CHALMERS


# Simulation and Visualization of Hair for Real-Time Games

*Master of Science Thesis in the Programme Computer Science – Algorithms, Languages and Logic*

MATHIAS HÄLLMAN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2011

Simulation and Visualization of Hair for Real-Time Games

MATHIAS HÄLLMAN



Examiner: ULF ASSARSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2011

## Preface

This is a master thesis for the Computer Science and Engineering department at Chalmers University of Technology. The project was carried out at MindArk PE AB in their Gothenburg office, during the period January 2011 to May 2011.

Special thanks goes to my supervisor and examiner at Chalmers University of Technology, Ulf Assarsson, and to my supervisor at MindArk PE AB, David Skyfall, for all the help and feedback provided throughout the project. I would also like to thank Carl Henriksson and Peter Broqvist at MindArk PE AB for making the hair models required during this project. And finally, thanks to all the rest of MindArk PE AB for providing an excellent working environment.

## Abstract

This report presents an implementation of techniques for hair simulation and rendering meant to be run in real-time along side everything else in a performance heavy game by modern standards.

The simulation method is derived from a mass-spring system with added functionality to maintain a default pose and collide against the head while staying stable even at non-interactive frame rates. Patches of hair are extracted from a polygonal model of the hair. These patches are then sorted against the head in order to render them in the proper order for alpha blending to be performed correctly. A specialized hair shading model is also presented.

The results of the implementation show only a small drop in performance, which arguably is out-weighted by the pleasing visual results of having livelier hair for the in-game characters.

## Sammanfattning

Denna rapport presenterar en implementation av tekniker för hår-simulering och rendering ämnade för att köras tillsammans med resten av ett prestanda-krävande spel med dagens standard.

Simulerings-metoden är härledd från mass-spring-system med till-lagd funktionalitet för att behålla en standard pose samt kollidera mot huvudet medan systemet hålls stabilt även med icke-interaktive frame rates. Patchar av hår är extraherade från en polygon-modell av håret. Dessa patchar sorteras sedan mot huvudet för att uppnå lämpling renderings-ordning för alpha blending att utföras korrekt. En specialiserad hår-shading-modell presenteras också.

Resultaten från implementationen visar endast en liten minskning i prestandan, vilket diskuterbart vägs ut av de tillfredsställande visuella resultaten av att ha livligare hår för in-game-karaktärerna.

## Table of Contents


**1. Introduction** 6
1.1. Related Work ..... 6
1.1.1. Hair Simulation ..... 6
1.1.2. Hair Rendering ..... 6
1.2. Outline ..... 7

**2. Hair Modelling** 8
2.1. Strands ..... 8
2.2. Wisps ..... 8
2.3. Strips ..... 9

**3. Hair Simulation** 10
3.1. Mass-Spring Systems ..... 10
3.2. One Dimensional Projective Equations ..... 11

**4. Hair Rendering** 12
4.1. Transparency ..... 12
4.1.1. Screen-Door ..... 12
4.1.2. Alpha to coverage ..... 13
4.1.3. Alpha blending ..... 13
4.2. Shadowing ..... 14
4.2.1. Deep Shadow Maps ..... 14
4.2.2. Opacity Shadow Maps ..... 15
4.2.3. Deep Opacity Maps ..... 16
4.3. Shading ..... 16
4.3.1. Kajiya and Kay ..... 16
4.3.2. Marschner et al. ..... 17

**5. Implementation** 18
5.1. Model ..... 18
5.2. Simulation ..... 18
5.3. Rendering ..... 22
5.3.1. Sorting ..... 22
5.3.2. Shading ..... 24

**6. Results** 26

**7. Discussion** 27

**8. Conclusion** 28
8.1. Future Work ..... 28

**References** 29
**Gallery** 30

## List of Figures


1 Strand model ..... 8
2 Wisp model ..... 9
3 Strip model ..... 9
4 Skinning ..... 10
5 Mass-Springs System ..... 11
6 Hair segment for smulation using One-Dimensional Projective Equations ... 11
7 Screen-Door artifacts ..... 12
8 Patches sorted towards head ..... 14
9 Transmittance function ..... 15
10 Opacity Shadow Maps ..... 15
11 Deep Opacity Maps ..... 16
12 Hair strand on microscopical level ..... 17
13 Simulated bone-chain ..... 18
14 Distance correction ..... 19
15 Default pose ..... 21
16 Jitter by alpha to coverage ..... 22
17 Patch- and indice-lists ..... 24


