Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Hair Simulation

Liliane Rodrigues de Almeida

JUIZ DE FORA
NOVEMBRO, 2014

Hair Simulation

Liliane Rodrigues de Almeida

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Marcelo Bernardes Vieira

JUIZ DE FORA
NOVEMBRO, 2014

Hair Simulation

Liliane Rodrigues de Almeida

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Marcelo Bernardes Vieira
Doutor em Ciência da Computação, ENSEA-UCP

Rodrigo Luis de Souza da Silva
Doutor em Engenharia de Sistemas, UFRJ

Marcelo Caniato Renhe
Mestre em Engenharia de Sistemas, UFRJ

JUIZ DE FORA
DE NOVEMBRO, 2014

Aos meus amigos.

Resumo

A simulação de cabelo tem uma grande diversidade de abordagens para modelagem, dinâmica e renderização. Este trabalho visa apresentar a evolução destes três processos ao longo dos anos, estudar o modelo capaz de simular o básico de cada um deles e identificar os requisitos para se construir o simulador. O foco deste trabalho é a simulação da dinâmica do cabelo e de suas características como não extensibilidade do fio, volume e atenuação do movimento. Cabelos cacheados e ondulados são gerados, e o modelo de iluminação específico para fios é implementado com *shaders*.

Palavras-chave: simulação de cabelo, renderização, modelagem, dinâmica.

Abstract

Hair simulation has a broad diversity of approaches for modeling, dynamic simulation, and rendering. This work aims to present the evolution of these three processes along the years, to study the model capable of simulating the basic of each one of them and identify the requisites for building a hair simulator. The focus of this work is the simulation of hair dynamics and the hair properties as strand non-extensibility, volume and movement damping. Curly and wavy hairs are generated, and the specific lighting model for strands is implemented with shaders.

Keywords: hair simulation, rendering, modeling, dynamics.

Acknowledgments

I thank all my friends of the Group for Computer Graphics, Image and Vision for somehow helping me with this work. I also thank my friends and teachers from GET (Tutorial Education Group) for everything during my graduation.

My crazy-curly-hair friends (Isis and Babi too) for letting me make part of their group. And some other "hairy" friends.

My dog, Sophie, for the ability to cheer me up without any effort.

I also thank Hugo for showing me several new worlds. The good and the ugly ones.

All the teachers of the DCC for the lessons.

And I thank many other important people to me that I won't forget, specially my mom for her work, and my dad for his family.

Man can live without science, he can live without bread, but without beauty he could no longer live, because there would no longer be anything to do to the world. The whole secret is here, the whole of history is here.

Dostoevsky (The Demons)

Contents

List of Figures

List of Symbols

$\vec{p}$	Particle position $\vec{p} = (x, y, z)$ and $x, y, z \in \mathbb{R}$
$\vec{v}$	Particle velocity $\vec{v} = (x, y, z)$ and $x, y, z \in \mathbb{R}$
$\vec{a}$	Particle acceleration $\vec{a} = (x, y, z)$ and $x, y, z \in \mathbb{R}$
$\vec{x}_i$	Last valid position of particle i
$\vec{d}_i$	Difference between the updated and the corrected position of particle i
Δt	Time step
i	Particle index
D_{xyz}	Density on the voxel vertex (x, y, z)
$\vec{V}_{xyz}$	Velocity on the voxel vertex (x, y, z)
$D(x, y, z)$	Function that returns the density value at (x, y, z)
g_x^i	Coordinate x of the gradient of particle i

Abbreviation List

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
BCSDF	Bidirectional Curve Scattering Distribution Function
DFTL	Dynamic Follow The Leader
DOF	Degree of Freedom
FDM	Finite Difference Method
FTL	Follow The Leader
LOD	Level of Detail
SPH	Smoothed Particle Hydrodynamics
TDM	Tridiagonal Matrix Formulation

1 Introduction

In the field of computer animation, hair simulation has been a recurrent topic of research, composed by the processes of modeling, dynamics and rendering (Ward et al., 2007). It is one of the components that increase the level of realism shown on movies (Iben et al., 2012) and video-games (Han et al., 2012). In these virtual worlds, the sense of reality lies on the details. One of them is the character's hair. The properties of this material are unique, yet can be found in other objects such as grass, fur, or any set of strings that interact with each other (threads, elastic rods, cables and ropes).

Usually, a complex scene has a large number of elements, so hair simulation must not be computationally expensive and must keep a believable movement and look. Many approaches have been developed, from approximations to physically based simulation (but not in an exact manner), as presented in the survey of Ward et al. (2007). Their work also highlights that the development of an approach between these extremes will be driven by the type of application, and so the research about the hair simulation as whole.

In the entertainment industry there are two applications for hair simulation: animated movies and video games. These sub-lines of research can be represented by Iben et al. (2012) and Müller et al. (2012) respectively. Both of them simulated straight and curly hair, hair volume and response to external forces through simplifications of the real hair behavior.

This work aims to present the existent methods for hair simulation. As there is a broad diversity of methods in the literature, Chapter 2 describes the evolution of approaches along the years for theoretical basis. Chapter 3 explains the adopted approach and implementation issues. Chapter 4 shows the results. And Chapter 5 presents the conclusion and future works.

1.1 Problem Definition

The problem addressed in this work involves methods to improve the realism of hair dynamics by simulating as large a number of strands as possible, and using approximations of some known characteristics of hair, such as:

- Strand non-extensibility and bending;
- Mutual repulsion to create volume;
- Energy loss due to friction.

1.2 Objectives

The primary objective of this work is to study hair simulation methods based on the dynamic behavior of real hair. The secondary objectives are:

- Represent several kinds of hair;
- Implement one of the most recent methods of dynamic modeling hair simulation without considering performance;
- Present hair simulation fundamentals.

2 State-of-Art

This chapter presents related works of the three processes to recreate hair on the computer as defined in Ward et al. (2007) and also highlights the methods that lead to the scheme implemented in this work.

2.1 Modeling

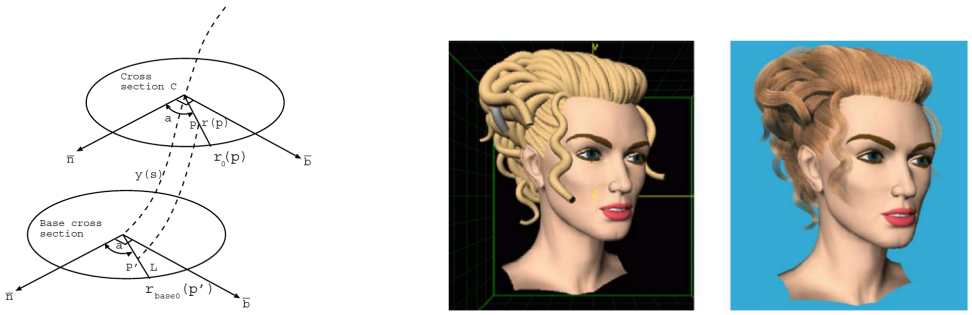
The scope of the modeling process is the hairstyling. Here we specify the global and local shape, density, distribution and orientation, i.e. the hair geometry. There are many ways to produce these characteristics, so Ward et al. (2007) divided the hair modeling approaches that are based on geometry, physics and images. The evaluation was made considering the shape flexibility (number of hairstyles), user control (hair details definition), and time for manual setup (how fast the user can get the final style). The Table 2.1 classifies these features.

Method	**Hair Shapes**	**User Control**	**Manual Time**
Gen. Cylinders	flexible	high	slow
Surfaces	limited to straight	high	fast
Physical Volumes	limited, details hard	cumbersome	medium
Photos	limited, must exist	none	fast
Sketches	limited, details hard	medium	fast

Table 2.1: Analysis of global shaping methods (Ward et al., 2007)

On the geometry-based approaches, the generalized cylinders were introduced by Yang et al. (2000), that fits in the category of wisp model approach, presented by Watanabe and Suenaga (1992). The hair is modeled through a one-dimensional curve to represent the center of the cylinder, and a cylinder shell for the wisps. Along the hair length, the cross sections can vary their size, so braids, ponytails can be recreated. Curliness is also possible, but they concluded that it would take too much time to model and render. To change the cylinder's shape and to better render the hair details, they selected a volume density model. They generated a discrete image, called grid map, to randomly

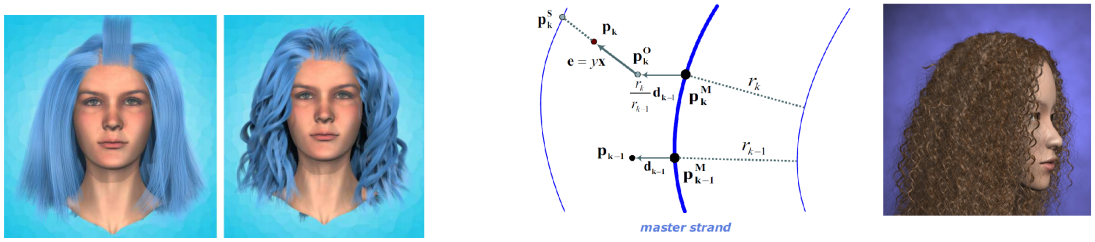
place virtual strands on each cross section of the cylinder. The strands are oriented by these dots to build the density distribution field for volume construction. Finally, they developed a ray-tracing method that samples points by finding the intersection between the ray and the cylinder's boundary, and renders the volume by determining the corresponding density value on the grid map. Kim and Neumann (2002) extended the work of



(a) Hair wisp represented by a cylinder. (b) Full model.

Figure 2.1: The Cluster Hair Model (Yang et al., 2000).

Yang et al. (2000) by adding an hierarchy of generalized cylinders, manipulated through a multiresolution technique. Their system has editing tools to subdivide, move, copy, twist the clusters over the scalp, and rendering options. Choe and Ko (2005) proposed a static hairstyling in three steps: generation of wisps from single strands (keyhair); wisp deformation to orient the strands of this subset; global constraints to produce more hairstyles. Each of these steps require several parameter setting by the user. As result, they showed a model able to create braids, curliness and ponytails. The parametric surface approach

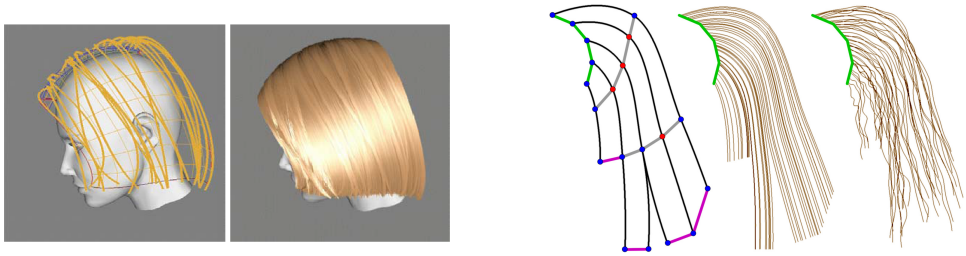


(a) Results from Kim and Neumann (2002). (b) Keyhair and full model of Choe and Ko (2005).

