UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Real-Time Hair Simulation and Rendering

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Martin Rünz

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Diana Röttger (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Mai 2012

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Bachelor Thesis Martin Rünz
Thesis Research Plan
(Mat. Nr.: 208 210 309)

Topic: Real-time Hair Simulation and Rendering

Due to the amount of hair strands a human head possesses, it's a challenging task to simulate realistic looking hair. With the abilities of modern graphic cards it became possible to simulate and render convincing hair in real-time.

Within the scope of this bachelor thesis a hair simulation should be implemented. This simulation should present realistic looking hair, which is reacting to wind, gravity and motions of the head. Furthermore the simulation could react to collisions with objects or provide an artist friendly shading system.

Key aspects of the work are:

1. Studying DirectX
2. Researching and analyzing available techniques to simulate and render hair
3. Creating a concept
4. Building a prototype based on the concept
5. Demonstration and Evaluation of the results
6. Documentation

Koblenz, Oct. 14, 2011

- Martin Rünz- – Prof. Dr. Stefan Müller –

Abstract

The following thesis presents real-time techniques for virtual hair simulation, generation and rendering. It refers to a prototype which has been implemented within the scope of this bachelor thesis. After discussing properties of human hair in section 2, section 3 outlines simulation methods and explicitly explains mass-spring systems. All simulation methods are based on particles, which are used to generate geometry and subsequent to render hair strands. This generation process is explained in chapter 4 *geometry generation*. Besides the *Kajiya and Kay's Hair Model*, the *Marschner Shading Model*, and a third, artist friendly shading system, section 5 will describe shadow and self-shadowing techniques, such as *deep opacity maps*. While the subjects of the first sections are platform-independent methods and properties, section 6 presents *DirectX 11* oriented implementation details. Finally, the prototype is used to analyze the quality as well as the efficiency of covered techniques.

Zusammenfassung

Die folgende Arbeit behandelt Techniken zur virtuellen Echtzeitdarstellung von Haaren. Dabei wird eine Unterteilung zwischen Simulations- und Rendertechniken vorgenommen. Die Kapitel beziehen sich auf einen Prototyp, der im Rahmen dieser Bachelor-Arbeit entstanden ist. Nachdem in Kapitel 2 die Eigenschaften des menschlichen Haares behandelt wurden, greift Kapitel 3 verschiedene Simulationsverfahren und ausdrücklich Masse-Feder Systeme auf. Alle erläuterten Simulationsverfahren basieren auf Partikeln, welche zur Geometrieerzeugung und damit zum Rendern genutzt werden. Während dieser Erzeugungsvorgang in Kapitel 4 *Geometry Generation* beschrieben wird, werden Rendertechniken in Kapitel 5 besprochen. Neben den Darstellungsmodellen *Kajiya and Kay's Hair Model*, *The Marschner Shading Model* und einem artistenorientiertem Model wird die Realisierung von Selbst-Verschattung und Schattenschlag behandelt. Während die ersten Kapitel plattformunabhängige Methoden und Eigenschaften vorstellen, geht Kapitel 6 auf *DirectX 11* orientierte Implementierungsdetails ein. Des Weiteren werden die Qualität der Techniken und der involvierte Rechen- aufwand anhand des Prototyps analysiert.

Contents

1 Introduction

Virtual characters appear in various contexts, such as in animation films or computer games and are expected to look realistic or in the way a director specified. This involves a convincing hair simulation, which is a challenging task. Cosmetic companies are interested in hair simulations as well, requiring detailed systems for prototyping. Since the human head has around 100.000 hairs, each one reacting to forces such as wind or friction, physical representations are only approximating the behaviour of hair and are computationally expensive. Further, hair fibers exhibit complex scattering properties which have to be considered to present natural results. These tasks have concerned researchers for more than two decades and can be addressed continually better with increasing processing power. Some simulation techniques are dedicated to offline renderers and are primarily used in motion pictures. The animation film *Final Fantasy: The Spirits Within* [12], released in 2001, set new standards by animating 60.000 separate hairs. *Tangled*, an animation film released in 2010, even focuses on hair animation. The shader used in this film will be discussed in section 5.3. In 2003 Nvidia presented the *Nalu Demo* which proved that believable hair can also be simulated in real-time. Most computer games use very basic methods to present hair or simply avoid hair by distributing head coverings, such as helmets.
This thesis presents different approaches to the simulation and rendering aspects of human hair and explains which ones are adaptable for real-time applications. Long smooth hair is emphasized, but other hair styles are considered as well. After comparing techniques a prototype gets developed, outlining implementation details. It is based on DirectX 11 and uses the capabilities of modern graphic cards to be as efficient as possible.

2 Properties of human hair

Before discussing ways to simulate human hair styles, it is necessary to understand the properties of human hair. A detailed description of physical and chemical hair behaviour is given by C. R. Robbins [54].
Figure 1 illustrates the body of a human hair shaft. It consists of two crucial parts: The cortex and the cuticle. The cortex consists of spindle-shaped keratin filaments and contributes about 90% of the total hair weight, see [26]. It often encloses one or more regions, which are called medulla (lat. *medulla* = „marrow"), located near the centre of the strand. The boundary of a hair strand is the cuticle. When light rays hit a fiber they interfere with this layer, which forms scales that have a significant effect on light scattering. Because of the small cross-section of hair fibers, which is between $50 - 120\mu m$ in diameter, depending on the age and the ethnic background of a person, they are subject to bending. On the other hand, hair strongly resists to shearing and stretching forces. This is due to the amount of keratin, which is a stiff material. The elastic modulus (Young's modulus) of hair fibers averages $3.89GPa$ and is comparable to the elastic modulus of wood [14, 54]. Robbins classifies hair properties by three ethnic groups: Caucasians, Mongolians and Ethiopians. While people with typical European hair would be part of the first group, Asians would belong to the Mongolian category and people with typical African hair would be part of the latter group, namely Ethiopians. Characteristic values for these groups are listed in table 1.

Ethnic group	Diameter[1]	Eccentricity	Curvature	Color
Ethiopians	$90.62\mu m$	0.82	Wavy to wooly	Brown-black to black
Caucasians	$63.93\mu m$	0.67	Straight to curly	Blond to dark brown
Mongolians	$79.53\mu m$	0.60	Straight to wavy	Dark brown to brown-black

Table 1: Average hair characteristics classified by ethnic groups [54, 58].

Ethiopian hair is coarse and primarily black. The highly eccentric cross-section causes this type of hair to be wavy to wooly. On average Caucasian hair strands are less elliptical and can be straight to curly. Mongolians posses hair that's cross-section is most similar to a circle. Hence, Mongolians have straight to wavy hair. The color of hair is determined by the composition of melanin, which is enclosed in cortex cells. Red and blond strands exhibit a higher concentration of pheomelanin. Eumelanin is brown to black and is more abundant in hair of dark skinned people.

[1] Measured along the long principal axis of the elliptical cross-section.

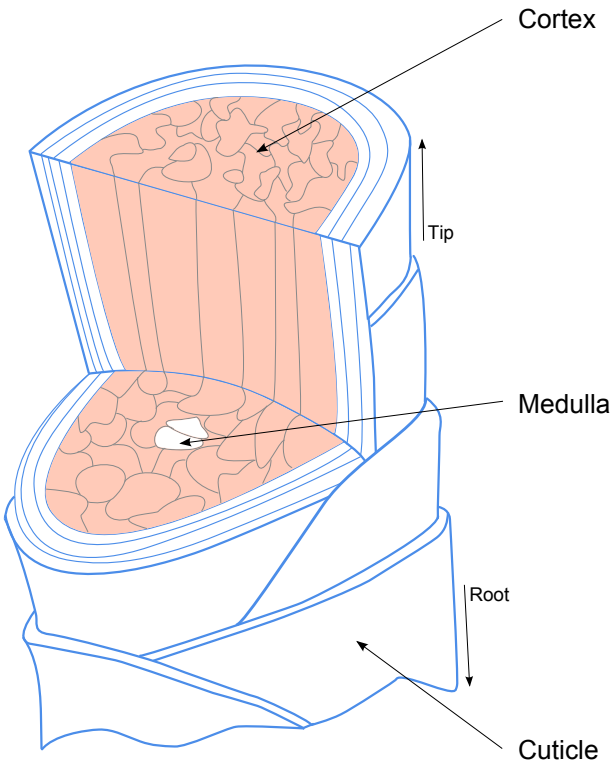


Figure 1: Schematic illustration of a single hair.

With increasing age the amount of melanin decreases until the hair is grey or white. Around 200 hair strands are located on the scalp of a Caucasian person per square centimetre [45]. The index of refraction of human hair is approximately 1.55 [40]. A detailed image of a hair strand taken with a scanning electron microscope can be found in appendix A.1.

3 Simulation

Simulating a single hair is a problem of *elastic rod* theory [35,59]. Most elastic rod models are well suited to present the physical behaviour of hair in a realistic way, but are only capable of running in real-time for individual threads. It is a common approach to discretize each strand by a set of nodes. These nodes (also called particles) are subject to constraints and their kinematics describe the motion of a thread or a hair strand. For rendering purposes nodes get connected by geometry, as described in section 6.2.

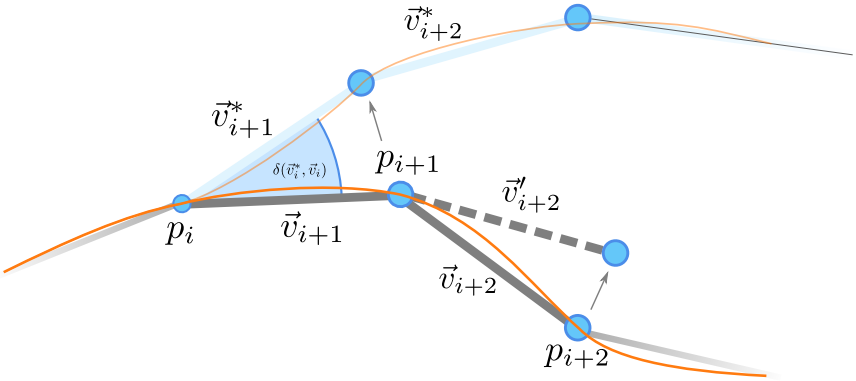


Figure 2: Hair strands are represented by particles. This illustration presents the initial as well as the current state of a strand, accompanied by notations.

When expressing strands as particles, the simulation process and the process of geometry generation occur autonomously. Simulated particles form guide hairs, which can be used to create more than a single hair strand. Section 4.2 presents interpolation techniques for guide hairs. It is also possible to create flat meshes with hair textures on the basis of particles [36], which is efficient, but produces unnatural appearing hair. The succeeding section will discuss various simulation methods, related to particles.
Throughout this thesis following notations will be used (see figure 2):

- p_0 .. $p_n \in \mathbb{R}^3$: Position of a strand particle, where p_0 is located at the hair root and $n \in \mathbb{N}$ is the number of particles per strand minus one.
- n_p is the number of particles and n_s is the number of segments per strand.
- $\vec{v}^*_1$ .. $\vec{v}^*_{n_s} \in \mathbb{R}^3$: Initial position of a strand particle, relative to its prior particle. At the beginning of the simulation following condition is met: $\vec{v}^*_i = p_i - p_{i-1} \quad \forall i \in \{1..n_s\}$.
- s_i: Segment between particle p_i and p_{i-1}, where $i \in \{1..n_s\}$.

3.1 Methods

In the last two decades several methods for hair simulation were acquired. But due to complex hair characteristics, only a few are applicable for real-time environments. In 1992 Anjyo et al. introduced *One Dimensional Projective Equations* for hair dynamics [3,31]. Similar to other methods, dynamics are mapped on nodes that receive kinetic energy. Each node p_i spans its own spherical coordinate system containing p_{i+1}. Given a force F, that is acting on s_{i+1}, the translation of p_{i+1} is computed by projecting the force vector onto two perpendicular planes. This yields two forces F_θ, F_ϕ which are acting on the azimuth and inclination angle of the polar coordinates. The angular accelerations can then be solved by two ordinary differential equations:

$$\begin{aligned} \frac{d^2\theta_i}{dt^2} &= c_i u_i F_\theta \\ \frac{d^2\phi_i}{dt^2} &= c_i v_i F_\phi \end{aligned} \tag{1}$$

where c_i is a constant related to the inertia moment of S_i and u_i, v_i are length specifications, with the result that $u_i F_\theta, u_i F_\phi$ combine into a moment of force respectively.

By iterating over all particles, while solving the angular acceleration, the hair motion is described. One dimensional projective equations are simple, stable and efficient, but are not able to simulate torsion. Furthermore, the original algorithm ignores hair-hair interaction and isn't able to handle collisions in a sufficient way [63]. Improved versions such as those by Lee and Ko [1] and Jung et al. [67] approach these problems, but are inaccurate, especially when accounting for hair-hair interaction of long strands.

Another method for hair simulations are *Free-Form Deformations* [47]. 2004 Volino et al. [62] showed that their lattice model can be used to approximate the behaviour of hair in real-time. To implement this method a grid is built around the head, which is subject to distortion. This deformation is reproduced to the hair geometry. Because it can be non-linear it is more capable than affine transformations, but since distortion treats hair in a continuous way, it is unable to predict the behaviour of single wisps.

In 2001 Hadap et al. introduced a *continuum based approach* to model hair dynamics [28]. Continuum based simulations assume that the object of interest is not constructed by discrete elements or particles but rather by a steadily spread medium. They are commonly used to model the behaviour of fluids. Because in essence, each medium is arranged by atoms, there is an imprecision with continuum based systems, but at distant observation they are very accurate. Strands retain their individual dynamics however, which does not fit into a continuum based approach, but partly react continuously due to hair-hair interaction. Hence Hadap et al. simulate single

hairs using *rigid multi-body serial chains* and account for inter hair characteristics by utilizing continuum dynamics. They represent the density of the hair continuum by the number of hair strands per volume unit. Although the dimension of atoms is not comparable to the dimension of hair strands with respect to the scaling of the simulation, Hadap et al. produced realistic results with this approach. Their method uses *Langrangian* formulations of fluid dynamics. *Eulerian* formulations are possible as well [48] and McAdams et al. [42] presented a *hybrid* approach. Langrangian and Eulerian representations differ in their viewpoint. While in a Langrangian representation properties are expressed relative to moving particles, they are expressed relative to grid points in Eulerian representations. Rigid multi-body serial chains descend from forward kinematics, which have extensively been investigated in the context of robot dynamics [7, 18]. Because mass-spring systems are more efficient and often used in real-time simulations, they are explicitly discussed in the the following section, whereas rigid multi-body serial chains are neglected in this thesis.

3.2 Mass-Spring System

Mass-Spring systems are often used to animate deformable objects. They are well studied, because they have been deployed in cloth simulations for a long time. Advantages of mass-spring systems are, that they are easy, efficient and produce realistic results. To apply such a system to the hair simulation, each particle functions as a mass-point and each segment as a spring. The mass-points are subject to forces and accordingly to Newton's second law of motion receive acceleration:

$$\vec{F} = m\vec{a} \Longleftrightarrow \vec{a} = \frac{\vec{F}}{m} \tag{2}$$

When time progresses, this results in a change of velocity and consequentially in a change of position. A force that is acting continuously on all points is the gravitational force. Other external forces derive from wind or friction. Springs allow particles to react on forces, but also introduce constraints to maintain the hair's shape. The spring forces are depicted in figure 3.
The arrows in the first row of this figure illustrate in which directions forces of coil springs are acting. If the distance between two particles p_1, p_2 is lower than the original distance d, the spring force $\vec{f_1}$ is directed from p_2 to p_1 and $\vec{f_2}$ in the opposite direction. Would the distance be higher than the rest distance, the forces would point in reversed direction. In both cases the forces cause an acceleration, that attempts to restore the original distance between p_1 and p_2. These forces are described by Hooke's law:

$$
\begin{aligned}
F &= -kx \\
\vec{f_1} &= -0.5kx\frac{p_1 - p_2}{\|p_1 - p_2\|} = -\vec{f_2} \\
&= -0.5k(\|p_1 - p_2\| - d)\frac{p_1 - p_2}{\|p_1 - p_2\|} \qquad \forall p_1, p_2 \in \mathbb{R}^3 \wedge p_1 \neq p_2
\end{aligned} \tag{3}
$$

In this law k is called the spring constant and defines the stiffness of a spring. The higher this constant, the higher the reacting force. x is the displacement of the particles relative to their rest distance. Since the spring force F is distributed on both particles, each gains half the force in reversed direction. Equation 3 states how Hook's law can be applied to particles. If p_1 equals p_2 the direction of forces $\vec{f_1}$ and $\vec{f_2}$ are undefined, a case one has to prevent in a mass-spring system.
Materials that are simulated by mass-spring systems can experience arbitrary high deformation and thus are called *super-elastic*. As mentioned previously, hair is a stiff material especially resistant to shearing and stretching forces. At room temperature the *extension at break* of hair amounts to 48% [54]. Hair that's length is further extended will break. This however, would require very high stress and hairs would be ripped out before breakage. Even strong wind would not cause any noticeable hair extension and hence common hair simulations try to suppress any change of hair length. Such behaviour can be approached by choosing high values for k. Would the value be to low, strands could not maintain their length, when forces are acting. On the other hand a high spring constant involves small time steps or else the governing equations may not converge and therefore become unstable. Furthermore, stiff springs cause a stronger oscillation, which is unnatural for most materials. Finally, a loss of angular momentum [32] can be caused when using larger k-values in combination with implicit integration. Provot addressed this problem in 1995 [53] and introduced a post-

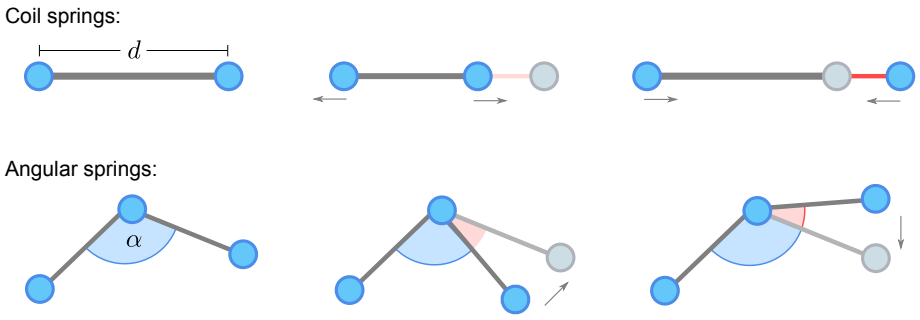


Figure 3: Coil spring and angular springs

processing mechanism for spring based dynamics. He defines a maximal extension factor for coil springs τ. Springs that exceed the specified extension are contracted to meet the constraint. This also allows to use small k-values, larger time steps and hence fewer iterations. It would also be possible to adjust the velocities of particles, instead of the positions. Such an approach is given by Bridson et al. [6]. The simulation however described in section 6 uses Provot's method.