## List of Algorithms


1 Simulation ..... 20
2 Patch-creation ..... 23
3 Shading ..... 25


## List of Tables


1 Performance results ..... 26

# 1. Introduction

Simulating and visualizing hair while getting visually pleasing results is a great challenge, and even more so when adding the time restraints imposed by real-time rendering. This is mostly due to the complex geometry of hair and the interaction between individual hair strands. There are several problems to be solved when attempting to create a realistic looking set of hair in games. The first being how to model the geometry of the hair; an explicit representation of every hair strand is rarely feasible. Then there is the rendering of the hair; light interaction with a set of hair is far more complex than the interaction with objects with simpler geometry and no transparency. Finally there is the simulation of the hair; there is a lot of physical interaction which is difficult to describe in the form of an algorithm. This is again due to the complex geometry of a full set of hair.

## 1.1. Related Work

Hair rendering and simulation are active areas of research with plenty of unsolved problems. Most research goes into simulating realistic looking hair, although the presented solutions are hardly suitable for real-time rendering in a game environment by today's processing power.

### 1.1.1. Hair Simulation

Plenty of models for simulating a full set of hair have been proposed, although only a few of these are viable today for real-time purposes in a game environment. Two noteworthy approaches are the ones proposed by Koh and Huang [1] and Thalmann et al. [2].

The paper by Koh and Huang models the hair using NURB[1]-surfaces, then each control-point of each surface is subjected to physics simulation. This causes each hair strip to be simulated independently of the others, unless forces are explicitly added between neighbouring strips.

Thalmann simulates the hair in a similar fashion to Koh and Huang, except wisps[2] are being simulated instead of control points along NURB-surfaces. Hair strands are then generated around the wisps to create the full set of hair.

### 1.1.2. Hair Rendering

Kajiya and Kay [3] introduced a hair shading algorithm which was capable of capturing a convincing diffuse term and a specular highlight. Marschner et al. [4] later studied the structure of a strand of hair to better understand how light interacts with it. Among other things they discovered that hair in fact has two specular highlights, and that these are shifted differently along the direction of the strand.

Scheuermann at ATI Research [5] devised a phenomenological shading algorithm that

1 Non-uniform rational B-spline – a parametric description of curves
2 See Section 2.2

also captured the components observed by Marschner et al. While not being physically accurate, this algorithm is faster and gives satisfactory visual results for most purposes.

The most successful shadowing techniques used for hair rendering have been variations of Shadow Maps [6]. Lokovic and Veach at Pixar [7] first introduced Deep Shadow Maps for the production of their movie Monsters Inc. Kim and Neumann [8] tried to achieve real-time performance with their Opacity Shadow Maps technique by making additional assumptions about the shadows (see Section 4.2.2). Recently, Yuksel and Keyser [9] introduced the Deep Opacity Maps algorithm which builds further on the idea of Opacity Shadow Maps and is both faster and produces higher quality shadows. Sintorn and Assarsson beat these results when they took another direction by improving a different technique called Occupancy Maps [10].

## 1.2. Outline

The report will begin by describing the various problems and a few of their possible solutions. In particular, the problems of modelling, simulation and rendering will be covered in Section 2, 3 and 4 respectively.

Section 5 will then describe the work performed during this project and the justification for the choices made. The results are then published in Section 6, followed by a discussion in Section 7 and conclusion in Section 8.

## 2. Hair Modelling

The model used to represent the geometry of hair has a large impact on what type of hair and styles would be allowed to be created by the artists. It also affects how the hair is to be simulated, and of course its performance.

Hair strands can be modelled either explicitly or through a wide range of implicit representations, meaning that some cruder model would be used to represent a set of strands. This is to overcome the issue of an overwhelming amount of geometry.

### 2.1. Strands

Perhaps the most obvious way to model hair is to draw individual line segments for each strand. A human head of hair has on average 100,000 strands. Suppose that each strand is represented by 40 line segments. That makes the entire head of hair consist of 4,000,000 line segments. Even if a simpler model consisting of only 20,000 strands is used, it takes 800,000 line segments, which is still a lot for modern hardware to handle [1]. An example of such a model can be seen in Figure 1.