Figure 2.2: Hair modeling through multiresolution in Figure 2.2(a) and through keyhair in Figure 2.2(b).

represents a group of strands, also referred as hair strips, by a two-dimensional surface. It reduces the number of objects manipulated during the simulation and rendering. For that

reason, it is fast enough for real time applications. Kim and Neumann (2000) developed the Thin Shell Volume method to create the strands inside on a surface defined by the user and to simulate the hair-hair interaction, but did not modeled curls. Later, Yuksel et al. (2009) used this idea to create an interactive tool to model within minutes straight and curly hair by wisp curves "extracted" and edited from the polygon mesh.



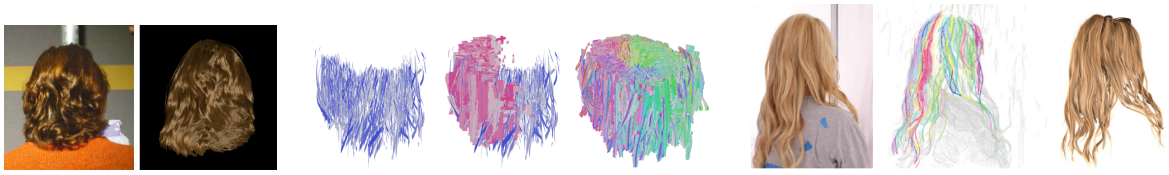
(a) Thin Shell Volumes Kim and Neumann (2000). (b) Hair Meshes Yuksel et al. (2009).

Figure 2.3: Approaches for modeling hair using surfaces and meshes.

The physically-based approaches consists in modeling hair appropriately for the dynamic simulation, predicting the strands' positions. For this reason, techniques such as fluid flow (Hadap et al., 2000, 2001), vector field (Choe and Ko, 2005; Petrovic et al., 2005; Müller et al., 2012) and rigid-body (Hadap et al., 2001) were adapted to manipulate hair data. More details about these techniques are given in Section 2.2.

Finally, the image-based approaches are the fastest in terms of modeling by the user. The methods can be: photograph-based for creation of real hair and sketch-based for cartoon hair. According to Ward et al. (2007), the first one started with Kong et al. (1997). They recreated the hair volume from pictures of four different viewpoints of a real hair model. Grabli et al. (2002) chose to infer the strands orientation based on the hair reflectance from a single viewpoint and a moving light source. They get the strand orientation in two dimensions using Sobel filter on horizontal and vertical directions. To recover the 3D direction, they extract the light components from the color and highlights of hair, and elect the most reliable strand. As their approach depends on detecting high contrast regions, they reconstructed only part of the hairstyle. Paris et al. (2004) continued the work of Grabli et al. (2002) recreating the geometric information of a full hairstyle from an orientation vector field by four viewpoints. For that, they used several filters for edge and line detection. Some discontinuities can still appear, so

they correct it based on local coherence. Wei et al. (2005) presented a more practical method eliminating some conditions as controlled illumination and fixed camera. They adapted one of the filters of Paris et al. (2004) and used multiple viewpoints combination to reconstruct one hairstyle. Jakob et al. (2009) recreated all the fibers of a sparse wisp rather than determining the whole hair volume, as previous works. They vary the focus of a macrophotograph through the wisp volume in many viewpoints to generate data for late fiber growing. Herrera et al. (2012) used thermal images to reconstruct the hair on the visual hull volume. Their approach dismiss manual hair-skin segmentation and special care about light. Luo et al. (2013) presented a system to reconstruct the underlying hair wisps from a set of static images independently from the light information. They also used different viewpoints. However, the accuracy depended strongly on the initial point cloud extracted from the images and the approach still needed manual interference for the image input, as Hu et al. (2014) explains. To eliminate this requisite, Hu et al. (2014) built a data driven wisp reconstruction with a voting-based algorithm.



(a) Paris et al. (2004) (b) Jakob et al. (2009) (c) Hu et al. (2014)

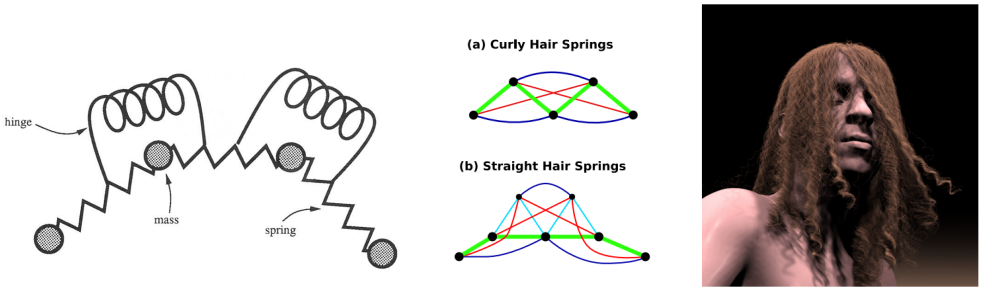
Figure 2.4: Different approaches for hair reconstruction based on photographs.

2.2 Dynamics

The dynamic process concerns the movement of a single hair strand and of the hair at large. The items pointed out in the problem definition belongs to this step of hair simulation. They can be achieved depending on the choice of the hair modeling. To clarify these relations, Ward et al. (2007) separated the lines of research in four sub-areas: mass-spring system, projective dynamics, rigid multi-body serial chain and dynamic Super Helices.

The mass-spring system was one of the primary approaches to recreate the hair dynamics. It was proposed by Rosenblum et al. (1991) and have been studied and modified

along the years for its simplicity (Plante et al. (2001); Petrovic et al. (2005); Selle et al. (2008); Iben et al. (2012), and others). The strand is a series of point masses linked by strong linear springs, to prevent stretching in straight hair, and hinges (angular spring) for bending. These properties fits better in cloth simulation, and that is why it keeps being improved on cloth research field (see Liu et al. (2013)). Baraff and Witkin (1998) and some previous works concluded that the motion of cloth (and so the hair's) results in a stiff differential equation. Which means solving it demands small steps to find the solution. Therefore, implicit time integration (Backward Euler method), for instance, can ignore this restriction while maintaining the stability. However, solving linear systems is not fast enough for real time applications (Han et al., 2012). Without considering that as a goal, Selle et al. (2008) represented all hairs explicitly and introduced a framework completely based on mass-spring model to handle bend, torsion, collision and curly hair. The springs are arranged in different ways to simulate these properties and can be created and undone during the simulation.



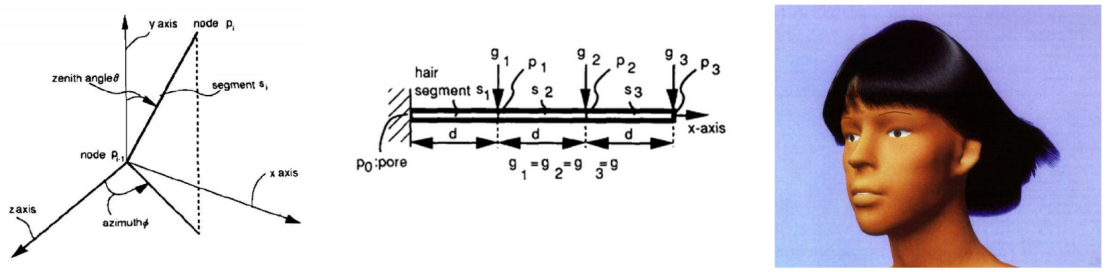
(a) Mass-spring-hinge system of Rosenblum et al. (1991).

(b) Different types of springs to handle hair curve, bend, twist, torsion and to prevent collapse, and the resulting simulation (Selle et al., 2008).

Figure 2.5: Two types of mass-spring system.

The projective dynamics is represented by the work of Anjyo et al. (1992). They modeled the strands as a single cantilever beam (straight beam fixed in one side and loose on the other) divided in segments. The hair bending is interpreted as the beam deformation given by a two-dimensional differential equation, and two one-dimensional differential equation to rule the dynamics with the variables are the azimuth and zenith angles. The segment needs to be projected on the planes spanned by these angles. Each strand segment (called "stick") is ruled by these equations and its numerical solution is a

second order recurrence formula. The angles in next time step are obtained by knowing the two last ones and the strand update is done from the pore to the tip. In the end, all positions are valid with respect to length constraint. External forces are embedded in the equations, and farther from the scalp, more intense it gets. Although the modeling and animation are well solved, the original hairstyle is not recovered after the simulation starts (Hadap et al., 2001).



(a) Projecting external forces on the strand segment.

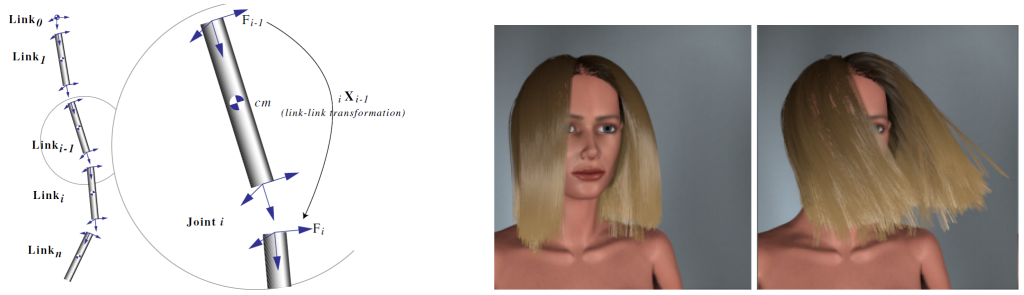
(b) Gravity distribution over the cantilever beam.

(c) Final simulation.

Figure 2.6: Strand structure and final simulation of the model of Anjyo et al. (1992)

Since both methods cited above can not simulate torsions/twisting (before Selle et al. (2008)), another one, quite similar to the cantilever beam modeling, originated from the robotics field: rigid multi-body serial chain. As the name indicates, the strand is divided in non-deformable solid bodies positioned in series. This approach for hair animation was introduced by Hadap et al. (2001). Each strand segment has the same size and is connected to other segment through a three DOF spherical joint. The joints have angular accelerations induced from the external forces by a linear equation. To model the hair as a whole, and its interaction with other objects and itself, Hadap et al. (2001) chose the smoothed particle hydrodynamics (SPH), one of the numerical methods of the Lagrangian viewpoint of fluid dynamics. This method takes into account of neighbor particles, so it can simulate hair-hair and hair-air interaction. They considered hair as continuum, i.e. every point of a medium has a defined physical property. At large scale this assumption can produce realistic results, since the distances between strands "are much smaller than the overall volume". However, strand decoupling is lost due to the continuum approach. McAdams et al. (2009) solved this using a hybrid approach of the Eulerian and Lagrangian approach for fluid dynamics, and implicitly manipulating the

velocities.