Coil springs are practical to simulate deformable objects and to meet *distance constraint*s. On the downside, they are inappropriate to restore an initial state of a hair strand. Since hair styles tend to recover their original state, angular springs are utilized to introduce a second, *angular constraint*. Whenever mass-points move in space, angular springs can be used to force them back to their original position, relative to the prior particle. One could imagine a gust of wind that moves the hair of a person with a short hairstyle. It is likely that after the gust the hair strands of the person fall back in place. Such situations can be handled with angular springs. The structure of angular springs is depicted in the second row of figure 3.

Hook's law can be applied to angular springs in the same manner as to coil springs. Assuming, that segment s_j is of unit length, the (angular) displacement x of particle p_{j+1} to its rest position is equal to the angle between segment s_j being in rest position and segment s_j, measured in radians. Since $x = \delta \cdot |s_j|$, where δ is the referred angle, adjustments to k can account for variations in segment length. A difficulty that appears, when connecting segments by springs is, that each particle needs a *frame of reference* to identify the angular rest position of a subsequent particle. The simulation belonging to this thesis stores the initial position vector $\vec{v}_i^*$ of particles relative to prior particles. Whenever a segment s_j is rotated, the rotation is also applied to initial position vectors of subsequent particles, yielding $\vec{v}_i' \quad \forall i \geq j$. $\vec{v}_i$ is defined as $\vec{v}_i = p_{i+1} - p_i$. Let $\delta(\vec{a}, \vec{b})$ be the angle between two vectors $\vec{a}$ and $\vec{b}$ of same length and $Q(\vec{a}, \vec{b}, \omega)$ be a quaternion that results in a rotation of ω around $\vec{a} \times \vec{b}$, clockwise, then $\vec{v}_i'$ can be formulated as[2]:

$$\vec{v}_i' = \begin{cases} v_1^* & \text{if } i = 1 \\ \prod_{k=1}^{i-1} \Big(Q\big(\vec{v}_k', \vec{v}_k, \delta(\vec{v}_k', \vec{v}_k)\big) \Big) \cdot \vec{v}_i^* & \text{otherwise} \end{cases} \tag{4}$$

When $\vec{v}_i'$ is constituted, x can simply be calculated by $x = \delta(\vec{v}_i', \vec{v}_i) \cdot |s_i|$, while the force is oriented by $\vec{d}_f = \vec{v}_i' - \vec{v}_i'^p$. Here $\vec{v}_i'^p$ is the projection of $\vec{v}_i'$ onto the vector $\vec{v}_i$. The force can be directed more efficiently by choosing $\vec{d}_f = p_{i-1} + \vec{v}_i' - p_i$, with the side benefit of supporting distance constraints, even though this is an approximation. Equation 4 exposes, that several rotations have to be accumulated to gain $\vec{v}_i'$. Hence, the use of quaternions

[2]It is presumed, that neither $\vec{a}$, nor $\vec{b}$ are of length zero. If $\vec{a}$ equals $\vec{b}$, no rotation is applied.

helps to reduce the total amount of calculations. In [13] Melax describes a stable procedure to create quaternions based on two vectors, which has been used for this simulation.
Some hair simulations [61] approximate angular constrains with *flexion*-springs. Flexion springs are identical to coil springs, but connect distanced mass-points, rather than neighboring points. By this topology they react to bending, since bending forces change the relative positions of specific particles. Flexion springs are very common in cloth simulation systems, but have major drawbacks when applied to hair systems. Firstly, multiple different rest configurations can occur, that cause hair strands to adopt malformed shapes. Furthermore, they introduce more than a solely angular constraint and thus cause a loss of angular momentum.

3.3 Numerical Integration

The previous section described that forces are acting on particles and that these forces result in velocity and hence in a change of position. A naive approach to applying forces would only regard the current position p, the current velocity $\dot{p}$, the timestep Δt and the force f:

$$\begin{aligned} \ddot{p}(t) &= \frac{f(p(t))}{m} = \frac{d^2p}{dt^2} \\ \dot{p}(t+\Delta t) &= \dot{p}(t) + \ddot{p}(t)\Delta t \\ p(t+\Delta t) &= p(t) + \dot{p}(t)\Delta t \end{aligned} \tag{5}$$

where $\ddot{p}$ is acceleration and t an instant of time. These equations could be solved after each simulation time step, starting from an initial configuration. Solving $p(t+\Delta t)$ obviously includes solving *ordinary differential equation*s (ODEs), which take the form:

$$\begin{aligned} p^{(n)}(t) &= g(t, p, p', ..., p^{(n-1)}) \quad (explicit) \\ 0 &= g(t, p, p', ..., p^{(n)}) \quad (implicit) \end{aligned} \tag{6}$$

The formulations above (equation 5) are equivalent to the *explicit Euler method* and can be derived as follows:

$$\begin{aligned} \dot{p}(t) &= g(t, p) \\ \Rightarrow \frac{p(t+\Delta t) - p(t)}{\Delta t} &= g(t, p) \\ \Rightarrow p(t+\Delta t) &= p(t) + \dot{p}(t)\Delta t \end{aligned} \tag{7}$$

This is the most basic method for numerical integration of ordinary differential equations. It assumes that $\dot{p}(t)$ is constant during a timestep Δt. The error, that arises because of this assumption, can be measured by comparing the approximation p with the infinite Taylor series expansion of the

exact solution p_e to the ODE [24]. Subtracting p from p_e yields:

$$\begin{aligned} p_e &= \sum_{n=0}^{\infty} \frac{p^{(n)}(t)}{n!}\Delta t^n = p(t) + \dot{p}(t)\Delta t + \frac{1}{2}\ddot{p}(t)\Delta t^2 + ... \\ p_e - p &= \frac{1}{2}\ddot{p}(t)\Delta t^2 + \frac{1}{6}\dddot{p}(t)\Delta t^3 + ... \end{aligned} \tag{8}$$

The above polynomial shows, that the lowest order error in O-notation is $O(\Delta t^2)$. Hence the explicit Euler method offers a first order approximation. Preferable numerical methods have three properties: They converge, which means that smaller time steps produce results closer to the real solution, they are of high order to reduce the error and they are stable. An improvement of the explicit Euler method is the *implicit* or *backward Euler*. The implicit Euler method computes $p(t + \Delta t)$ by equating:

$$p(t + \Delta t) = p(t) + g(t + \Delta t, p(t + \Delta t))\Delta t \tag{9}$$

which has to be regrouped and solved as an algebraic equation. Baraff and Witkin showed [4] that this allows large time-steps in mass-spring systems. A more popular method to solve ordinary differential equations in interactive applications is the *Verlet method*. Advantages of this method are efficiency and accuracy. It is derived by adding two third order Taylor series:

$$\begin{aligned} p(t + \Delta t) &= p(t) + \dot{p}(t)\Delta t + \frac{1}{2}\ddot{p}(t)\Delta t^2 + \frac{1}{6}\dddot{p}(t)\Delta t^3 \\ p(t - \Delta t) &= p(t) - \dot{p}(t)\Delta t + \frac{1}{2}\ddot{p}(t)\Delta t^2 - \frac{1}{6}\dddot{p}(t)\Delta t^3 \\ \Rightarrow p(t + \Delta t) &= 2p(t) + \ddot{p}(t)\Delta t^2 - p(t - \Delta t) \end{aligned} \tag{10}$$

while the velocity, if required, can be expressed by a first order approximation:

$$\dot{p}(t + \Delta t) = \frac{p(t + \Delta t) - p(t)}{\Delta t} \tag{11}$$

Since $p(t + \Delta t)$ depends on $p(t - \Delta t)$ as well as $p(t)$, the last position has to be stored beside the current position. More precise velocity values can be gained by employing the related *Velocity Verlet method*. Its velocity is based on a half time-steps, resulting in these equations:

$$\begin{aligned} p(t + \Delta t) &= p(t) + \dot{p}(t)\Delta t + \frac{1}{2}\ddot{p}(t)\Delta t^2 \\ \dot{p}(t + \Delta t) &= \dot{p}(t) + \frac{1}{2}(\ddot{p}(t) + \ddot{p}(t + \Delta t))\Delta t \end{aligned} \tag{12}$$

where the acceleration has to be computed at the time of t and $t+\Delta t$. This is possible because forces are only position-dependent. In order to treat both

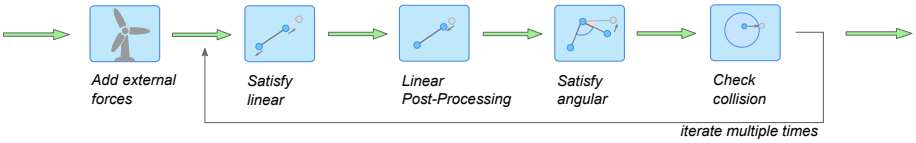
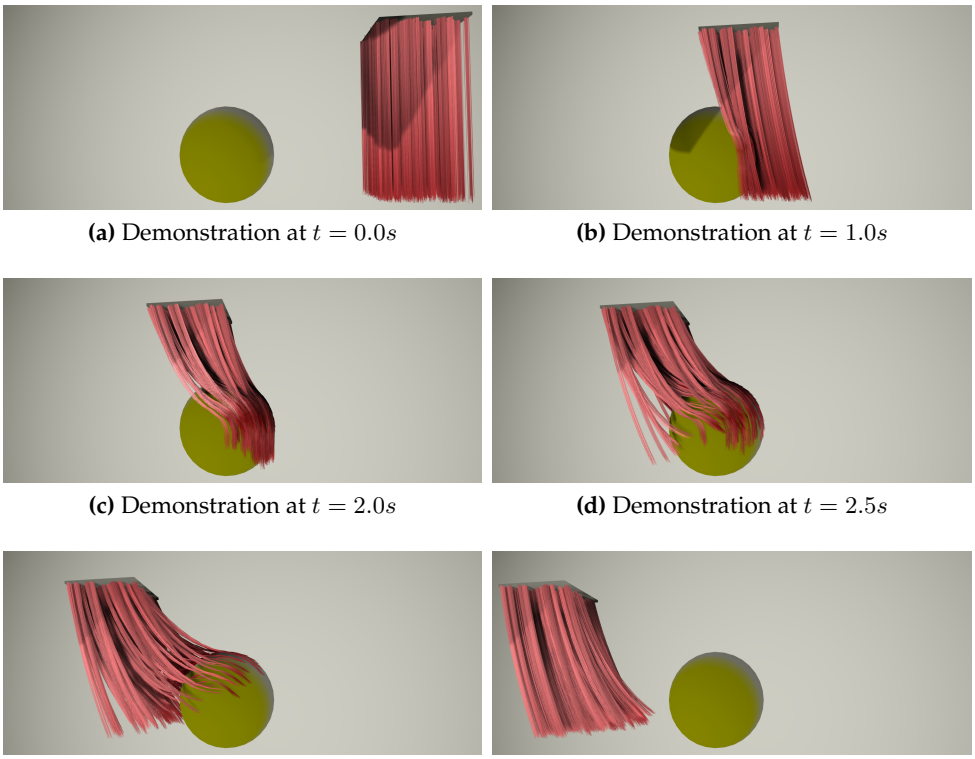


Figure 4: Constraints are satisfied multiple times per simulation step.

methods in a similar way, one could attach a buffer to each particle which either stores the old position $p(t - \Delta t)$ or $\ddot{p}(t)$ according to the method. The process of integration is illustrated in figure 4. To produce smooth results, constraints are satisfied multiple times per simulation step. Whenever constraint forces are applied to particles, the numerical integration is executed to change properties accordingly. Common external forces emerge from gravitation or wind. Gravitation can be introduced by accelerating all particles along the negative y-axis. If particles are of uniform mass, this is equal to adding a constant force to them. The simulation, further described in section 6, adds smoothed random force vectors to particles to simulate wind. These force vectors are scaled depending on their orientation towards tangent vectors, to account for windage. The appendix contains a series of pictures to demonstrate the influence of wind (Appendix A.4, figure 31).
Which of the two Verlet variants is better suited depends upon the application. The latter provides more precise velocity, but less precise position values. Other popular methods are the fourth-order Runge–Kutta method *RK4* or *leapfrog* integration.

3.4 Collision

Without convincing collision handling virtual scenes become clearly artificial. This holds true for hair simulations, where hair-hair and hair-body interaction are of interest. Because hair-hair interaction is approached by special simulation methods such as fluid dynamics, see section 3.1, hair-body interaction is the focus here. The process of handling collisions is separated in *collision detection* and *collision response*. To detect collisions between body and hair, the body mesh is usually approximated by simple geometric objects, because exact determinations are not possible in real-time environments. For the same reason hair strands too are represented by simpler objects. Since distance constraints are most efficient with spheres, the simulation described in this thesis approximates characters solely by spheres. With regards to collision detection, strands can be represented by its segments, its nodes or by generalized cylinders [8]. While general cylinders would model best on hair wisps, they are neglected here for performance reasons. Nguyen and Donnelly [49] describe a *pearl configuration* based on particles which is faster than collision tests with segments and works well.



