therefore need to introduce a dynamic representation of the wisp as well. Studying the behaviour of real hair in hairstyles which form flat wisps, we've made the following observations:

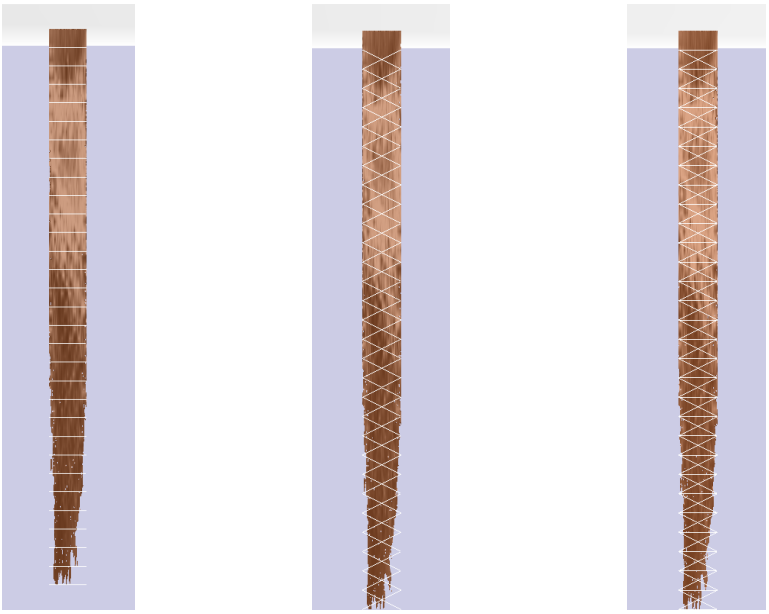
1. Wisps stretch or compress somewhat in their cross section, and this deformation can vary along the wisp's length.

2. It is very rare for a single wisp to come apart completely, and usually requires interaction with an external object such as a comb.

3. Wisps can become entangled with each other, but these tangles are usually weaker than those which hold the hair strands in a wisp together.

We want to find a model for this behaviour which will capture it with sufficient accuracy without introducing too much extra work into the simulation. As our integration method relies on low equation stiffness, we must also choose a model which will not violate this precondition.

Point 3 concerns interaction between different wisps. It implies that we do not need to consider wisps merging permanently, but otherwise does not directly



(a) (b) (c)

Figure 3.19: Different patterns of wisp springs between nodes of the rim strands, with springs rendered as white lines.

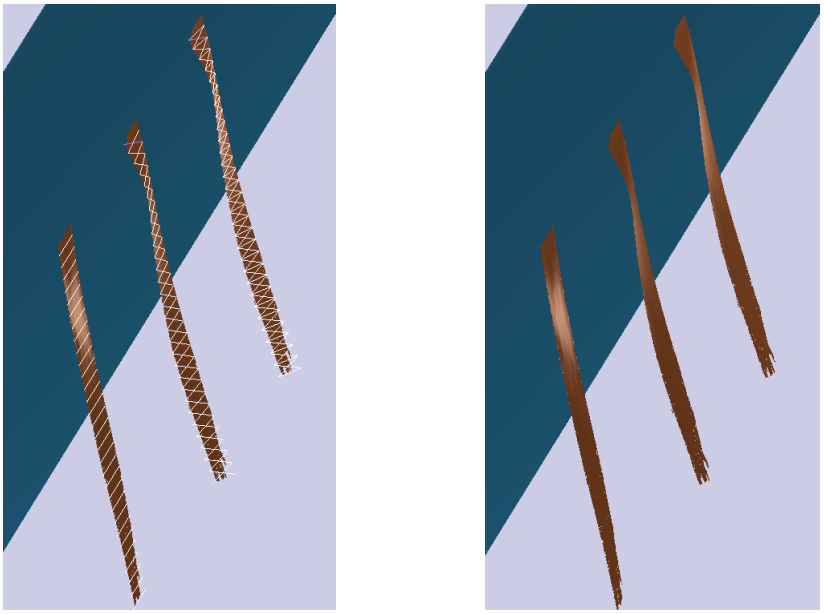
affect the dynamic model of a single wisp. We therefore defer representing this behaviour to Chapter 4, where we describe our hair–hair collision handling. Together with point 2, they also tell us that we can generally treat the set of existing wisps as stable.

We go even further and treat the number, distribution, and identity of wisps as properties which are defined at initial scene set-up and do not change during the simulation. This is a simplification to save on having to somehow detect when wisps merge. Implementing such a merge could also destabilise the simulation. Once we have prohibited wisps from permanently merging, we also prohibit them from splitting up as that would then be an irreversible change.

Note that this is a deliberate design decision rather than an essential characteristic of our wisp representation. It would be entirely possible to add wisp splitting/merging functionality; we simply choose not to do that for performance reasons. In this, we rely on observations 2 and 3 above as assurance that such occurrences are rare in real hair and we are thus not missing important behaviour. We are nonetheless aware that applying this choice does somewhat limit the scope of situations we can represent; if we were to simulate a hairstyling process, for example, wisp splitting and formation would definitely have to be added.

Having dealt with inter-wisp behaviour, we still need to find an appropriate dynamic model for the wisp itself. We notice that point 1 can easily be modelled by connecting the rim strands with springs; we call these *wisp springs*. This approach is similar to the static links use by Chang et al. (2002), and was partially inspired by them.

If we keep the wisp springs' stiffness low, they will allow all the deformations we want to capture, such as the wisps expanding, compressing, or twisting, while at the same time preventing the wisp from coming apart altogether. Low stiff-



(a) (b)

Figure 3.20: Effect of different wisp spring patterns on wisp deformation; the spring patterns are arranged the same as in Figure 3.19. Notice how the left-hand wisp "shears" while its springs remain at rest length; this does not correspond to real hair behaviour. The diagonal springs in the other two configurations lead to the wisps twisting correctly under the deformation (here, gravity pull). Spring compression is rendered as a blue tint proportional to the compression, which can be be observed near the root of the two right-hand side wisps (a). The presence of latitudinal springs in the right-hand wisp increases its rigidity, causing it to twist less. This can be best observed in the amount of specular reflection present in the hair in (b).

ness will also keep the equations integrable with our chosen method. The exact stiffness value is related to the hair's tendency to entangle. The rest length of the wisp springs is computed from the length they achieve when the rim strands of the wisp are in their rest configuration.

We have experimented with several patterns of connecting the rim strands with wisp springs; a few notable ones are shown in Figure 3.19. The most basic configuration simply connects with the same indices (Figure 3.19a); we call such springs *horizontal*. This has the advantage of having relatively few springs to compute; on the downside, it is possible for the wisp to "shear" with the springs staying at rest length (see Figure 3.20). Such a deformation does not correspond to real wisp behaviour and disturbs the visual impression from the simulation. To correct for this, it is advisable to introduce diagonal springs (Figure 3.19b). The wisp's shear would compress or expand these; the resisting force they exert on the wisp leads to a more natural deformation, also shown in Figure 3.20.

It is also possible to combine both of these spring patterns, as in Figure 3.19c. This increases the overall rigidity of the wisp, so adding the horizontal springs can be used as an extra parameter controlling behaviour of the simulated hair.

Next, we formalise the inclusion of wisp springs in the system dynamics. We use $\mathcal{S}$ to denote the set of springs attached to node i. Each spring j in this set has the following parameters: indices a_j^{L} and a_j^{R} of the nodes connected by the spring, stiffness $\underline{k}_j$, and rest length $\hat{\underline{d}}_j$ computed as follows:

$$\hat{\underline{d}}_j = \left\| \hat{\underline{\mathbf{x}}}^{\mathrm{L}}_{a_j^{\mathrm{L}}} - \hat{\underline{\mathbf{x}}}^{\mathrm{R}}_{a_j^{\mathrm{R}}} \right\| \tag{3.173}$$

Wisp springs are created and attached as part of scene set-up and do not change during the simulation. We incorporate them into the equations of motion (equation 3.134) in a straightforward way by adding them to the external force term:

$$\mathbf{F}_i^{\mathrm{springs}} = \sum_{j=1}^{|\mathcal{S}|} -\underline{k}_j \frac{\left\| \mathbf{x}^{\mathrm{L}}_{a_j^{\mathrm{L}}} - \mathbf{x}^{\mathrm{R}}_{a_j^{\mathrm{R}}} \right\|}{\hat{\underline{d}}_j} \left(\mathbf{x}^{\mathrm{L}}_{a_j^{\mathrm{L}}} - \mathbf{x}^{\mathrm{R}}_{a_j^{\mathrm{R}}} \right) \tag{3.174}$$

The same equation is also used for springs representing mutual entanglement of two wisps, as discussed in Section 4.2.

3.7 GPU Implementation

One of the goals of our hair animation method is to provide good performance on hardware on par with a higher-end consumer workstation. Fully programmable Graphics Processing Units (GPUs) have been a part of this class of hardware for several years now. These units present a vast resource of floating-point processing power. Being aware of this, we have designed our method for easy and efficient implementation on a modern GPU, and proven this concept by implementing a subset of the simulation as a closed-loop GPU system. In this section, we will describe the details of this implementation, concentrating on specifics which make the implementation optimised for the GPU's computation model.

3.7.1 CUDA

The technology we use to implement our method on the GPU is nVidia's CUDA. We give a brief overview of the relevant parts of this framework in this section; for more information, please refer to (NVIDIA Corporation 2015).

Computation Model

CUDA's computation model is based on the notion of a *kernel,* which is a function started from host (CPU) code, but executed on the GPU (a "device" in CUDA terminology). Each invocation of a kernel executes the same device code in a number of device threads. Threads are organised into *blocks* of one, two or three dimensions.

The GPU's cores are grouped into *multiprocessors.* The exact number of cores depends on the particular device, usually ranging in the tens. Each thread block is assigned to one multiprocessor and executes on that multiprocessor only. The threads in a block all have access to an area of *shared memory* dedicated to that block. It is possible for more than one block to be assigned to the same

multiprocessor, but the multiprocessor's resources (registers and shared memory) have to be split between blocks—resources are allocated to blocks for the entire duration of the kernel invocation. Using too many resources can therefore reduce opportunities for parallelism by reducing the number of blocks which can be assigned to a multiprocessor.

Threads in a block are split into *warps* for execution. Each warp consists of 32 threads, consecutive within the block structure. In one processing step, all threads in a warp must perform the same instruction. If that is not possible because of e.g. branching, some threads are masked out and the divergent branches are performed sequentially, reducing processor utilisation. It is therefore desirable to structure code so that branching is aligned with warp size: all threads in a warp should ideally follow the same branch.

Blocks themselves can be arranged into a one- or two-dimensional grid. This is largely organisational—while threads within one block can communicate through shared memory and mutual synchronisation, there is very little synchronisation possible between threads in different blocks.

Memory Model

There are different memory areas available to device code, with different access characteristics.

The fastest memory are registers, unique to each thread. The GPU has a large register pool to support its purpose as a fast massively parallel computing device. Of similar speed as registers is shared memory, an area accessible to all threads in a block.

The main memory of the GPU is called global memory. Compared to shared memory and registers, access to global memory is significantly slower and as such has to be optimised so as not to degrade overall performance. Alignment and access locality is also important in global memory, because accesses happen in transactions. A single transaction generally reads or writes 128 contiguous bytes from a 128-byte aligned address; details depend on the capabilities of the actual hardware device. Newer GPUs offer an automatically managed L1 and L2 cache for global memory.

In addition to these general-purpose memory areas, there are two special types of memory. Constant memory is a an area of read-only memory optimised for frequent reads of small pieces of data, with its own dedicated cache; cache misses have the same latency as global memory. Texture memory is accessed through the GPUs texturing units. It is cached, optimised for spatial locality of reads, and can be accessed with automatic linear interpolation in the same way textures are accessed in graphics processing. Just like constant memory, it can only be written from host code, and its latency in case of a cache miss is the same as for global memory.

It should be noted that using *graphics interoperability,* CUDA allows kernels both reading and writing access into OpenGL Buffer Objects stored in GPU memory. Access to these is no different from normal access to global memory.

Execution Model

While GPUs are programmed in common programming languages (C++ in the case of CUDA), the underlying hardware architecture is different enough from today's CPUs to make it necessary to consider entirely different paradigms with regards to performance. CPUs generally try to compensate for stalls dues to branching, memory latency, and similar occurrences by sophisticated instruction scheduling strategies, out-of-order instruction execution, memory pre-fetching, branch prediction etc. Basically, a desktop CPU consists of units (or tens at best) of cores, relying on their sophistication for performance. Similarly, CPU threads are heavyweight—switching context from one thread to another is generally an expensive operation which requires saving and restoring register state, memory maps, and other information.

GPUs take a very different approach. Where a CPU is sophisticated, the GPU is designed to be simple and streamlined. A GPU processor is a simple in-order pipeline processor. Any performance benefits which can be obtained by instruction re-ordering, such as exposing instruction-level parallelism, have to be discovered and implemented by the compiler when compiling the kernel code. The GPU itself will execute the code in exactly the order the compiler produced it.

GPU threads are lightweight as well; the entire system is designed so that a context switch is an extremely fast operation. All of that is because GPUs follow a different philosophy regarding latency. They are designed for scenarios where the number of threads is much bigger than the number of cores, and a lot of these threads follow the same or very similar paths of execution. The individual cores are simple, but the GPU derives its performance from the sheer number of them—hundreds or thousands of cores on a mid-range through high-end GPU board. Any instruction or memory latency is then masked by switching out the blocked threads and executing other ones in the meantime. To fully utilise a GPU, enough threads must always be available for execution to mask latency.

Performance

In summary, these points are important for getting good performance from the GPU:

- Have enough threads and enough work to do on them to mask instruction and especially memory latency.
- Access global memory within contiguous, aligned blocks, to prevent access fragmenting into multiple transactions.
- Minimise access to global memory, preferring storage in shared memory or registers.
- Organise branching in code to minimise the number of divergent warps (warps where threads follow different branches).

Adhering to these principles generally requires adapting the block size, for things like memory access alignment of branch divergence. When constructing blocks,

resource requirements should also be considered, to potentially allow more blocks to share a multiprocessor.

While we use CUDA for our implementation, it would be a matter of straightforward programming work to port the implementation to a different GPU-programming API such as OpenCL. The optimisations we have presented in this section are aimed at GPU architecture in general and are not tailored to specifics of CUDA in any way. It should be noted that our method is therefore not tied to a particular API of a particular hardware vendor.

3.7.2 Our GPU Processing Pipeline

We have implemented the basic simulation loop on the GPU, using several kernels (Kmoch et al. 2010). This implementation includes integration of strand dynamics, constraint enforcement, and updating material frame orientation. All data required for these computations and for rendering is always present in the GPU memory. This means that when not considering hair–hair collisions, the implementation forms a closed-loop GPU system.

We have not implemented hair–hair collision detection on the GPU. Collision detection is a task which is generally not well suited to the architecture of today's GPUs, because it necessarily requires accessing data at random locations in memory. Even so, GPU algorithms for collision detection acceleration do exist (Le Grand 2007). It should therefore be possible to implement this part of our hair animation method on the GPU as well, thus arriving at a truly GPU-only solution. How well the GPU could accelerate collision detection of the many hair primitives normally found in close proximity during hair simulation would have to be subject to further investigation.