(a) Strand structure. (b) Final simulation.

Figure 2.7: The rigid multi-body serial chain of Hadap et al. (2001)

To complete the desirable properties of a hair model, there is the curliness, not to be confused with torsion. Bertails et al. (2006) noticed that previous works did not simulate exactly the deformations of curly hair, but simply rendered the helical/helix shape over an straight extensible strand. So, relying on the mechanics and structure of real hair, they applied the Kirchhoff equations to an inextensible elastic rod to predict the hair motion. The strand is shaped by a centerline curve divided in segments with every point associated to a material frame (three orthonormal axis aligned with the tangent), to track the twist vector. The Kirchhoff equations "describes the dynamics of a rigid body in an ideal incompressible fluid", so bending and torsion are included in their model. After many trials, they opted for the Lagrangian mechanics to derive the motion equations and then integrating them over time with a semi-implicit Newton method. To simulate a full head, they modeled hair with a sparse set of guide strands, or keyhairs.

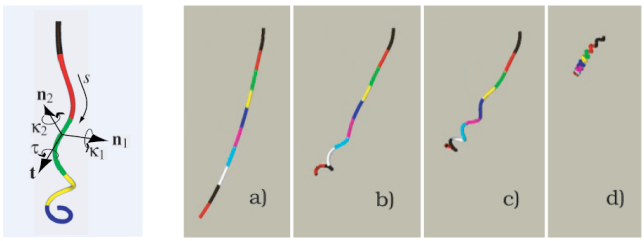
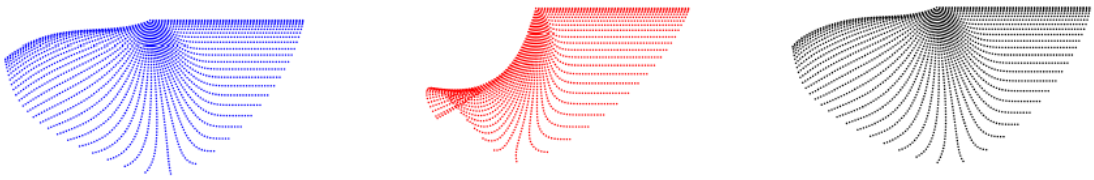


Figure 2.8: Super-Helix geometry and different curvatures and twist of the strand (Bertails et al., 2006).

In the subsequent years, other kind of approaches began to appear based on previous works with more basis on physics. For instance, the Dynamic Follow The Leader

(DFTL) of Müller et al. (2012) is an adaptation for hair and fur simulation of the Position Based Dynamics (PBD) of Müller et al. (2006). The method of Müller et al. (2012) simulates thousands of hair strands in real time with the support of the GPU (Graphics Processing Unity). The core of their approach is on limiting the stretch in a single iteration per frame. To do that, the distance constraint is geometric, i.e., no physics basis were used to correct the particle positions. This was the cause of the problems documented in their paper. Since their work is studied here, more details about the implementation are given in Chapter 3.

The paper of Han et al. (2013) is also about simulating inextensibile hair. They compared their work with the PBD, the physically correct simulation, and with the DFTL. They formulated a linear system to limit the distance of two connected vertices. The adaptation to the strand geometry resulted in a tridiagonal symmetric matrix, that can be solved without actually building a matrix, as they explain. A very low stretch still appears but it is not visually significant. The simulation is very similar to the PBD after 40 iterations, which means that there is less damping added on the simulation. And the bending constraint is solved by 8 iterations of the PBD algorithm. Some limitations are related to the small time step to guarantee stability of the constraint linearization.



(a) TMF. (b) DFTL. (c) PBD - 40 iterations.

Figure 2.9: Simulation of a single strand using different methods for comparison (Han et al., 2013).

Iben et al. (2012) based their method on the extensible elastic rods to model curly hair to attend the artists demands. Their method was developed for animated movies. So due to the exaggerated movements of the characters, the curls may get straight if a pure physical model was simulated. Instead, they added linear springs to control the bending and to maintain the curl shape.

2.3 Rendering

Rendering is the final step to recreate hair, and so for other objects. In general, rendering deals with generating two-dimensional images from three-dimensional models in a scene. Geometry, textures, viewpoint, illumination and shading are used to compute the color and intensity of every pixel on screen. In the case of hair, self-shadows, light scattering and reflectance are the main aspects to render it with an appearance similar to the real one. These effects can be achieved depending on the global and local representation of hair, as Ward et al. (2007) describes, resulting in two kinds of renderers: for explicit and implicit geometry. Both of them have its advantages and disadvantages, but most recent methods deal well with their past rendering limitations, as described below.

Ward et al. (2007) explains that in the explicit representation, hair fibers are curves in space. Since a strand is very thin, it can be smaller than a pixel and cause aliasing. This concept from the signal processing field is treated as a noise and side effect of the sampling operation. Research on these visibility issues focus mainly on point or line sampling. Barringer et al. (2012) shows that the number of samples and the distance between the camera and the hair are the base conditions to get an accurate visibility of hair. So they represent the strands as three-dimensional Bézier splines with varying thickness, use spatial line sampling, and test their intersection. As result, their work shows images with almost no noise in several viewpoints and scales.

(a) Complete rendering of the hair.

(b) Hair rendering when the camera is near the strands.

Figure 2.10: Curve rendering of Barringer et al. (2012)

The implicit geometry is represented by a scalar-valued three-dimensional grid. This data structure is used to store densities to create volumes or surfaces through signed

distance fields. Petrovic et al. (2005) used this information to model hair response to illumination, observing that light reflection of hair corresponds to surface normals, like in the classical Phong model. Implicit representation is also suitable to render with texels, the 3D textures introduced by Kajiya and Kay (1989).

(a) Hair isosurface. (b) Normals illumination. (c) Kajiya shading.

Figure 2.11: Hair rendering through volumes (Petrovic et al., 2005)

Besides the geometry rendering detail, hair renderers need to model light interaction of each strand. Real hair filaments are composed of amorphous proteins that act as a transparent medium and other ones that contains the pigment that absorb the light (Ward et al., 2007).

Kajiya and Kay (1989) was one of the first works to render details of a furry surface. For that, they generalized the volume densities with *texels*, a three dimensional array of parameters approximating the visual properties of a collection of microsurfaces. The parameters are: a density scalar function, measuring how much of a volume cell is covered by microsurfaces; a local orientation of the microsurface (orthogonal basis formed by the normal, the tangent, and the bi-normal), and a lighting model (bidirectional lighting reflection function). Texels for hair rendering can be simplified by storing the density at each point of the volume. If the hair does not change color, the reflectance function is the same for all strands. The lighting model uses only the tangent of the strand to compute the color of its thin cylindrical surface. The ray tracing technique is applied to texels. The ray intersection with the texel's rectangular shape is calculated by solving a quadratic equation. The diffuse component is the Lambertian shading model applied to a cylinder. The light is reflected in all directions and its intensity is weighted by angle of reflection. The highlights of hair are defined by the specular component. On a cylinder, the maximum brightness can be seen when the observer is exactly looking at

the mirror angle of the incident light.

(a) Kajiya shading. (b) Marschner shading. (c) Real picture.

Figure 2.12: Hair shading comparison from Marschner et al. (2003)

Marschner et al. (2003) extended the Kajiya's model for physically based rendering. They based their model on two researches about real hair and remade their experiment on measuring the light scattering. The three works concluded that the specular white peak does not occur on the specular angle, and there is a second colored light peak from the internal reflection of the back side of the strand. When the beam of light hits the strand, part of it is reflected (first peak), the other is transmitted, reflected on the back side and transmitted again (second peak, called by them as glint).

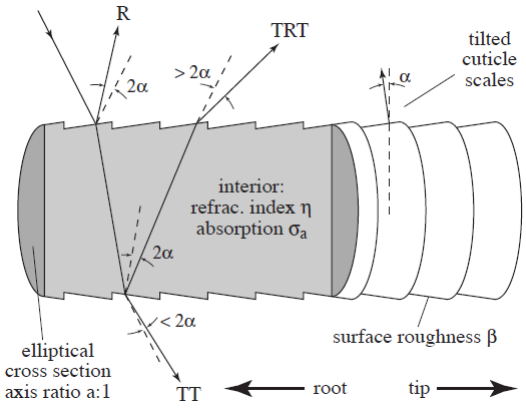


Figure 2.13: Light over the hair strand (Marschner et al., 2003)

Marschner et al. (2003) also captured the intensity of the red, green and blue colors of the reflected light and observed differences between blond and dark hair. Blond hair showed a secondary highlight that varies considerably with the incident light angle, and that depends mostly on the red color. All these behaviors are due to the strand geometry: a

cuticle (tilted cylinder) that covers the cortex and medulla (colored cylinder). Marschner et al. (2003) approximated these optical properties through Bravais's law, Snell's law and Fresnel's equations, as described in physics literature. To fit the computation on a rendering system, they build their model considering only what is useful for a renderer, introducing the longitudinal and azimuthal scattering functions. The longitudinal ones are normalized Gaussians to model the blur and the angle deviation effect. The azimuthal ones models the highlights and glint effects smoothed by the surface roughness. The validation of their work is made through qualitative match of the model and the measurements.

The work of Marschner et al. (2003) was improved in many other papers of physically based rendering. Zinke and Weber (2007) contributed by defining the BCSDF (Bidirectional Curve Scattering Distribution Function). These 'BxDF' functions relates the angles of the incident and outgoing rays over an infinitesimal surface patch, and returns the fraction of reflected or transmitted light energy. The work of Marschner et al. (2003) was considered as a brief introduction of the BCSDF, since they did not take in account the movements or proximity of light. So, in order to render hair in any camera distance, Zinke and Weber (2007) defined a near and far field shading model.

Sadeghi et al. (2010) presented a shading system for 3D animated movies. They explained that in film production, the artists needs to control the hair parameters like color, highlights intensity and width and frequency that glints appear. The physically-based shading models are able to simulate these features, but the parameters are tied to each other because of the energy conservation. Thus, they chose to decouple the single scattering model of Marschner et al. (2003) and the multiple scattering approximation of Zinke et al. (2008), the Dual Scattering. Also, they separated the parameters of the single and multiple scattering to attend requests of the art direction of changes in only one component.

3 Hair Simulation

This work presents the method of Müller et al. (2012) for hair and fur simulation, because it simplifies the physical properties of hair. Therefore, every strand is simulated, a voxel grid is implemented, and the strands are modeled as a serial chain of point mass. Each of these structures can mimic the hair properties described on Section 1.1 and to achieve the goals presented in Section 1.2.

The system developed in this work is classified as Dynamical, because it defines the motion rules over time. Thus, it works with time integration in steps. In this scenario, frequently the time step needs to be very small to result in a smooth simulation, and it is not necessary to set it fixed.