(a) Demonstration at $t = 0.0s$

(b) Demonstration at $t = 1.0s$

(c) Demonstration at $t = 2.0s$

(d) Demonstration at $t = 2.5s$

(e) Demonstration at $t = 3.0s$

(f) Demonstration at $t = 4.0s$

Figure 5: Series of pictures to demonstrate collision detection and response, where $r_p = 0$. 180 guide hairs with 10 segments were used to create 5580 interpolated hairs. Strands are shaded with the artist friendly system described in section 5.3

Particles receive a circumference, which increases towards the hair tips, resulting in collision tests between spheres. Let k be the number of collision spheres that belong to a character, $c_j \in \mathbb{R}^3$ the related centres and r_j the related radii, where $j \in \{0..k-1\}$. Then a collision between a particle and a sphere is detected if and only if:

$$|p_i - c_j| - (r_j + r_p) < 0 \tag{13}$$

or to avoid a square root operation:

$$|p_i - c_j|^2 - (r_j + r_p)^2 < 0 \tag{14}$$

The variable r_p allows customized circumferences and has been introduced to prevent segments from penetrating the body. Higher r_p values reduce the accuracy of the collision detection, but also degrease undetected collisions. If segments are small, low r_p are sufficient and the imprecision is not

noticeable. Figure 5 demonstrates the process of collision detection and response, where no pearl configuration was necessary due to small segments. In order to resolve collisions, impulsive responses are used. They are based on *Newton's law of restitution for instantaneous collisions with no friction*, which makes three simplifying assumptions about collisions [24]: The collision has no duration, meaning that attributes are changed instantaneously. Tangential forces due to friction are neglected. A quantity known as the *coefficient of restitution* accounts for submolecular interactions and energy loss. The law states that after the collision, the linear momentum is conserved:

$$\begin{aligned} q_c + q_p &= q'_c + q'_p \\ \rightarrow q'_c &= q_c + \hat{q} \\ \rightarrow q'_p &= q_p - \hat{q} \end{aligned} \tag{15}$$

where q_c, q_p are the momenta before, q'_c, q'_p are the momenta after the collision and $\hat{q}$ is an idealized impulse. Let $\dot{p}$ be the velocity of a particle, $\dot{c}$ be the velocity of a collision sphere and $\vec{n}$ the normal to the collision plane, then $\dot{p}'$ and $\dot{c}'$ can be expressed by:

$$\begin{aligned} \dot{c}' &= \dot{c} + \frac{\hat{q}}{m_c}\vec{n} \\ \dot{p}' &= \dot{p} - \frac{\hat{q}}{m_p}\vec{n} \end{aligned} \tag{16}$$

The coefficient of restitution indicates how elastic two objects collide. While a value of 1 means that both objects collide perfectly elastically smaller values belong to inelastic collisions. It is defined as ratio between velocity difference before and after the collision:

$$\varepsilon = \frac{\vec{n} \circ (\dot{p}' - \dot{c}')}{\vec{n} \circ (\dot{c} - \dot{p})} \tag{17}$$

Combining equation 15 and equation 16 yields changed velocities by solving $\hat{q}$:

$$\hat{q} = \vec{n} \circ \frac{(\varepsilon + 1)(\dot{p} - \dot{c})}{\frac{1}{m_c} + \frac{1}{m_p}} \tag{18}$$

Because the body mass is many times higher than the mass of hair strands, m_c can be assumed to be infinite, simplifying equation 18:

$$\hat{q} = \vec{n} \circ (\varepsilon + 1)(\dot{p} - \dot{c})m_p \tag{19}$$

Finally the new velocity of a particle, colliding with a sphere is described by:

$$\dot{p}' = \dot{p} - \big(\vec{n} \circ (\varepsilon + 1)(\dot{p} - \dot{c})\big)\vec{n} \tag{20}$$

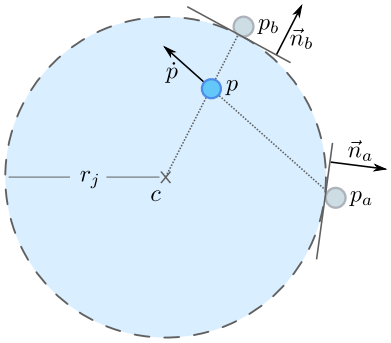


Figure 6: Collision detection between a pearl with radius r_p and a sphere with radius r_j. Here, two response methods are presented.

In addition to updating the particle's velocity, the actual collision has to be dissolved. This happens either by repositioning the particle along $p - c$ or backward to its direction of movement. Both methods result in a different collision plane, as depicted in figure 6.
The updated position p_a or p_b can be expressed by:

$$p_a = (r_j + r_p)\frac{p - c}{|p - c|} \tag{21}$$

$$p_b = s\dot{p} + p$$

Here $s\dot{p} + p$ is a parametric representation of a straight line. s can be computed by solving $|s\dot{p} + p - c| = r_j + r_p$ and using the s value lesser than zero. Another way of handling collision responses are *penalty forces*. At their essence, penalty forces are spring forces acting against the interpenetration of objects. In terms of particles, which are penetrating collision spheres, penalty forces could be introduced by inserting a stiff spring between p_b or p_a and p. The lower the spring damping would be, the more elastic the collision response. One advantage of penalty forces is, that they react well, when multiple objects are colliding at one time. On the downside they allow penetration and become inaccurate with high velocities. For that reason impulsive responses have been chosen for the simulation belonging to this thesis. If a particle is placed inside a collision sphere after a response, the newly introduced penetration is handled at the next simulation step. Such misplacement could be suppressed with additional tests, but the simple method described here works well enough and is efficient. Collision responses can also be integrated into a *constraint solver*, depending on the given framework or physics engine.

4 Geometry Generation

Hairstyles may have a complex structure and it is difficult to create them with classic modelling procedures. When hair should behave dynamically the creation process is even more demanding. In the last years several generation methods were presented. In [65] Yuskel et al. present hair meshes. Their method is intended to simplify the task for artists by introducing a work-flow that is analog to modeling polygonal surfaces. Another approach is multi-resolution editing [34], that constructs a level of detail representation for hair manipulation. By subdividing clusters one can either operate on single strands or at a higher level in the topology. Grabli et al. [23] introduced a procedure that constructs hair styles based on photographs. Other methods are physical based, including proceedings that are based on fluid flow [27].

4.1 Particle Creation

The software belonging to this thesis implements two generation procedures. The first one distributes hair strands randomly on a scalp, the second one imports a configuration created with the hair tool of Cinema4D [41]. The former method iterates through scalp faces, and places a number of wisps dependent on the face's area. The random positions are selected as described in the textbook GPU Gems 2 [49]. A more sufficient algorithm would suppress density variations, such as the dart throwing algorithm [9] or the one described in [34]. After placing the hair roots, subsequent particles are positioned by adding a scaled and interpolated normal to the root particle. This yields straight hairs and is only reasonable when using long hair with weak angular springs.

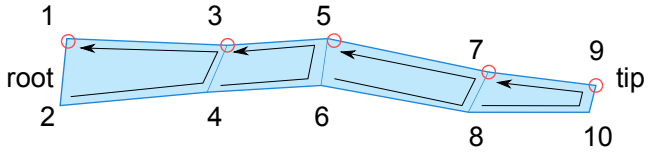
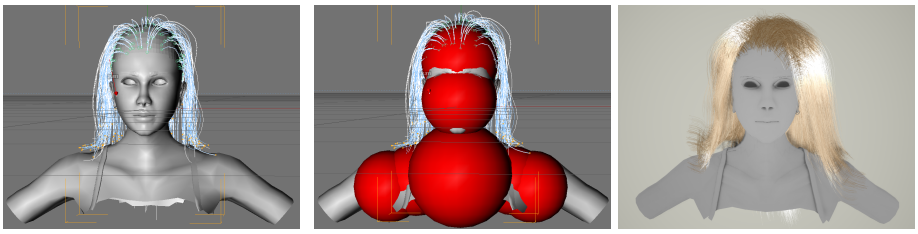


Figure 7: Hair vertices exported to a Wavefront Object File. Here faces are defined counter-clockwise.

With the second method it is possible to create detailed hair styles. The software provides tools that are based on items such as a brush and a comb, parted with facile selection modes. In order to export a hair model from Cinema4D, the software offers the ability to generate vertices along the hair strands. These can be stored in several formats, including the *Wavefront Object File* format. Hair strands that are stored to a wavefront object file have a predefined layout. The number of segments $n-1$ per strand can be spec-



(a) Create character and hairstyle **(b)** Add collision sphere **(c)** Import in real-time application

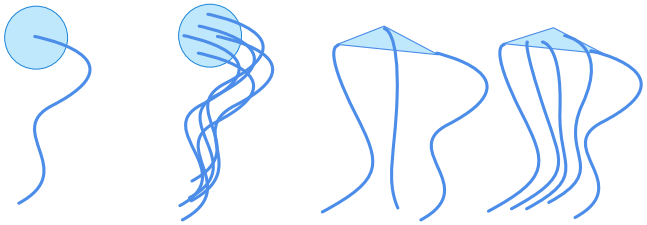
Figure 8: Character creation workflow

ified within Cinema4D. While the count of (indexed) vertices describing one strand is $2n$, the total amount of hair strands amounts $2n/n_v$, where n_v is the total number of vertices. The indices are ordered from root to tip, as illustrated in figure 7. If the number of segments per hair is known, the simulation can import object files and separate strands. Particles that are placed at each second vertex, reproduce the original hair shape, please see the red dots in figure 7. Figure 8 outlines the character creation work-flow. After the character and hair-style has been designed, collision spheres can be added to introduce collision handling. Afterwards, the scene is ready to be imported into the application.

4.2 Particle Interpolation

To save computation time, particle positions can be used to produce more than a single hair strand. Tariq [60] distinguishes between *single strand based interpolation* (or clump based interpolation) and *multi strand based interpolation*. Both methods create new particles based on existing ones. The single strand based technique duplicates a strand several times and translates the new hairs by a random offset. As a result a wisp (or a clump) gets constructed. The translation has to take place in the xy-plane of the local coordinate frame of a hair strand. If strands are generated along the normals of a scalp mesh, the plane is obviously defined by the normal. When an object file is imported however, the plane can be computed by finding the three nearest vertices to the hair root and interpolating their normals. Figure 9 shows both techniques.
Visual results of single strand based interpolation can be optimized by manipulating the duplicated particles based on the particle speed, as described by Choe et al. [8]. Choe et al. increase the wisp radius, the faster it moves, to account for hair-air interaction. This scaling factor ξ also increases towards the tip. It can be formulated as follows:

$$\xi = 1 + \sigma|\vec{v}| \tag{22}$$



(a) Single Strand Based Interpolation **(b)** Multi Strand Based Interpolation

Figure 9: caption

Here σ is a monotonically increasing function of the distance to the root. When increasing the wisp radius it is expedient to move the cross section back along the speed direction. Otherwise an increment of the cross-section would raise and a decrement would reduce the speed for a short moment, which can result in flickering. Let c_0 be the original and c_v be the new centre of the cross-section. Then c_v can be expressed as:

$$c_v = c_0 - r_0(\xi - 1)\frac{\vec{v}}{|\vec{v}|} \tag{23}$$

Where r_0 is the original wisp radius. These formulations are not based on physical fundamentals, but afford sufficient visual results for real-time environments. Interpolation based on multiple strands consults three hairs to place the particles of a new strand. Each attribute of a new strand, such as velocity and mass, are interpolated values of those three hairs. To identify three appropriate strands, hair roots can either be placed at scalp vertices which allows one to apply the triangulation of the scalp mesh. Or, the Delaunay algorithm can be used to create a triangulation between roots.
Single strand based interpolation and multi strand based interpolation produce different visual results. The former approach can be used to simulate wisps, that occur due to hair friction. The latter method is useful for soft hair styles, that appear voluminous with relatively few hairs. Which method to employ is an artistic choice, one can also combine both methods.

5 Rendering

Since hair fibers are very thin objects, they are often regarded as one dimensional lines. Classic *Bidirectional Reflectance Distribution Function*s (BSDFs) define a relationship between incoming irradiance and outgoing radiance with respect to a surface normal vector. Various shading models are based on this principle to determine the surface color of an object. Lines however have an infinite set of normals and therefore different approaches for hair rendering has been established. The next sections present three shading models for hair rendering. The early one by Kajiya and Kay, the physical based approach Marschner et al. introduced in 2003, and finally a model that is artist oriented.

5.1 Kajiya and Kay's Hair Model

In 1989 Kajiya and Kay presented a lightning model for human hair [30] formed by a diffuse and specular component. While the specular component was an adaptation of the Phong shading model for cylindrical surfaces, the diffuse component was derived of the Lambertian shading model. Instead of using a single surface normal, Kajiya and Kay use the normal plane to build a lightning model. This plane is defined by the point of interest of the hair fiber x_0 and the fiber's tangent $\vec{t}$. The lightning $\vec{l}$ and view vector $\vec{v}$, starting from x_0, are then projected onto this plane, yielding $\vec{v_p}$ and $\vec{l_p}$, where $\vec{l}$ and $\vec{v}$ are assumed to be of unit length. The vectors $\vec{l_p}$, $\vec{t}$ and $\vec{l_p} \times \vec{t}$ span an orthonormal basis. Kajiya and Kay use this basis to derive their shading equation. They use $\vec{l_p}$ as normal for diffuse and $\vec{v_p}$ as normal for specular lightning. The original derivatives can be condensed as follows:

$$\begin{aligned} L_O &= (\Psi_d + \Psi_s) \otimes B \\ L_O &= \left(c_d \cdot \sin(\vec{t}, \vec{l}) + c_s \cdot \cos^m(\vec{v}, \vec{v_p})\right) \otimes B \end{aligned} \tag{24}$$

where B is the brightness, m the shininess, Ψ_d the diffuse component, Ψ_s the specular component and c_d, c_s are the diffuse and specular reflection coefficients respectively. Here the trigonometric functions use the angle between two vectors. The symbol $\otimes$ means piecewise vector multiplication. By calculating the appropriate input angles, this equation can be interpreted as phong shading equation. This is fulfilled in the textbook Real-Time Rendering [2][3]:

[3]The book's equation may differ slightly from this one, since the first printing of the $3rd$ edition of Real-Time Rendering contained a typing error.

$$
\begin{aligned}
\overline{\cos}(\theta_i) &= \sin(\vec{t},\vec{l}) = \sqrt{1-(\vec{l}\cdot\vec{t})^2} \\
\overline{\cos}(\alpha) &= \max\Big(\sqrt{1-(\vec{l}\cdot\vec{t})^2}\sqrt{1-(\vec{v}\cdot\vec{t})^2}-(\vec{l}\cdot\vec{t})(\vec{v}\cdot\vec{t}),0\Big) \\
L_O &= \big(c_d\cdot\overline{\cos}(\theta)+c_s\cdot\overline{\cos}^m(\alpha)\big)\otimes B
\end{aligned}
\tag{25}
$$

Here $\overline{\cos}$ refers to the clamped cosine function.
The hair model of Kajiya and Kay is still used widely, but has two major drawbacks when compared to more advanced methods. First, the original model is not energy conserving, which is important for physically based rendering. Secondly, it disregards that hair is translucent and hence missing visual effects.
While the first drawback can be overcome by normalization, the second one is inveterate. The normalization of *bidirectional reflectance distribution function*s (BRDFs) can be done by dividing by the maximum of a function called *directional-hemispherical reflectance function*. The latter function measures the amount of absorption a BRDF exhibits with a predefined input direction. Is the value zero, all light is absorbed, a value of one means that all light is reflected. Values higher than one indicate that the BRDF is not energy conserving, because more light is reflected than originally shined on the surface.
Let f be a bidirectional reflectance distribution function, then the corresponding directional-hemispherical reflectance is defined as:

$$
R_\Omega(l) = \int_\Omega f(l,v)\cos\theta d\omega = \int_0^{2\pi}\int_0^{\frac{\pi}{2}} f(l,v)\cos\theta\sin\theta d\theta d\phi \tag{26}
$$

where θ is the angle between surface normal and the outgoing light direction and the Ω subscript indicates an integration over the upper hemisphere. Since the shading model is defined over the entire sphere, R_Ω has to be modified. The adjusted function R_O, here referred to as *directional-spherical reflectance*, can be formulated as:

$$
R_O(l) = \int_O g(l,v)\sin\theta_t d\omega = \int_0^{2\pi}\int_0^{\pi} g(l,v)\sin^2\theta_t d\theta_t d\phi \tag{27}
$$

This function integrates over the entire sphere, where g is the distribution function of the Kajiya and Kay shading model. The well-known integration scheme of BRDFs is oriented toward the surface normal, the scheme here is oriented toward the hair tangent, which is reflected in using $\sin\theta_t$ instead of $\cos\theta$, as illustrated in figure 10. $\sin\theta_t$ equals $\cos\theta$, because:

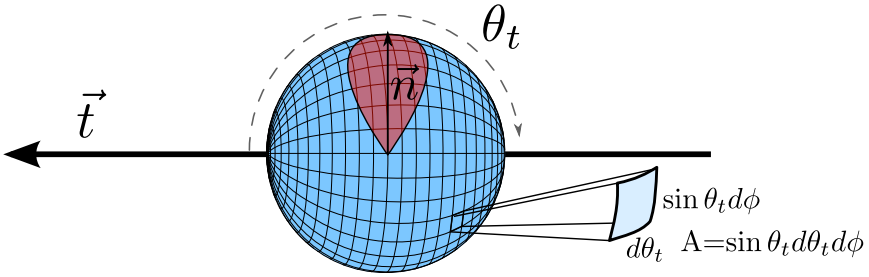


Figure 10: Illustration of the integration scheme. The red area visualizes the specular part of the phong BRDF, which is rotated around the hair axis.

$$\begin{aligned}
\cos(\frac{\pi}{2}-\alpha) &= \cos(\alpha-\frac{\pi}{2}) = \sin\alpha \\
\cos\theta &= \cos(|\frac{\pi}{2}-\theta_t|) \\
\rightarrow \cos\theta &= \begin{cases} \cos(\frac{\pi}{2}-\theta_t) & \text{if } \theta_t \leq \frac{\pi}{2} \\ \cos(\theta_t-\frac{\pi}{2}) & \text{otherwise} \end{cases} \\
\rightarrow \cos\theta &= \sin\theta_t
\end{aligned} \tag{28}$$

Equation 25 showed, that the shading model can be expressed as phong shading term, which can be transformed to the distribution function g[4]:

$$g(l,v) = \frac{c_d}{\pi} + \frac{c_s \overline{\cos}^m(\alpha)}{\pi} \tag{29}$$

To calculate the maximum of R_O it is sufficient to consider the specular term only. Then, to guarantee energy conservation, the condition $c_d + c_s \leq 1$ must be fulfilled. It is established, that $R_\Omega(l)$ for the phong BRDF reaches its maximal value, when the light vector is equal to the surface normal, which is also true for the g, since the distribution of the specular component is a rotation of the phong distribution around the hair axis. Assuming, that $\vec{l}\cdot\vec{t} = 0$ and $c_s = 1, c_d = 0$, one has:

$$\overline{\cos}(\alpha) = \sin(\vec{v},\vec{t}) = \sin\theta_t \tag{30}$$

Combining equations 27, 29, 30 yields:

$$R_O(l)_{Max} = \frac{1}{\pi}\int_0^{2\pi}\int_0^{\pi} \sin^{(m+2)}\theta_t d\theta_t d\phi \tag{31}$$

[4]The original phong BRDF states, that $f(l,v) = \frac{c_d}{\pi} + \frac{c_s\overline{\cos}^m(\alpha)}{\pi\overline{\cos}\theta}$. Modern approaches exclude the division by $\overline{\cos}\theta$, because it has no physical plausibility and causes the maximum of $R_\Omega(l)$ to be infinite. Hence, this BRDF could not be normalized. Physical plausible shaders are discussed here [37]

Exponent	$R_O(l)_{Max}$	Exponent	$R_O(l)_{Max}$	Exponent	$R_O(l)_{Max}$
(1	19.7392)	8	8.47828	15	6.28219
2	15.5031	9	8.02101	16	6.08903
3	13.1595	10	7.63045	17	6.91265
4	11.6274	11	7.29183	18	5.75075
5	10.5276	12	6.99458	19	5.60146
6	9.68946	13	6.73092	20	5.46321
7	8.02364	14	6.49497	21	5.33472

Table 2: $R_O(l)_{Max}$ function values for various exponents.