An alternative approach to this problem would be to utilise the fact that our simulation uses a short time step (usually 1–2 ms) and perform collision detection on the CPU in parallel to GPU-based simulation. In effect, the CPU would be detecting collisions in the system configuration at time t_n in parallel to the GPU computing the new state t_{n+1}. Detected collisions would then be applied to the next step. This way, collision response would be one step "behind" the simulation, but thanks to the short time step and thus minor changes in system state in one simulation step, we would not expect any noticeable artefacts. We have not implemented this mixed CPU/GPU approach, but believe it viable and consider it a good direction for future evolution of our method.

In the remainder of this section, we describe our GPU implementation.

Rendering Data

Our GPU simulation is coupled with rendering of individual simulated strands as camera-facing billboards. Some hair data, most importantly positions of centre-line nodes, must be shared between the simulation and rendering. Hair shading also generally requires the directional vector of the strands, so tangents must be accessible to rendering as well. We store such shared data in an OpenGL vertex buffer object (VBO) and use CUDA graphics interoperability to access it from the kernels.

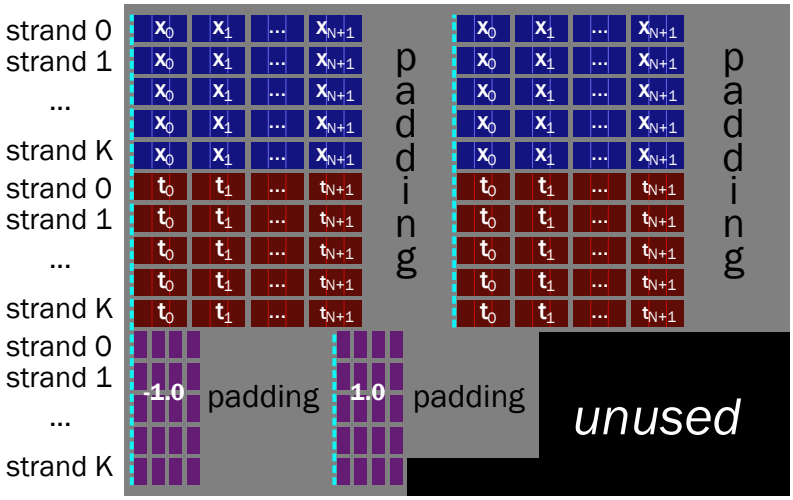


Figure 3.21: Layout of vertex buffer object storing data which is computed by the simulation kernels and accessed during rendering. The layout was chosen to provide a good access pattern to the kernels: the dashed cyan lines indicate 192-byte alignment boundaries which make the data properly aligned both for CUDA kernel access and OpenGL rendering.

The camera-facing billboard used for strand rendering are computed in the vertex shader. For this purpose, two geometry vertices are assigned to each centreline node. During simulation, the coordinates of both of them are updated to move along with the node.

When rendering, the vertex shader is used to project them using the ordinary model–view matrix; then, the projected vertices are offset slightly in the projection plane, providing a very efficient way of obtaining a camera-facing billboard in the shape of the strand.

In addition to node positions and tangent vectors, this also requires an "offset" attribute for each vertex. The value of this attribute is -1 or 1 and it controls the direction of the offset along the $\mathbf{x}$ axis of the projection plane. Figure 3.21 describes the data layout of a vertex buffer object used for individually rendered strands.

Kernel Overview

Computing one simulation step is split into multiple kernels based on thread configurations and memory size required to compute the necessary pieces of data. The individual phases are:

1. **Integration kernel**, which evaluates forces and computes new node positions.

2. **Constraint kernel**, which computes the values of constraints and their gradients. This kernel is called once for each constraint enforcement iteration (see below).

3. **Curvature binormal kernel**, which computes curvature binormals, their gradients, and the gradient of holonomy.

4. **Twist kernel**, which computes twist and updates the material frame.

5. **Bending kernel**, which computes bending.

6. **Bending gradient kernel**, which computes bending gradient.

Each of these kernels is described in more detail below. For each of them, we specify its interface: *Arguments* describe value objects (such as a scalar or a `float3` object) which are passed as parameters directly to the kernel. *Inputs* are data stored in device memory, which the kernel reads through pointers passed as parameters. *Outputs* are likewise stored in device memory, but we use it for data which the kernel writes into memory. It is perfectly possible for a kernel to both read and write the same data, in which case we list it in both inputs and outputs.

Integration Kernel

Arguments: $\Delta t, \mathbf{g}$
Inputs: $\mathbf{x}, \dot{\mathbf{x}}, \boldsymbol{\omega}, \hat{\underline{\boldsymbol{\omega}}}, \underline{\mathrm{B}}, l, \nabla\boldsymbol{\omega}$
Outputs: $\mathbf{x}, \dot{\mathbf{x}}$

The integration kernel evaluates forces acting on the strand and integrates the equations of motion, computing the (unconstrained) position and velocity of nodes. Each thread block computes one strand. Block size is $3 \times (N + 2)$, so that each thread deals with one component of a node's position or velocity (which are vectors of size 3). A thread computing the c-th coordinate of node i thus computes the c-th coordinate of the force using equation 3.104:

$$\nabla_i U[\![c]\!] = \sum_{k=1}^{N} \frac{1}{\hat{\underline{l}}_k} \sum_{j=k-1}^{k} \left(\nabla_i \boldsymbol{\omega}_k^j\right)^{\mathrm{T}} [\![c\,0, c\,1]\!]\, \underline{\mathrm{B}}^j \left(\boldsymbol{\omega}_k^j - \hat{\underline{\boldsymbol{\omega}}}_k^j\right) \tag{3.175}$$

There are two interesting points in this kernel, both related to memory organization. The kernel's input consists of these arrays: bending $\boldsymbol{\omega}_i^j$, rest-state bending $\hat{\underline{\boldsymbol{\omega}}}_i^j$, bending matrices $\underline{\mathrm{B}}^j$ and domain length $\hat{\underline{l}}_i$; $13N + 22$ floats in total. If we make use of the symmetry of $\underline{\mathrm{B}}^j$, we reduce the size to $12N+21$ floats. By loading a `float4` per GPU thread, the entire block can read $12N + 24$ floats from global memory in parallel. So storing this data in one continuous array for each strand leads to efficient, fully coalesced reads.

The kernel also reads $(\nabla_i \boldsymbol{\omega}_j^k)^{\mathrm{T}}$, a 3D array of 3×2 matrices, which doesn't fit into shared memory. However, we notice that each element of $(\nabla_i \boldsymbol{\omega}_j^k)^{\mathrm{T}}$ is used by exactly one thread, and exactly once (the c-th thread of node i uses the c-th row once for each j and k). Therefore, when arranged correctly, the elements can be read directly from global device memory when needed. Coalesced reads are achieved by storing the gradient as a 4D array of `float2` rows, with dimensions sorted like this, from the fastest varying index to the slowest: column, i, j, k. Note that this is the opposite of the "natural" storage of a 4D array.

Constraint Kernel

Arguments: $\Delta t, \underline{H}^2$
Inputs: $\mathbf{x}, \|\underline{\hat{\mathbf{e}}}\|^2$
Outputs: $\mathbf{CI}, \mathbf{CH}, (\nabla \mathbf{C})^{\mathrm{T}}, \underline{\mathrm{M}}^{-1} (\nabla \mathbf{C})^{\mathrm{T}}$

The constraint kernel is called once for each iteration of constraint enforcement; inextensibility constraints (equation 3.126) and equality-based hair–head collision constraints (equation 3.147) are supported. Each thread block computes one strand. Block size is $3 \times (N + 2)$; the motivation is that each thread can compute one component (column) of the constraint gradient matrix $\nabla \mathbf{C}$. The kernel starts by computing the values of all constraints and comparing them against the convergence threshold. Logical AND is applied to the results of these comparisons via parallel reduction. If all constraints are satisfied, the kernel terminates.

Otherwise, the kernel computes matrices which will then be used by the linear equation solver. The fist of these is the matrix of constraint gradients $\nabla \mathbf{C}$ of size $\left(N + |\mathcal{P}|\right) \times 3\,(N + 2)$. Notice that while the number of columns of the matrix is fixed, the number of its rows is not known when the kernel starts, because it depends on the number of nodes which collide with the head (these collisions are detected by the kernel itself while computing constraints). To access the matrix in a coalesced manner, however, we would need neighbouring threads to access neighbouring matrix rows. For this reason, the kernel actually computes the transpose of the constraint gradient matrix, $(\nabla \mathbf{C})^{\mathrm{T}}$. At the same time, the premultiplied matrix $\underline{\mathrm{M}}^{-1} (\nabla \mathbf{C})^{\mathrm{T}}$ is also computed.

It is for this computation that the size of the block was chosen, because $3\,(N + 2)$ is the number of the matrices' rows. This is important, as the matrices are only written to once and therefore stored directly in global memory. Aligning the block size with matrix size thus allows the threads to fill the matrices in a simple loop with perfectly coalesced memory accesses.

When this kernel finishes execution, two outcomes are possible. One is that the kernel exited early because all constraints were satisfied (beneath the convergence threshold). In such case, constraint enforcement is over.

The other possible outcome is that at least one constraint was violated, in which case a step of minimisation must be performed. We do this using CULA[3], a GPU implementation of the LAPACK linear algebra library. Remember that all matrices involved are already present in the GPU memory, having just been computed by this kernel.

Curvature Binormal Kernel

Arguments: none
Inputs: $\mathbf{x}, \|\underline{\hat{\mathbf{e}}}\|$
Outputs: $\kappa\mathbf{b}, \nabla\kappa\mathbf{b}, \nabla\Psi$

In addition to computing curvature binormals, this kernel takes advantage of the fact that it has curvature binormal data in shared memory, and also computes other quantities for whose computation curvature binormal is one of the inputs.

[3] `http://www.culatools.com`

Each thread block computes one strand; block size is $3 \times (N+2)$, so that each thread deals with one component of a node's position.

The kernel runs in three phases. In the first one, $\kappa\mathbf{b}$ and $\nabla\kappa\mathbf{b}$ are computed (using equation 3.75 and equations 3.110–3.112) and stored in shared memory. In the second phase, $\nabla\psi$ is computed using equation 3.77 and stored in shared memory as well. The data is laid out so that $\nabla\psi$ is at the start of shared memory, with curvature binormals and their gradients following. Finally, $\kappa\mathbf{b}$ and $\nabla\kappa\mathbf{b}$ are written to global memory.

The third phase computes $\nabla\Psi$. The only input to this computation is $\nabla\psi$, stored at the start of shared memory. The kernel therefore re-uses all shared memory space beyond that to store the large 2D array which is $\nabla\Psi$. When computed, it is written to global memory.

Notice that $\nabla\psi$ is never needed outside of this kernel and is thus never stored in global memory.

Twist Kernel

Arguments: none
Inputs: $\mathbf{x}, \mathbf{u}_1^0, \mathbf{t}^0, \kappa\mathbf{b}, \hat{\underline{\tau}}$
Outputs: $\mathbf{u}_1^0, \mathbf{t}, \mathbf{m}_1, \mathbf{m}_2$

The twist kernel computes the material frame, using the original version of the algorithm from (Kmoch et al. 2009). Each thread block computes one strand, with block size $3\times(N+2)$: one thread for each component of a 3D vector assigned to a node or an edge.

The kernel starts by updating the Bishop axis $\mathbf{u}_1^0$ at the root based on the value of $\mathbf{t}^0$ from the previous integration step using equation 3.80. The rest of the Bishop frame is then computed into shared memory; it is local to this kernel.

With the Bishop frame ready, tangents are computed and the algorithm for computing θ is applied. Finally, $\mathbf{m}_1$ and $\mathbf{m}_2$ are computed, and all output data written to global memory.

Bending Kernel

Arguments: none
Inputs: $\kappa\mathbf{b}, \mathbf{m}_1, \mathbf{m}_2$
Outputs: $\boldsymbol{\omega}$

The bending kernel is extremely straightforward, computing bending using equation 3.82. Each thread block computes bending for one strand. Block size is $3 \times (N+2)$, to guarantee coalesced access to input data.

Recall that for each node i, there are two bending vectors $\boldsymbol{\omega}_i^{i-1}$ and $\boldsymbol{\omega}_i^i$. Being 2-vectors, they are stored in memory as one `float4` vector.

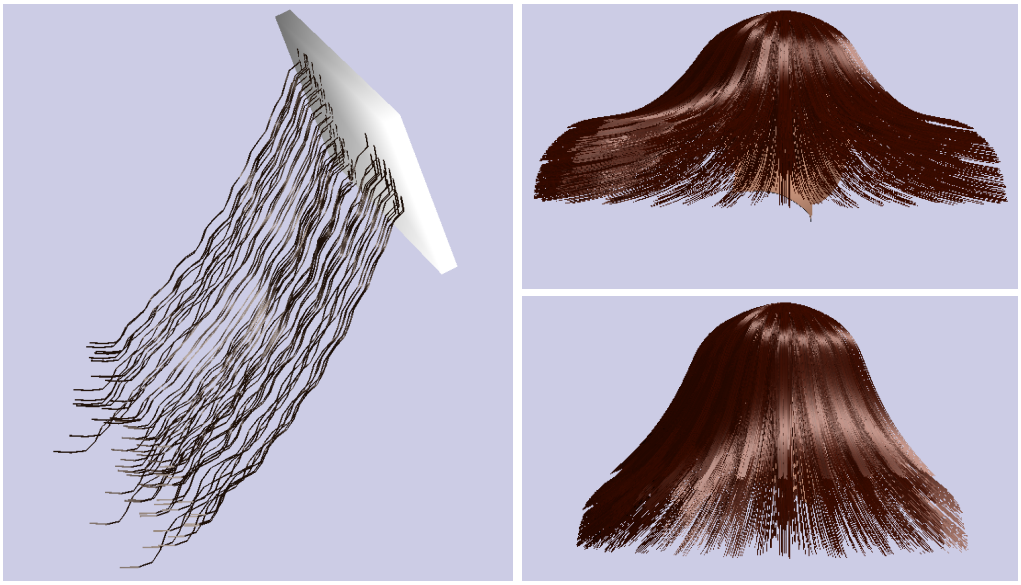


Figure 3.22: Hair simulated and rendered by our GPU implementation.

Bending Gradient Kernel

Arguments: none
Inputs: $\kappa\mathbf{b}, \mathbf{m}_1, \mathbf{m}_2, \boldsymbol{\omega}, \nabla\Psi$
Outputs: $\nabla\boldsymbol{\omega}$

The bending gradient kernel computes $\nabla\boldsymbol{\omega}$. We've introduced the optimum memory layout for it when discussing the integration kernel. Each thread block computes one strand. In this kernel, we use 3-dimensional blocks of $3 \times 3 \times Z$ threads. The 3×3 part of the block size is related to the structure of $\nabla\kappa\mathbf{b}$ data—for each j, k pair, there are only 3 values of i such that $\nabla_i \left(\kappa\mathbf{b}\right)_k j$ is non-zero (per equations 3.110–3.113); this is covered by one block dimension. The other is simply for working with 3-dimensional vectors.

The entire 4D array of 2-dimensional vectors which stores $\nabla\boldsymbol{\omega}$ does not fit into shared memory and is too large to be computed by block threads directly, which means iteration in the kernel is necessary. The thread block dimension Z is therefore chosen for maximum utilisation. On CUDA devices with block size limited to 512 threads, maximum Z is 56 (as $3 \times 3 \times 57 = 513 > 512$). On newer compute capabilities, the maximum value of Z is 64 (which is the maximum $\mathbf{z}$ size of a thread block). $\nabla\boldsymbol{\omega}$ is then computed in blockwise iteration, Z columns at a time.