In this work, the first particle of the strand is attached to a vertex of the scalp mesh, for simplicity. Any rotation or translation of the head is replicated to the strand root. Due to the distance constraint, the next particle is moved towards or away from the previous one. The dynamics of a single strand is explained in Section 3.1, and the hair interaction, in Section 3.2. The rendering of the color and the two highlights of hair are described in Section 3.3 for a better visualization of several strands.

3.1 Strand Dynamics

Firstly, the dynamics of a single strand needs to be defined. Müller et al. (2012) proposed an algorithm inspired by the Position Based Dynamics (Müller et al., 2006) that manipulates the positions directly, and infers the velocities from the difference between the last and the current position. Their algorithm ensure zero stretch in one iteration, as the Follow The Leader of Brown et al. (2004). Müller et al. (2012) presents the structure of the Position Based Dynamics as in Algorithm 1.

Algorithm 1: Position Based Dynamics - structure

1 $\vec{p} \leftarrow \vec{x} + \Delta t \cdot \vec{v} + \Delta t^2 \cdot \vec{F}$
2 $\vec{p} \leftarrow$ SolveConstraints($\vec{p}$)
3 $\vec{v} \leftarrow \dfrac{\vec{p} - \vec{x}}{\Delta t}$
4 $\vec{x} \leftarrow \vec{p}$

As stated in Section 3, the strand is divided in segments of equal length, and between them there are points with the same mass. Each of these particle store its position, velocity, and acceleration at a given time step. When the simulation starts, all of the external forces, like gravity and wind, are summed and stored as an acceleration vector ($\vec{F}$ in line 1), according to the Second Law of Newton of Motion, $\vec{a} = \vec{F}/mass$. The current position became the old position ($\vec{x}$), then the particles position are updated by:

$$\vec{v}_{tmp} = \vec{v} + \vec{a} \cdot \Delta t$$

$$\vec{p} = \vec{p} + \vec{v}_{tmp} \cdot \Delta t$$

$\vec{v}_{tmp}$ is a temporary vector, since the velocity $\vec{v}$ does not need to be saved in this stage because it will be overwritten in line 3 of the Algorithm 1.

After this update, probably the distance constraint is broken. To recover the valid state, a single pass of the Follow the Leader algorithm inside the "SolveConstraints" procedure could solve this, as Müller et al. (2012) says. Even with the distance corrected, their experiment showed a strange behavior of the strand, which started to spin quickly around a particle. Over time, the piece of strand spinning shrinks, as if the center of rotation had passed to the next particle. The Figure 3.1 from Müller et al. (2012) shows the result of this simulation.

Searching for the cause of these results, Müller et al. (2012) examined closely their system. The FTL algorithm analyzes the strand segments one by one inside the loop over the strand. It corrects only the position of the second particle and then passes to the next segment. This operation corresponds to a system in which the previous particle has infinite mass, therefore it can not be moved. The spins of the following particles is due to

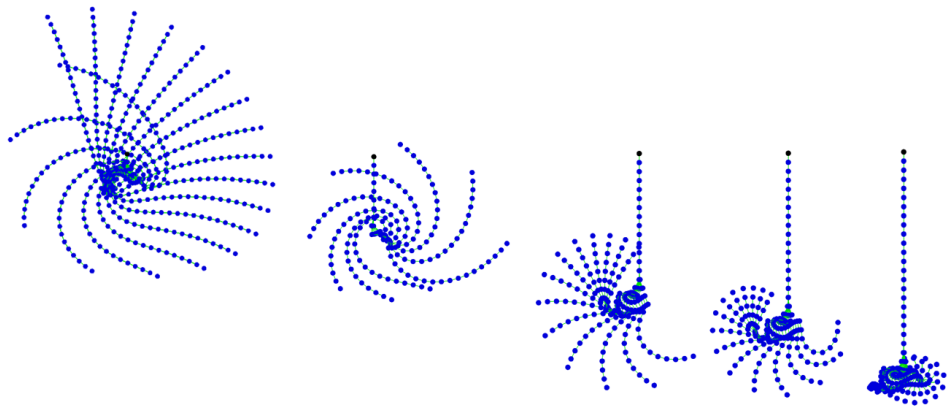


Figure 3.1: Simulation a single strand using the Follow The Leader from Müller et al. (2012)

the accumulated kinetic energy of the strand, affecting mostly the particles on the tail, since the sum of their masses is very small. Another reason for the strange behavior is that the simulation of Dynamic Systems is a coupled problem. Their constraints almost always can not be solved separately.

3.1.1 Dynamic Follow the Leader

Müller et al. (2012) explains that they tried many methods to derive the dynamics without modifying the original FTL algorithm. But all of them failed because of the uneven mass distribution inherent to the FTL projection. During the research they found a simple solution: change the velocity update by adding a correction factor. Their main contribution is

$$\vec{v}_i \leftarrow \frac{\vec{p}_i - \vec{x}_i}{\Delta t} + s_{\text{damping}} \cdot \frac{-\vec{d}_{i+1}}{\Delta t}, \tag{3.1}$$

where $\vec{v}_i$ is the new velocity, $\vec{p}_i$ is the current particle position, $\vec{x}_i$ is the last valid position, $s_{\text{damping}} \in [0, 1]$ is a coefficient to control the damping, $\vec{d}_{i+1}$ is the correction vector for the position of the next particle produced by the FTL, and Δt is the time step.

The idea behind the Equation 3.1 is explained in Müller et al. (2012). To mimic the movement of two particles with same mass, the correction vector of the FTL, $\vec{d}_i$, must also corrects the position of the previous particle ($\vec{p}_{i-1} \leftarrow \vec{p}_{i-1} - \vec{d}_i$). But this will violate the distance constraint between particles $i-2$ and $i-1$. To avoid this, instead of correcting the positions, the velocity of particle i is updated using the correction vector

of particle $i + 1$. This way the distance constraint is still solved in one iteration over the strand, but with the price of adding numerical damping. The simulation of the DFTL yields the motion represented in Figure 3.2 (from Müller et al. (2012)). The full algorithm is presented in Algorithm 2.

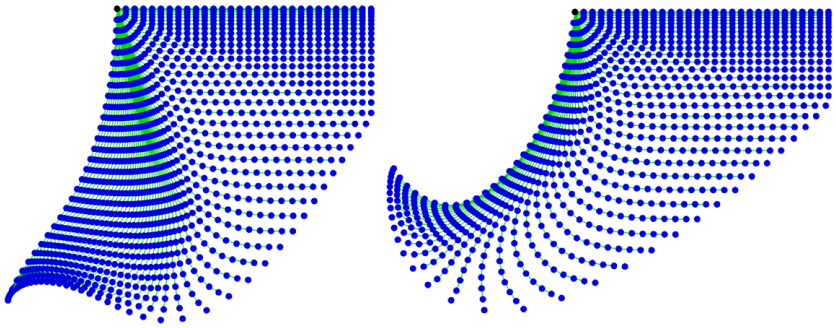


Figure 3.2: Trajectory of a strand using the DFTL algorithm with damping values of 0.9 and 1.0 from Müller et al. (2012).

Algorithm 2: Dynamic Follow The Leader(Δt)

```
1  /* The first particle does not need to be update */
2  for i ← 1 to totalControlPoints − 1 do
3  |  /* FTL */
4  |  posDiff ← particle[i].position - particle[i − 1].position
5  |  newPosition ← normalize(posDiff) · distance
6  |  particle[i].position ← newPosition + particle[i − 1].position
7  |  /* Getting the correction vector of the next particle */
8  |  posDiff ← particle[i + 1].position - particle[i].position
9  |  newPosition ← normalize(posDiff) · distance
10 |  d_{i+1} ← newPosition - posDiff
11 |  /* Setting velocity */
12 |  vel ← (particle[i].oldPosition - particle[i].position)/Δt
13 |  vel ← vel - s_damping/Δt
14 |  particle[i].velocity ← vel
15 end for
```

3.1.2 Curly Hair

It is possible to render curly hair over the structure of a serial chain of point masses. No modifications are needed on the DFTL algorithm. As Müller et al. (2012) defines, each particle is associated with a normal and a bi-normal and the curl points are calculated within the plane formed by these two vectors.

In order to compute the first particle normal, initially we set it as the normal of the mesh vertex. The bi-normal is defined as the cross product of the segment direction and the normal. And the cross product of the bi-normal and the segment direction updates the normal direction. The three resulting vectors form an orthonormal basis.

Since any non zero vector can be the normal of a point, the normal of the previous particle is propagated to the next one. To smooth the curvature when the straight strand bends, Müller et al. (2012) estimated the normal of particle i by averaging the perpendicular vectors of the segments between particles $(i-1,\ i)$ and $(i,\ i+1)$, as represented in Figure 3.3. At last, the bi-normal is the cross product of the strand direction with the normal.

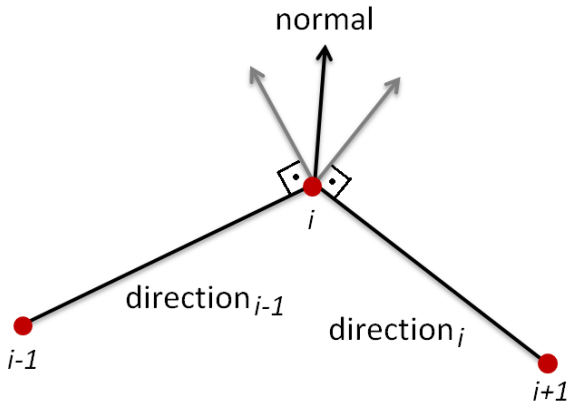


Figure 3.3: Smoothing the normal of particle i by averaging the perpendicular vectors of the adjacent segments.