Solving this equation is not as easy as solving the equivalent term of the original phong shading model. Computational software programs, such as *Mathematica* are capable of deriving the general solution. In order to normalize Kajiya and Kay's model efficiently, one could pre-compute $R_O(l)_{Max}$ values and store relevant results in look-up tables or approximate the function. Table 2 lists function values for exponents from 0 to 21. The final normalized shading model is expressed by:

$$L_O = \frac{1}{R_O(l)_{Max}} \big(c_d \cdot \overline{\cos}(\theta) + c_s \cdot \overline{\cos}^m(\alpha)\big) \otimes B \tag{32}$$

Marschner et al. introduced a method for hair shading, which is energy conserving and produces more realistic results.

5.2 The Marschner Shading Model

In their Paper [40] Marschner et al. exploited properties of dielectric cylinders to derive a scattering function S for hair fibers. This function behaves like a *Bidirectional Scattering Distribution Function* (BSDF) but is not defined as ratio between radiance L and irradiance E, rather as *curve intensity* $\bar{L}$ per *curve irradiance* $\bar{E}$. These units are one dimensional equivalents to L and E and are chosen because a hair fiber is regarded as one dimensional object. Further, S extends over the entire sphere, not only over the upper hemisphere. The function considers three reflectance paths, that light can travel through a hair. Figure 11 illustrates these paths. The first one, called **R**-path contributes light that impinges on the surface of the hair and gets reflected immediately. The **TRT**-path represents rays which get transmitted by the hair, travel inside of the strand, are reflected at the backside and finally transmitted again. This component produces a secondary highlight. The **TT**-path covers light that is transmitted by the hair, travels inside of the hair until it gets transmitted again. This component is crucial in backlighted situations. Figure 11 also illustrates tilted cuticle scales, that cause a shift of the highlights by an angle α. The scattering function $S_p(\theta_i, \theta_o, \phi_i, \phi_o)$ is a 4D transformation, that can be factored into a product of two 2D functions

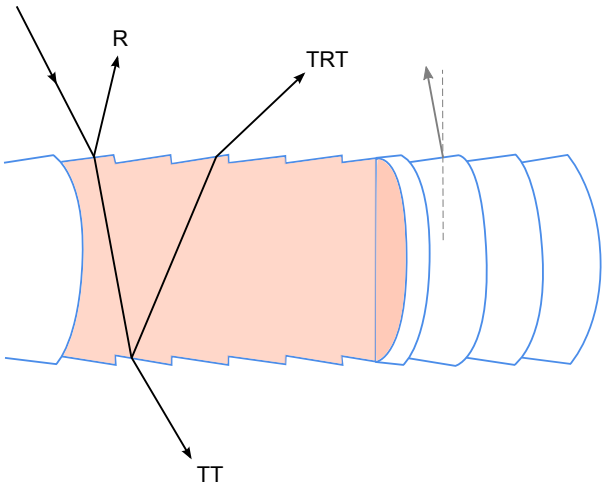


Figure 11: Lightscattering within a human hair fiber

$M_p(\theta_i, \theta_o)$ and $N_p(\theta_d, \phi_d)$. Where θ_i is the angle of inclination and ϕ_i the azimuth of the incident light relative to the normal plane, which is perpendicular to the hair axis. θ_o and ϕ_o are the angles to the outgoing direction, respectively. The difference angles θ_d, ϕ_d are defined as $\theta_d = (\theta_o - \theta_i)/2$ and $\phi_d = (\phi_o - \phi_i)$. Where $P = \{R = 0, TT = 1, TRT = 2\}$ and $p \in P$ depends on the relevant path. Hence the complete scattering function can be expressed as:

$$S(\theta_i, \theta_o, \phi_i, \phi_o) = \frac{1}{\cos^2 \theta_d} \sum_{p \in P} M_p(\theta_i, \theta_o) N_p(\theta_d, \phi_d) \tag{33}$$

The division by $\cos^2 \theta_d$ projects the solid angle of incident beams onto the hair surface. While M describes longitudinal scattering, N represents azimuthal scattering. After analysing measurement results Marschner et al. discovered, that a normalized Gaussian density function $g(x; \sigma^2)$ can be used to approximate M_p. This function should have its maximum when $\theta_i = -\theta_o$ shifted by an angle α_p due to the tilted cuticle scales, see figure 11. Let θ_h be $\theta_h = (\theta_i + \theta_o)/2$ and β_p be the longitudinal width (or standard deviation) then the various M functions are defined as:

$$M_p(\theta_h) = g(\theta_h - \alpha_p, \beta_p^2) \tag{34}$$

or:

$$M_p(\theta_h) = \frac{1}{\beta_p \sqrt{2\pi}} e^{-\frac{(\theta_h - \alpha_p)^2}{2\beta_p^2}} \tag{35}$$

Assuming a hair fiber has a circular cross section, which is illuminated either complete or not at all, the azimuthal scattering functions can be de-

rived from principles of energy conservation. The next section will present approximations of the N_p-components, figure 14 of section 5.3 gives a visual impression of the function values. Let $\bar{E}$ be the irradiating power per unit length (*curve irradiance*). Then the outward intensity per unit length (*curve intensity*) $\bar{L}$ equals $\bar{E}$ in order that:

$$\bar{L}_p(\phi_d)d\phi_p = A_p(h)\frac{1}{2}\bar{E}dh \tag{36}$$

where dh is an interval of the incident beam and $d\phi_p$ the angle, resulting when this part of the beam leaves the hair, see figure 12. In fact, equation 36 equates the energy of $\bar{L}$ with the energy E. Because $\bar{E}$ is one dimensional, it corresponds to E multiplied with the fiber's diameter. When considering a unit circle, the factor $\frac{1}{2}$ arises. The function $A_p(h)$ evaluates the impact of the different paths on the intensity, it accounts for attenuation by absorption as well as attenuation by reflection. When regarding the cross section as a unit circle, h is the distance between the ray with width dh and the centre of the circle. To calculate the amount of light, that travels towards an observer, it is necessary to solve h for each possible path. Afterwards, it is possible to compute the corresponding absorption. A bundle of rays that enter the circle by this offset h will leave it, so that the outgoing directions are spread over an angle $d\phi$, in radians. Because a circular cross section is presumed, the described conception does not depend on the actual values of ϕ_i or ϕ_o. The ratio between curve intensity and curve irradiance expresses a distribution function parametrized by ϕ_d:

$$\frac{\bar{L}_p(\phi_d)}{\bar{E}} = \frac{1}{2}A_p(h)\left|\frac{dh}{d\phi_p}\right| \tag{37}$$

Both, θ_d and ϕ_d are required to compute h and $d\phi_p$. Summating these functions for $p \in P$ yields N:

$$N(\theta_d, \phi_d) = \sum_{p\in P}\frac{1}{2}A_p(h)\left|\frac{dh}{d\phi_p}\right| \tag{38}$$

This function can be solved by computing h and the derivative $\frac{dh}{d\phi_p}$. Figure 12 illustrates that the angle between an incoming and outgoing ray $\Delta\phi_p$ is:

$$\begin{aligned} \Delta\phi_R &= -2\gamma_i \\ \Delta\phi_{TT} &= 2\gamma_t - 2\gamma_i + \pi \\ \Delta\phi_{TRT} &= 4\gamma_t - 2\gamma_i \end{aligned} \tag{39}$$

Note that while ϕ_d is the actual angle between the directions, $\Delta\phi_p$ represents the angle dependent on γ_i, γ_t. When the output of $\eta'(\theta_d)$ is the index

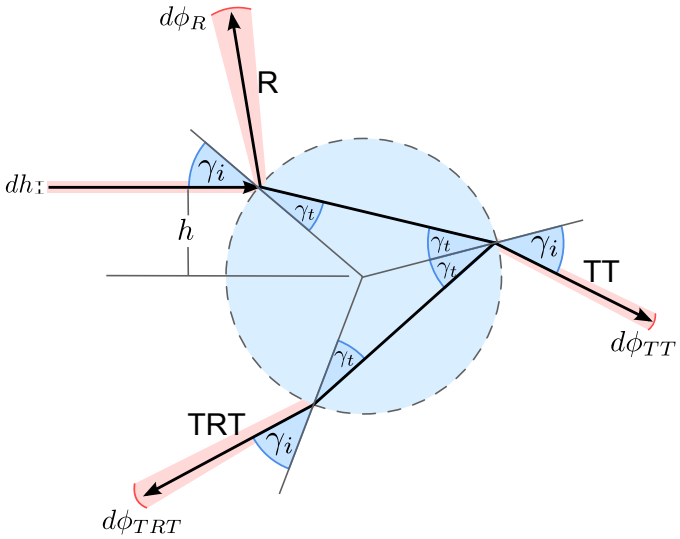


Figure 12: Scattering from a unit circle

of refraction with reference to the normal plane[5], γ_t can be formulated as $\gamma_t = arcsin(\frac{sin(\gamma_i)}{\eta'})$ and $\gamma_i = arcsin(h)$. In this way $\Delta\phi_p$ can be expressed as a function depending on h. With the condition $\Delta\phi_p - \phi_d = 0$, which means that $\Delta\phi_p$ must result in a change of direction equal to ϕ_d, one can proceed to solve h. The resulting equation:

$$2p \cdot arcsin(\frac{h}{\eta'}) - 2 \cdot arcsin(h) + p\pi - \phi_d = 0 \tag{40}$$

can be approximated using Taylor series or similar methods. Marschner et. al presented the following equation to approximate γ_t:

$$\gamma_t = \frac{3c}{\pi}\gamma_i - \frac{4c}{\pi^3}\gamma_i^3 \tag{41}$$

to gain a cubic function[6] $\Delta\hat{\phi}_p$ that resembles $\Delta\phi_p$:

$$\Delta\hat{\phi}_p(\gamma_i) = -\frac{8pc}{\pi^3}\gamma_i^3 + \left(\frac{6pc}{\pi} - 2\right)\gamma_i + p\pi \tag{42}$$

In these functions c is defined as $c = arcsin(\frac{1}{\eta'})$. Now h can be computed by calculating $\Delta\hat{\phi}_p - \phi_d = 0$ and taking the arcsines of the roots. There will be one root when solving h for the R and TT component, but one or three roots when solving the TRT component. If the TRT component

[5]This index is called Bravais index and can be calculated using θ_d: $\eta'(\theta) = cos(\theta)(\eta^2 - sin^2(\theta))^{-\frac{1}{2}}$ where η is the refractive index of hair.

[6]Cubic functions are relatively easy and efficient to solve. Solving methods, accompanied by an interesting historical background, are documented by Dunham [16]

provides three roots or that is to say three *sub*-paths, each of them has to be considered separately. Using the chain rule for finding derivatives, $\Delta\hat{\phi}_p(\gamma_i)$ can also be used to get $\frac{dh}{d\phi_p}$:

$$\frac{dh}{d\phi_p} = \left(\frac{d\phi_p}{dh}\right)^{-1} \approx \left((\Delta\hat{\phi}_p \circ \gamma_i)'(h)\right)^{-1} \tag{43}$$

while:

$$(\Delta\hat{\phi}_p \circ \gamma_i)'(h) = \frac{1}{\sqrt{1-h^2}}\left(-\frac{24pc}{\pi^3}\gamma_i^2 + \frac{6pc}{\pi} - 2\right) \tag{44}$$

Up to now, methods to compute h and $\frac{dh}{d\phi_p}$ of equation 38 have been presented. The last step is to compute the absorption factors $A_p(h)$. There are two effects that result in absorption, when considering a specific path. First, volume absorption in the fiber interior and second, Fresnel reflection, that determines what amount of light takes the current path. Let $F_p(\gamma_t)$ account for the Fresnel effect and $T(\sigma, \gamma_t) = exp(-2\sigma\cos\gamma_t)$ account for volume absorption, where the absorption coefficient σ is a color vector whose values range from zero to ∞ and $2\cos\gamma_t$ is the path's inner length[7]. Then the absorption functions can be expressed as:

$$\begin{aligned} A_R(h) &= F_R(\gamma_t) \\ A_{TT}(h) &= F_{TT}(\gamma_t)T(\sigma,\gamma_t) \\ A_{TRT}(h) &= F_{TRT}(\gamma_t)T(\sigma,\gamma_t)^2 \end{aligned} \tag{45}$$

By expanding *coefficients of reflection*, the functions F_p can be gained. The governing equations are explained in textbooks [21]. Marschner proved [39] that the refractive index of the parallel coefficient differs from the one of the perpendicular coefficient. The coefficients of reflection are:

$$\begin{aligned} r_s &= \frac{\cos(\gamma_1) - \frac{\eta^2}{\eta'}\cos(\gamma_2)}{\cos(\gamma_1) + \frac{\eta^2}{\eta'}\cos(\gamma_2)} \\ r_p &= \frac{\cos(\gamma_2) - \eta'\cos(\gamma_1)}{\cos(\gamma_2) + \eta'\cos(\gamma_1)} \end{aligned} \tag{46}$$

Here r_s is *s-polarized* and r_p *p-polarized*, where γ_1 is the angle between surface normal and reflected ray and γ_2 the angle between normal and transmitted ray.
Finally, Marschner et al. presented two implementation oriented optimizations. First, the N_{TRT}-component has caustics at certain angles. These

[7] In [40] Marschner et al. mistakenly state that the inner path length is $2+2\cos(2\gamma_t)$, figure 30 in appendix A.3 illustrates that this length has to be $2\cos(2\gamma_t)$

caustics are removed and replaced by a gaussian density function.To maintain continuity, the gaussian is faded over a range of incidence angles. The width of the gaussian density function and the fade range are input parameters to the shading model. Second, because hair strands are rather elliptical than circular, an eccentricity parameter has to be integrated. Eccentricity is approximated by changing the refractive index of hair. Following formulas were presented:

$$\begin{aligned} \eta_1^* &= 2(\eta-1)a^2 - \eta + 2 \\ \eta_2^* &= 2(\eta-1)a^{-2} - \eta + 2 \\ \eta^*(\phi_d) &= \frac{1}{2}((\eta_1^* + \eta_2^*) + \cos(2\phi_d)(\eta_1^* - \eta_2^*)) \end{aligned} \tag{47}$$

η^* is the adapted index of refraction and a is the eccentricity parameter. In order to evaluate the scattering function S, the curve radiance $\bar{L}$ has to be solved:

$$\begin{aligned} d\bar{L} &= S(\theta_i, \theta_o, \phi_i, \phi_o) d\bar{E} \\ d\bar{E} &= \bar{L}_i(\omega)\cos\theta_i d\omega = DL_i(\omega)\cos\theta_i d\omega \\ \Rightarrow \bar{L} &= \int_O S(\theta_i, \theta_o, \phi_i, \phi_o) d\bar{E}(\theta_i, \phi_i) d\omega \\ &= D\int_O S(\theta_i, \theta_o, \phi_i, \phi_o) L_i(\omega)\cos\theta_i d\omega \end{aligned} \tag{48}$$

where D is the diameter of hair fibers and the subscript O indicates, that the integral extends over the entire sphere. $\bar{L}$ can be used when hair is rendered as line primitive. Because the simulation that is discussed here generates surfaces, the radiance L is derived:

$$\begin{aligned} S(\theta_i, \theta_o, \phi_i, \phi_o) &= \frac{d\bar{L}}{d\bar{E}} = \frac{D \cdot dL}{D \cdot dE} = \frac{dL}{dE} \\ \Rightarrow dL &= S(\theta_i, \theta_o, \phi_i, \phi_o) dE \\ dE &= L_i(\omega)\cos\theta_i d\omega \\ \Rightarrow L &= \int_O S(\theta_i, \theta_o, \phi_i, \phi_o) E(\theta_i, \phi_i) d\omega \\ &= \int_O S(\theta_i, \theta_o, \phi_i, \phi_o) L_i(\omega)\cos\theta_i d\omega \end{aligned} \tag{49}$$

When area lights are neglected, $\bar{L}$ and L can be expressed by adding up the radiance for each point light source:

$$\bar{L} = D \sum_{k=1}^{n_l} S(\theta_i, \theta_o, \phi_i, \phi_o) E_L(\theta_i) \cos\theta_i \quad (50)$$

$$L = \sum_{k=1}^{n_l} S(\theta_i, \theta_o, \phi_i, \phi_o) E_L(\theta_i) \cos\theta_i$$

Here the irradiance E_L is measured in a plane perpendicular to the light source and n_l is the number of point lights.
The shading model of Marschner et al. is more sophisticated than the one of Kajiya and Kay and produces natural as well as energy conserving results. A drawback however is that control parameters are hardly comprehensible and understandable.