This approach has a lot of problems. First of all, the amount of line segments required is a lot of geometry to be passed through the rendering pipeline. This would also make a lot of fine geometry cover individual pixels, resulting in aliasing artifacts [3].

Using this technique, the artists would find it difficult to place all the hair on the scalp without specialized tools to do so. They would also have a hard time stylizing the hair.

***Figure 1**: Strand model. (Image courtesy of Ulf Assarsson.)*

### 2.2. Wisps

An alternative to creating line segments for each strand is to create line segments only for a few strands and sparsely distribute them over the scalp. These strands, often called control hairs or wisps, can then be used to generate more strands between them [2]. An example of such a model can be seen in Figure 2.

This methods still has the problems of an overwhelming amount of geometry, although it makes it easier for the artists to place on the scalp and stylize. Perhaps the largest gain compared to modeling every strand, however, is that this solution would allow fairly efficient simulation of individual strands.

***Figure 2**: Wisp model. (Image courtesy of NVIDIA corporation.)*

## 2.3. Strips

A strip, sometimes called patch, is a flat surface used to represent a collection of strands. This of course does not give the same level of detail as the previously discussed models. Using proper rendering techniques, however, the method can often give sufficient visual results. The strips may be either polygonal or tessellated[1] from some sort of parametric surface [1], the main difference being in how they are later simulated. NURB-surfaces[2] can be simulated much like a regular strand, whereas a polygonal model has no inherent strand-like structure. An example of such a model using polygons can be seen in Figure 3.

As justified in Section 5, this last model using polygonal strips was the one chosen for implementation during this project. The rest of the report will focus mainly on techniques relevant for this model.

***Figure 3**: Strip model. (Image courtesy of ATI Research, Inc.)*

1 To generate triangles at run-time
2 Non-uniform rational B-spline – a parametric description of curves

## 3. Hair Simulation

As stated in Section 2, the model used for the geometrical representation will affect what type of hair and styles are possible to represent. These are also affected by the method used for simulation. It is therefore important to use a flexible method in order to not further restrict the artists.

It is common to animate polygonal models using a technique called skinning. What this means is that the geometrical model is seen as a skin to another model, called a skeleton. The skeleton consists of a bone chain, and when the chain changes its pose, the geometrical model changes with it. By letting each vertex in the geometrical model be weighted against one or several of the skeleton's bones, modern hardware can efficiently determine new positions for the vertices, resulting in the newly taken form. Figure 4 shows a model being skinned to a skeleton, including bones through its hair.

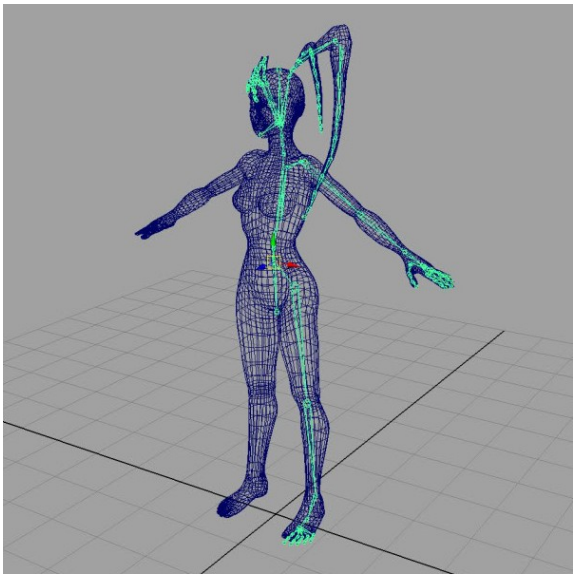


***Figure 4**: Skinning. (Image courtesy of Dan Lindberg.)*

This section will discuss different techniques for simulating such a bone chain, taking character motion and external forces such as gravity and wind into account. The bone chain is assumed to branch out in an acyclic tree-like structure as shown in Figure 13, thus allowing the chains to be treated as strands of hair.