Discussion

Our implementation serves to demonstrate that the method we have chosen is suitable for implementation on the specific architecture of modern GPUs. While not fully optimised, it does outperform our CPU implementation on larger scenes, where the massively parallel nature of the GPU pays off. Figure 3.22 shows several screenshots from an animation computed and rendered fully on the GPU; hair–hair collisions are not detected.

Chapter 4

Handling Hair–Hair Collisions

In Section 3.2, we have discussed how important collective hair effects are in shaping overall hair behaviour. The most important of these effects is the constant contact between individual hair strands. We capture a lot of these effects in the very efficient abstraction of combining hair strands into flat wisps instead of simulating their contact individually. The wisps themselves are still subject to collisions, however, and we need to handle these. We present our method in this chapter.

4.1 Flat Hair Wisps

In Section 3.2.1, we have presented our observation that in a large class of hairstyles, hair strands tend to clump into more or less flat wisps. We will elaborate on it here, because it forms a cornerstone of our collision handling scheme. The core observation is that hair tends to form "layers" or "sheets" on the head, in effect clustering along the latitudinal direction. These layers are then usually broken down into wisps along the longitudinal direction. While these are not perfectly flat, their longitudinal dimension is much larger than their cross-section thickness. See Figure 4.1 for examples of hairstyles where this phenomenon occurs. Notice that these cover a large class of hair varieties—long and short, straight and wavy.

For our hair–hair interaction scheme, we restrict ourselves to hairstyles composed entirely from such flat wisps. Furthermore, we take the wisps as primitives of our simulation; they are created as part of model set-up and we do not allow them to form up or split during simulation runtime. This is certainly a simplification, but it allows us simulate a full hairstyle efficiently. Importantly, this simplification is not arbitrary, but based on modelling an actual phenomenon readily observed in real hair.

Our method is not the first one to use triangle or quad strips in hair simulation; similar approaches have been taken by Daldegan et al. (1993), Chang et al. (2002), Liang and Huang (2003), and Ward et al. (2003). However, to the best of our knowledge, we are the first to use them for modelling actual behaviour of real hair, as opposed to just an arbitrary model reduction. This is reflected in capturing the wisp behaviour in the dynamics of the simulation, and in our specific collision responses presented later in this chapter.

Figure 4.1: Various hairstyles showing flat wisp formation[1].

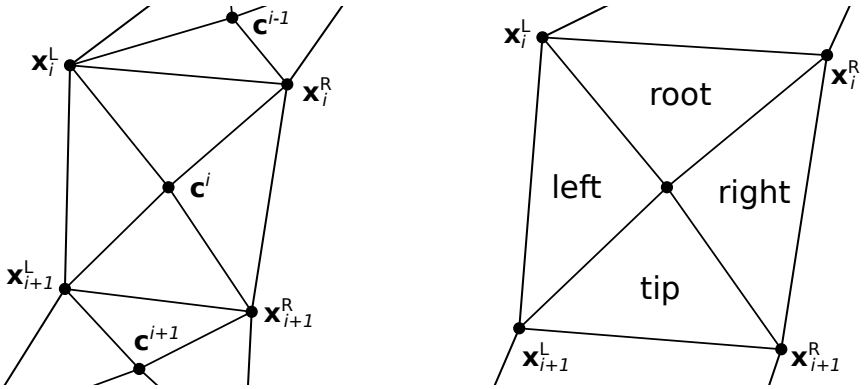
4.1.1 Representation for Collision Detection

Our representation of these flat wisps was sketched out in Section 3.6.3. For the purposes of simulating dynamics, the wisp consists of a pair of rim strands connected by springs. For collision handling, we want to view the wisp as a continuous strip delimited by the rim strands, like a ribbon. Edges of the rim strands form natural segmentation of the wisp; segment i is formed by the four vertices $\mathbf{x}_i^{\mathrm{L}}$, $\mathbf{x}_{i+1}^{\mathrm{L}}$, $\mathbf{x}_{i+1}^{\mathrm{R}}$, $\mathbf{x}_i^{\mathrm{R}}$.

Because the rim strands are only loosely coupled through wisp springs but otherwise act independently, the segments of the wisp are seldom planar. Indeed, they are a simplification of the actual shape of hair strands comprising the wisp, each of which deforms slightly differently. The overall shape remains that of a thin wisp, but that while we describe the wisps as "flat," they are not 2-dimensional. Their thickness is small compared to their longitudinal size, but still non-zero. To model wisp interactions accurately, we need to capture the shape and volume of the wisps somehow. Because we only compute dynamics of the rim strands, our simulation does not provide us with any information about the shape of the hair strands between the rims. We therefore need to approximate it somehow. We however know that it will roughly remain a flat wisp. We therefore create

[1]Image sources and licenses:

- Top left-hand image: By ␣joshuaBentley, licensed under Creative Commons BY-ND-2.0, taken from `https://www.flickr.com/photos/joshwachaos/281027390/in/photostream/`.
- Bottom left-hand image: By Carlos Espinoza Leon, licensed under Creative Commons NC-BY-SA-3.0 , taken from `http://desmatrix.deviantart.com/art/Flowing-hair-exercise-134659516`.
- Middle image: By Girls hair, licensed under Creative Commons BY-SA-3.0, taken from `http://commons.wikimedia.org/wiki/File:Beautiful_healty_hair.JPG`.
- Right-hand image: By Titus Tscharntke, in the public domain, taken from `http://commons.wikimedia.org/wiki/File:Blonde_hair_detailed.jpg`.



(a) Skeleton triangles (b) Triangle names

Figure 4.2: The skeleton structure of a wisp segment.

a skeleton structure for the wisp; first, the centroid point of each segment is computed:

$$\mathbf{c}^i = \frac{1}{4}\left(\mathbf{x}_i^{\mathrm{L}} + \mathbf{x}_{i+1}^{\mathrm{L}} + \mathbf{x}_{i+1}^{\mathrm{R}} + \mathbf{x}_i^{\mathrm{R}}\right) \tag{4.1}$$

The skeleton for the segment then consists of a fan of four triangles sharing the centroid as their apex (see Figure 4.2a). We introduce names for the individual triangles comprising a segment (see also Figure 4.2b):

Triangle	**Name**
$\triangle \mathbf{x}_i^{\mathrm{L}} \mathbf{c}^i \mathbf{x}_i^{\mathrm{R}}$	root triangle
$\triangle \mathbf{x}_{i+1}^{\mathrm{L}} \mathbf{c}^i \mathbf{x}_i^{\mathrm{L}}$	left triangle
$\triangle \mathbf{x}_{i+1}^{\mathrm{R}} \mathbf{c}^i \mathbf{x}_{i+1}^{\mathrm{L}}$	tip triangle
$\triangle \mathbf{x}_i^{\mathrm{R}} \mathbf{c}^i \mathbf{x}_{i+1}^{\mathrm{R}}$	right triangle

Notice that we consistently mark the triangles in counter-clockwise order when looking down the negative minor axis of the wisp, with tangent pointing down (this orientation is also used in Figure 4.2).

Chang et al. (2002) also use triangle strips between simulated strands for collision handling. For them, however, the strip is an auxiliary data structure representing non-simulated hair in an abstract way. Hair–hair collisions are captured by detecting collisions between the auxiliary triangle strips and individual strands, even those interpolated for rendering only. The triangles thus serve as a purely abstract representation of some volume of non-simulated hair. We take a different approach, treating the triangle strip as a concrete representation of a flat wisp of hair. This is also reflected in the fact that we detect collisions between these wisps as first-class participants, where Chang et al. (2002) use triangle strips just to simulate strand–strand collisions in an accelerated way. For this, we also assume the wisps to have volume.

To model this fact that the wisp has non-zero thickness, the actual representation of the wisp is obtained as a swept-sphere volume of the skeleton triangles: centre a sphere of a given radius on each point within the triangle, and take the convex envelope of this set of spheres. The resulting shape is depicted in Figure 4.3. These sphere-swept triangles are used as the wisp's representation for

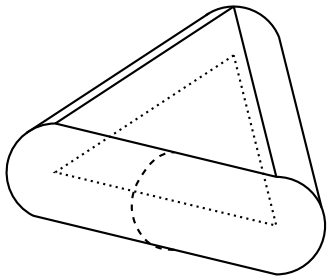


Figure 4.3: Sphere-swept triangle, the basic primitive comprising the collision-handling representation of wisps. The corresponding skeleton triangle is shown at the centre in dotted line.

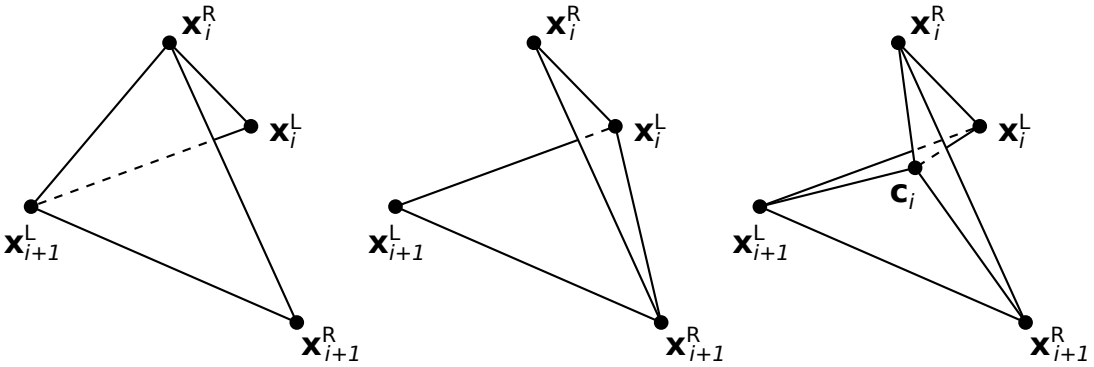
collision detection. We use $\underline{\sigma}^i$ to denote the radius of the sphere swept around triangles in segment i. This radius can vary along the wisp, normally getting smaller closer to the tip. Different wisps can naturally use different radius values.

There are two reasons why we have chosen this representation. The major one is computation efficiency of collision detection. Determining whether two sphere-swept triangles intersect is equivalent to finding the distance between the skeleton triangles, which is a reasonably efficient operation.

The second reason is related to rendering. As we've discussed in Section 3.6.3, many ways of rendering wisps need to represent them as triangle meshes, so it is advantageous if we can reuse elements of the representation both for collision detection and rendering. The easiest way to triangulate a non-planar quad is to choose a diagonal and form two triangles from the quad. However, this has the downside of being asymmetric—the representations of the same segment with the other diagonal chosen can be quite different. See Figure 4.4 for an illustration. Additionally, the centroid-based representation is always smoother than the diagonal-based one. As we know, the wisp stays largely flat during motion, so the smoother representation is also more accurate.

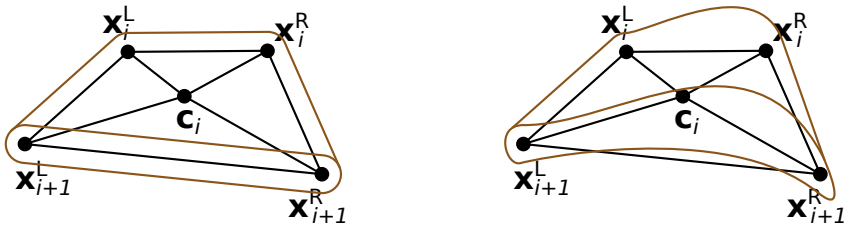
We are fully aware that it is still only an approximation, though. It will smooth out any minor "creases" in the wisp's shape. For example, if the two edges comprising the segment lie in the same plane, the segment will lie in that plane as well, even if the corresponding wisp would arch in its cross section a bit (see Figure 4.5). We choose to ignore this drawback, because we perceive it as minor. The wisp width is generally small enough that the neglected deformations are small as well, and our simulation does not give us any indication of actual wisp shape in between the rim strands anyway. Furthermore, it should be noted that such deformations are only ignored for collision detection purposes. It is still perfectly possible to render the wisp as arched, using virtual geometry techniques such as normal or relief mapping.

The radius of the swept spheres corresponds to thickness of the wisp. Notice that the spheres extend to the sides of the triangles as well as above and below them, so the wisp representation is effectively wider than the distance between the rim strands (by this small margin of half the wisp's thickness). We accept



(a) Diagonal $\mathbf{x}_{i+1}^{\mathrm{L}}$–$\mathbf{x}_i^{\mathrm{R}}$ (b) Diagonal $\mathbf{x}_i^{\mathrm{L}}$–$\mathbf{x}_{i+1}^{\mathrm{R}}$ (c) Centroid

Figure 4.4: Splitting a segment into just two triangles can give vastly different results based on segment diagonal chosen: (a), (b). Splitting using the centroid guarantees consistency: (c). A rather extreme deformation of the segment is depicted, to make the difference obvious.



(a) Collision-detection shape (b) Actual shape

Figure 4.5: If the segment edges are coplanar, the segment is considered flat for collision detection purposes (a), even if real hair in the wisp would arch out of the plane (b).

this slight imprecision for sake of efficiency. The collision detection computation (presented in Section 4.1.2) is substantially simplified by using a sphere-swept volume without exceptional cases. Do not forget that the whole wisp model is a simplification in the first place. It even makes more sense to base the movement of the entire wisp on strands which are not on the absolute edge of the wisp, where they could be more easily influenced by neighbouring hair. Treating the rim strand as slightly inside the wisp, where the wisp interaction dominates, is consistent with the model.

4.1.2 Detecting Collisions

With the wisp representation in place, we have to actually detect when collisions between wisps occur. Because the wisps are generally long and flexible, we have to account for the wisp looping back and colliding with itself as well.

To accelerate collision detection, we use a hierarchy of bounding volumes constructed over the wisp's segments. We use axis-aligned bounding boxes (AABBs) in our implementation, but any other collision detection acceleration structure could be used as well. An axis-aligned bounding box is defined as a rectangular cuboid whose edges are parallel to axes of the coordinate system. It is uniquely defined by two points $\mathbf{a}_{\min}$ and $\mathbf{a}_{\max}$ on opposite ends of its body diagonal, such that $\mathbf{a}_{\min} \leq \mathbf{a}_{\max}$ component-wise.

Our choice of AABBs was motivated by their efficiency: both in terms of update and intersection testing. The trade-offs AABBs normally make for fast update and testing is a lot of false positives, i.e. occasions when the boxes intersect but their actual contents don't. Tighter bounding volumes generally offer less false positives at the cost of computationally more intensive intersection testing and updates. However, we notice that hair tends to be in very tight contact, with many wisps close to each other. We can therefore see that false positives are very likely to occur with all but the tightest-fitting bounding volumes, whose cost to compute an intersection or update could easily be prohibitive in our scenario of short time steps.