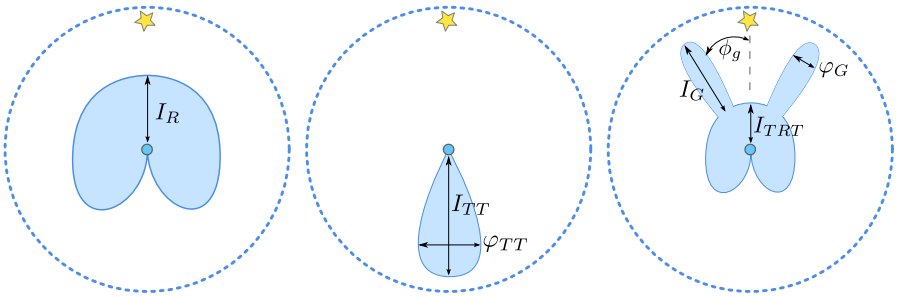
5.3 An Artist Friendly Hair Shading System

(a) Red-colored hair **(b)** Blue-colored hair **(c)** Pink-colored hair

Figure 13: Using the artist friendly shading system by Sadeghi et al., different hair looks can be created within seconds.

Sadeghi et al. recognized, that the shading model by Marschner et al. is cumbersome and developed a system [56] that places more value on controllability than on physical correctness. By intention, they created a model which can produce realistic and unrealistic results, depending on the concerns of art directors. They approached this task by defining a pseudo scattering function S' with *artist friendly controls* that approximates Marschners scattering function. The controls are divided by the possible paths:

- **R**: Color, intensity, longitudinal position and longitudinal width
- **TT**: Color, intensity, longitudinal position, longitudinal width and azimuthal width
- **TRT** (without glints): Color, intensity, longitudinal position and longitudinal width
- **Glints**: Intensity and azimuthal width



(a) N'_R-Component **(b)** N'_{TT}-Component **(c)** N'_{TRT}-Component

Figure 14

Figure 13 illustrates three different parameter configurations. Like the scattering function described by Marschner et al., S' is separated into an azimuthal function N' and a longitudinal function M':

$$S'(\theta_d, \phi_d) = \frac{1}{\cos^2 \theta_d} \sum_{p \in P} C_p I_p M'_p(\theta_d) N'_p(\phi_d) \tag{51}$$

Here C_p corresponds to the color of a given path and I_p to the respective intensity. The azimuthal function N' does not depend on θ, because Fresnel terms are ignored for simplicity. Sadeghi et al. also use gaussian functions to approximate M', but instead of unit area probability density functions they adopt unit height functions. The difference between the two is, that the latter have unit height at the maximum, whereas the area integral of the former functions is one. While unit height probability density functions do not have to be energy conserving, their advantage is that they are more controllable, because changing the width will not affect the brightness of the highlight. The M' functions are defined as:

$$\begin{aligned} M'_p(\theta_d) &= g'(\theta_d - \alpha_p, \beta_p^2) \\ g'(\nu, x) &= e^{-\pi \nu^2 x^{-2}} \end{aligned} \tag{52}$$

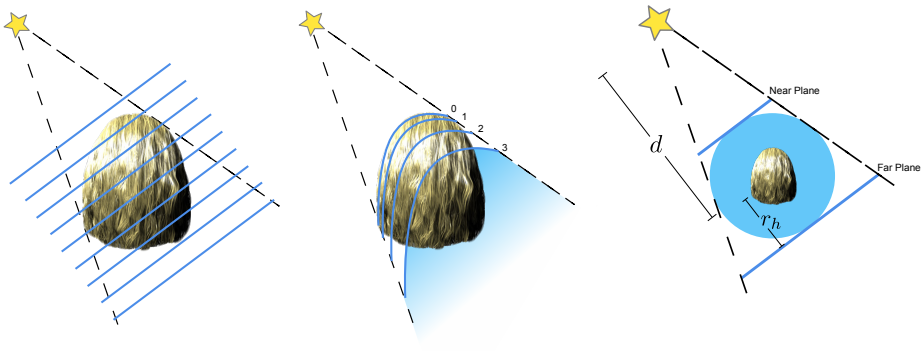
The azimuthal N'_p functions are visualized in figure 14. Each of them is motivated by the azimuthal functions of Marschners shading model. They can be formulated in the following way:

$$\begin{aligned} N'_R(\phi_d) &= \cos(\frac{\phi_d}{2}) \\ N'_{TT}(\phi_d) &= g'(\pi - \phi_d, \varphi_{TT}^2) \\ N'_{TRT}(\phi_d) &= \cos(\frac{\phi_d}{2}) + I_g g'(\phi_g - \phi_d, \varphi_g^2) \end{aligned} \tag{53}$$

Where φ_{TT}, φ_g are azimuthal width parameters, ϕ_g is the azimuthal position of glints and I_g is the intensity of glints. The appearance of glints can be randomized by choosing different ϕ_g values between $30°$ and $45°$ for each strand. Sadeghi et al. also show how the *Dual Scattering* model of Zinke et al. [72] fits into the artist friendly environment. For performance reasons, multiple scattering has not been implemented in the course of this thesis. Self-shadowing however has been taken into account, as described below.

5.4 Shadows

Shadows are important for the realism of virtual scenes. Scenes without shadow do not only feel unnatural, they are lacking an important proportion and spacing indicator. Hair can be regarded as a volumetric object and in order to perceive it as such, self-shadowing is crucial. Further, hair casts shadow on its surroundings and vice versa. Classic *shadow mapping* algorithms can be used for shadow casting from the environment to the hair (*environment to hair*), but are impractical for self-shadowing or *hair to environment* casting, because it performs a binary test, that determines whether a pixel is occluded or receives no shadow. In hair volume however, a large number of strands cast shadows, resulting in a variety of shading intensities. This cannot be realized with typical shadow mapping. *Opacity shadow maps* are an adaptation of shadow maps for volumetric objects. They were presented by Kim and Neumann in 2001 [33] and have since been used in hair rendering implementations. Analog to classic shadow mapping, opacity shadow maps apply renderings of the scene from the light's position. But instead of storing depth values, opacity values are stored at different depths from the light. This requires multiple rendering passes, depending on the granularity of depth levels. Storing opacity values has the advantage, that they can produce different shadow intensities and by association reduce the perceptibility of aliasing. In 2008 Yuksel and Keyser presented *deep opacity maps* [64], an updated version of opacity shadow maps. In contrast to opacity shadow maps, deep opacity maps do not separate the depth range of the hair volume in uniform layers. The layer density of deep opacity maps decreases with increasing depth. The major benefit of deep opacity maps is however, that those layers are adjusted to the the hair shape. A deep opacity map is generated in two succeeding steps, *A* and *B*. In step *A*, a common shadow map is rendered. The second step uses this map to determine at which depth z_0 the first opacity layer begins. Subsequent layers are placed at the depth values $z_1..z_k$, where *K* is the number of opacity layers and $0 \leq k < K$. One could also imagine, that after a light ray hits the first hair, it inducts the first layer, which has a fixed width and is followed by subsequent laysers. In step *B*, the hair is rendered a second time from the light's position. At each pixel-shader invokation a look-up at the



(a) Opacity shadow map layers **(b)** Deep opacity map layers **(c)** View frustrum setup

Figure 15: While opacity shadow maps use regular spaced layers, deep opacity maps use fewer layers that are adjusted to the hair shape. On the right view frustrum adaptations for the generation process are depicted.

shadow map returns z_0 and defines the fragment depth relative to the hair volume, with respect to the light. Each fragment can be associated to a layer by evaluating its depth. Let z be the depth of the current fragment, then the fragment is associated with layer l, if $z_l \leq z < z_{l+1}$, where $0 \leq l < K - 1$. This implies that the camera is looking along the positive z-axis. Each channel of a deep opacity map represents the opacity values of one layer. Hence up to four opacity layers can be stored in one *RGBA*-texture. A texel in the map is filled by adding the opacity value of hair to the channel that represents the layer associated with the fragment and subsequent channels. This accumulation pattern can be achieved by utilizing additive blending.
More than four layers can be used when employing multiple render targets or performing multiple draw calls. Because the frequency of opacity layers can be chosen to be highest at locations where shadow is fading in, the total amount of layers can be reduced. To achieve this, the layer width $z_{l+1} - z_l$ increases with progressing l. As a result deep opacity maps require less layers than opacity shadow maps. The assembly of layers is illustrated in figure 15.
The insertion of shadow in the final image happens in a similar way to other shadow mapping techniques. While hair is rendered from the camera's point of view, the pixel-shader evaluates the projection of the fragment onto the shadow and deep opacity map. A look-up at the shadow map provides z_0, which is enough to associate the fragment to an opacity layer, and by implication to a channel of the deep opacity map. A texture look-up at the deep opacity map then provides an opacity value for the fragment in question. Eventually the transmittance value τ can be ex-

pressed as:

$$\tau = exp(-\Omega) \tag{54}$$

Here Ω denotes the opacity. Typically the final color value is scaled by the transmittance value τ to receive shaded regions. Since this technique relies on sampling, aliasing artifacts are compulsory. In order to use the resolution of the deep opacity map efficiently, the view frustrum can be adapted to the hair volume, when rendering from the light's point of view. When r_h is the maximal distance of a particle to the hair origin and d is the distance between the light source and the hair origin, the view frustrum can be configured as following: $w_n = h_n = 2d_h(1 - \frac{r_h}{d}), d_n = d - r_h, d_f = d - r_h$. Where w_n, h_n are the width and height of the near plane, d_n is the distance from the light source to the near plane and d_f the distance to the far plane, see figure 15c.
Another technique for volumetric objects, called *adaptive volumetric shadow maps* (AVSM), were introduced by Intel Corporation in 2010 [17, 57]. Because Intel's reference implementation of AVSM required $9.7ms$ per frame, deep opacity maps have been used here. Performance implications and implementation details of deep opacity maps are discussed in section 6.3.

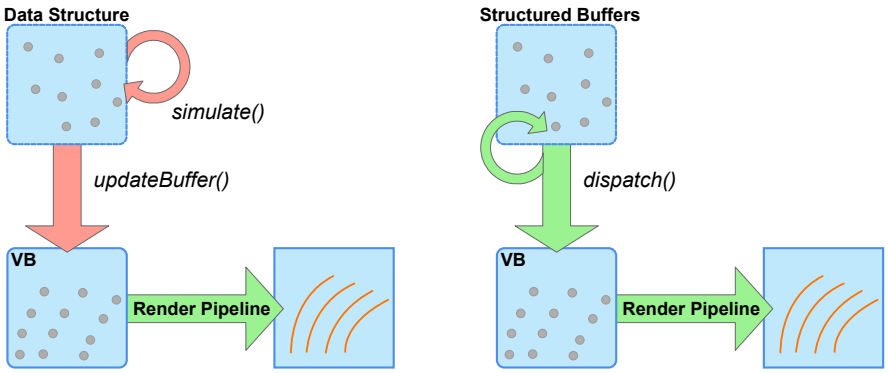
6 Implementation

The application provided with this thesis compares various techniques and demonstrates how the capabilities of modern hardware can be exploited. The implementation is based on DirectX 11 and utilizes the Qt SDK [10] for GUI creation and Assimp [22] in order to import various 3D file formats. The resulting software is a Win32 application, the programming language has been C++ and shaders were developed in HLSL, more precisely with shader model 5. Significant parameters are mapped to graphical input widgets, to allow comfortable testing. Some parameters are passed as *shader macros* and hence involve a shader recompilation, which is executed on the fly if necessary. Screenshots of the application and of each input element are attached in appendix A.4, A.6. The interpolation is based on single strands, as earlier discussed in section 4.2.

6.1 General-Purpose GPU

Many real-time applications are limited in their possibilities due to a high CPU workload. Since programmable GPUs were introduced in 2001, the processing capabilities of shaders increased steadily, as described by Montrym and Montrym [43]. This allows developers to shift work from the CPU to the GPU, releasing processing time for other tasks. One has to consider that while the CPU is a serial processor, graphic cards perform highly parallel, which has design as well as performance implications. GPUs are optimized to execute several thousand threads simultaneously, by using *single instruction, multiple data* (SIMD) control mechanisms. Hence, incoherent branching per thread results in performance penalties, see [49]. To implement algorithms designed for GPUs, either the graphics pipeline or the general purpose interface (GPGPU) of modern graphic cards can be utilized, see [46]. Microsoft enhanced DirectX 11[8] with a GPGPU API called DirectCompute. Adopting GPGPU Programming has the benefit that each thread has access to each input element and a shared memory, which belongs to a group of threads. These access patterns are hard to implement and are often expensive, when using the normal graphics pipeline. Similar to common shaders, *compute shaders* can be bound to the graphics card. Then the pipeline is also referred to as the *computation pipeline* [69].
The simulation presented here implements reference algorithms running on the CPU and corresponding variants that rely on DirectCompute. Performance differences between the CPU an GPU implementation are discussed in section 7. After their generation, particles are stored in an array-like data structure. The reference implementation operates on this structure and modifies entries with each simulation step. Subsequently the content is

[8] Even though Microsoft released DirectCompute with DirectX 11, it is also compatible to DirectX 10 graphic cards.



(a) CPU based simulation **(b)** GPU based simulation

Figure 16: Dataflow comparison of GPU and CPU based implementation.

mapped to a vertex buffer and serves as an input to the rendering pipeline. This implies, that the vertex buffer must be send to the GPU after each update. Figure 29a illustrates the dataflow of the reference implementation. While red arrows depict CPU operations, green arrows visualize GPU operations.
The GPU implementation stores generated data into *structured buffer*s and provides it to the computation pipeline via *unordered access view*s. Because the vertex buffer is also bound to the computation pipeline, one dispatch call suffices to update particle positions as well as the vertex buffer, see figure 29b. Here the vertex buffer stores particle positions, which are transformed to geometry within the rendering pipeline, as with the CPU implementation. Because the CPU does not operate on particles, no particle data has to be transferred between CPU and GPU.
DirectCompute provides a thread addressing system. Threads hold a 3D coordinate within their *thread group*. Thread groups on the other hand have a 3D coordinate on their own. This helps developers to realize access patterns. Regarding a fluid simulation for example, the 3D coordinates could be used to access entries in a grid. The simulation discussed here creates one thread group per wisp and one thread for each hair. If the y-coordinate of a thread equals zero, a guide hair is simulated and results are stored to a *group shared memory*. Otherwise, the shader accesses the group shared memory and creates single strand interpolated values. Since threads are working with the same data in parallel, the group shared memory has to be synchronized after the guide hair has been simulated. In pseudo-code, the compute shader looks as follows:

```
StructuredBuffer<PerHairData> HairBuffer;        // Input
```

```
RWStructuredBuffer<Particle>  ParticleBuffer;    // Input
RWBuffer<float4>              VertexBuffer;      // Output

groupshared Particle guide[PARTICLES_PER_HAIR]; // Cache

[numthreads(1, WISP_NEIGHBORS+1, 1)]
void CSSimulateHair( )
{
    if(DispatchThreadID.y == 0)
        simulateHair();

    GroupMemoryBarrierWithGroupSync();

    if(DispatchThreadID.y > 0)
        createInterpolatedStrand();
}
```

6.2 Tessellation

Up to this point hair has been represented by particles. Section 3 described techniques to simulate these. To render particles, the render pipeline generates actual strands out of them. The data enters the pipeline as a *control point patchlist*. In this section, the expressions *particle* and *control point* will be used interchangeable, since both refer to the same data. The input assembler forwards the data further to the vertex-shader, which performs the multiplication with the model matrix. Then, control points are passed to the tessellation stage, where they are converted to actual hairs. Figure 17 depicts the dataflow of this generation process.