### 3.1. Mass-Spring Systems

Rosenblum et al. [11] were among the first to propose a simulation model for individual hair strands. Essentially, they suggested that each strand should be modelled as a series particles connected by springs and hinges. Each particle would be subjected to fundamental laws of physics and the springs would hold the strands together. They also used angular springs to avoid hair bending at unrealistic angles. A strand represented by particles connected by springs and angular springs can be seen in Figure 5.

Although this method is easily implemented, spring models are generally hard to

control. Unless a very small time step is used during simulation, the system will become unstable. Another problem is that hair strands do not stretch, but obviously springs do. Stiff springs could be used, although this would make it even harder to keep the system stable.

Implicit integration, also called the backward Euler method, could be used to keep the system stable even when using stiff springs and large time steps [12]. This is however a considerably more expensive operation than the commonly used Euler method.

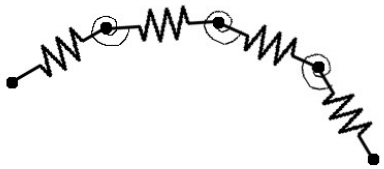


***Figure 5**: Mass-Spring System.*

## 3.2. One-Dimensional Projective Equations

Anjyo et al. [13] considered a strand of hair as though being built up of rigid sticks, or segments, rather than particles connected by springs. These segments are modelled as a distance from the end of a parent segment and two angles (azimuth and zenith) representing a direction to the parent. This idea can be seen in Figure 6 where segment $S_i$ is connected to its parent segment $S_{i-1}$ with azimuth $\Phi$ and zenith $\Theta$. During simulation, external forces are applied to the segments. These are then converted to torques. The external torques along with torques from spring forces between the segments to keep rigidity of the entire strand are then applied to the segment to calculate its new angles and thereby position.

Unlike Mass-Springs Systems, this method can efficiently simulate a large set of hairs while keeping the system stable without having to use a very small time step. It also prevents stretching of hair strands, which can be a significant issue with the previous method.

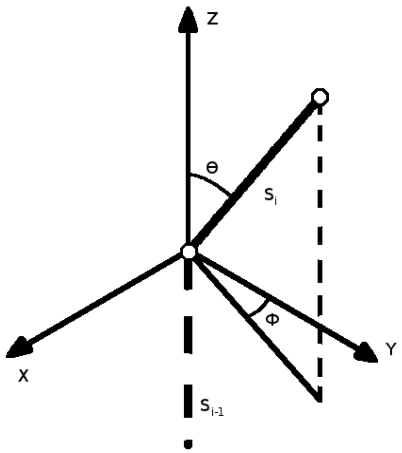



***Figure 6**: Hair segment for smulation using One-Dimensional Projective Equations.*

# 4. Hair Rendering

Three basic problems need to be addressed when rendering hair strips; transparency, shadowing and shading. Transparency is important mainly for the lower parts of the strips, where the hair is supposed to end but the actual triangles being rendered go on. Shadowing is particularly difficult for hair due to the complex geometry. Finally, the shading model needs to consider the special geometry and material of individual hair strands. Depending on the pigment, hair fibers can also be quite translucent, which affects all of the above mentioned problems.

## 4.1. Transparency

A more appropriate term for transparency which stays true to real life objects is translucensy, meaning the transmittance of light through an object. The term translucensy is however more commonly used when speaking of more sophisticated methods than those that will be discussed in this section. The methods in this section will only take the surface of an object into account, paying no attention to the thickness or material of the object. In translucent real life objects, light will refract in complicated angles and the intensity of the light will dissipate as it travels within the object. These effects are not captured by the methods below. Instead, light will pass straight through objects as though they had a uniform thickness.

Many methods have been proposed to achieve transparency in computer graphics. Three common techniques called screen-door, alpha to coverage and alpha blending have historically been used extensively in games.

### 4.1.1. Screen-Door

One of the simplest methods of achieving transparency is called screen-door. It works by only drawing half the pixels of a surface in a check-board pattern. That way, the viewer will perceive the surface as half transparent unless he or she is very close to the screen. This of course has a few drawbacks. For one, a surface can only be completely opaque or 50% transparent. Secondly, only two surfaces can ever be seen on a part of the screen covered by a transparent object. This artifact is illustrated in Figure 7.