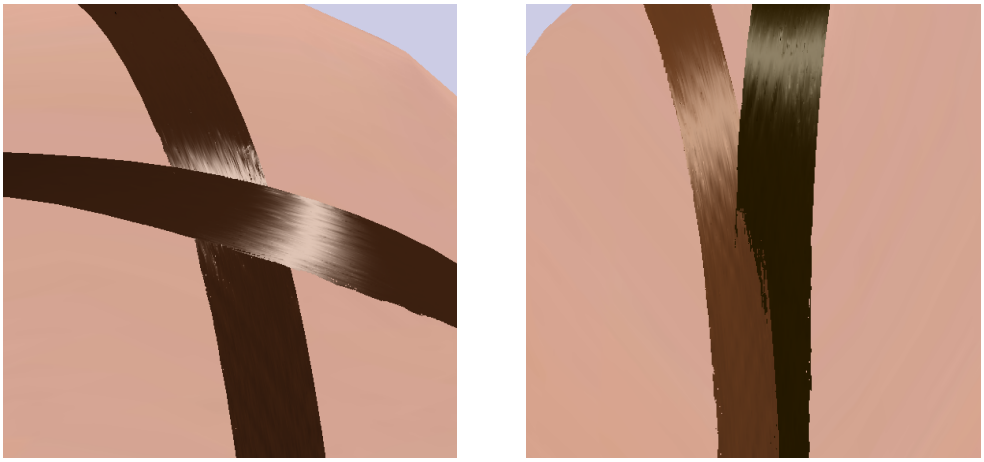
Still, we try to minimise false positives by making use of these specifics of hair when constructing the AABB hierarchy. We start by creating an AABB for each triangle of each segment. Then we build a binary hierarchy above them recursively, creating a parent AABB *par* encompassing its two children 0 and 1:

$$\left(\forall i \in \{0,1,2\}\right) \quad \begin{aligned} \mathbf{a}^{par}_{\min}[\![i]\!] &= \min\left\{\mathbf{a}^{0}_{\min}[\![i]\!], \mathbf{a}^{1}_{\min}[\![i]\!]\right\} \\ \mathbf{a}^{par}_{\max}[\![i]\!] &= \max\left\{\mathbf{a}^{0}_{\max}[\![i]\!], \mathbf{a}^{1}_{\max}[\![i]\!]\right\} \end{aligned} \tag{4.2}$$

On the first level above triangles, we consistently group the root and left triangles under one parent and the right and tip ones under another, but this choice is arbitrary.

The hierarchy is constructed as part of scene set-up and its structure is not modified during simulation, to keep its update fast. The positions and dimensions of the bounding boxes are updated in each simulation step, but the parent-child relationships between the bounding volumes stay the same.

As we have mentioned, hair is constantly in close proximity. Furthermore, the length of a wisp is generally much larger than its width and thickness. Above a certain level in the hierarchy, the AABB contains a large length of the wisp and it



(a) Unaligned (b) Aligned

Figure 4.6: Examples of commonly-occurring wisp collision configurations. In both cases, the wisps collide with their flat sides. In (a), strands in the wisps are not mutually aligned, resulting in a blocking or sliding collision. In (b), the strands are aligned and so tangling occurs. Notice how the entangled wisps penetrate each other to some degree.

is generally so big that intersection is pretty much guaranteed to occur for nearby wisps, making the test pointless. We therefore limit the maximum height of the hierarchy; at the very high levels, the AABBs are just too huge to be useful.

We process the hierarchy using a standard recursive descent algorithm (Eberly 2005, pp. 540–553). On the lowest level, we have to determine whether the wisp representations actually intersect. This is where our choice of sphere-swept triangles for representation pays off. Let us assume we're currently processing triangles $\triangle^i$ and $\triangle^j$ from segments i and j. The two sphere-swept triangles intersect if and only if the distance between the base triangles is less than $\underline{\sigma}^i + \underline{\sigma}^j$. This is equivalent to stating that the square of the triangles' distance is less than $\left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2$. Such reformulation is significant, as it can save us from having to compute a square root to get the final distance, which is a computationally expensive operation compared to most normal floating-point operations such as addition or multiplication (Eberly 2015).

In our implementation, we use the software library Wild Magic (Eberly 2015) for computing the squared distance $\text{dist}^2\left(\triangle^i, \triangle^j\right)$ between the triangles. Then, if $\text{dist}^2\left(\triangle^i, \triangle^j\right) < \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2$, we mark both the triangles and their containing segments as in collision.

Collision detection is performed once in each simulation step, after equations of motion were integrated but before constraint enforcement occurs. The reason is that we use constraints to handle certain types of collisions, as presented in Section 4.3.

4.1.3 Collision Classification

We've shown in the preceding section how we detect when two wisp segments collide. For individual strands, a collision would mean they have penetrated each

other, a physical impossibility which would have to be corrected. In the case of wisps, the matter is not so clear-cut.

Remember that our wisp model is an abstraction for many hair strands close together, without specifics on where each individual strand is within the triangle-based wisp representation. So we must determine what happens when two such wisps come in contact with one another.

We have made several experiments with a synthetic-hair wig, analysing the outcomes of wisps colliding under different velocities and angles. Results of these match anecdotal observations we've made of real hair. The factor determining the outcome of such a collision is mutual orientation of the colliding wisps.

One possible mutual alignment is depicted in Figure 4.6a. The wisps come in contact through their flat sides, and the strand directions are not aligned. In such case, the wisps will slide over each other. One can potentially push the other if it has much larger energy on impact, but otherwise they just act as a barrier for themselves and once they separate, there is no lasting effect of the collision.

In the same flat side contact, if the strands of the two wisps are aligned in a roughly similar direction, a very different effect is observed (shown in Figure 4.6b). The individual hair strands from the two wisps entangle, forcing the wisps to stick together and follow each other during movement. Note that we have observed that during casual hair movement, this effect is temporary—the strands are not entangled so strongly as to merge the wisps, and sufficient separation can cause them to disentangle and move freely again.

It is also possible for the flat side of a wisp to come up against the rim of another one, or vice versa. In such case, the outcome again depends on the relative orientation of strands inside the wisps. If they are aligned, the wisps will entangle slightly. If they are not aligned, they simply block each other, but can slide along themselves or move apart easily.

There are two more configurations possible, which occur very rarely in real hair motion. The tip of one wisp can move against the flat side or rim of another. In such case, the wisps do not entangle readily. Our explanation is that doing so would require force acting along the length of the strands in the first wisp. Given the relatively low effective bending stiffness of a single hair strand, the strands bend under this force instead of being pushed into the other wisp and becoming entangled. Almost no entanglement is observed in a tip–flat side contact, and only minimal one in case of a tip–rim contact.

We need a way to capture these different responses in our model. Analysing the different scenarios presented above, it becomes clear that the primary factor responsible for the collision response quality is mutual strand orientation. When the wisps entangle, the degree to which they do so depends somewhat on the configuration of the collision (whether it's a flat–flat or a flat–side contact), but this effect is no prominent and is hard to measure. As a simplification, we therefore choose to ignore it altogether and use just two possible responses to a wisp collision:

- The strands are aligned, which means the wisps entangle and should affect each other during movement.
- The strands are not aligned, which means the wisps can not come closer to each other than they are now, but are otherwise not restricted in movement.

A similar collision classification scheme is used by Bertails et al. (2003) and Ward and Lin (2003).

For each collision, we need to decide which response to apply. Let use assume a collision of a triangle from segment i with a triangle from segment j; these can come from different wisps or even from the same wisp, if it loops unto itself. We first compute how well the strands in the colliding segments are aligned. The cosine of their angle is a good enough measure of this, and very efficient to compute:

$$A^{i,j} = \mathbf{t}^i \cdot \mathbf{t}^j \tag{4.3}$$

The actual measure of how aligned the strands are is then $\left|A^{i,j}\right|$. We need a threshold for this value. Below the threshold, the strands are not aligned and the wisps should block; above the threshold, the strands are aligned and entangling should occur. This threshold is a parameter of the simulation, related to the collective properties of the hair we simulate. Clumpy hair with tightly-pressed strands in a wisp is less likely to entangle than more loose hair. The composition of the wisp is also a factor. For this reason, we actually assign a potentially different value of this entanglement threshold to each wisp, within a range of values based on the hairstyle properties. We then average these thresholds before comparing with the alignment from equation 4.3.

There is an imprecision here in that we're averaging the cosines instead of the angles, but these thresholds are largely empirical values anyway, so this matters little. Furthermore, the threshold angles are usually in the $\left[\pi/6, \pi/3\right]$ range, where the discrepancy between the angle and its cosine is less pronounced.

Handling collisions of wisps with aligned strands (alignment value above the threshold) is described in Section 4.2. Unaligned collisions are the subject of Section 4.3.

4.2 Aligned Collisions

In this section, we describe how we handle collisions where the strands in the colliding wisps are aligned. Let us assume the colliding segments are i and j, the colliding triangles being $\triangle^i = \triangle \mathbf{a}_i \mathbf{c}^i \mathbf{b}_i$ and $\triangle^j = \triangle \mathbf{a}_j \mathbf{c}^j \mathbf{b}_j$. In both triangles, $\mathbf{a}$ and $\mathbf{b}$ are nodes from one or both rim strands of the respective wisp. An aligned collision occurs when $\left|\mathbf{t}^i \cdot \mathbf{t}^j\right|$ is above the tangle threshold. Such collisions result in the two wisps entangling.

4.2.1 Representing Entanglement

When two wisps entangle, they start to move in a joint fashion—not only can they not move closer together, as in the case of a blocking collision, but each one is also able to pull the other to a certain degree. Vastly different influences (such as being actively pulled apart) can of course disentangle the wisps and make them behave independently again, but subjected to less dramatic effects, the entangled wisps tend to drag one another along.

Our model already contains a representation for entangled hair—the wisps themselves. Recall these are just an approximation of a clump of hair sticking together because of mutual entanglement. Wisps are represented as sphere-swept

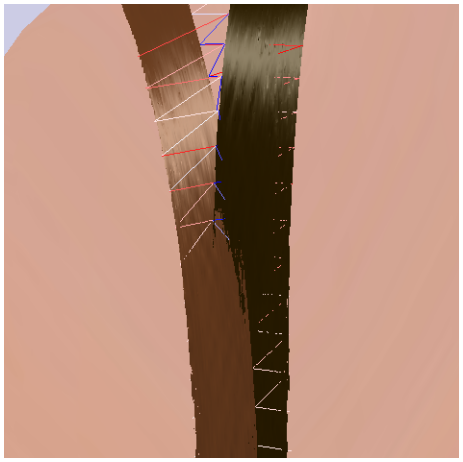


Figure 4.7: Springs representing entanglement of two wisps, created as a reaction to an aligned collision. Springs in rest-state length are rendered in white. Red and blue are used for springs extended or compressed, respectively, with saturation proportional to the deformation size.

triangles for collision detection, but for dynamics, which is the important part here, the tangle effect is captured by the *springs* connecting the rim strands of the wisp.

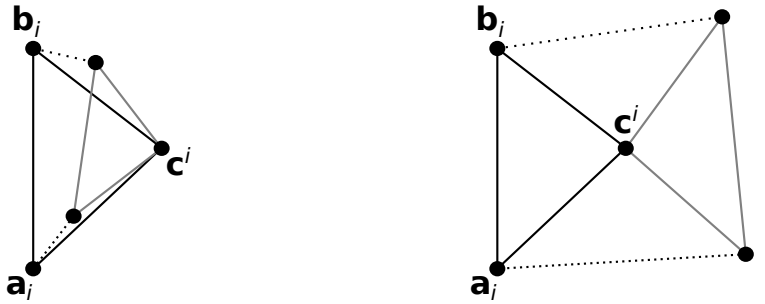
Springs have a number of advantages. They are simple to set up and fast to evaluate; and most importantly, they readily represent all the interactions we need for entangled wisps. They allow one to pull the other along when moving, and prevent the colliding wisps from coming closer together. Furthermore, springs are not a "hard" effect, but a gradual one. This again matches behaviour of entangled wisps, where the effects of interaction are also "fuzzy." We therefore choose springs to model wisp entanglement caused by aligned collisions. Figure 4.7 shows the tangle springs resulting from the aligned collision depicted in Figure 4.6b.

4.2.2 Attaching Springs

To use springs, we have to attach them to something. This is not straightforward, because our dynamics only works on the nodes of the wisps' rim strands. But the collision we have detected is between two triangles, one from each wisp. In each of these triangles, two vertices are nodes of a strand and the third one is the centroid of the segment.

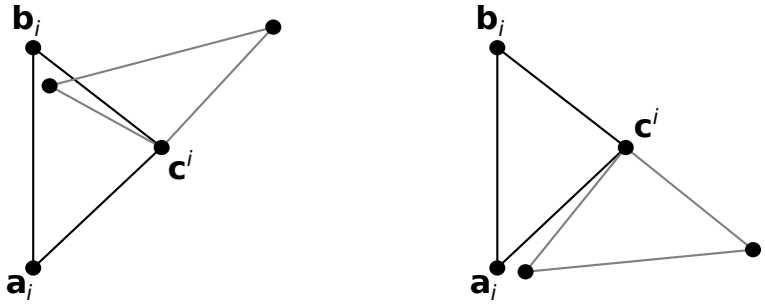
The simplest solution would be to attach a spring to the centroids of the colliding segments, but segment centroids are not simulated dynamically, so they cannot be subject to spring forces. What we do instead is attach springs to the nodes which form the vertices of the colliding triangles.

For this, we have to correctly identify which node to connect with which. This depends on which triangles make up the collision we're processing, and the mutual orientation of the wisps in the place of collision. For the purpose of this discussion of mutual wisp configuration, we recognise two *flavours* of triangles: we will refer to root and tip triangles as *transverse* triangles and to left and right triangles as *longitudinal* triangles (refer to Figure 4.2b for the triangle name definition).



(a) Overlapping triangles

(b) Opposite triangles



(c) Adjacent triangles

(d) Adjacent triangles

Figure 4.8: Possible configurations of colliding triangles. Projected $\triangle \mathbf{a}_j \mathbf{c}^j \mathbf{b}_j$ is shown in grey; where unambiguous vertex correspondence can be found, it is marked in dotted lines.

Notice that each triangle shares two sides with triangles of flavour opposite to itself; this fact will be important in the discussion below.

To match up the nodes correctly, we need to (virtually) map them to a common reference frame. For this, we take the following mental exercise (note that the transformations described are done purely for the purpose of discussion and do not happen in code). We start by projecting one of the triangles into the plane of the other; their relationship is symmetric so the choice of triangle to project is arbitrary and does not affect the result. Without loss of generality, let us assume we project $\triangle^j$ into the plane of $\triangle^i$. Then, we translate the projected $\triangle^j$ so that the projected $\mathbf{c}^j$ becomes collocated with $\mathbf{c}^i$.

Because we're dealing with an aligned collision, the two triangles will be roughly "adjacent" to each other if they are of different flavours (one transverse and one longitudinal), and they will be roughly "overlapping" or "opposite" if they are of the same flavour. See Figure 4.8 for examples. We can see that in the same-flavour case, an unambiguous way exists to establish simple correspondence between the two triangles' vertices (also depicted in Figure 4.8). In the case of different-flavour triangles colliding, there is no obvious reason to prefer one pairing over another, and neither of them promises consistent and satisfactory behaviour. The choice faces the same problems as trying to triangulate a segment into two triangles (as discussed in Section 4.1.1 and Figure 4.4).

However, by analysing the situation further, we find that we can safely ignore collisions of a transverse triangle with a longitudinal one, without losing any information from the wisps' contact. The reason is simple: on and near the triangle's sides, the sphere-swept volume of the triangle overlaps the sphere-swept

volume of its neighbouring triangle sharing the same side. Geometrically, the only way in which $\triangle^i$ could collide with $\triangle^j$ without colliding with any of $\triangle^j$'s neighbour triangles is for a vertex of $\triangle^i$ to be within the collision distance of the internal area of $\triangle^j$. But there are two sides of $\triangle^i$ which meet at the vertex, and least one of them will be shared with a triangle of a flavour different from that of $\triangle^i$, that is, the same as the flavour of $\triangle^j$. Which means that that triangle will also register a collision with $\triangle^j$, and that collision will involve two triangles of the same flavour.