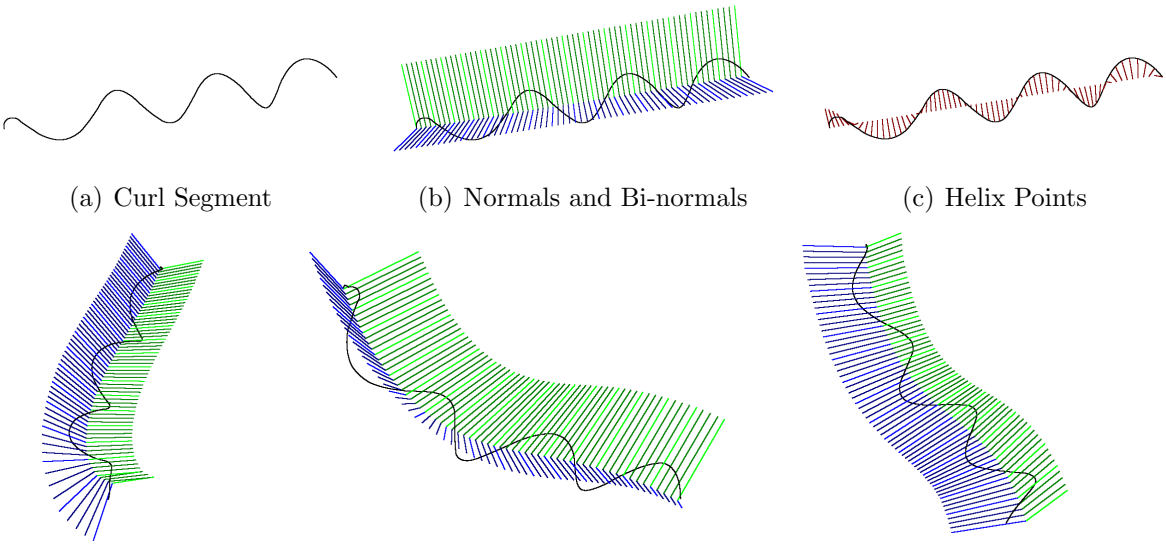
To get more details, these vectors are linearly interpolated along the segment. The spiral points are computed on every subdivision, and the frequency increases from the top to tip.

In Algorithm 3, the normal of the first particle starts with the same value as the normal of the mesh vertex. To ensure the three vectors are orthonormal, the normal is redefined as the cross product of the bi-normal and the strand direction. During the simulation, the dot product of the first strand segment and the initial normal can be less than zero. This can cause an abrupt change in the normal and bi-normal orientations. To avoid that, changing the sign of the y coordinate of the initial normal is sufficient. The last particle receives the same normal and bi-normal from the previous particle.

Algorithm 3: Draw Curl Strand

```
1  angle ← 0
2  for i ← 0 to totalControlPoints − 2 do
3  |  strandDirection_1 ← particle[i + 1].position - particle[i].position
4  |  strandDirection_2 ← particle[i + 2].position - particle[i + 1].position
5  |  /* Projecting the normal on the strand segment */
6  |  proj ← strandDirection_2 · (normal_i · strandDirection_2)
7  |  /* Computing the normal for the next particle */
8  |  normal_{i+1} ← normal_i - proj
9  |  binormal_{i+1} ← normal_{i+1} × strandDirection_2
10 |  normal_{i+1} ← (normal_{i+1} + normal_i)/2
11 |  binormal_{i+1} ← (binormal_{i+1} + binormal_i)/2
12 |  /* Computing helix point on segment (i,i + 1) */
13 |  x⃗ ← {1, 0, 0}
14 |  y⃗ ← {0, 1, 0}
15 |  r ← {A · cos(angle/B), A · sin(angle/B), 0}
16 |  x⃗ ← normal_i · (x⃗ · r⃗)
17 |  y⃗ ← binormal_i · (y⃗ · r⃗)
18 |  helixPoint ← (normalize(x⃗ + y⃗) · A) + particle[i].position
19 |  angle ← angle + angleStep
20 |  /* Increasing frequency */
21 |  angleStep ← angleStep + d
22 end for
```

Considering Algorithm 3, A is the helix radius and B rules the width of a full helix turn. The helix point on the plane produced by the normal and bi-normal is represented by a vector r in line 14. To get the correct position, this vector is compared with the x and y axis. The resulting coefficients of the dot product are utilized to scale the normal and bi-normal. The generated point is translated to the origin of the frame, i.e. the particle position. The same can be applied to the subdivision. Figure 3.4 shows the result with lines connecting the helix points.



(a) Curl Segment (b) Normals and Bi-normals (c) Helix Points

(d) Interpolation of the normals and the bi-normals during movement.

Figure 3.4: The structure to draw curly hair with 4 subdivisions, 17 control points and distance of 0.3. In Figure 3.4(d) the normals are green lines and the bi-normals are blue lines. The interpolated normals and bi-normals are represented in dark green and dark blue lines.

3.2 Hair Interaction

Hair also presents a collective behavior. The strands tend to follow the movement of those nearby, and suffer from damping because of the numerous inter collisions. The technique used on the work of Müller et al. (2012) was inspired in the work of Petrovic et al. (2005), which is a Voxel Grid.

3.2.1 Voxel Grid

The Voxel Grid is an uniform space subdivision that can help the search for neighbors objects, in this case, the control points of the strands. Each voxel vertex can store the density and the velocity, for example, and form a field created by the particles to influence them later.

Müller et al. (2012) generates the density field as in Petrovic et al. (2005). Each particle has an influence field represented by a three-dimensional tent function with value one at the particle position, that decreases linearly to zero, when the distance is higher than the size of the voxel side. The density on the voxel vertex, as defined on Petrovic et

al. (2005), is given by

$$D_{xyz} = \sum_i (1 - |P_x^i - x|) \cdot (1 - |P_y^i - y|) \cdot (1 - |P_z^i - z|), \tag{3.2}$$

where D_{xyz} is the density on the vertex at (x, y, z), (P_x^i, P_y^i, P_z^i) represents the world coordinates of the i^{th} particle, and the sum is over all the particles inside the 8 adjacent voxels. During the tests, the Equation 3.2 will probably return negative values if the size of the grid is greater than one. So, $|P_x^i - x|$ and the other factors are divided by the size of the voxel. Figure 3.5 shows the 2D case of the sum, and the particle influence on the grid vertices.

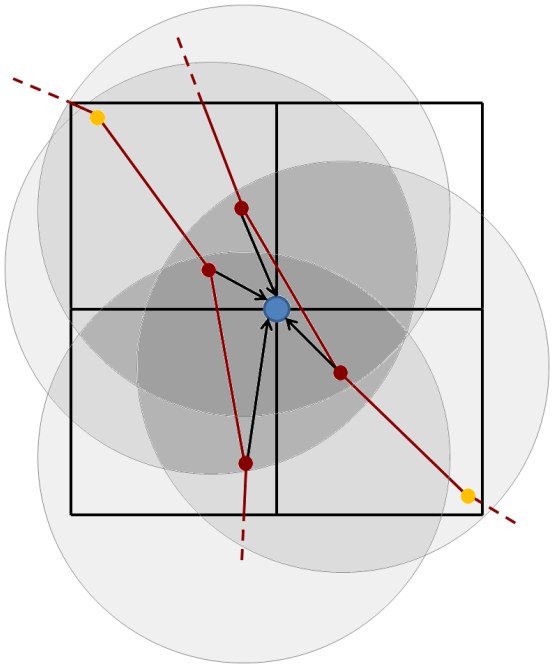


Figure 3.5: Density at a voxel vertex exemplified in 2D case. The four gray spheres are the influence of the four dark red particles. The dark red lines are the strand segments. The two yellow particles are too far to influence the blue vertex. The arrows represents that there is an influence on the blue vertex.

A velocity field is also necessary to smooth the velocities of the nearest particles. Consequently, the hair will behave as a collective material. Petrovic et al. (2005) calculated the velocity at the voxel vertices by

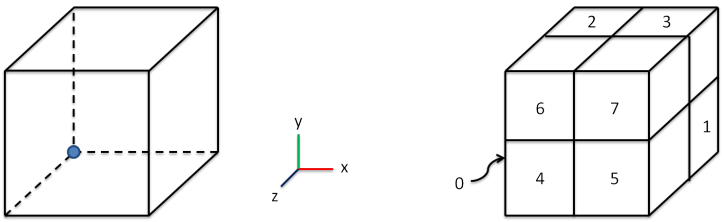
$$\vec{V}_{xyz} = \frac{\sum_i (1 - |P_x^i - x|) \cdot (1 - |P_y^i - y|) \cdot (1 - |P_z^i - z|) \cdot \vec{v}_i}{D_{xyz}}, \tag{3.3}$$

where D_{xyz}, (x, y, z), (P_x^i, P_y^i, P_z^i) are the same as defined in Equation 3.2, $vecv^i$ is the particle velocity, the sum is over all the particles inside the 8 adjacent voxels, and $\vec{V}_{xyz}$ is the velocity at the voxel vertex.

3.2.2 Octree

To save memory, an Octree was combined with the Voxel Grid. Some advantages and disadvantages of this approach are described in the book of Ericson (2004) dedicated to collision detection in real time. An uniform space subdivision needs an appropriate cell size when the objects inside the grid have different sizes. If the grid is too fine, the grid update can be slower specially when dealing with moving objects. If the grid is too coarse, the collision test can degenerate to the worst case (all objects tested with all the others), and some details can be lost when generating fields. In our work, the cell size affects not only the update of the velocities and densities in the grid, but the hair repulsion, discussed on Subsection 3.2.4.

The Octree is constructed with a top-down approach and the data are the control points of the hair strand. The voxel (or the octree leaf) is identified by the position of the vertex at the right, back and down point of the cube, as illustrated on Figure 3.6(a). The octants of the other nodes are numbered from zero to seven as shown on Figure 3.6(b).



(a) Voxel position

(b) Octants of a octree node.

Figure 3.6: Voxel representation.

The search for the bounding voxel utilizes the particle position as parameter. If the x, y, z coordinates of the point are above the voxel center, the octant is summed with $1, 2$ and 4 respectively.

To update the octree dynamically, all voxels are visited and the particles in it are tested. If the search for the voxel by the particle position is different from the voxel it

belongs, the particle is transferred to the other voxel. In this process, some voxels may be left behind empty.

In some simulations, the data exchange between voxels is not sufficient to produce a stable system in the end of the update. This is because of the voxel search, guided by the particle and the voxel positions to decide which octant the particle is contained. Numerical error occurs specially when one of the particle's coordinates is close to zero. The correct voxel is found but the test to ensure if the particle is inside the voxel fails in this case. Another way to update the grid is inserting the hair data and deleting it at each simulation step. This simplifies the program since the algorithm to join empty voxels will no longer be necessary. Also deleting and reconstructing the grid is a little faster than checking if the particles moved to another voxel, and this approach was used on the simulations.