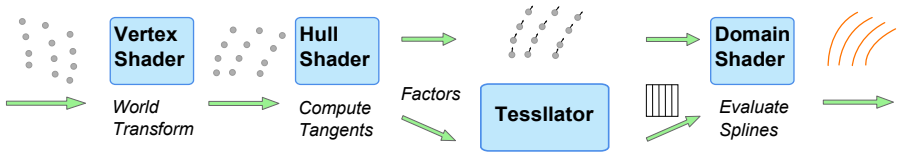


Figure 17: Dataflow through the tessellation stages

Control points entering the tessellation stages are evaluated and a tessellated triangle strip is passed on to proceeding stages. The hull-shader program is used to compute the tangents of control points and to forward them to the *tessellator*. Since one control point patch can cover a maximal amount of 32 points, this simulation can run with up to 31 segments per hair. The tessellator creates $3f + 1$ vertices for each strand, where f is an adaptable tessellation factor. By evaluating conditions such as distance of

a hair strand to the camera or visibility, the tesselation factor and hence the level of detail can be regulated. This is achieved by using the following patch constant function:

```
ConstantFunction( )
{
     InsideTessellationVertical = EdgeTessellationLeft
        = EdgeTessellationRight = 1;
     InsideTessellationHorizontal = EdgeTessellationTop
        = EdgeTessellationBottom  = f;
}
```

The *domain-shader* is invoked per tessellated vertex. Its general task is to evaluate parametric surfaces or curves to position vertices accordingly. This principle is adopted to form hair fibers based on a quadratic domain. Specifying the domain type to be a *quad* instead of *isoline* produces two dimensional hairs. This has the advantage that strands appear equally thick from different distances and that fewer strands create voluminous hair. Because hair is represented by a limited number of particles, segments are rather visualized by splines as by straight lines. *Cubic hermite splines* connect two control points with each other, given that the tangent of both points is known. An arbitrary amount of control points can be connected with piecewise cubic hermite splines and thus they are suitable to describe strands by a set of particles. More precisely, *Catmull–Rom splines* are used in this simulation, according to the method with witch the tangents were chosen in the hull-shader. While various techniques to create splines are presented in [55], Yuksel et al. [66] discuss Catmull–Rom splines in detail. The domain-shader receives three parameters: The domain location, the control point patch and hair specific attributes. The first defines the two dimensional location of the new vertex within the domain. Using the x-coordinate of this parameter as a progress variable, it is easy to calculate a spline polynomial. Let n_s be the number of tessellated hair segments and $w_s = \frac{1}{n_s}$ the width of a segment in domain coordinates, then the progress t_s between two control points is given by:

$$t_s = \lfloor x - \lfloor x \cdot n_s \rfloor \cdot w_s \rfloor \cdot w_s \tag{55}$$

where $\lfloor x \rfloor \doteq x - (x \ mod \ 1)$ is the floor operator. This variable can be used to calculate the hermite basis functions, which are defined as follows:

$$\begin{aligned} p(t_s) &= h_0(t_s)p_a + h_1(t_s)p_b + h_2(t_s)m_a + h_3(t_s)m_b \\ m(t_s) &= h'_0(t_s)p_a + h'_1(t_s)p_b + h'_2(t_s)m_a + h'_3(t_s)m_b \end{aligned} \tag{56}$$

$$h_0(t) = 2t^3 - 3t^2 + 1 \qquad h_0'(t) = 6t^2 - 6t$$
$$h_1(t) = 3t^2 - 2t^3 \qquad h_1'(t) = 6t - 6t^2$$
$$h_2(t) = t^3 - 2t^2 + t \qquad h_2'(t) = 3t - 4t + 1$$
$$h_3(t) = t^3 - t^2 \qquad h_3'(t) = 3t^2 - 2t$$

Here p_a, p_b is the position of the first and second control point respectively and m_a, m_b are the corresponding tangents. While $p(t_s)$ yields the position of new vertices, $m(t_s)$ yields their tangents. Within a control point patch p_a has the index $\lfloor x \cdot n_s \rfloor$ and p_b the same index plus one. Since the x-axis is used as a progress variable, one can imagine that the domain becomes stretched to form a single hair, as depicted in figure 18. After the hermite basis functions have been interpreted, the hair surfaces are folded and correlate to a Catmull-Rom spline.

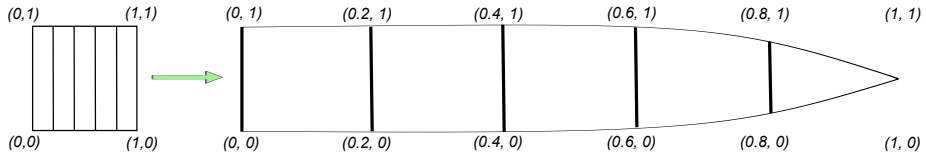
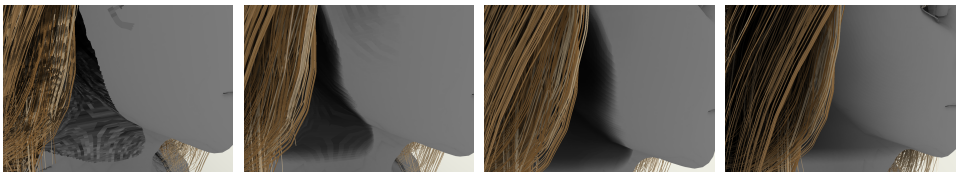


Figure 18: The x-coordinate of a vertex in domain space is used as a progression variable. Hence, strands are stretched along the x-axis.

To create the impression that hair strands are three dimensional rather than flat, hair surfaces are oriented towards the camera. Let d_h be the diameter of a hair strand, m_p the tangent of the vertex of interest and c_v the view vector of the camera in world space. Then the surfaces are spanned by translating vertices by $v_t = (y - 0.5)(c_v \times m_p)d_h$. Due to the fact that $y - 0.5$ is either -0.5 or 0.5, two vertices at the same progress are each translated by the half diameter in opposite direction. Finally the domain-shader performs operations to reduce the amount of computations at the pixel-shader. These are mainly related to texture look-ups, as further explained in section 6.6.2.

6.3 Deep Opacity Maps

Section 5.4 described that deep opacity maps are able to reproduce *hair to environment* shadow casting as well as self-shadowing. One could also argue that *environment to hair* casting is possible by adding high opacity values to the map, while rendering environmental objects or the character's body. However, using high opacity values would introduce intensive shadows, which may be unsuitable. Low values on the other side introduce artefacts as pictured in figure 19a. These artefacts can be reduced by employing *percentage-closer filtering* [20], see figures 19b and 19c. As mentioned in section 5.4 one *RGBA* texture suffices to store four opacity



(a) Deep opacity maps only, with no filtering

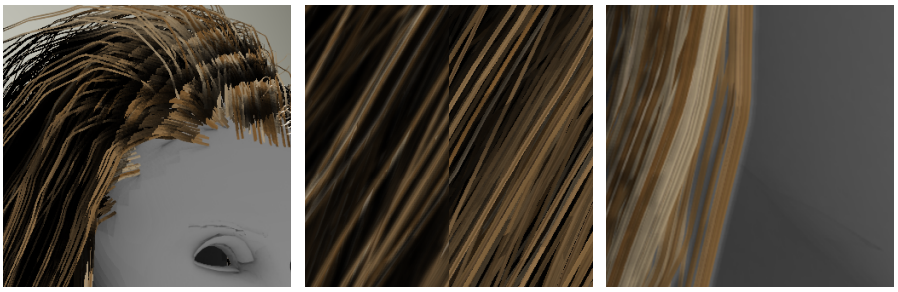
(b) Deep opacity maps only, with pcf radius 5 and 36 samples

(c) Deep opacity maps only, with pcf radius 6 and 49 samples

(d) Deep opacity maps and simulated absorption, each with pcf radius 6 and 49 samples

Figure 19: Shadow artefacts and the use of filtering

layers. Here three opacity layers were stored in one texture together with hair depth values. This has performance benefits in the context of filtering, since depth and opacity sampling is applied in one step. To avoid artefacts due to a low depth resolution, see figure 20a, the *DXGI_FORMAT* of the deep opacity map should provide one channel with at least 16 bit. A *DXGI_FORMAT_R16G16B16A16_UNORM* texture has been used here, which requires a scaling of opacity values to the range from zero to one.
Storing depth values within the deep opacity map also allows the efficient implementation of additional shadow mapping techniques, which require a second depth buffer of the complete scene.
Performance critical applications may demand faster algorithms than deep opacity mapping. On that account a second technique, essentially a simplification of deep opacity maps, has been implemented. It requires a depth map from the light's point of view and interprets the difference between these depth values to projected depth values, from the eye's point of view, as absorption values. This approach has already been used to render translucent materials [19]. It is very similar to shadow mapping but evaluates the differences in depth, rather than performing a binary test. The costs of a second draw call of the hair structure are saved, because no DOM-layers are needed. In contrast to deep opacity mapping, absorption based shadow mapping can be used for *environment to hair* shadow casting without introducing artefacts. A disadvantage is however, that strands are assumed to be spread evenly within the hair volume. Since this is not true for most scenes, shadows might appear less natural as with deep opacity mapping. By default the simulation described here uses DOM for self and *hair to environment* shadow casting, and the absorption based approach for *environment to hair* shadow casting.
It has also been tested to replace percentage-closer filtering by implementing *screen-space soft shadows*. Screen-space soft shadows [25] involve the rendering of shadows to a texture which is blurred, usually by applying a



(a) The hairline above the eye shows shadow artefacts due to a 8 bit depth buffer.

(b) Screen-space blurring affects the image quality. Left with, right without blur.

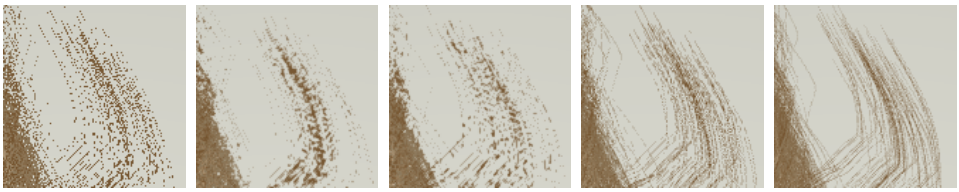
(c) Shadow-bleeding due to screen-space blurring

Figure 20: Further shadow artefacts

Gaussian kernel, and multiplied with the color buffer. During the course of this, two different artefacts appeared, making them deficient. Firstly, the human eyes are specialized in identifying differences in light intensity. As it is likely that the kernel radius covers multiple hairs, intensities are blurred over the hair volume. The result is that hair seems to be out of focus, see figure 20b. Figure 20c illustrates the second artefact, which is shadow bleeding: Shaded regions near to regions without shadow lose their shading, while unshaded regions falsely receive shadow.

6.4 Anti-Aliasing

Since hair strands are thin fibers, comparable to one dimensional lines, aliasing effects are strongly noticeable. They occur because of the limited sample rate during rasterization. Within the scope of this thesis, two anti-aliasing techniques have been tested. *Fast Approximate Anti-Aliasing* (FXAA), and *multisample anti-aliasing* (MSAA). Where the former one operates on the final output image and can thus be regarded as post-processing filter, MSAA supersamples the depth and stencil buffer. To implement FXAA the scene has to be rendered to a texture, which then is drawn onto a fullscreen quad. While the texture is rendered to the quad, the function $FxaaPixelShader$ is called once per pixel to reduce aliasing [38]. This technique has been used in several high-class computer games since 2011[11], because it is efficient, works seamlessly with deferred rendering and performs well on most scenes. MSAA shades each pixel once but samples multiple depth and stencil values to encounter aliasing. This method is provided by most graphic cards and can be enabled with few DirectX or OpenGL interface calls. MSAA produces excellent results but is less efficient than FXAA and cannot be easily combined with deferred shading, because the lightning pass can not be anti-aliased. The tests showed, that



(a) (b) (c) (d) (e)

Figure 21: A wisp of hair is rendered with different anti-aliasing techniques. Following techniques were used: (a) no anti-aliasing, (b) FXAA with default settings, (c) FXAA with high quality settings, (d) MSAA with four samples and (e) MSAA with eight samples.

FXAA has quality issues with the delicate nature of hair styles. Dithering effects were even visible when using quality preset 39, the highest quality level of FXAA 3.11. MSAA on the other hand performed as expected. Figure 21 compares various MSAA and FXAA configurations. Image 21a shows a rendering of some hairs without any anti-aliasing. 21b applies FXAA with default, 21c FXAA with preset 39 quality. Image 21d and 21e show strands rendered with 4 MSAA samples at quality level 2 and 8 samples at quality level 4 respectively. At a resolution of 1706x979 the rendering process with default FXAA took $0.458ms$ longer than without anti-aliasing. High quality FXAA took $0.814ms$ longer. The MSAA configuration which produced image 21d took $3.333ms$ and the configuration to produce image 21e $6.356ms$ longer. These values have been averaged over a period of 2 minutes and repeated several times. Measurement methods will be further discussed in section 7. A comprehensive summary of anti-aliasing techniques was given at Siggraph 2011[29].

6.5 Integration into frameworks

In order to integrate hair simulations into a framework or engine an easy representation is desirable. The optimal solution would add hair support to existing characters without further modifications. Since most systems rely on a scene-graph, *hair nodes* are required. Would such a hair node be attached to the *character node*, then all character transformations would also be applied to the hair. This might seem proper, but would result in inelastic hair which imitates the character's motions. If only root particles are attached to the character instead, then the remaining strands are dragged behind as with real hair. To achieve this, one can add a *character wrapper node* to the scene graph which is parent to the character as well as the hair node. Further, the moment that the character is moving, according transformations must be applied to the hair's root particles. While wrapper nodes work on stationary grounds or in isolated moving objects such

as trains, they inherently do not cover opposing wind effects, occurring in open moving objects like convertibles. Opposing wind can be addressed by placing the hair node further upwards in the scene-graph, which is only possible when the complete hair is subject to opposing wind, or by introducing additional external forces.
The implementation discussed here possesses a character (actor) class which completely abstracts hair interfaces. The class diagram in figure 22 outlines this structure.

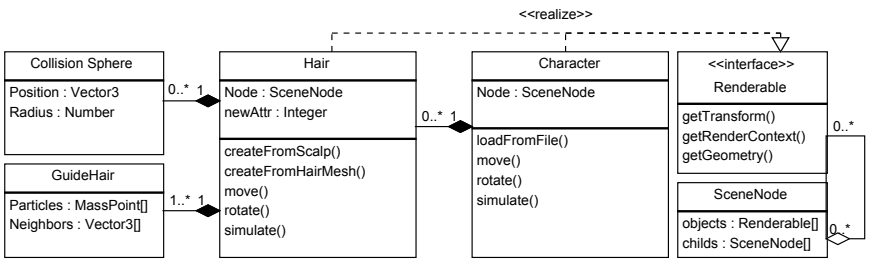


Figure 22: Schematic class diagram of hair related classes.

Characters are created by calling the *loadFromFile()* method, which reads various 3d formats and detects whether a scalp or hair mesh is present. Following this, either *createFromScalp()* or *createFromHairMesh()* gets called. Finally the accomplished wrapper nodes gets attached to the scene-graph.

6.6 Optimizations

6.6.1 Stream-Out

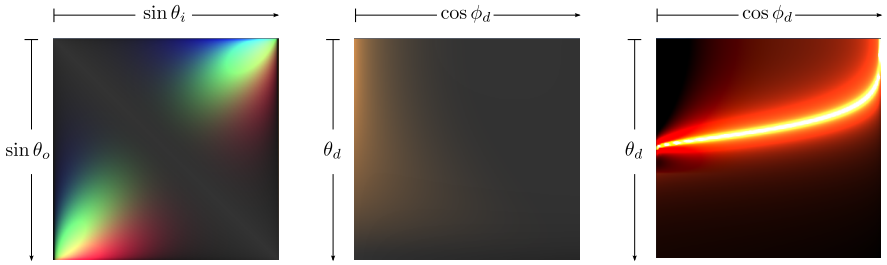
With Direct3D 10 the *geometry-shader* and the *stream output pipeline* were introduced. The stream output pipeline enables DirectX to stream vertices to a buffer instead of the rasterizer. On subsequent draw calls this buffer can be bound as an input vertex buffer. Because all vertex, hull, domain and geometry-shader operations were already applied before sending vertices to the buffer, the capabilities of the stream ouput stage can lower GPU operations, when multiple render passes are necessary. If a high amount of vertices is streamed out however, the buffer size can overcharge the graphic card. This can happen quickly, because vertices are not stored in a space-saving arrangement and vertices can occupy a relatively large amount of space, depending on the pixel-shader input requirements. The pixel-shader inspired by Marschner et al. as well as the pixel-shader inspired by Sadeghi et al. require $96bytes$ per fragment. Assuming that each hair is represented by 100 triangles, and that a hairstyle consists of 30.000 hairs, then the buffer already occupies $288MB$. The simulation discussed here actually performed faster without using stream-output buffers.