We have now determined which vertex to join with which in the projection. However, we still need to determine which unprojected vertex corresponds to the projected one, which depends on the wisps' mutual orientation in 3D space. For this, we count the number of *flips* required to align the projected $\triangle^j$ to $\triangle^i$. Each of these counts as one flip:

- $\triangle^i$ and $\triangle^j$ have different names (e.g. $\triangle^i$ is a tip triangle and $\triangle^j$ is a root one)
- The wisps point in opposite directions: $A^{i,j} < 0$ (equation 4.3)
- The wisps are touching with opposite flat sides:

$$\left(\mathbf{t}^i \times \left(\mathbf{x}_i^{\mathrm{R}} - \mathbf{x}_i^{\mathrm{L}}\right)\right) \cdot \left(\mathbf{t}^j \times \left(\mathbf{x}_j^{\mathrm{R}} - \mathbf{x}_j^{\mathrm{L}}\right)\right) < 0$$

The term $\mathbf{t}^i \times \left(\mathbf{x}_i^{\mathrm{R}} - \mathbf{x}_i^{\mathrm{L}}\right)$ represents the "normal" of the flat side of the wisp at segment i.

When the number of flips is even, the triangles are aligned in the projection plane and we attach one spring between $\mathbf{a}_i$ and $\mathbf{a}_j$, and another spring between $\mathbf{b}_i$ and $\mathbf{b}_j$. If the number of flips is odd, the triangles are opposite each other in the projection plane and the attachments are as follows: $\mathbf{a}_i$ with $\mathbf{b}_j$, $\mathbf{b}_i$ with $\mathbf{a}_j$. Care has to be taken not to attach a second spring between two nodes if they are already connected by one.

The rest length of the spring is set to its length at creation time, i.e. when the collision first occurs. Depending on the current deformation of the colliding wisps, it is possible for the triangles to collide on one end, while their vertices on the other end are quite distant from each other. Introducing tangle springs between such widely separated nodes would give unnatural-looking results of wisps behaving as entangled even when they are visually far apart. To prevent this, we introduce a maximum length of tangle springs, and simply do not create springs whose rest length would exceed this maximum. The value can vary between wisps and it is related to the "fuzziness" of the wisp—the higher the rest length allowed, the more likely the wisp is to tangle. This corresponds to a less compact wisp, with more hair sticking out.

At scene set-up, each wisp is also assigned a value for tangle stiffness. This value is inversely proportional to the wisp's tangle threshold—the less likely a wisp is to entangle, the easier it is for it to disentangle again, which means that its tangle stiffness is lower. The stiffness of the actual spring being created is then the average of the tangle stiffness values of the colliding wisps.

4.2.3 Spring Management

Once the spring is created, it becomes part of the simulation starting in the next time step (remember that collision detection follows after integration).

As we've stated in our observations in Section 3.6.3, wisp entanglement is temporary. If their motion forces the wisps sufficiently apart, they disentangle. To model this effect in our simulation, we compare the tangle spring's length to its rest length at the beginning of each time step, and if it past a certain threshold, we remove the spring.

Suddenly removing a spring which was considerably elongated (and thus exerted considerable force) could easily introduce instabilities and unrealistic effects into the simulation. To remedy this, it is possible to use an approach introduced for solving a very similar issue by Bando et al. (2003). The effective stiffness of the spring is scaled by a reciprocal of its elongation, which means that it gradually becomes weaker as it extends. This helps keep the simulation stable, prevents undesired motion artefacts, and realistically models the entanglement weakening as the wisps pull farther apart and more and more strands come loose from the other wisp. Note that no stiffness scaling should be applied in case the spring compresses—a steady response is desired in such case, preventing the wisps from coming too close together.

4.3 Unaligned Collisions

This section is dedicated to the description of handling collisions in which the strands of the colliding wisps are not aligned. We again assume the colliding segments are i and j, and the colliding triangles $\triangle^i = \triangle \mathbf{a}_i\mathbf{c}^i\mathbf{b}_i$ and $\triangle^j = \triangle \mathbf{a}_j\mathbf{c}^j\mathbf{b}_j$. A collision is unaligned when $\left|\mathbf{t}^i \cdot \mathbf{t}^j\right|$ is below the tangle threshold. Such a collision means the wisps physically block each other's movement, without entangling.

This means that the wisps should be prevented from moving closer to one another, without otherwise restricting their movement (they are free to slide along each other). Worded differently, their movement in the direction to each other is constrained. This suggests we could model these unaligned conditions as constraints and use our constraint-enforcement mechanism to resolve them.

4.3.1 Hair–hair Collision Constraints

A collision is detected when the swept-sphere volumes associated with the two triangles intersect. As we've covered in Section 4.1.2, this can be reformulated using (squared) distance of the triangles: $\mathrm{dist}^2\left(\triangle^i, \triangle^j\right) < \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2$. We use this formulation as the basis; ideally, we would like the constraint to be:

$$\mathrm{dist}^2\left(\triangle^i, \triangle^j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 = \mathrm{dist}^2\left(\triangle \mathbf{a}_i\mathbf{c}^i\mathbf{b}_i, \triangle \mathbf{a}_j\mathbf{c}^j\mathbf{b}_j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 \geq 0 \quad (4.4)$$

We will now show that while achieving this exact formulation would be impractical, we can use it to guide us to a working solution.

$\mathrm{dist}^2\left(\triangle^i, \triangle^j\right)$ depends on all vertices of both the triangles. However, the centroids $\mathbf{c}^i$ and $\mathbf{c}^j$ are not subject to dynamics and therefore cannot directly participate in constraint enforcement. Instead, their coordinates depend on all

four nodes comprising their respective segment. This means that the condition expressed in equation 4.4 depends on 8 nodes in total. Most importantly, however, these nodes come from two rims of two wisps—four distinct strands in total (or two, in the case of a wisp's self-collision).

Directly implementing this constraint would therefore force us to combine constraint enforcement of the involved strands into one set of equations. This would be a fundamental change to our method, which has so far been able to process each strand independent from all other strands. Furthermore, since each wisp normally collides with several other wisps, it is perfectly possible that the transitive closure of the *collides-with* relation would be the entire hairstyle. Recall from Section 3.5.4 that constraint enforcement requires repeated solution of a system of equations whose size depends on the number of nodes and constraints involved. Trying to do this for the entire hairstyle at once would far exceed the memory and computation power available. We have to find an alternative formulation of the constraint which will enable us to keep processing each strand separately.

Equation 4.4 considers both triangles reacting to the constraint simultaneously. We will first introduce a simplification which allows separate handling for each wisp. When processing $\triangle^i$, we will treat $\triangle^j$ as fixed in space, and vice versa. During such processing, we use the term *active* to refer to the wisp whose constraints are being enforced; the wisp treated as fixed is called *blocking.*

The active/blocking distinction basically transforms the constraint like this:

$$\begin{aligned} \operatorname{dist}^2\left(\triangle \mathbf{a}_i \mathbf{c}^i \mathbf{b}_i, \triangle \breve{\mathbf{a}}_j \breve{\mathbf{c}}^j \breve{\mathbf{b}}_j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 \geq 0 \\ \operatorname{dist}^2\left(\triangle \breve{\mathbf{a}}_i \breve{\mathbf{c}}^i \breve{\mathbf{b}}_i, \triangle \mathbf{a}_j \mathbf{c}^j \mathbf{b}_j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 \geq 0 \end{aligned} \tag{4.5}$$

We again use the notation introduced in Section 3.5.4: $\breve{\mathbf{z}}$ refers to the unconstrained value of vector $\mathbf{z}$, that is, the value after integration but before constraint enforcement.

Equation 4.5 allows us to treat each wisp independently, but the two rim strands of each wisp are still connected in these equations. As the next step, we treat $\mathbf{c}^i$ as an independent variable, ignoring its dependence on the strands' nodes. This amounts to fixing its position:

$$\begin{aligned} \operatorname{dist}^2\left(\triangle \mathbf{a}_i \breve{\mathbf{c}}^i \mathbf{b}_i, \triangle \breve{\mathbf{a}}_j \breve{\mathbf{c}}^j \breve{\mathbf{b}}_j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 \geq 0 \\ \operatorname{dist}^2\left(\triangle \breve{\mathbf{a}}_i \breve{\mathbf{c}}^i \breve{\mathbf{b}}_i, \triangle \mathbf{a}_j \breve{\mathbf{c}}^j \mathbf{b}_j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 \geq 0 \end{aligned} \tag{4.6}$$

Notice that this has already solved our problem when the triangle we're solving for is longitudinal (as defined in Section 4.2.2). In such triangles, both $\mathbf{a}_i$ and $\mathbf{b}_i$ are nodes of the same strand.

For transverse triangles, we apply the same trick once more, considering the position of the node from the other strand fixed. This is what the constraint formula looks like when $\triangle^i$ is a root triangle, that is, when $\mathbf{a}_i = \mathbf{x}_i^{\mathrm{L}}$ and $\mathbf{b}_i = \mathbf{x}_i^{\mathrm{R}}$:

$$\operatorname{dist}^2\left(\triangle \mathbf{x}_i^{\mathrm{L}} \breve{\mathbf{c}}^i \breve{\mathbf{x}}_i^{\mathrm{R}}, \triangle \breve{\mathbf{a}}_j \breve{\mathbf{c}}^j \breve{\mathbf{b}}_j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 \geq 0 \tag{4.7}$$

Thus, we have transformed a joint constraint for 8 nodes from 4 strands into a set of 2–4 constraints for 1–2 nodes each, each constraint limited to one strand.

The next problem we have to tackle is the fact that to enforce the constraints, we need to be able to express their gradient as well as their value. Computing the distance of two triangles has no nice closed-form solution to differentiate, which makes a triangle-based formula unsuitable for our purposes. Our goal is therefore to transform the constraints into expressions for which computing the gradient efficiently is possible.

Let us deal with transverse triangles first (equation 4.7). We take another simplification and base the constraint on the distance from $\triangle^j$ to the *node* involved on segment i, rather than to the entire triangle $\triangle^i$. Equation 4.7 for the root triangle thus becomes:

$$\operatorname{dist}^2\left(\mathbf{x}_i^{\mathrm{L}}, \triangle\breve{\mathbf{a}}_j\breve{\mathbf{c}}^j\breve{\mathbf{b}}_j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 \geq 0 \tag{4.8}$$

For the tip triangle, $\mathbf{x}_{i+1}^{\mathrm{L}}$ appears instead of $\mathbf{x}_i^{\mathrm{L}}$.

For longitudinal triangles, where both non-centroid vertices of the triangle are nodes of the same strand, we formulate the constraint as the distance between $\triangle^j$ and the *edge* connected by these nodes. For the left triangle, the formula is as follows:

$$\operatorname{dist}^2\left(\overline{\mathbf{x}_i^{\mathrm{L}}\mathbf{x}_{i+1}^{\mathrm{L}}}, \triangle\breve{\mathbf{a}}_j\breve{\mathbf{c}}^j\breve{\mathbf{b}}_j\right) - \left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2 \geq 0 \tag{4.9}$$

These simplifications actually weaken the constraints, but we find the imprecision is acceptable given the overall "soft" nature of hair.

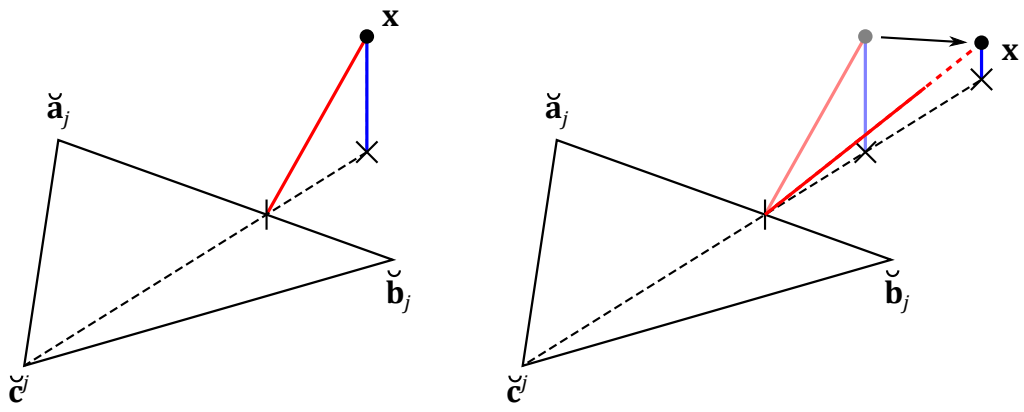
We still cannot apply these constraints in this form, though. At the moment the collision is detected, the triangles are already closer to each other than $\left(\underline{\sigma}^i + \underline{\sigma}^j\right)^2$, which means the nodes in question could be closer as well. However, due to the way we handle inequality constraints, they must never actually be violated. The slack variable corresponding to a violated inequality constraint would be negative. The objective function (given in equation 3.152) contains the logarithm of each slack variable, so it would not be well-defined in such case.

We solve this by dropping the swept-sphere radius term from the constraints, simply constraining the nodes not to touch the triangle. This basically turns the hard constraint into a soft one: it allows the wisps to penetrate each other slightly, but the closer the skeleton triangles get, the more the collision constraint will dominate the objective function. This in turn causes constraint enforcement to drive them away from each other more strongly.

While the primary motivation here is implementation validity, it does actually capture a real phenomenon in hair. When wisps get into an unaligned collision, they can compress somewhat; they are not a rigid shape. In our simulation model, this translates to the skeleton triangles getting closer to each other, which is precisely what these relaxed constraints allow. This further vindicates the simplification to node- and segment-based constraints. It is even possible to store information about the depth of penetration with the wisps and use it during rendering to compress the virtual shape of the wisp.

Putting this all together, we arrive at the following collision constraint formulae for the left rim strand participating in collision of triangle $\triangle_i$:

$$\mathrm{CC}^{\triangle^i,\triangle^j} = \begin{cases} \operatorname{dist}^2\left(\mathbf{x}_i^{\mathrm{L}}, \triangle\breve{\mathbf{a}}_j\breve{\mathbf{c}}^j\breve{\mathbf{b}}_j\right) & \triangle^i \text{ is root triangle} \\ \operatorname{dist}^2\left(\overline{\mathbf{x}_i^{\mathrm{L}}\mathbf{x}_{i+1}^{\mathrm{L}}}, \triangle\breve{\mathbf{a}}_j\breve{\mathbf{c}}^j\breve{\mathbf{b}}_j\right) & \triangle^i \text{ is left triangle} \\ \operatorname{dist}^2\left(\mathbf{x}_{i+1}^{\mathrm{L}}, \triangle\breve{\mathbf{a}}_j\breve{\mathbf{c}}^j\breve{\mathbf{b}}_j\right) & \triangle^i \text{ is tip triangle} \end{cases} \tag{4.10}$$



(a) (b)

Figure 4.9: Figure (a) shows difference between node–triangle distance (red) and node–plane distance (blue). Figure (b) demonstrates movement of node $\mathbf{x}$ which moves it further from the triangle, but closer to the plane. When such movement occurs, it is incorrectly penalised by the simplified node–plane distance constraint.