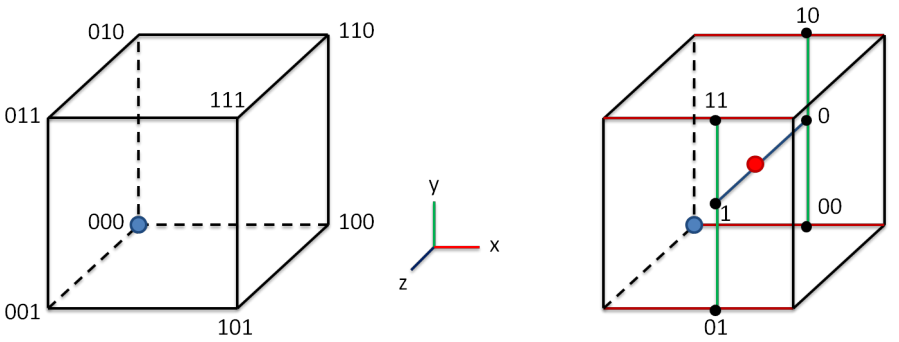
3.2.3 Trilinear Interpolation

After the grid is filled through Equations 3.2 and 3.3, the velocity of the particles are smoothed by a filter kernel and a drag term on Petrovic et al. (2005), and by a weighted average between the current velocity of the particle and the interpolated velocity from the grid on Müller et al. (2012). The Equation 3.4 is the velocity smoothing from the second work, where $s_{friction}$ is a value in the closed interval $[0, 1]$ to control the friction between the strands.

$$\vec{v} \leftarrow (1 - s_{friction}) \cdot \vec{v} + s_{friction} \cdot \vec{v}_{grid} \tag{3.4}$$

One of the fastest ways to approximate the values in a three-dimensional and continuum medium is through the trilinear interpolation. This method can be applied in scalar and vector fields. As in the work of Müller et al. (2012), to simulate hair friction and repulsion, we need to calculate the interpolated velocity ($\vec{v}_{grid}$) and the interpolated density, respectively.

To calculate the trilinear interpolation, it is necessary to have a rectangular prism (a cube in the Voxel Grid case) with values associated with the 8 vertices. They can be labeled as shown on Figure 3.7(a), and the interpolated points, on Figure 3.7(b).



(a) Labels of the vertices and coordinate system orientation

(b) Labels o the interpolated points.

Figure 3.7: Labels for Trilinear Interpolation.

A linear interpolation is done on the three axis. Following the axis order x, y, z and a simple and easy way to implement, the algorithm will need to compute 7 linear interpolations. In Algorithm 4, the variables C_{xyz} are the values on the voxel vertices. The linear interpolation, represented by the function "Linear(v_0, v_1, t)", is calculated by $(1-t) \cdot v_0 + t \cdot v_1$.

Algorithm 4: Trilinear Interpolation(point)

```
1  /* Translating to the voxel position */
2  tpoint ← point - voxel.position
3  C_00 = Linear(C_000, C_100, tpoint.x)
4  C_01 = Linear(C_001, C_101, tpoint.x)
5  C_11 = Linear(C_011, C_111, tpoint.x)
6  C_10 = Linear(C_010, C_110, tpoint.x)
7  C_0 = Linear(C_00 , C_10 , tpoint.y)
8  C_1 = Linear(C_01 , C_11 , tpoint.y)
9  C = Linear(C_0 , C_1 , tpoint.z)
10 return C
```

3.2.4 Hair Repulsion

Müller et al. (2012) simulated the hair repulsion by correcting the particle velocity by

$$\vec{v} \leftarrow \vec{v} + s_{repulsion} \cdot \vec{g}/\Delta t, \tag{3.5}$$

where $\vec{g}$ is the normalized gradient of the density field and $\vec{v}$ is the particle velocity.

The gradient is a vector that points to the maximum of a function, and can be

approximated by the Finite Difference Method (FDM). The Central FDM gives a more stable calculus:

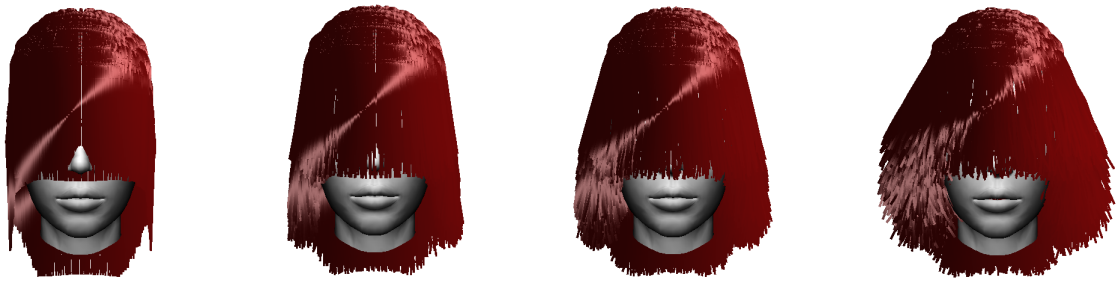
$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2 \cdot h}, \tag{3.6}$$

where h is a value close to zero, and x is the independent variable of the function.

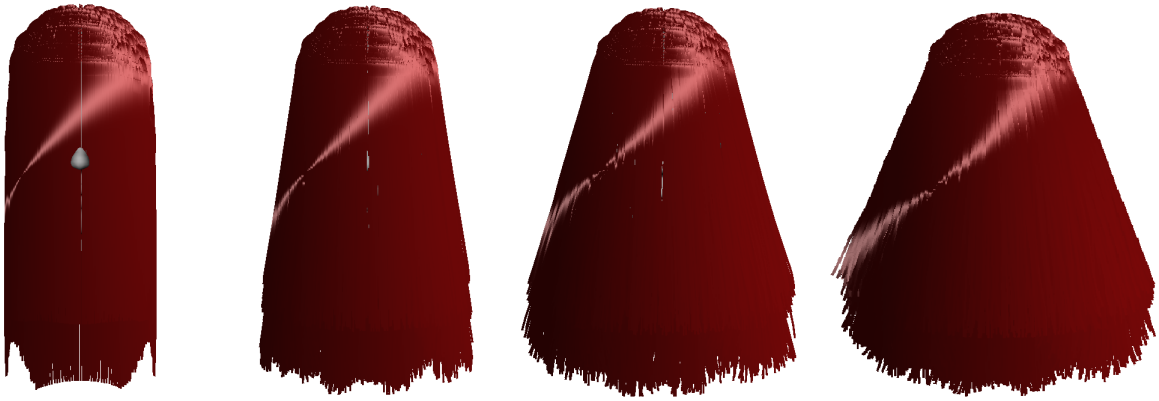
The components of gradient of the density field are calculated by

$$\begin{aligned} g_x^i =& (D(x+r, y, z) - D(x-r, y, z))/(2 \cdot r) \\ g_y^i =& (D(x, y+r, z) - D(x, y-r, z))/(2 \cdot r) \\ g_z^i =& (D(x, y, z+r) - D(x, y, z-r))/(2 \cdot r), \end{aligned} \tag{3.7}$$

where $D(x, y, z)$ is the density at the point (x, y, z), r is the radius of the sampling, in this case is the size of the side of the voxel. After this step, the resulting gradient is normalized. The Figures 3.8(a) and 3.8(b) shows how the repulsion works with different values for $s_{repulsion}$ acting on short and long hair.



(a) Varying $s_{repulsion}$ in short hair $(0.0, 0.04, 0.07, 0.15)$.



(b) Varying $s_{repulsion}$ in long hair $(0.0, 0.04, 0.07, 0.15)$.

Figure 3.8: Gradient approximation using Equation 3.6 for short and long hair.

The gradient direction varies during the simulation resulting in instability of the

strands, when $|s_{repulsion}|$ is high. In Figures 3.8(a) and 3.8(b), the greater the repulsion, more the strands moves away and towards each other without any other external force.

The gradient calculus is linked with the size of voxel. If size of the cell is small, the gradient is more accurate, and if it is large, the gradient is smoother, but miss details.

3.3 Hair Rendering

The rendering of hair defines the light behavior over the strands, and consequently, the shading. The model of Kajiya and Kay (1989) was one of the firsts to render furry surfaces and it is the simplest shader that can be applied to the hair. As stated on Section 2.3, Marschner et al. (2003) extended the Kajiya's model giving the hair a more accurate appearance especially due to the secondary colored highlight. In this work, to keep the simplicity and to give a better visualization of several strands, the shading model implemented is between the aforementioned works, as the one developed by Scheuermann (2004).

The illumination is implemented with the OpenGL Shading Language (GLSL). The data related to vertex (position, normal, tangent, camera position) are written on the vertex shader and passed to the fragment shader, where the color per pixel is calculated.

3.3.1 Shading

The model of Kajiya and Kay (1989) is alike the traditional Phong shading, but instead of the object's normal, the tangent of the strand is used. The diffuse color is computed by

$$Diffuse = K_d \cdot \sin(\vec{t}, \vec{l}) = K_d \cdot \sqrt{1 - (\vec{t} \cdot \vec{l})^2}, \tag{3.8}$$

where K_d is the diffuse reflection coefficient, $\vec{t}$ is the tangent vector of the strand and $\vec{l}$ is the light direction from the hair to the camera. To get the cosine value of the angle between the vectors from the dot product, both vectors must be normalized.

The specular component is defined as

$$Specular = K_s \cdot (\cos(\vec{t}, \vec{l}) \cdot \cos(\vec{t}, \vec{e}) + \sin(\vec{t}, \vec{l}) \cdot \sin(\vec{t}, \vec{e}))^p, \tag{3.9}$$

where K_s is the specular reflection coefficient, $\vec{t}$ and $\vec{l}$ are the normalized tangent and light direction respectively, $\vec{e}$ is the normalized vector from the hair to the camera position, and p is the exponent from Phong shading to control the sharpness of the highlight. The cosine can be calculated as the dot product, and the sine with the trigonometric identity shown in Equation 3.8.

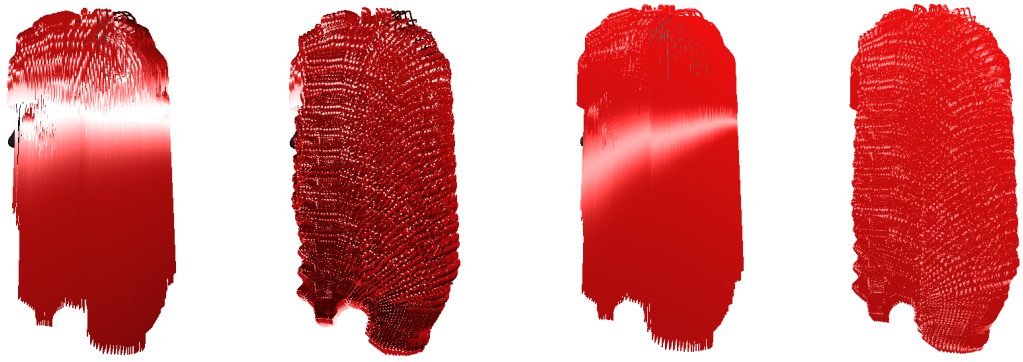
Even with these specifications, the hair gets too bright because there is no self-shadowing. Scheuermann (2004) made a model between the ones of Kajiya and Kay (1989) and Marschner et al. (2003). Even though they used a polygonal hair model and textures, their shader can be applied to the line strip implemented in this work. So, the texture info is substituted by the base color of the strand. The idea of Scheuermann (2004) is to simulate the tilted scales of real hair by shifting the tangent of the strand in the direction of the normal. Therefore the two highlights can be simulated, as shown on Algorithms 5 and 6 for the vertex and fragment shader respectively. The vertex shader passes down the tangent, normal, view vector, and light direction. The ambient occlusion term was unconsidered in this work. The fragment shader computes the diffuse lighting, the two shifted specular highlights and sums all the terms.