6.6.2 Look-Up Textures

Look-up tables are a common instrument to reduce the number of calculations by storing precomputed function values. *Look-up Textures* are the equivalent to GPU programming and have been utilized to reduce the workload of the Marschner and Artist Friendly hair shader. Otherwise, both would be computationally excessive, especially the former one. Whenever a re-computation is necessary due to the change of a shading parameter, the look-up textures are updated on the fly. The shading model of Marschner et al. is implemented by using three look-up textures, which are similar to the two described by Nguyen and Donnelly [49]. The first texture stores M_R, M_{TT} and M_{TRT} values to the RGB color channels and a value related to θ_d to the alpha channel. To fetch texels, $\sin\theta_i$ is mapped to u-coordinates and $\sin\theta_o$ is mapped to v-coordinates. Accessing the textures by sines instead of actual angles avoids expensive inverse trigonometric functions, because of the following:

$$\begin{aligned} \sin\theta_i &= \vec{l} \circ \vec{t} \\ \sin\theta_o &= \vec{v} \circ \vec{t} \end{aligned} \tag{57}$$

$$\begin{aligned} \vec{l}_\perp &= \vec{l} - \vec{t}\sin\theta_i \\ \vec{v}_\perp &= \vec{v} - \vec{t}\sin\theta_o \\ \cos\phi_d &= \frac{(\vec{l}_\perp \cdot \vec{v}_\perp)}{\sqrt{(\vec{l}_\perp \cdot \vec{l}_\perp)(\vec{v}_\perp \cdot \vec{v}_\perp)}} \end{aligned}$$



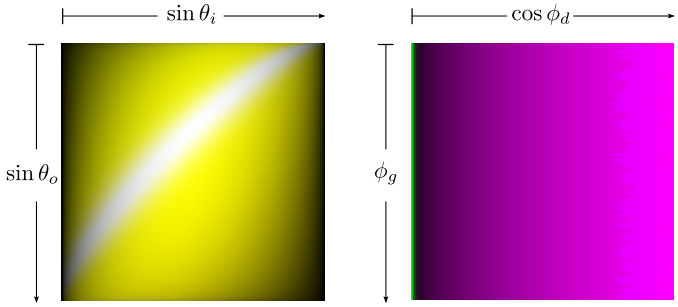
(a) M-components multiplied by projection term in RGB-channels, θ_d related term in alpha channel

(b) N_{TT}-component in RGB-channels, N_R-component in alpha channel

(c) N_{TRT}-component in RGB-channels

Figure 23: Look-up textures used to implement the shading model by Marschner et al. The background of figure *a* and *b* is dark grey, such as on the right side of figure *b*. This color is visible whenever the value of the alpha channel is low.

In this manner all required input angles are computed efficiently. Also, these computations can be applied within an arbitrary shader program, as soon as strands are tessellated. The values of the longitudinal functions M_R, M_{TT}, M_{TRT} are multiplied by $\cos\theta_i / \cos^2\theta_d$ to account for the solid angle projection and the projection of the irradiance (see equation 33 and 50 earlier). The value of the alpha channel ($2|\theta_d|/\pi$) is mapped to the v-coordinate of the second and third look-up texture, where the factor $2/\pi$ scales $|\theta_d|$ to a range from zero to one. $|\theta_d|$ is chosen instead of θ_d for the reason that the function is symmetric. The v-coordinate of the second and third texture is mapped to $\cos\phi_d$. While texture two stores N_{TT} values in the RGB channels and N_R in the alpha channel, texture three reproduces N_{TRT}. N_{TRT} as well as N_{TT} occupy three channels, since both components store color information due to absorption. Figure 23 illustrates these look-up textures. An analogous approach had been elected to implement the Artist friendly hair shader. Two textures serve as look-up memories, one storing M'_R, M'_{TT}, M'_{TRT} and one storing N'_R, N'_{TT}, N'_{TRT}. In contrast to N_{TT}, N_{TRT}, the components N'_{TT}, N'_{TRT} only reserve one channel, because color and intensity are multiplied after function evaluations, see equation 51. The first texture is also addressed by $\sin\theta_i$ and $\sin\theta_o$, the second texture on the other hand is addressed by $\cos\phi_d$ and ϕ_g. To randomize the value of ϕ_g, the fetch operation uses a pseudo random number between zero and one, based on the strand ID. Each row of the texture represents a different angle between $30°$ and $45°$. The resulting textures are depicted in figure 26.



(a) M'-components in RGB-channels **(b)** N'-components in RGB-channels

Figure 24: Look-up textures used to implement the artist friendly shading model by Sadeghi et al.

7 Evaluation

Real-time computer simulations are supposed to create a convincing virtual environment. Since humans are able to distinguish between image sequences of movements and continuous movements if the image rate is lower than 24 *frames per second*, real-time simulations must produce an image at least every $42ms$. Profiling techniques can be used to ensure that an application fulfils this requirement. The application presented by this thesis utilizes a profiling technique very similar to the one described by Preshing [52]. A *profiling module* is used to track each relevant function call and to analyse the execution duration. The profiling overhead per function call is less than $100ns$.

7.1 Visual Quality

The distinction between simulation and rendering is also reasonable, when evaluating the visual quality of the application. Since wind forces are the primary external forces of this simulation, the reaction to them is compared with the behaviour of real hair. Figure 25a shows a real person whose long hair is subject to wind. It is apparent, that strands form wisps, which fly in the same direction. This also is applicable to the virtual character demonstrated in figure 25b. The movement of real hair is more differentiated though, due to friction forces between neighboring hair strands [51] and because the simulation misses flow characteristics of wind. In the near future it will not be possible to simulate hair accurately, but the visual impression of simplified models is already satisfying.
The appearance of hair depends on the underlying shading system. While the model of Kajiya and Kay is rather inexact, the model by Sadeghi et al. and Marschner et al. are capable of presenting realistic results. Figure 25b

(a) The long hair of a person is subject to wind

(b) A similar setup, where the hair of a virtual character is subject to wind

Figure 25: Comparison between a real and a virtual character.

(a) Offline Rendering with 30.000 hairs

(b) Real-time rendering with 20.500 hairs and 500 guide hairs

Figure 26: Comparison between two hair renderings. The left one has been rendered over a period of 44 seconds, while the right is simulated at 30 frames per second.

is rendered using the artist-friendly system and shows natural color gradations. Marschner's shading model had been used to render the series of pictures in appendix A.4, which also appear real. The last picture of the series seems to be to bright, which might either be related to parameter settings, or to the neglect of elaborate scattering. Multiple-scattering eminently effects the appearance of light coloured hair, see [44], but has been ignored for performance reasons. While the physical based approach by Marschner et al. is bound to produce realistic hair, the model of Sadeghi et al. is able to produce super-realistic hair. Figure 26 pictures the attempt to create hair that looks like fire. Here, illustration 26a is more convincing, also because the application is unable to make gradients. All shading techniques discussed in this thesis are far-field approaches. Strands shaded by far-field models satisfy at a given distance but appear flat when inspected close enough. Because near-field approaches are more involved and still subject to research, far-field methods, which work well in most cases, were deployed here. In his Ph.D. thesis [70,71] Zinke discusses near-field methods in detail.

7.2 Computation Time

To estimate the efficiency of the simulation, the processing time of each computation step were averaged over a period of two minutes. Table 3 lists measurements taken with the reference (CPU-based) implementation. The test system was made up of a *Intel Core^{TM} i7-920* processor and a *nVidia GeForce GTX 560 Ti* graphics card. Each row contains the processing time per frame of the respective simulation step, followed by a percentage which relates to the total time per frame. The total time includes all simulation steps along with time needed for anti-aliasing and the presentation. Shadow had been disabled. The first three columns - 100 to 300 guides with 30 neighbors, which equates to 3.100 to 9.300 hairs - indicate that all methods are of linear complexity. The percentage values increase from row one to three, because the fraction of operations is unrelated to the simulation decreases. Prominent are high costs of collision-handling and angular forces. Since the character was rotating and the hair has been subject to wind and gravity, many hairs were colliding with one (or multiple) of the ten collision spheres in the test scene. This involved collision responses, which might explain the costs. A reduction of costs arising from collision-

Set-up	100 guides 30 neighbors	200 guides 30 neighbors	300 guides 30 neighbors	300 guides 60 neighbors
Add linear forces	$0.1973ms$ (2.44%)	$0.3985ms$ (4.01%)	$0.5907ms$ (4.27%)	$0.5935ms$ (2.94%)
Add angular forces	$1.2046ms$ (14.88%)	$2.4012ms$ (24.14%)	$3.6037ms$ (26.05%)	$3.6015ms$ (17.83%)
Add external forces	$0.0542ms$ (0.67%)	$0.1100ms$ (1.10%)	$0.1652ms$ (1.19%)	$0.1686ms$ (0.83%)
Integration	$0.3124ms$ (3.86%)	$0.6269ms$ (6.30%)	$0.9403ms$ (6.80%)	$0.9417ms$ (4.66%)
Post-Processing	$0.1094ms$ (1.35%)	$0.2159ms$ (2.17%)	$0.3441ms$ (2.49%)	$0.3561ms$ (1.76%)
Collision-Handling	$0.8614ms$ (10.57%)	$1.6741ms$ (16.83%)	$2.5928ms$ (18.75%)	$2.6141ms$ (12.94%)
Character Animation	$0.0108ms$ (0.13%)	$0.0201ms$ (0.20%)	$0.0288ms$ (0.20%)	$0.0337ms$ (0.17%)
Interpolation	$0.7276ms$ (8.97%)	$1.4497ms$ (14.56%)	$2.1637ms$ (15.65%)	$4.1803ms$ (20.67%)
Total time / FPS	$8.0967ms$ $123.5fps$	$9.9293ms$ $100.7fps$	$13.8292ms$ $72.31fps$	$20.1306ms$ $49.7fps$

Table 3: Performance measurements of the reference implementation.

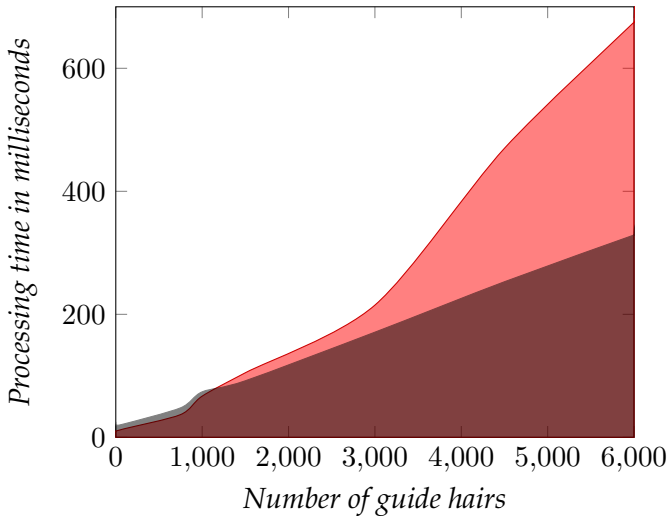


Figure 27: The red graph presents measurements of the reference implementation, while the black graph is presenting measurements of the GPU-based implementation.

handling could be achieved by using a *hierarchy of bubbles* [68] instead of collision spheres. The computation of angular forces is expensive due to rotational transformations, see section 3.2. A deeper investigation of flexion-springs might yield an alternative, which is better suited for real-time systems. Expenses of the interpolation step comprises adjustments of wisp radii that are related to speed of movements. The last column lists measurements taken with 18.300 hairs, consisting of 300 guides with 60 neighbors. Because twice as many neighbors were used in comparison with the previous column, the costs of the interpolation step doubled, while other steps are unaffected within a small tolerance range.
Figure 27 illustrates performance differences between the CPU and GPU-based implementation. Since the application's only task is to present hair, the CPU is not working to full capacity, if the simulation is moved to the GPU. This is reflected in the beginning of both graphs: The processing time of the GPU-based variant increases linearly right from the start, whereas the processing time of the CPU-based implementation increases slower at the beginning. At around 1.000 guide hairs, the graphs intersect, which means that the GPU based approach is faster thence. Due to the fact that typical hair styles are approximated by less than 1.000 guide hairs, the GPGPU implementation introduced no improvements here. CPU bound applications however can benefit from the general purpose interface of modern graphic cards. Furthermore, one has to control whether different thread allocations can increase the performance of the GPU-based implementation.

8 Conclusion

The simulation of virtual hair is a difficult task, especially when targeting real-time environments. While some effects, such as hair-hair interaction due to friction are not yet predictable fast enough, the overall behaviour of hair can be visualized in real-time. Even though a stand alone simulation has been presented here, it is likely that only a few characters near the camera require a detailed hair simulation, which makes the described techniques applicable for other applications, such as games. Depending on the hairstyle and the number of guide hairs, single characters can be simulated leaving space for additional computations.

The thesis showed, that the visualization of hair requires special efforts in many respects. In the beginning, modelling tools have to be designed for hair generation and need to support file formats that are usable for simulations. In addition to choosing and implementing animation methods, one has to develop geometry generation processes and adapt rendering techniques. Common shadow mapping is not suitable for hair rendering and some anti-aliasing procedures fail as well. Since much research has been spent on hair visualization, dedicated techniques are available.

The shading model presented by Marschner et al. enables visual pleasing hair and the derivation by Sadeghi et al. provides easy control parameters. State of the art simulations employ mass-spring systems [32] or a combination with auxiliary methods. Spring systems are well understood, providing many resources to draw on. Although simulated particles were used to generate hair strands, one could use the same methods to produce different kinds of geometry, such as tentacles (Figure 32a in the appendix outlines that tentacles can be produced by using strands with high diameter). Some features of the application still need more details. For example, artists may need the possibility to create color gradients or streaked hair.

Ducheneaut et al. [15] discovered that "hair style and color are considered 'high impact' features by users" (p. 4) and that users spend most time at adjusting them during avatar customization. Therefore, and because processing power increases, it is likely that within the next few years hair simulations become advertisement for high-class computer games. The simulation presented here demonstrates that natural hair has already arrived in real-time applications.

A Additional Figures

A.1 SEM Image of a Hair Fiber

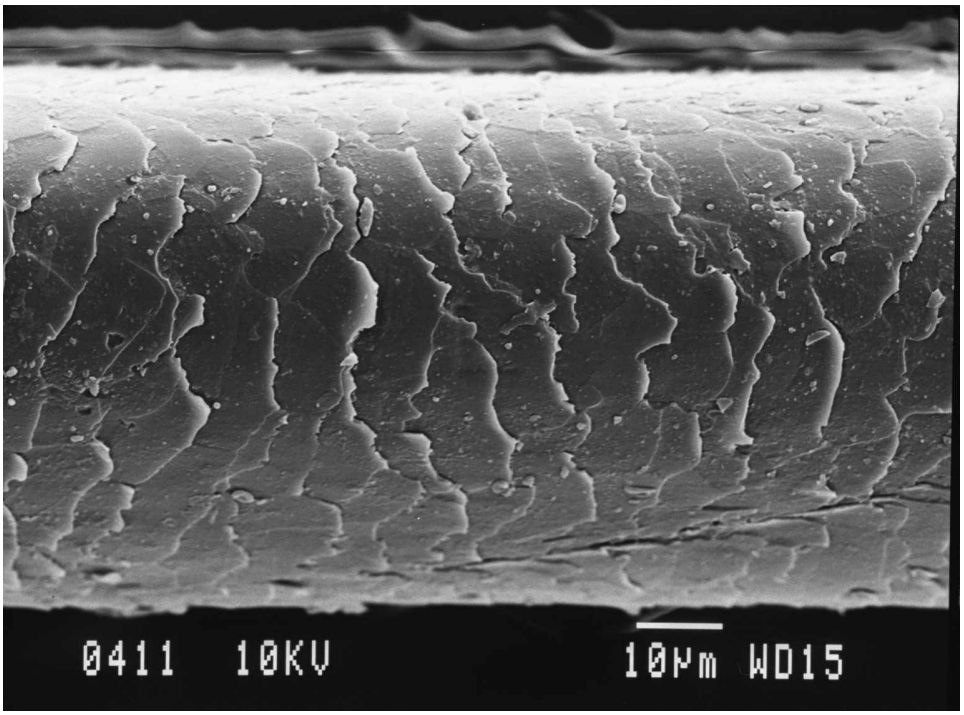


Figure 28: Image of a hair fiber taken with a scanning electron microscope. The cuticle scales are clearly visible. While the root points to the left, the tip points to the right. The image has been kindly provided by the Institute of Biological, Environmental and Rural Sciences of the Aberystwyth University and Dr. Stephen Clayton Wade.