Constraints for the right rim strand and for the other wisp's strands are constructed analogously.

We have implemented these constraints, again using the Wild Magic library (Eberly 2015) to compute the constraint values. For computing gradients, the formulae were obtained by symbolic differentiation, and the computation itself implemented based on the same principles as Schneider and Eberly (2002) use for computing point–triangle and segment–triangle distances.

This approach works, but the amount of computation required to evaluate a collision constraint and its gradient is disproportionate to that required by other types of constraints. Inextensibility constraints (equation 3.126) and hair–head collision constraints (equation 3.161), as well as their gradients (equation 3.170), require little more than a dot product. In comparison, the distance-based collision constraints presented in equation 4.10 require substantial computation and branching (Schneider and Eberly 2002).

Given the fact that hair wisps are far from rigid, it is a question whether such constraint accuracy at a cost of substantially more computation is necessary. Let us analyse the situation in which collision constraints apply, and see whether we can devise a suitable simplification.

Collision constraints are effected by unaligned wisp collisions—flat sides of two unaligned wisps collide, or the flat of one wisp perpendicularly hits the rim of another. In both of these cases, the desired outcome is the same: the affected segments of the wisp should not be allowed to move closer to the other wisp. In the equations above, we have represented this as the node/segment in question not coming closer to the other wisp's segment triangle. We now consider what the effect would be if we broadened the constraint to be the distance between the node/segment and the entire *plane* of the triangle (demonstrated in Figure 4.9a). This way, the constraint is stronger—there are some trajectories for the node/segment which bring it closer to the plane without approaching the

blocking triangle, as shown in Figure 4.9b. Most of these trajectories correspond to the active wisp bending over the blocking one. This is behaviour we would not want to lose, as it is readily observable in real hair.

Fortunately, it turns out that the behaviour is preserved even when using the entire plane as the source of the constraint. The reason is the fact that the collision is unaligned. The wisps' tangents are therefore close to orthogonal, which means segments adjacent to the colliding ones are unlikely to be in collision as well. They can therefore move freely, which allows the wisps to bend over each other. It is important to bear in mind that while we use the entire plane as the constraint, it is still only the colliding node to which the constraint is applied. Nodes not participating in the collision can move towards the constraint plane uninhibited.

The complexity of computing the distance between a point and a plane is comparable to that of other constraints in the system, which was our goal. We can thus safely use it as the constraint value for latitudinal triangles, from which only one node on each strand participates in the collision.

The direct equivalent for longitudinal triangles would be the distance between the plane and the edge connecting the triangle's vertices. Unfortunately, this would again bring undesired complexity to the constraint computation—computing the distance between a plane and a line segment requires branching, which leads to complexity in computing gradients. Constraint enforcement is an iterative process, which means constraints and their gradients are evaluated multiple times in each simulation time step; we therefore want these calculations to be as straightforward as possible. For this reason, we add one more simplification. When a collision of a longitudinal triangle $\triangle^i$ is first detected, we compute the distance of both the nodes on $\triangle^i$ (its non-centroid vertices) from the blocking plane, and treat only the closer one of them as participating in that collision. This is computed once and does not change as long as that particular collision (triangle–triangle pair) lasts. We find that this simplification does not degrade the simulation behaviour in any perceivable way. If the difference between the nodes' distances from the plane is large (that is, the edge is more or less orthogonal to the plane), changes in the distant node's position will not affect the triangle's distance from the plane. Conversely, if the nodes are roughly equidistant from the plane (the edge being close to parallel to the plane), the farther node is likely to be involved in another collision with the same blocking wisp as well, which can provide a constraint for it also.

With all these simplifications in place, each unaligned collision is represented by 2–4 constraints based on point–plane distance, one constraint for each participating strand. We now present the relevant formulae, again assuming an active triangle $\triangle \mathbf{a}_i \mathbf{c}^i \mathbf{b}_i$ colliding with a blocking triangle $\triangle \breve{\mathbf{a}}_j \breve{\mathbf{c}}^j \breve{\mathbf{b}}_j$. The collision plane is defined as follows:

$$\mathbb{P}^{i,j} = \left\{ \mathbf{z} : \mathbf{N}_{\mathbb{P}}^{i,j} \cdot \mathbf{z} - D_{\mathbb{P}}^{i,j} = 0 \right\} \tag{4.11}$$

When the collision is detected, the plane's normal vector $\mathbf{N}_{\mathbb{P}}^{i,j}$ and absolute term are computed from the triangle:

$$\mathbf{N}_{\mathbb{P}}^{i,j} = \frac{\left(\breve{\mathbf{a}}_j - \breve{\mathbf{c}}^j\right) \times \left(\breve{\mathbf{b}}_j - \breve{\mathbf{c}}^j\right)}{\left\| \left(\breve{\mathbf{a}}_j - \breve{\mathbf{c}}^j\right) \times \left(\breve{\mathbf{b}}_j - \breve{\mathbf{c}}^j\right) \right\|} \tag{4.12}$$

$$D_{\mathbb{P}}^{i,j} = \mathbf{N}_{\mathbb{P}}^{i,j} \cdot \breve{\mathbf{c}}^j \tag{4.13}$$

These are updated once each time step, during collision detection. The values are then used for that time step's constraint enforcement.

The collision constraint itself is then pretty straightforward. Similar to equation 4.10, we present formulae for the left rim strand of triangle $\triangle^i$:

$$\mathrm{CC}^{\triangle^i,\triangle^j} = \begin{cases} \mathbf{N}_{\mathbb{P}}^{i,j} \cdot \mathbf{x}_i - D_{\mathbb{P}}^{i,j} & \triangle^i \text{ is root triangle} \\ \mathbf{N}_{\mathbb{P}}^{i,j} \cdot \mathbf{x}_{i+1} - D_{\mathbb{P}}^{i,j} & \triangle^i \text{ is tip triangle} \\ \mathbf{N}_{\mathbb{P}}^{i,j} \cdot \mathbf{x}_i - D_{\mathbb{P}}^{i,j} & \triangle^i \text{ is left triangle, } \mathbf{x}_i \text{ is closer } \mathbb{P}^{i,j} \\ \mathbf{N}_{\mathbb{P}}^{i,j} \cdot \mathbf{x}_{i+1} - D_{\mathbb{P}}^{i,j} & \triangle^i \text{ is left triangle, } \mathbf{x}_{i+1} \text{ is closer to } \mathbb{P}^{i,j} \end{cases} \tag{4.14}$$

Denote the participating node as $\mathbf{x}_k$; the constraint gradient is simply the collision plane normal:

$$\nabla_k \mathrm{CC}^{\triangle^i,\triangle^j} = \mathbf{N}_{\mathbb{P}}^{i,j} \tag{4.15}$$

4.3.2 Constraint Conflicts

When we've implemented the above mentioned collision constraints for the first time, we've observed frequent instabilities in the constraint enforcement system. We've traced these to a conflict between hair–head and hair–hair collision constraints. When hair wisps stack on top of each other on the head, the bottom wisp is subject to a constraint from the head which prevents it from moving down, and to a constraint from its collision partner which prevents it from moving up. This leads to instabilities or convergence failures.

To alleviate this issue, we introduce a thin proximity zone around the head, inside which unaligned collisions are considered a one-way interaction. That is, when two wisps collide in an unaligned way inside this proximity zone, the collision constraint is only applied to the top wisp. The bottom wisp is not considered constrained and can move towards its collision partner if enforcement of head collision constraints requires it. Zone thickness on the order of millimetres has proven sufficient to eliminate this problem. Any wisp interpenetration artefacts which can result from these one-way collisions are generally obscured by higher layers of hair.

A wisp "sandwiched" between two other wisps does not cause the same instability issues, because hair–hair collision constraints are soft in nature: constraint enforcement causes both wisps to move away from the collision, something which is not possible when colliding with the immovable head.

Chapter 5

Results

5.1 Method Summary

This section describes both hair animation systems we have developed based on our method. Two distinct systems were developed as proof that our method does not depend on any particular underlying system. They also helped to test applicability of the hair bending principle (introduced in Section 3.1.2) independent of other features of our method (such as flat wisps).

Throughout the previous chapters, the order in which we have presented the individual parts of our hair animation method was driven by the context required for introducing and explaining them. In this section, we repeat them in the order they are executed at runtime.

5.1.1 Simplified Super-Helices

Our first hair animation method is based on the Super-Helix discretisation of Kirchhoff rod theory presented by Bertails et al. (2006). We have presented it in Sections 3.3 and 3.4.

The strand being simulated is discretised into N helical segments. Each segment Q is parametrised by its twist τ_Q and bending $\boldsymbol{\omega}_Q$; bending happens over the major axis of the cross section. These $2N$ variables are gathered into vector $\mathbf{q}$ and form the dynamic variables of the system.

System initialisation consists of setting the initial values of generalised coordinates $\mathbf{q}$ and generalised velocities $\dot{\mathbf{q}}$ which correspond to the initial shape and movement of the hair.

The equations of motion, formulated using Lagrangian mechanics, are given by equation 3.40. The mass matrix is dense and depends non-linearly on $\mathbf{q}$, which makes the solution rather costly.

After each step of the integration, if rendering is requested, the shape of the strand must be reconstructed. This means performing the process from Section 3.3.3 (with equation 3.43 applied) to compute $\mathbf{x}, \mathbf{t}, \mathbf{m}_1$, and $\mathbf{m}_2$. Reconstruction is also necessary if collision handling is required.

5.1.2 Explicit Hair Strands

The method we present here is based on the Discrete Elastic Rod model of Bergou et al. (2008). We have previously described it in Sections 3.5, 3.6, 3.7, and Chapter 4.

The simulated strand is discretised into a polyline of $N+1$ edges $\mathbf{e}^0 \dots \mathbf{e}^N$. The nodes of the polyline, $\mathbf{x}_0, \dots, \mathbf{x}_{N+1}$, form the dynamic variables of the system. Material frame orientation is assigned to edges and expressed as a single scalar θ, the angle of rotation around the tangent between the material frame and a canonical Bishop frame at the edge. This material frame orientation is computed quasistatically after each integration step and is not a free variable in the equations of motion.

For each strand, scene set-up requires setting the initial position $\mathbf{x}_i$ and velocity $\dot{\mathbf{x}}_i$ of each node i. The Bishop frame axes at the root segment $\mathbf{u}_1^0$ and $\mathbf{u}_2^0$ are initialised with the material frame axes $\mathbf{m}_1^0$ and $\mathbf{m}_2^0$ at the root segment, and the rest of the Bishop frame is computed using parallel transport (Section 3.5.3).

For strands which form rims of flat wisps (Section 3.6.3), wisp springs are attached to their nodes and the springs' stiffness coefficients $\underline{k}$ and rest lengths $\underline{\hat{d}}$ are initialised. In our implementation, we also compute the bounding volume hierarchy of axis-aligned bounding boxes over wisps at this point (Section 4.1.2).

Simulation Loop

With the set-up complete, the simulation loop then proceeds as follows:

1. For all strands, *compute all constituent terms of the equations of motion:*
   - 1.1. $(\forall i)$ compute curvature binormal $(\kappa\mathbf{b})_i$ from $\mathbf{e}^{i-1}$, $\mathbf{e}^i$ using equation 3.75.
   - 1.2. $(\forall j)$ compute Bishop frame $\mathbf{u}_1$, $\mathbf{u}_2$ from $\mathbf{t}^0$, $\mathbf{e}^0$ using equations 3.80 and 3.63.
   - 1.3. Compute material frame (θ, $\mathbf{m}_1$, $\mathbf{m}_2$) using algorithm presented in Section 3.6.1.
   - 1.4. $(\forall i)$ $\big(\forall j \in \{i-1, 1\}\big)$ compute bending $\boldsymbol{\omega}_i^j$ from $(\kappa\mathbf{b})_i$, $\mathbf{m}_1^j$, $\mathbf{m}_2^j$ using equation 3.82.
   - 1.5. $(\forall i)$ compute curvature binormal gradients $\nabla_{i-1}(\kappa\mathbf{b})_i$, $\nabla_i(\kappa\mathbf{b})_i$, $\nabla_{i+1}(\kappa\mathbf{b})_i$ from $(\kappa\mathbf{b})_i$, $\mathbf{e}^{i-1}$, $\mathbf{e}^i$ using equations 3.110–3.112.
   - 1.6. $(\forall i)$ compute holomy gradients $\nabla_{i-1}\psi_i$, $\nabla_i\psi_i$, $\nabla_{i+1}\psi_i$ from $(\kappa\mathbf{b})_i$, $\mathbf{e}^{i-1}$, $\mathbf{e}^i$ using equation 3.77.
   - 1.7. $(\forall i)$ compute gradients of total holonomy $\nabla_i\Psi^{i-1}, \dots, \nabla_i\Psi^N$ from $\nabla_i\psi_{i-1}$, $\nabla_i\psi_i$, $\nabla_i\psi_{i+1}$ using equation 3.78.
   - 1.8. $(\forall i, k)$ $\big(\forall j \in \{k-1, k\}\big)$ compute bending gradient $\nabla_i\boldsymbol{\omega}_k^j$ from $\mathbf{m}_1^j$, $\mathbf{m}_2^j$, $\nabla_i(\kappa\mathbf{b})_k$, $\boldsymbol{\omega}_k^j$, $\nabla_i\Psi^j$ using equation 3.105.
2. *Compute* new *velocities* $\dot{\mathbf{x}}$ and *positions* $\mathbf{x}$ of all strands by integrating the equations of motion (equation 3.134) using values computed in step 1. This includes evaluation of external forces such as gravity and springs.

3. Perform *collision detection,* as described in Section 4.1.2. This includes updating the bounding volume hierarchy.
   - For all aligned collisions, introduce collision springs into the dynamic system (Section 4.2).
   - For all unaligned collisions, register collision constraints into the constraint enforcement system (Section 4.3).
4. *Enforce constraints* on each strand:
   4.1. Compute values of constraints for inextensibility (equation 3.126), hair–head collisions (equation 3.161) and hair–hair collisions (equation 4.14).
   4.2. If all constraint values are below the convergence threshold, proceed to step 4.6.
   4.3. Compute constraint gradient matrix (equations 3.170 and 4.15).
   4.4. Compute one step of Newton minimisation (equations 3.159 and 3.160).
   4.5. Return to step 4.1.
   4.6. If any minimisation steps were computed, update velocities using equation 3.172.

5.2 Evaluation

In this section, we evaluate the results of using our explicit method for simulating a variety of scenarios summarised in Table 5.1. The basic set-up is always the same: a number of wisps is distributed in a uniform pattern with random jittering applied (an example distribution is shown in Figure 3.15). Parameters of the wisps such as thickness and wisp spring stiffness, as well as parameters of the strands such as length, cross-section scale and initial material frame orientation are randomly varied as well. Figure 5.1 shows screenshots from two tests of scenario F1.

All data presented here was obtained by running our implementation on a system with an Intel®Core™2 Quad 2.66 GHz CPU with 8 GB of RAM.

Table 5.2 lists the performance we have measured in the basic scenario of hair falling under gravity. One fact is immediately obvious from the table: for these scene sizes, the method does not run in real-time, and collision detection is the bottleneck. This happens to be an unfortunate artefact of our implementation.