Algorithm 5: Scheuermann Vertex Shader ()

```
1  /* Vector defined per vertex */
2  attribute vec3 tangent;
3
4  out vData {
5      vec3 normal, eyeTangent;
6      vec4 diffuse,ambientGlobal, ambient, ecPos;
7  }vertex;
8
9  void main(){
10     /* Vectors */
11     vertex.normal = normalize(gl_NormalMatrix * gl_Normal);
12     vertex.eyeTangent = normalize(gl_NormalMatrix * tangent);
13     vertex.ecPos = gl_ModelViewMatrix * gl_Vertex;
14     /* Colors */
15     vertex.ambient = gl_FrontLightProduct[0].ambient;
16     vertex.diffuse = gl_FrontLightProduct[0].diffuse;
17     vertex.ambientGlobal = gl_LightModel.ambient *
   gl_FrontMaterial.ambient;
18     gl_Position = ftransform();
19 }
```

Algorithm 6: Scheuermann Fragment Shader ()

```
1  vec4 color = vertex.ambient + vertex.ambientGlobal;
2  vec3 lightDir = normalize(vec3(gl_LightSource[0].position-vertex.ecPos));
3  vec3 normal = normalize(vertex.normal);
4  vec3 tangent = normalize(vertex.eyeTangent);
5  vec3 camera = vec3(gl_ModelViewMatrixInverse * vertex.ecPos);
6  camera = normalize(camera - vertex.ecPos.xyz);
7  vec3 halfVec = normalize(camera + lightDir);
8
9  /* Diffuse Light */
10 vec3 diffuse = vec3(gl_FrontMaterial.diffuse * max(0.0, 0.75*dot(normal,
   lightDir)+0.25));
11
12 /* Shifting Tangents */
13 float shift = noise1(gl_FrontLightProduct[0].diffuse);
14 vec3 primTang = tangent + (primShift + shift)* normal;
15 vec3 secTang = tangent + (secShift + shift)* normal;
16
17 /* Specular Light */
18 float specMask = noise1(gl_FrontLightProduct[0].specular);
19 float TdotH = dot(primTang, halfVec);
20 float sinTH = sqrt(1.0 - TdotH * TdotH);
21 float strandSpec = smoothstep(-1.0, 0.0, TdotH) * pow(sinTH, 128.0);
22
23 vec3 specular = strandSpec * vec3(gl_FrontLightProduct[0].specular);
24
25 TdotH = dot(secTang, halfVec);
26 sinTH = sqrt(1.0 - TdotH * TdotH);
27 strandSpec = smoothstep(-1.0, 0.0, TdotH) * pow(sinTH, 90.0)
28
29 specular += starndSpec * vec3(gl_FrontLightProduct[0].specular *
   gl_FrontLightProduct[0].diffuse);
30
31 /* Final color */
32 color += diffuse * gl_LightSource[0].diffuse.xyz;
33 color += specular * gl_LightSource[0].specular.xyz;
```



(a) Kajiya and Kay shading on straight and curly hair

(b) Scheuermann shading on straight and curly hair

Figure 3.9: Comparison of the shading model of Kajiya and Kay (1989) and Scheuermann (2004). There are 9 control points and distance of 0.4. The mesh scale was reduced to generate these images.

In Figure 3.9, the curls are more visible when using the Kajiya's shader because it gives more contrast, even though it is exaggerated in this implementation. The Scheuermann's shader produces a smooth transition between colors, and there are no dark areas in curly hair.

3.3.2 Strand smoothing

To cover the sharp edges of the line segments, the control points of the strand can be smoothed by a spline. There are many types of interpolation as the Bézier curves, B-splines, Cosine interpolation, and the Catmull-Rom spline. The two first compute the curve inside the control points. The last two, the curve passes through the control points, which is more suitable to see the real position of the strands. According to Muzic (2012), the cosine interpolation presents some discontinuities and deteriorate to linear interpolation when applied in each axis separately in higher dimensions. But the Catmull-Rom spline gives a reasonable approximation as a cubic interpolation. The Algorithm 7 shows how to compute the spline points between two strand segments. The tangent is also calculated here to be sent to the shaders.

Algorithm 7: Draw Catmull-Rom Spline(y_0, y_1, y_2, y_3)

```
1  num ← 25
2  dt ← 1.0/num
3  t ← 0.0
4  last ← y0
5  for i ← 0 to num do
6      a0 ← −0.5· y0 + 1.5· y1 − 1.5· y2 + 0.5· y3
7      a1 ← y0 − 2.5· y1 + 2· y2 − 0.5· y3
8      a2 ← −0.5· y0 + 0.5· y2
9      a3 ← y1
10     res ← ( a0 · t^3+a1 · t^2+a2 · t+a3 )
11     /*Sending vertex to OpenGL*/
12     glVertex3f(res)
13     tangent ← res - last
14     /*Sending tangent to the shaders*/
15     glVertexAttrib3f(tangent)
16     last ← res
17     t ← t + dt
18 end for
```

3.3.3 Angular constraint

The angular constraint makes the hairs near the scalp appear more rigid, that is, they are harder to bend. This constraint is computed in the DFTL style, segment by segment, and it is weaker as it gets near the tip. To not distort the DFTL simulation, this constraint is applied after the dynamics.

The strand segments can bend by an angle defined by the user. The cosine of this angle is saved to be compared with the dot product of the current segment and the previous segment. If the angle is greater than 90°, there are no more angle limitation.

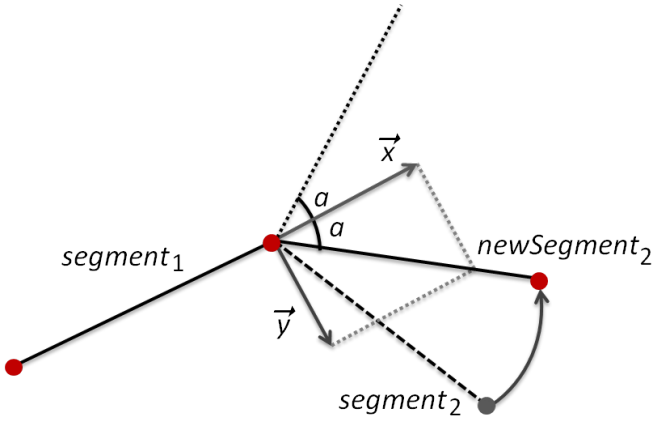


Figure 3.10: Angular constraint.

When a control point breaks this rule, the particle is pushed to the surface of an imaginary cone with angle aperture defined by the bending limit. To do that, an unit vector on the cone surface is computed on the plane covered by the strand direction of the previous and current segment.

In Figure 3.10, the black doted line and $newSegment_2$ forms the cone, and the black dashed line is the segment that needs to be corrected. The second segment is projected over the direction of the previous segment, in order to find the symmetrical axis of the cone, represented by the gray vector $\vec{x}$. The vector $\vec{y}$ is in the same plane of $segment_2$ and $\vec{x}$. The size of $\vec{x}$ and $\vec{y}$ is $\cos(a)$ and $\sin a$ respectively. Their sum gives the correct direction of the second segment, represented by $newSegment_2$. To maintain the distance restriction, $newSegment_2$ is normalized and scaled by the predefined distance.

4 Results

The methods present in this work were written in C++ and in GLSL (OpenGL Shading Language), for the rendering, and supported by the libraries OpenGL, GLU and GLUT for the graphics and window management, GLEW to compile the vertex and fragment shaders, and Assimp for mesh data manipulation. The tests were made on Windows 7, processor Intel Core i3, and integrated video card Intel HD Graphics 3000. The distance between the control points tested was between $[0.2, 1.0]$, the mass of the particles is 0.5 and the gravity is an unit vector in the $-y$ direction. The 3D model is an "obj" file from the TressFX11 2.0 code sample [1]. The scale was reduced on Blender 3D 2.68a[2] to improve the performance on CPU.

During the tests, some instabilities near the tip were found despite the size of the grid or the distance between the control points. When the head makes an abrupt movement, the end points of the strand increase its velocity and moves quickly in random directions. This effect occurred with more frequency when a single strand is simulated, and rarely with hundreds of strands, due to the size of the voxels and the velocity field generated in the grid. Without the voxel grid the same behavior occurs in about 10 of 4000 strands. This means that the size of the model and of the strands plus the s_{damping} in Equation 3.1 are also connected to this behavior. There are no errors while inserting the data on the voxel grid, even though the particles with high velocities do not really belong to the associated voxel. The error disappeared when the anomalous tips passed near several strands with the smaller velocities.

Except these considerations, the simulation ran around 3.8 frames per second (FPS) with 500 strands, 10 control points, distance of 1.0, and 25 samplings per segment to smooth the strand with the Catmull-Rom spline, and line width of 2.0. To fill the scalp, it is needed about 3800 strands.

[1] http://developer.amd.com/tools-and-sdks/graphics-development/amd-radeon-sdk/#downloadsamples

[2] http://www.blender.org/

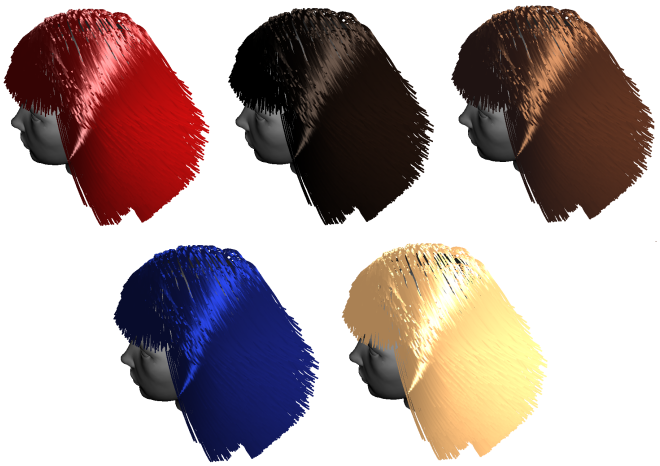


Figure 4.1: Results for red, black, brown, blue and blond short hair. There are 10 control points and distance of 1.0.

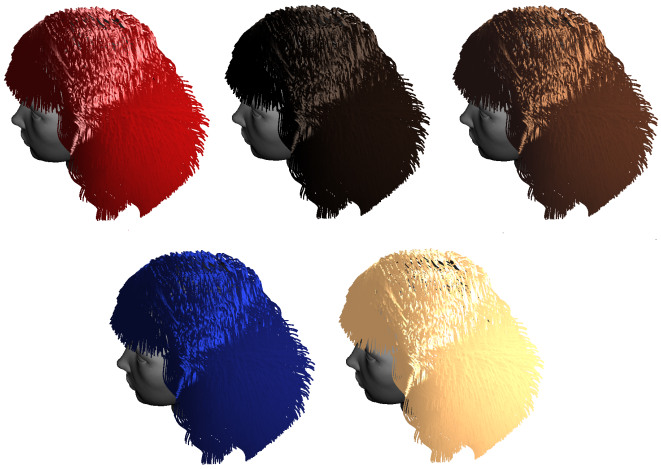


Figure 4.2: Results for red, black, brown, blue and blond curly hair.