A.2 Deep Opacity Map

(a) Hair-Rendering with the Marschner Shading model

(b) The corresponding deep opacity map

Figure 29: Hair-Rendering with its corresponding deep opacity map. Here the deep opacity map is inverted, hence yellow represents opacity values of the third layer, red values of the second and third layer, and black values of the first, second and third layer.

A.3 Internal Path Lengths in Unit Circles

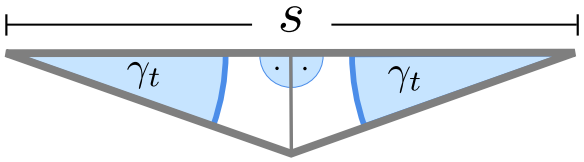
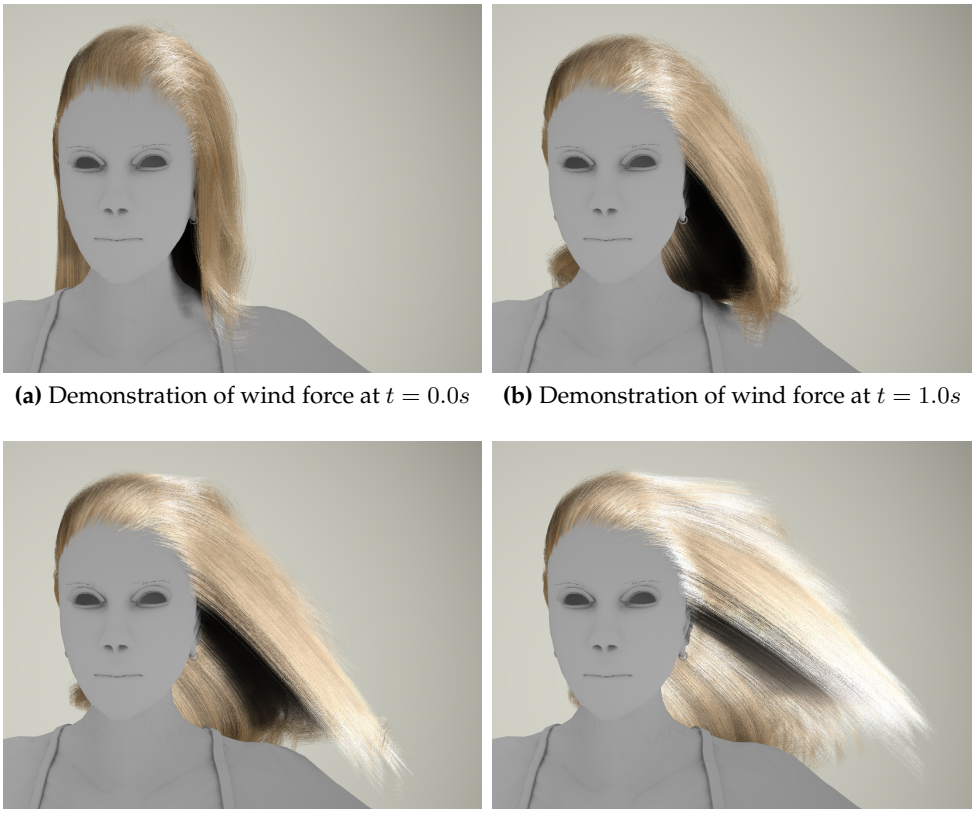


Figure 30: The internal path length in unit circles can be formulated as: $s = 2\cos(\gamma_t)$.

A.4 Demonstration of Wind



(a) Demonstration of wind force at $t = 0.0s$

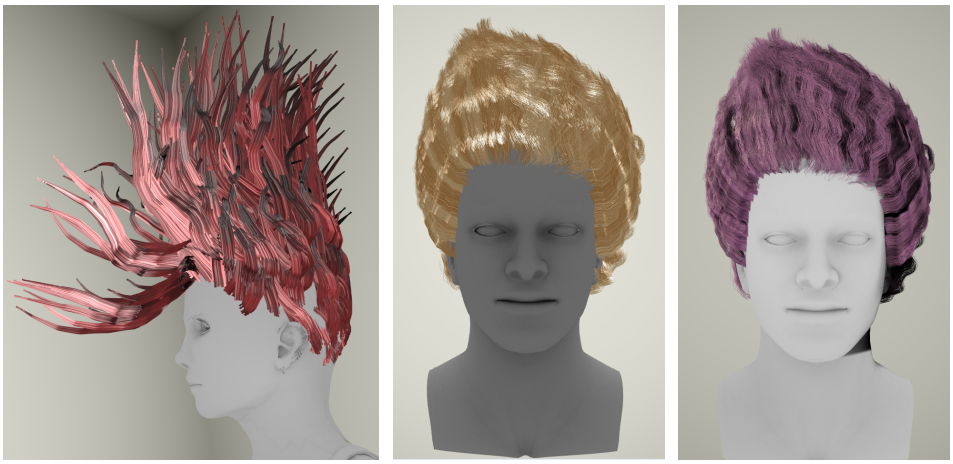
(b) Demonstration of wind force at $t = 1.0s$

(c) Demonstration of wind force at $t = 2.0s$

(d) Demonstration of wind force at $t = 3.0s$

Figure 31: Series of pictures to demonstrate hair under the influence of wind. Here, 50.000 strands are shaded with the model by Marschner et al.

A.5 Hair-Styles



(a) Tentacle-like hair structure (few hairs with high diameter)

(b) Hair without shadow seems to be flat

(c) Hair with self-shadowing on the other hand appears voluminous

Figure 32: Further hair-styles. Figure a) illustrates the usage of extreme parameter settings. Figure b) and c) show the effect of self-shadowing and rendering settings.

A.6 Graphical User Interface

Figure 33: Arranged screen-shots of the graphical user interface. The first three rows of control items address the different shading methods. Row four provides widgets which affect the physics of the simulation. Miscellaneous parameters, can be tweaked in row five, while the last row provides control over the process of geometry generation. All elements are rotated by 90°, due to space reasons.

B List of Figures

C List of Tables

9 References

[1] Natural hairstyle modeling and animation. *Graph. Models*, 63:67–85, March 2001.

[2] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering 3rd Edition: Advanced Shading*, chapter VII, page 1045. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[3] K.-i. Anjyo, Y. Usami, and T. Kurihara. A simple method for extracting the natural beauty of hair. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, pages 111–120, New York, NY, USA, 1992. ACM.

[4] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 43–54, New York, NY, USA, 1998. ACM.

[5] F. Bertails, S. Hadap, M.-P. Cani, M. Lin, T.-Y. Kim, S. Marschner, K. Ward, and Z. Kačić-Alesić. Realistic hair simulation: animation and rendering. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 89:1–89:154, New York, NY, USA, 2008. ACM.

[6] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 594–603, New York, NY, USA, 2002. ACM.

[7] J. T. Chang, J. Jin, and Y. Yu. A practical model for hair mutual interactions. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pages 73–80, New York, NY, USA, 2002. ACM.

[8] B. Choe, M. G. Choi, and H.-S. Ko. Simulating complex hair with robust collision handling. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 153–160, New York, NY, USA, 2005. ACM.

[9] D. Cline, S. Jeschke, A. Razdan, K. White, and P. Wonka. Dart throwing on surfaces. *Computer Graphics Forum*, 28(4):1217–1226, 6 2009.

[10] N. Corporation. The qt sdk. http://qt.nokia.com/products/, Apr. 2012.

[11] N. Corporation. Technology overview nvidia geforce gtx 680. http://www.nvidia.fr/content/PDF/product-specifications/GeForce_GTX_680_Whitepaper_FINAL.pdf, 2012.

[12] Culture.com. Final fantasy: The spirits within : Production notes. http://culture.com/articles/507/final-fantasy-the-spirits-within-production-notes.phtml, Apr. 2012.

[13] M. Deloura. *Game Programming Gems: The Shortest Arc Quaternion*, chapter II.11. Charles River Media, Inc., Rockland, MA, USA, 2000.

[14] Department of Engineering, University of Cambridge. Property information - young's modulus and specific stiffness. http://www-materials.eng.cam.ac.uk/mpsite/properties/non-IE/stiffness.html, Apr. 2012.

[15] N. Ducheneaut, M.-H. Wen, N. Yee, and G. Wadley. Body and mind: a study of avatar personalization in three virtual worlds. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 1151–1160, New York, NY, USA, 2009. ACM.

[16] W. Dunham. *Journey through genius: the great theorems of mathematics*. Wiley science editions. Penguin Books, 1991.

[17] W. Engel. *GPU Pro 2: Adaptive Volumetric Shadow Maps*, chapter IV.3, page 225—241. A. K. Peters, Ltd., Natick, MA, USA, 1st edition, 2011.

[18] R. Featherstone. *Robot dynamics algorithms*. Kluwer international series in engineering and computer science: Robotics. Kluwer, 1987.

[19] R. Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics: Real-Time Approximations to Subsurface Scattering*, chapter III.16. Pearson Higher Education, 2004.

[20] R. Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[21] G. Fowles. *Introduction to Modern Optics*, pages 40–45. Dover Books on Physics Series. Dover Publications, 1989.

[22] A. Gessler, T. Schulze, K. Kulling, and D. Nadlinger. Open asset import library. http://assimp.sourceforge.net/, Apr. 2012.

[23] S. Grabli, F. Sillion, S. R. Marschner, and J. E. Lengyel. Image-based hair capture by inverse lighting, May 2002.

[24] J. Gregory. *Game engine architecture: Rigid Body Dynamics*, chapter XII.4. Ak Peters Series. A K Peters, 2009.

[25] J. Gumbau Portalés. *Techniques for improving the visualization of natural scenes*. PhD thesis, Universitat Jaume I. Departament de Llenguatges i Sistemes Informàtics, Feb 2011.

[26] S. Hadap, M.-P. Cani, F. Bertails, M. Lin, K. Ward, S. R. Marschner, T.-Y. Kim, and Z. Kacic-Alesic. Strands and hair: Modeling, animation, and rendering, August 2007. Award: Best Course Notes for a New Course.

[27] S. Hadap and N. Magnenat-Thalmann. Interactive hair styler based on fluid flow. In *Eurographics Workshop on Computer Animation and Simulation 2000*, pages 87–99. Springer, August 2000.

[28] S. Hadap and N. Magnenat-Thalmann. Modeling dynamic hair as a continuum. *Computer Graphics Forum*, 20(3):329–338, 2001.

[29] J. Jimenez, D. Gutierrez, J. Yang, A. Reshetov, P. Demoreuille, T. Berghoff, C. Perthuis, H. Yu, M. McGuire, T. Lottes, H. Malan, E. Persson, D. Andreev, and T. Sousa. Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH 2011 Courses*, SIGGRAPH '11, pages 6:1–6:329, New York, NY, USA, 2011. ACM.

[30] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '89, pages 271–280, New York, NY, USA, 1989. ACM.

[31] T. K. Ken-Ichi, K. ichi Anjyo, and D. Thalmann. Hair animation with collision detection. In *In Models and Techniques in Computer Animation*, pages 128–138. Springer-Verlag, 1993.

[32] T.-Y. Kim. Writing a hair dynamics solver. http://www.rhythm.com/~tae/Kim_Simulation.ppt, Apr. 2012.

[33] T.-Y. Kim and U. Neumann. Opacity shadow maps. In *In Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 177–182. Springer-Verlag, 2001.

[34] T.-Y. Kim and U. Neumann. Interactive multiresolution hair modeling and editing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 620–629, New York, NY, USA, 2002. ACM.

[35] P. Kmoch, U. Bonanni, and N. Magnenat-Thalmann. Hair simulation model for real-time environments. In *Proceedings of the 2009 Computer Graphics International Conference*, CGI '09, pages 5–12, New York, NY, USA, 2009. ACM.

[36] C. K. Koh and Z. Huang. A simple physics model to animate human hair modeled in 2d strips in real time. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, pages 127–138, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[37] R. R. Lewis. Making shaders more physically plausible. In *In Fourth Eurographics Workshop on Rendering*, pages 47–62, 1994.

[38] T. Lottes. Fxaa. technical report. http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf, 2011.

[39] S. Marschner. Derivations for appendix of "light scattering from hair fibers".

[40] S. R. Marschner, H. W. Jensen, M. Cammarano, S. Worley, and P. Hanrahan. Light scattering from human hair fibers. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 780–791, New York, NY, USA, 2003. ACM.

[41] MAXON Computer GmbH. Schönes haar ist ihnen gegeben. http://www.maxon.net/de/products/cinema-4d-studio/hair.html, Apr. 2012.

[42] A. McAdams, A. Selle, K. Ward, E. Sifakis, and J. Teran. Detail preserving continuum simulation of straight hair. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 62:1–62:6, New York, NY, USA, 2009. ACM.

[43] J. Montrym and H. Moreton. The geforce 6800. *IEEE Micro*, 25:41–51, March 2005.

[44] J. T. Moon, B. Walter, and S. Marschner. Efficient multiple scattering in hair using spherical harmonics. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 31:1–31:7, New York, NY, USA, 2008. ACM.

[45] C. Orfanos and R. Happle. *Hair and hair diseases*. Springer-Verlag, 1990.

[46] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[47] R. Parent. *Computer animation: algorithms and techniques*, chapter III.7. The Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, 2002.

[48] L. Petrovic, M. Henne, and J. Anderson. Volumetric methods for simulation and rendering of hair. *Pixar Animation Studios*, (06-08):1–6, 2005.

[49] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*, chapter III.23, IV.34. Addison-Wesley Professional, 2005.

[50] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[51] E. Plante, M.-P. Cani, and P. Poulin. Capturing the complexity of hair motion. *Graphical Models (GMOD)*, 64(1):40–58, january 2002. submitted Nov. 2001, accepted, June 2002.

[52] J. Preshing. A c++ profiling module for multithreaded apis. http://preshing.com/20111203/a-c-profiling-module-for-multithreaded-apis, Apr. 2012.

[53] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *IN GRAPHICS INTERFACE*, pages 147–154, 1995.

[54] C. Robbins. *Chemical and physical behavior of human hair*. Springer, 2002.

[55] D. Rogers. *An introduction to NURBS: with historical perspective*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling Series. Morgan Kaufmann Publishers, 2001.

[56] I. Sadeghi, H. Pritchett, H. W. Jensen, and R. Tamstorf. An artist friendly hair shading system. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 56:1–56:10, New York, NY, USA, 2010. ACM.

[57] M. Salvi, K. Vidimče, A. Lauritzen, and A. Lefohn. Adaptive volumetric shadow maps. In *Eurographics Symposium on Rendering*, pages 1289–1296, June 2010.

[58] G. Sobottka and A. Weber. Geometrische und physikalische eigenschaften von human-haar. Technical Report CG2003/1, Institut für Informatik II, Universitaet Bonn, 2003.

[59] J. Spillmann and M. Teschner. 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*, SCA '07, pages 63–72, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[60] S. Tariq. Hair. Technical report, 2010.

[61] H. D. Taşkıran and U. Güdükbay. Physically-based simulation of hair strips in real-time. In *Proceedings of the 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG) Short Papers*, pages 153–156, Plzen - Bory, Czech Republic, February 2005.

[62] P. Volino and N. Magnenat-Thalmann. Animating complex hairstyles in real-time. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '04, pages 41–48, New York, NY, USA, 2004. ACM.

[63] K. Ward, F. Bertails, T.-Y. Kim, S. R. Marschner, M.-P. Cani, and M. C. Lin. A survey on hair modeling: Styling, simulation, and rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13:213–234, March 2007.

[64] C. Yuksel and J. Keyser. Deep opacity maps. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008)*, 27(2), 2008.

[65] C. Yuksel, S. Schaefer, and J. Keyser. Hair meshes. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 166:1–166:7, New York, NY, USA, 2009. ACM.

[66] C. Yuksel, S. Schaefer, and J. Keyser. On the parameterization of catmull-rom curves. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, SPM '09, pages 47–53, New York, NY, USA, 2009. ACM.

[67] O. K. T. L. Yvonne Jung, Alexander Rettig. Realistic real-time hair simulation and rendering. In *Proceedings*, Vision, Video and Graphics 2005, 2005.

[68] A. Zelinsky. Using path transforms to guide the search for findpath in 2d. *International Journal of Robotics Research*, 13:315–325, 1994.

[69] J. Zink, M. Pettineo, and J. Hoxley. *Practical Rendering and Computation with Direct3D 11*. Taylor and Francis, 2011.

[70] A. Zinke. *Photo-Realistic Rendering of Fiber Assemblies*. Dissertation, Universität Bonn, Jan. 2008.

[71] A. Zinke and A. Weber. Light scattering from filaments. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):342–356, 2007.

[72] A. Zinke, C. Yuksel, A. Weber, and J. Keyser. Dual scattering approximation for fast multiple scattering in hair. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 32:1–32:10, New York, NY, USA, 2008. ACM.