The axis-aligned bounding box hierarchy in the way we have implemented it (Section 4.1.2) is clearly insufficient as an acceleration structure for collision detection in our case. However, it is important to bear in mind that the method of collision detection is not coupled to our animation method in any way. For us, collision detection is merely a tool which we use as a "black box" to obtain the list of colliding wisp triangles, on which we then apply our method's specific collision handling. We apparently chose the wrong tool for the job; unfortunately, we have not been able to implement a better collision detection method due to time constraints. Still, we feel that this does not invalidate our method itself—if a more efficient collision detection mechanism was put in place instead of the

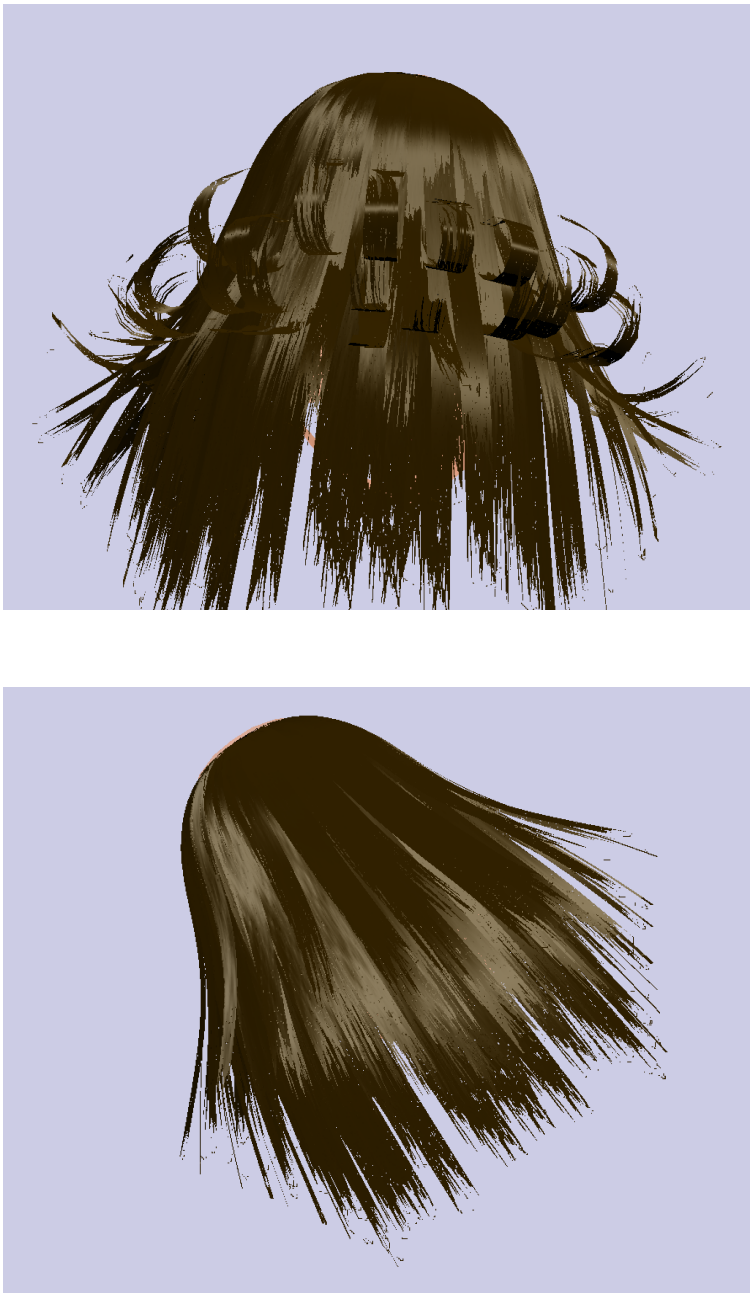


Figure 5.1: Screenshots from two tests of scenario F1.

Scene	Shape	Length	Wisps	Segment length	Nodes
F1	straight	30 cm	120	1 cm	7440
F2	straight	30 cm	60	1 cm	3720
F3	wavy	30 cm	60	1 cm	3720
F4	curly	30 cm	60	1 cm	3720
R1	straight	30 cm	60	2 cm	1920
R2	straight	30 cm	15	2 cm	480
R3	straight	50 cm	15	1 cm	1530
R4	wavy	30 cm	15	2 cm	480
R5	curly	30 cm	15	2 cm	930
I1	straight	15 cm	10	1 cm	320
I2	straight	30 cm	2	1 cm	124
I3	wavy	30 cm	6	1 cm	372

Table 5.1: Our test scenes. **F** scenes feature a full head of hair, and are intended to measure our method's performance on a plausible representation of a normal hairstyle. **R** scenes use a reduced set of hair; they can be used for comparison with **F** scenes to assess impact of scene complexity on performance. They also showcase some parameter variations which are easier to test and evaluate in a reduced scene. Finally **I** scenes were chosen of a size which offers interactive rates with our method.

Scene	Int.	CD	CR	Const.	Update	Total
F1	47.88	1123.77	5.63	38.07	26.18	1239.60
F2	23.59	240.70	1.04	19.20	13.06	297.13
F3	23.58	241.72	1.14	20.49	13.19	299.81
F4	23.85	241.82	0.87	18.91	13.11	294.50
R1	6.28	130.89	0.50	5.04	5.29	148.24
R2	1.45	9.37	0.04	1.42	1.35	13.63
R3	15.35	9.28	0.02	14.70	6.68	46.09
R4	1.36	1.12	0.01	1.10	1.38	4.96
R5	1.23	8.75	0.03	1.30	1.30	12.63
I1	0.68	2.50	0.01	1.01	0.80	5.03
I2	0.57	0.48	0.00	0.78	0.38	2.22
I3	1.90	0.65	0.00	2.30	1.27	6.15

Table 5.2: Average time taken in each scenario by different parts of our simulation. Times are given in milliseconds. Times include random user interaction (rotating the head) in **I** scenes. The columns give times for different stages: **Int.**—computing forces, integrating node velocities and positions. **CD**—collision detection and update of AABB hierarchies. **CR**—collision response, i.e. registering collision constraints or creation of collision springs. **Const.**—constraint evaluation and enforcement, including velocity update. **Update**—Bishop frame and material frame update, plus computing bending and holonomy gradients for the next integration step.

Scene	Int.	CR	Const.	Update
F1	41.3%	4.9%	32.9%	22.6%
F2	41.8%	1.8%	34.0%	23.1%
F3	40.6%	2.0%	35.3%	22.7%
F4	45.3%	1.7%	35.9%	24.9%
R1	36.2%	2.9%	29.0%	30.5%
R2	34.0%	0.9%	33.3%	31.7%
R3	41.7%	0.1%	39.9%	18.1%
R4	35.4%	0.3%	28.6%	35.9%
R5	31.7%	0.8%	33.5%	33.5%
I1	26.9%	0.4%	39.9%	31.6%
I2	32.8%	0.0%	44.8%	21.8%
I3	34.5%	0.0%	41.8%	23.1%

Table 5.3: Average time taken by each stage as percentage of total step duration*.

current one, no other part of our simulation would have to change and all collision responses could be applied exactly as they are now. With this in mind, for the rest of this chapter, we will not consider the time spent on collision detection and AABB hierarchy update when evaluating the performance of our method. To avoid an unintentionally misleading impression, all total times from which collision detection is excluded are marked with an asterisk*.

Table 5.3 presents the simulation step times in relative form as percentage of the total duration of the step. The relative effect of constraint enforcement is higher in the **I** scenes because the interaction was geared towards making the wisps collide with each other or slide along the head. Unsurprisingly, the effect of collision response is closely related to the density of hair in the scene.

In Table 5.4, we present the time required for features unique to our method. This means the time to compute spring forces arising from both wisp and tangle springs. The time taken by our twist computation is also shown. The data shows that our twisting algorithm is largely linear in the number of nodes in the scene.

5.3 Conclusion

In this thesis, we have presented a novel method of dynamic hair animation which specifically exploits properties and behaviour of real hair to improve both realism and efficiency. At the core of our method is the observation of Swift (1995) that due to its elliptical cross section, human hair bends preferably over its minor cross-section axis. This property influences twisting and bending behaviour of hair, including curliness. To the best of our knowledge, no existing hair animation method makes explicit use of this behaviour.

We have shown how this observation and its implications can be applied to a theoretical model of hair. We have further demonstrated this application in practice on two different approaches to simulation, an implicit representation model by Bertails et al. (2006) and an explicit approach by Bergou et al. (2008). We have thus obtained two distinct animation models both utilising our method's

Scene	Springs	/Int.	Twist	/Total*	/Nodes
F1	2.81	5.9%	3.96	3.4%	5.32
F2	0.95	4.0%	1.96	3.5%	5.27
F3	1.20	5.1%	2.09	3.6%	5.62
F4	1.13	4.7%	2.06	3.9%	5.54
R1	0.50	7.9%	1.04	6.0%	5.42
R2	0.13	9.0%	0.27	6.3%	5.63
R3	0.28	1.8%	0.78	2.1%	5.10
R4	0.10	7.3%	0.29	9.2%	6.04
R5	0.10	8.1%	0.28	7.2%	3.01
I1	0.06	8.8%	0.18	7.1%	5.63
I2	0.04	7.0%	0.07	4.0%	5.65
I3	0.08	4.2%	0.21	3.8%	5.65

Table 5.4: Time spent on computing spring forces and our twisting algorithm, given in milliseconds. The time for spring forces relative to the whole integration stag is also given, and likewise the relative effect of our twist computation on total step time. The last column is twist computation relative to the number of nodes, scaled by a factor of 10^4 for presentation purposes.

core principles. This proves that our approach is independent of the underlying simulation principle, making it applicable to a wide range of scenarios.

In addition to the bending behaviour mentioned above, we have identified other phenomena commonly found in real hair which can be used to improve animation, namely tendency of hair to form flat wisps. Incorporating this behaviour is easier with an explicit representation, so we have applied it to our method based on explicit rods. This improves the method by reducing the number of primitives which must be simulated to obtain satisfactory visual results.

In addition, we have proposed a new natural and efficient collision response mechanism for hair–hair collisions, based on the same phenomenon of wisp formation. This mechanism was also demonstrated in our explicit animation model.

Our explicit approach is designed with considerations for today's massively parallel architectures such as GPUs. We have proven this concept by implementing the core part of our simulation on the GPU. We have also proposed how this implementation could be extended to a closed-loop GPU system or to a system where the CPU and GPU collaborate for maximum performance.

Bibliography

ALPHALAB INC. 2015. *The TriboElectric Series*. Salt Lake City, UT, USA. Available from WWW: ⟨`http://www.trifield.com/content/tribo-electric-series/`⟩.

AUDOLY, Basile; POMEAU, Yves. 2010. *Elasticity and Geometry: From hair curls to the non-linear response of shells*. Oxford: Oxford University Press. ISBN 978-0-19-850625-6.

BANDO, Yosuke; CHEN, Bing-Yu; NISHITA, Tomoyuki. 2003. Animating Hair with Loosely Connected Particles. *Computer Graphics Forum*. Vol. 22, no. 3, pp. 411–418. ISSN 1467-8659.

BEER, Ferdinand Pierre. 2010. *Vector Mechanics for Engineers: Statics and Dynamics*. 9th ed. New York: McGraw-Hill Companies. ISBN 978-0-07-352940-0.

BERGOU, Miklós; WARDETZKY, Max; ROBINSON, Stephen; AUDOLY, Basile; GRINSPUN, Eitan. 2008. Discrete elastic rods. *ACM Trans. Graph.* Vol. 27, no. 3, pp. 63:1–63:12. ISSN 0730-0301.

BERTAILS, Florence; KIM, Tae-Yong; CANI, Marie-Paule; NEUMANN, Ulrich. 2003. Adaptive Wisp Tree: A Multiresolution Control Structure for Simulating Dynamic Clustering in Hair Motion. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville, Switzerland: Eurographics Association, pp. 207–213. ISBN 1-58113-659-5.

BERTAILS, Florence; MÉNIER, Clément; CANI, Marie-Paule. 2005a. A practical self-shadowing algorithm for interactive hair animation. In *GI '05: Proceedings of the 2005 conference on Graphics interface*. Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, pp. 71–78. ISBN 1-56881-265-5.

BERTAILS, Florence; AUDOLY, Basile; QUERLEUX, Bernard; LEROY, Frédéric; LÉVÊQUE, Jean-Luc; CANI, Marie-Paule. 2005b. Predicting Natural Hair Shapes by Solving the Statics of Flexible Rods. In *Eurographics Short Presentations*. Dublin, Ireland: Eurographics Association, pp. 81–84. ISSN 1017-4656.

BERTAILS, Florence; AUDOLY, Basile; CANI, Marie-Paule; QUERLEUX, Bernard; LEROY, Frédéric; LÉVÊQUE, Jean-Luc. 2006. Super-helices for Predicting the Dynamics of Natural Hair. *ACM Trans. Graph.* Vol. 25, no. 3, pp. 1180–1187. ISSN 0730-0301.

BLOOMENTHAL, Jules; BAJAJ, Chandrajit. 1997. *Introduction to Implicit Surfaces*. San Francisco: Morgan Kaufmann. The Morgan Kaufmann Series in Computer Graphics. ISBN 9781558602335.

BONANNI, Ugo. 2010. *Haptic Interaction with Virtual Hair*. Genève, Switzerland. Doctoral thesis. Université de Genève.

BONANNI, Ugo; KMOCH, Petr. 2008. Virtual Hair Handle: A Model for Haptic Hairstyling. In *Eurographics 2008 — Annex to the Conference Proceedings*. Aire-la-Ville, Switzerland: Eurographics Association, pp. 311–314. ISSN 1017-4656.

BONANNI, Ugo; KMOCH, Petr; MAGNENAT-THALMANN, Nadia. 2009a. Haptic Interaction with One-dimensional Structures. In *VRST '09: Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*. New York: ACM, pp. 75–78. ISBN 978-1-60558-869-8.

BONANNI, Ugo; KMOCH, Petr; MAGNENAT-THALMANN, Nadia. 2009b. Interaction Metaphors for Modeling Hair using Haptic Interfaces. *International Journal of CAD/CAM*. Vol. 9, no. 1, pp. 93–102. ISSN 1598-1800.

BOYD, Stephen; VANDENBERGHE, Lieven. 2004. *Convex Optimization*. Cambrdige: Cambridge University Press. ISBN 978-0-521-83378-3.

BRADIE, Brian. 2005. *A Friendly Introduction to Numerical Analysis*. London: Pearson. ISBN 978-0130130549.

BUTCHER, J. C. 2003. *Numerical Methods for Ordinary Differential Equations*. 2nd ed. Hoboken, NJ, USA: Wiley. ISBN 978-0471967583.

CERF, Moran; HAREL, Jonathan; EINHAEUSER, Wolfgang; KOCH, Christof. 2008. Predicting human gaze using low-level saliency combined with face detection. In *Advances in Neural Information Processing Systems 20 (NIPS 2007*. Red Hook, NY, USA: Curran Associates, Inc., pp. 241–248. ISBN 978-1-60560-352-0.

CHAI, Menglei; ZHENG, Changxi; ZHOU, Kun. 2014. A Reduced Model for Interactive Hairs. *ACM Trans. Graph.* Vol. 33, no. 4, pp. 124:1–124:11. ISSN 0730-0301.

CHANG, Johnny T.; JIN, Jingyi; YU, Yizhou. 2002. A Practical Model for Hair Mutual Interactions. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. New York: ACM, pp. 73–80. ISBN 1-58113-573-4.