Figure 4.3: Results for red, black, brown, blue and blond long hair.

5 Conclusion

Hair simulation is a research field that make use of techniques from other areas to solve the modeling, simulation and rendering issues. The works mentioned on Chapter 2 were developed through the years by first implementing based on a simple observation, then improved with physics fundamentals, and finally simplifying the physically correct simulation to gain performance. In the modeling process, this can be represented by the generalized cylinders of Yang et al. (2000), the textured polygonal strips of Kim and Neumann (2000), and the ribbons of Hu et al. (2014). In the dynamics process, the maximum of strands are simulated as in Rosenblum et al. (1991); Anjyo et al. (1992), fluid simulation techniques were adapted to hair (Hadap et al., 2001; McAdams et al., 2009) and Kirchhoff equations for curly hair (Bertails et al., 2006), and Müller et al. (2012); Han et al. (2013) optimized the computation based on the hair geometry. In the rendering process, the model of Kajiya and Kay (1989) adapted the Phong shading for cylinders, Marschner et al. (2003) added optics fundamentals and studied the details of the hair structure, and Scheuermann (2004) modified the Kajiya and Kay (1989) model to recreate the second highlight described in Marschner et al. (2003).

To create a hair simulator, four problems must be solved: placing the strands on the head, defining the dynamics of a single strand and their interaction, rendering the strands. In this work, the modeling simply created the strands on the scalp mesh vertex, the dynamics of a strand is defined by the DFTL algorithm and the movement of the hair as a whole is supported by the voxel grid, which also helped in varying the hairstyle with the strands repulsion. The rendering smoothed the line segments with the Catmull-Rom spline, managed the helix points to draw curly hair, and described the light interaction, so the strands are visible when there are too many of them close to each other.

The approach of Müller et al. (2012) studied in this work dismissed the smoothing of the strands when using the support of the graphics card. Without it, it is harder to manipulate a few hundreds of hairs with the same parameter of distance ($0.02m$). After implementing the Catmull-Rom spline, it is perceptible that the dynamics stage is slower

than only rendering the points of the smoothed line.

For future works, the collision with objects can be implemented by approximating the mesh through spheres and/or cylinders. The voxel grid plays an important role in the approach studied in this work. The top-down building alongside with an octree may not be the best choice. The memory cost is high in the top-down building, so fine grids can not be executed. Thus, the voxel grid can be implemented differently, as the Sparse Voxel Octree of Laine and Karras (2010), and Ph.D. thesis of Crassin (2011), the Giga Voxel. The curly hair can be represented as a textured strip since the strands of a curl will hardly separate from each other, but the centerline would still be a straight strand under the DFTL rule. The shading model of Marschner et al. (2003) could be implemented. External forces as wind can be added to the voxel grid.

Bibliography

Anjyo, K.-i.; Usami, Y. ; Kurihara, T. A simple method for extracting the natural beauty of hair. **SIGGRAPH Comput. Graph.**, 1992.

Baraff, D.; Witkin, A. Large steps in cloth simulation. **Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques**, p. 43–54, 1998.

Barringer, R.; Gribel, C. J. ; Akenine-Möller, T. High-quality curve rendering using line sampled visibility. **ACM Trans. Graph.**, v.31, n.6, p. 162:1–162:10, Nov. 2012.

Bertails, F.; Audoly, B.; Cani, M.-P.; Querleux, B.; Leroy, F. ; Lévêque, J.-L. Super-helices for predicting the dynamics of natural hair. **ACM Trans. Graph.**, 2006.

Brown, J.; Latombe, J.-C. ; Montgomery, K. Real-time knot-tying simulation. **Vis. Comput. 20**, v.20, p. 165–179, Mai 2004.

Choe, B.; Ko, H.-S. A statistical wisp model and pseudophysical approaches for interactive hairstyle generation. **IEEE Transactions on Visualization and Computer Graphics**, 2005.

Crassin, C. **GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes**. July 2011. Tese de Doutorado - UNIVERSITE DE GRENOBLE. English and web-optimized version.

Ericson, C. **Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

Grabli, S.; Sillion, F. X.; Marschner, S. R. ; Lengyel, J. E. A practical model for hair mutual interactions. **Proceedings of Graphics Interface (GI)**, 2002.

Hadap, S.; Magnenat-Thalmann, N. Interactive hair styler based on fluid flow. **Computer Animation and Simulation**, 2000.

Hadap, S.; Magnenat-Thalmann, N. Modeling dynamic hair as a continuum. **Comp. Graph. Forum (Eurographics Proc.)**, 2001.

Han, D.; Harada, T. Real-time hair simulation with efficient hair style preservation. **Eurographics Association - VRIPHYS**, 2012.

Han, D.; Harada, T. Tridiagonal matrix formulation for inextensible hair strand simulation. **Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2013)**, p. 11–16, 2013.

Herrera, T. L.; Arno Zinke, A. ; Weber, A. Lighting hair from the inside: A thermal approach to hair reconstruction. **ACM Trans. Graph.**, 2012.

Hu, L.; Ma, C.; Luo, L. ; Li, H. Robust hair capture using simulated examples. **ACM Transactions on Graphics, Proceedings of the 41st ACM SIGGRAPH Conference and Exhibition 2014**, 2014.

Iben, H.; Meyer, M.; Petrovic, L.; Soares, O.; Anderson, J. ; Witkin, A. **Artistic simulation of curly hair**. Technical report, Pixar Animation Studios, 2012.

Jakob, W.; Moon, J. T. ; Marschner, S. Capturing hair assemblies fiber by fiber. **ACM Trans. Graph**, 2009.

Kajiya, J.; Kay, T. Rendering fur with three dimensional textures. **Computer Graphics Proceedings of ACM SIGGRAPH 89**, 1989.

Kim, T.-Y.; Neumann, U. A thin shell volume for modeling human hair. **Computer Animation 2000**, 2000.

Kim, T.-Y.; Neumann, U. Interactive multiresolution hair modeling and editing. **Proceedings of ACM SIGGRAPH, Transactions on Graphics, vol. 21, no. 3**, 2002.

Kong, W.; Takahashi, H. ; Nakajima, M. Generation of 3d hair model from multiple pictures. **Proceedings of Multimedia Modeling**, 1997.

Laine, S.; Karras, T. **Efficient sparse voxel octrees**. In: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '10, p. 55–63, New York, NY, USA, 2010. ACM.

Liu, T.; Bargteil, A. W.; O'Brien, J. F. ; Kavan, L. Fast simulation of mass-spring systems. **ACM Transactions on Graphics**, v.32, n.6, p. 209:1–7, nov 2013. Proceedings of ACM SIGGRAPH Asia 2013, Hong Kong.

Luo, L.; Li, H. ; Rusinkiewicz1, S. Structure-aware hair capture. **ACM Transactions on Graphics, Proceedings of the 40th ACM SIGGRAPH Conference and Exhibition 2013**, 2013.

Marschner, S. R.; Jensen, H. W.; Cammarano, M.; Worley, S. ; Hanrahan, P. Light scattering from human hair fibers. **ACM Trans. Graph.**, v.22, n.3, p. 780–791, Jul 2003.

McAdams, A.; Selle, A.; Ward, K.; Sifakis, E. ; Teran, J. Detail preserving continuum simulation of straight hair. **Proc. of ACM SIGGRAPH 2009 conference**, 2009.

Müller, M.; Heidelberger, B.; Hennix, M. ; Ratcliff, J. Position based dynamics. **Proceedings of Virtual Reality Interactions and Physical Simulations (VRIPhys)**, 2006.

Müller, M.; Kim, T. ; Chentanez, N. Fast simulation of inextensible hair and fur. **Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS**, 2012.

Muzic, M. **Real-time hair simulation and rendering with opencl and opengl**. 2012. Dissertação de Mestrado - University of Dublin, Trinity College.

Paris, S.; no, H. M. B. ; Sillion, F. X. Capture of hair geometry from multiple images. **ACM SIGGRAPH 2004 Papers**, 2004.

Petrovic, L.; Heme, M. ; Anderson, J. **Volumetric methods for simulation and rendering of hair**. Technical report, Pixar Animation Studios, 2005.

Plante, E.; Cani, M.-P. ; Poulin, P. A layered wisps model for simulating interactions inside long hair. **Proceedings of Eurographics Computer Animation and Simulation**, 2001.

Rosenblum, R.; Carlson, W. ; Tripp, E. Simulating the structure and dynamics of human hair: Modeling, rendering and animation. **Journal of Visualization and Computer Animation**, 1991.

Sadeghi, I.; Pritchett, H.; Jensen, H. W. ; Tamstorf, R. An artist friendly hair shading system. **ACM Trans. Graph.**, v.29, n.4, p. 56:1–56:10, Jul 2010.

Scheuermann, T. **Practical real-time hair rendering and shading**. In: ACM SIGGRAPH 2004 Sketches, SIGGRAPH '04, p. 147–, New York, NY, USA, 2004. ACM.

Selle, A.; Lentine, M. ; Fedkiw, R. A mass spring model for hair simulation. **ACM SIGGRAPH 2008 Papers**, 2008.

Ward, K.; Lin, M.; Bertails, F.; Kim, T.; Marschner, S. ; Cani, M.-P. A survey on hair modeling: Styling, simulation, and rendering. **IEEE Transactions on Visualization and Computer Graphics**, 2007.

Watanabe, Y.; Suenaga, Y. A trigonal prism-based method for hair image generation. **IEEE Computer Graphics and Applications**, 1992.

Wei, Y.; Ofek, E.; Quan, L. ; Shum, H.-Y. Modeling hair from multiple views. **ACM SIGGRAPH 2005 Papers**, 2005.

Yang, X. D.; Xu, Z.; Yang, J. ; Wang, T. The cluster hair model. **Graphical Models, Volume 62, Issue 2**, 2000.

Yuksel, C.; Schaefer, S. ; Keyser, J. Hair meshes. **ACM SIGGRAPH Asia 2009 papers**, 2009.

Zinke, A.; Weber, A. Light scattering from filaments. **IEEE Transactions on Visualization and Computer Graphics**, v.13, n.2, 2007.

Zinke, A.; Yuksel, C.; Weber, A. ; Keyser, J. Dual scattering approximation for fast multiple scattering in hair. **ACM Trans. Graph.**, v.27, n.3, p. 32:1–32:10, Ago. 2008.