CHOE, Byoungwon; CHOI, Min Gyu; KO, Hyeong-Seok. 2005. Simulating Complex Hair with Robust Collision Handling. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. New York: ACM, pp. 153–160. ISBN 1-59593-198-8.

CHUNG, J.; HULBERT, G. M. 1993. A Time Integration Algorithm for Structural Dynamics With Improved Numerical Dissipation: The Generalized-α Method. *Journal of Applied Mechanics*. Vol. 60, no. 2, pp. 371–375. ISSN 0021-8936.

CONTI, François; BARBAGLI, Federico; MORRIS, Dan; SEWELL, Chris. 2005. *CHAI 3D: An Open-Source Library for the Rapid Development of Haptic Scenes*. Demo paper presented at IEEE World Haptics 2005, Pisa, Italy.

DALDEGAN, Agnes; MAGNENAT-THALMANN, Nadia; THALMANN, Daniel. 1993. An Integrated System for Modeling, Animating and Rendering Hair. *Computer Graphics Forum*. Vol. 12, no. 3, pp. 211–221. ISSN 1467-8659.

DESBRUN, Mathieu; GASCUEL, Marie-Paule. 1996. Smoothed Particles: A New Paradigm For Animating Highly Deformable Bodies. In *Computer Animation and Simulation '96*. Vienna: Springer-Verlag, pp. 61–76. ISBN 978-3-211-82885-4.

DILL, Ellis Harold. 1992. Kirchhoff's Theory of Rods. *Archive for History of Exact Sciences*. Vol. 44, no. 1, pp. 1–23. ISSN 0003-9519.

EBERLY, David H. 2005. *3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic*. San Francisco: Morgan Kaufmann. The Morgan Kaufmann Series in Interactive 3D Technology. ISBN 0-12-229064-X.

EBERLY, David H. 2015. *Geometric Tools* [online]. Redmond, WA, USA. [Visited on 2015-05-28]. Available from WWW: ⟨`http://www.geometrictools.com/`⟩.

FEATHERSTONE, Roy. 1987. *Robot Dynamics Algorithms*. New York: Springer US. The Springer International Series in Engineering and Computer Science. ISBN 978-1-4757-6437-6.

GOLDENTHAL, Rony; HARMON, David; FATTAL, Raanan; BERCOVIER, Michel; GRINSPUN, Eitan. 2007. Efficient simulation of inextensible cloth. *ACM Trans. Graph.* Vol. 26, no. 3, pp. 49:1–49:7. ISSN 0730-0301.

GOLDSTEIN, Herbert. 1980. *Classical Mechanics*. 2nd ed. Boston, MA, USA: Addison-Wesley. Addison-Wesley series in physics. ISBN 0-201-02918-9.

GOYAL, S.; PERKIS, N. C.; LEE, C. L. 2003. Torsional Buckling and Writhing Dynamics of Elastic Cables and DNA. In *ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. New York: American Society of Mechanical Engineers, pp. 183–191. ISBN 0-7918-3703-3.

GRAY, Henry. 1918. *Anatomy of the human body*. 20th ed. Philadelphia, PA, USA: Lea & Febiger.

GUPTA, Rajeev; MAGNENAT-THALMANN, Nadia. 2005. Scattering-Based Interactive Hair Rendering. In *Proceedings of Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 489–496. ISBN 0-7695-2473-7.

HADAP, Sunil; MAGNENAT-THALMANN, Nadia. 2001. Modeling Dynamic Hair as a Continuum. *Computer Graphics Forum*. Vol. 20, pp. 329–338. ISSN 1467-8659.

HADAP, Sunil; CANI, Marie-Paule; LIN, Ming; KIM, Tae-Yong; BERTAILS, Florence; MARSCHNER, Steve; WARD, Kelly; KAČIĆ-ALESIĆ, Zoran. 2007. Strands and Hair: Modeling, Animation, and Rendering. In *ACM SIGGRAPH 2007 Courses*. New York: ACM, pp. 1–150. ISBN 978-1-4503-1823-5.

HAIRER, Ernst; LUBICH, Christian; WANNER, Gerhard. 2006. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. 2nd ed. Berlin: Springer. Springer Series in Computational Mathematics. ISBN 978-3-540-30663-4.

HERRERA, Tomas Lay; ZINKE, Arno; WEBER, Andreas. 2012. Lighting Hair from the Inside: A Thermal Approach to Hair Reconstruction. *ACM Trans. Graph.* Vol. 31, no. 6, pp. 146:1–146:9. ISSN 0730-0301.

HO, Chin-Chang; MACDORMAN, Karl F.; PRAMONO, Z. A. D. Dwi. 2008. Human Emotion and the Uncanny Valley: A GLM, MDS, and Isomap Analysis of Robot Video Ratings. In *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction*. New York: ACM, pp. 169–176. ISBN 978-1-60558-017-3.

HU, Liwen; MA, Chongyang; LUO, Linjie; LI, Hao. 2014. Robust Hair Capture Using Simulated Examples. *ACM Trans. Graph.* Vol. 33, no. 4, pp. 126:1–126:10. ISSN 0730-0301.

IRGENS, Fridtjov. 2008. *Continuum Mechanics*. 2008th ed. Berlin: Springer. ISBN 978-3-540-74297-5.

JAKOB, Wenzel; MOON, Jonathan T.; MARSCHNER, Steve. 2009. Capturing Hair Assemblies Fiber by Fiber. *ACM Trans. Graph.* Vol. 28, no. 5, pp. 164:1–164:9. ISSN 0730-0301.

KAJIYA, James T.; KAY, Timothy L. 1989. Rendering fur with three dimensional textures. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. New York: ACM, pp. 271–280. ISBN 0-201-50434-0.

KIM, Tae-Yong; NEUMANN, Ulrich. 2002. Interactive Multiresolution Hair Modeling and Editing. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. New York: ACM, pp. 620–629. ISBN 1-58113-521-1.

KIRCHHOFF, Gustav. 1859. Über das Gleichgewicht und die Bewegung eines unendlichen dünnen elastischen Stabes. *Journal für die reine und angewandte Mathematik (Crelle)*. Vol. 56, pp. 285–313. ISSN 0075-4102.

KMOCH, Petr; BONANNI, Ugo; MAGNENAT-THALMANN, Nadia. 2009. Hair Simulation Model for Real-Time Environments. In *Proceedings of the 2009 Computer Graphics International Conference*. New York: ACM, pp. 5–12. ISBN 978-1-60558-687-8.

KMOCH, Petr; BONANNI, Ugo; PELIKÁN, Josef. 2010. Towards a GPU-only Rod-based Hair Animation System. In *SA '10: ACM SIGGRAPH ASIA 2010 Posters*. New York: ACM, pp. 7:1–7:1. ISBN 978-1-4503-0524-2.

KOH, Chuan Koon; HUANG, Zhiyong. 2001. A Simple Physics Model to Animate Human Hair Modeled in 2D Strips in Real Time. In *Computer Animation and Simulation 2001*. Vienna: Springer-Verlag, pp. 127–138. ISBN 978-3-211-83711-5.

KOSTER, Martin; HABER, Jorg; SEIDEL, Hans-Peter. 2004. Real-Time Rendering of Human Hair Using Programmable Graphics Hardware. In *Proceedings of the Computer Graphics International '04*. Washington, DC, USA: IEEE Computer Society, pp. 248–256. ISBN 0-7695-2171-1.

LANGER, Joel; SINGER, David A. 1996. Lagrangian aspects of the Kirchhoff elastic rod. *SIAM review*. Vol. 38, no. 4, pp. 605–618. ISSN 1095-7200.

LE GRAND, Scott. 2007. Broad-Phase Collision Detection with CUDA. In *GPU Gems 3*. Boston, MA, USA: Pearson Education, Inc. ISBN 978-0-321-51526-1.

LIANG, Wenqi; HUANG, Zhiyong. 2003. An Enhanced Framework for Real-Time Hair Animation. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 467–471. ISBN 0-7695-2028-6.

LIN, Wei Chin; LIAO, Wei-Kai; LEE, Chao-Hua. 2011. Simulating and Rendering Wet Hair. In *SA '11: SIGGRAPH Asia 2011 Posters*. New York: ACM, pp. 39:1–39:1. ISBN 978-1-4503-1137-3.

LOKOVIC, Tom; VEACH, Eric. 2000. Deep Shadow Maps. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM Press/Addison-Wesley Publishing Co., pp. 385–392. ISBN 1-58113-208-5.

LOVE, Augustus Edward Hough. 1906. *A Treatise on the Mathematical Theory of Elasticity*. 2nd ed. Cambrdige: Cambridge University Press.

MAGNENAT-THALMANN, Nadia; MONTAGNOL, Melanie; GUPTA, Rajeev; VOLINO, Pascal. 2006. Interactive Virtual Hair-Dressing Room. Vol. 3, no. 5, pp. 535–546. ISSN 1686-4360.

MARSCHNER, Stephen R.; JENSEN, Henrik Wann; CAMMARANO, Mike; WORLEY, Steve; HANRAHAN, Pat. 2003. Light scattering from human hair fibers. *ACM Trans. Graph.* Vol. 22, no. 3, pp. 780–791. ISSN 0730-0301.

MORI, Masahiro. 2012. The Uncanny Valley [From the Field]. *IEEE Robotics & Automation Magazine*. Vol. 19, no. 2, pp. 98–100. ISSN 1070-9932.

MORIN, David. 2008. *Introduction to Classical Mechanics*. Cambridge: Cambridge University Press. ISBN 978-0521876223.

NVIDIA CORPORATION. 2015. *CUDA Toolkit Documentation*. Santa Clara, CA, USA. Available also from WWW: ⟨`http://docs.nvidia.com/cuda/cuda-c-programming-guide/`⟩.

PAI, Dinesh K. 2002. STRANDS: Interactive Simulation of Thin Solids using Cosserat Models. *Computer Graphics Forum*. Vol. 21, no. 3, pp. 347–352. ISSN 1467-8659.

PHONG, Bui Tuong. 1975. Illumination for Computer Generated Pictures. *Communications of the ACM*. Vol. 18, no. 6, pp. 311–317. ISSN 0001-0782.

POLICARPO, Fábio; OLIVEIRA, Manuel M. 2006. Relief Mapping of Non-height-field Surface Details. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. New York: ACM, pp. 55–62. ISBN 1-59593-295-X.

POLICARPO, Fábio; OLIVEIRA, Manuel M.; COMBA, João L. D. 2005. Real-time Relief Mapping on Arbitrary Polygonal Surfaces. *ACM Trans. Graph.* Vol. 24, no. 3, pp. 935–935. ISSN 0730-0301.

PRESS, William H.; FLANNERY, Brian P.; TEUKOLSKY, Saul A.; VETTERLING, William T. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press. ISBN 978-0521431088.

ROBBINS, Clarence R. 2002. *Chemical and Physical Behavior of Human Hair*. 4th ed. New York: Springer Science & Business Media. ISBN 978-0-387-95094-5.

SCHEUERMANN, Thorsten. 2004. Practical Real-time Hair Rendering and Shading. In *ACM SIGGRAPH 2004 Sketches*. pp. 147. ISBN 1-58113-896-2.

SCHNEIDER, Philip J.; EBERLY, David H. 2002. *Geometric Tools for Computer Graphics*. San Francisco: Morgan Kaufmann. The Morgan Kaufmann Series in Computer Graphics. ISBN 978-1558605947.

SELLE, Andrew; LENTINE, Michael; FEDKIW, Ronald. 2008. A Mass Spring Model for Hair Simulation. *ACM Trans. Graph.* Vol. 27, no. 3, pp. 1–11. ISSN 0730-0301.

SHARMA, P.; CHEIKH, F.A.; HARDEBERG, J.Y. 2009. Face saliency in various human visual saliency models. In *Proceedings of 6th International Symposium on Image and Signal Processing and Analysis*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 327–332. ISBN 978-953-184-135-1.

SOBOTTKA, Gerrit; LAY HERRERA, Tomás; WEBER, Andreas. 2008. Stable Integration of the Dynamic Cosserat Equations with Application to Hair Modeling. *Journal of WSCG*. Vol. 16, no. 1–3, pp. 73–80. ISSN 1213-6972.

SPILLMANN, J.; TESCHNER, M. 2007. CoRdE: Cosserat Rod Elements for the Dynamic Simulation of One-dimensional Elastic Objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Aire-la-Ville, Switzerland: Eurographics Association, pp. 63–72. ISBN 978-1-59593-624-0.

SWIFT, J. A. 1995. Some simple theoretical considerations on the bending stiffness of human hair. *International Journal of Cosmetic Science*. Vol. 17, no. 6, pp. 245–253. ISSN 0142-5463.

TINWELL, Angela. 2014. *The Uncanny Valley in Games and Animation*. Boca Raton, FL, USA: A K Peters/CRC Press. ISBN 978-1466586949.

VERNALL, Donald G. 1961. *American Journal of Physical Anthropology*. Vol. 19, no. 4, pp. 345–350. ISSN 1096-8644.

VOLINO, Pascal; MAGNENAT-THALMANN, Nadia. 2006. Real-Time Animation of Complex Hairstyles. *IEEE Transactions on Visualization and Computer Graphics*. Vol. 12, no. 2, pp. 131–142. ISSN 1077-2626.

VRIES, Renko de. 2005. Evaluating changes of writhe in computer simulations of supercoiled DNA. *The Journal of Chemical Physics*. Vol. 122, no. 6, pp. 064905–1–064905–5. ISSN 0021-9606.

WARD, Kelly; LIN, Ming C. 2003. Adaptive Grouping and Subdivision for Simulating Hair Dynamics. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 234–243. ISBN 0-7695-2028-6.

WARD, Kelly; LIN, Ming C.; LEE, Joohi; FISHER, Susan; MACRI, Dean. 2003. Modeling Hair Using Level-of-Detail Representations. In *Proceedings of 16th International Conference on Computer Animation and Social Agents (CASA'03)*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 41–47. ISSN 1087-4844.

WARD, Kelly; GALOPPO, Nico; LIN, Ming C. 2006. A Simulation-based VR System for Interactive Hairstyling. In *IEEE Virtual Reality Conference (VR 2006)*. Los Alamitos, CA, USA: IEEE Computer Society. ISSN 1087-8270.

WORTMANN, Franz J.; SCHWAN-JONCZYK, A. 2006. Investigating hair properties relevant for hair handle. Part I: hair diameter, bending and frictional properties. *International Journal of Cosmetic Science*. Vol. 28, no. 1, pp. 61–68. ISSN 0142-5463.

YUKSEL, Cem; KEYSER, John. 2008. Deep Opacity Maps. *Computer Graphics Forum*. Vol. 27, no. 2, pp. 675–680. ISSN 1467-8659.

YUKSEL, Cem; TARIQ, Sarah. 2010. Advanced Techniques in Real-time Hair Rendering and Simulation. In *ACM SIGGRAPH 2010 Courses*. New York: ACM, pp. 1:1–1:168. ISBN 978-1-4503-0395-8.

YUKSEL, Cem; SCHAEFER, Scott; KEYSER, John. 2009. Hair Meshes. *ACM Trans. Graph.* Vol. 28, no. 5, pp. 166:1–166:7. ISSN 0730-0301.

ZINKE, Arno; YUKSEL, Cem; WEBER, Andreas; KEYSER, John. 2008. Dual Scattering Approximation for Fast Multiple Scattering in Hair. *ACM Trans. Graph.* Vol. 27, no. 3, pp. 32:1–32:10. ISSN 0730-0301.