A significant benefit of using this method is that there is no need to sort the fragments before drawing them. The problem of sorting will be discussed in more detail in Section 4.1.3.

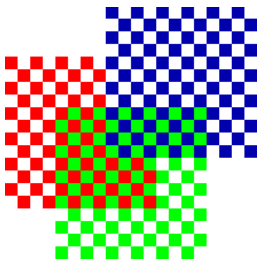


***Figure** 7: Screen-Door artifacts.*

### 4.1.2. Alpha to coverage

Alpha to coverage works much in the same way as screen-door, with the added advantage of allowing transparencies other than 50%. This is made possible through sub-pixel sampling. Instead of filtering out every other pixel of a fragment in a predefined check-board pattern, a randomized pattern is used to filter out a proportional set of the samples for a pixel.

Consider the case of rendering two surfaces covering the same pixel, one being opaque and the other being transparent and in front of the first. The second surface will have a mask associated with its alpha value, which will filter out samples taken for each pixel. No matter which surface is being rendered first, the opaque surface in the background will be visible only where the transparent surface was filtered out. The final color value for the pixel is then composed by taking the average out of the sub-pixel samples.

For an 8-bit alpha channel, the transparency of a surface may have one of 256 values. When creating the mask, this number is essentially reduced to the number of sub-samples taken for a pixel. What this means is that the discrete steps between adjacent transparency values become larger, and visual quality may suffer. If for instance only four sub-samples are taken per pixel, then a fragment may only have either of the transparencies 0%, 33.3%, 66.6% and 100%.

### 4.1.3. Alpha blending

Unlike the previously discussed methods to achieve transparency, alpha blending does not rely on filtering out samples or entire pixels [14]. As the name suggests, alpha blending uses an alpha value to blend the current fragment being drawn into the color buffer which is already holding the composite color of the previously drawn fragments. In the introduction to this section, it was stated that the thickness of an object will play no part when deciding how much light from behind the object will be transmitted through it. While this is technically true, the alpha value could vary over an object’s surface by using an alpha map, thus possibly giving the perception that the thickness varies as well.

The largest issue about using alpha blending is that fragments have to be rendered in a back-to-front order from the camera’s perspective in order to work properly. Ideally this sorting should be done for every sample. However, it is possible to get adequate results when sorting individual triangles or even entire objects.

It is also conceivable that the sorting is not even done back-to-front from the camera's perspective. Scheuermann realized this when implementing a hair rendering algorithm [5]. Instead, he sorted patches of hair, much like those described in Section 2.3, from the center of the head and outwards. By making use of proper culling in different render passes, he could then still render the patches in mostly back-to-front order from the camera. A big advantage of this method was that sorting only needed to be carried out once, as it no longer depended on the viewing angle of the camera. Figure 8 shows four patches from above and their render order as determined by their distance to the head.

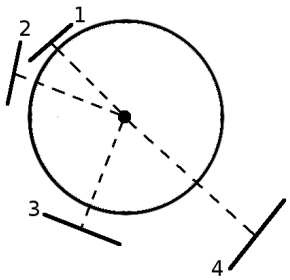



***Figure 8**: Patches sorted towards head.*

## 4.2. Shadowing

The complex geometry along with the translucent material of hair contributes to shadows which are exceptionally difficult to portray. Unlike a solid object of simple geometry, light penetrates and gradually drop off within a set of hair, rather than having its surface either in or out of light.

The most successful attempts at casting shadows for hair have been variations of Shadow Maps [6], a technique commonly used for shadowing of opaque objects. In short, the Shadow Maps technique work by rendering the scene from the view of the light source, storing only the depths at each texel. When rendering from the camera, the algorithm compares the distance between the light source and the fragment being rendered with the distance that the light reaches in the direction of the fragment. This second distance is stored as a depth value in the light's shadow map after having rendered the scene from the light's perspective. If an object lies between the light source and the fragment being rendered, the first distance will be larger than the second, and so the light falls short and the fragment must be in shadow.

### 4.2.1. Deep Shadow Maps

Rather than storing a single depth value at each pixel, Deep Shadow Maps store an approximation of the transmittance function [7]. This function describes how much light has dropped off at various distances from the light source in a certain direction.

A transmittance function is devised by merging two functions which are simpler to compute: a surface transmittance function and a volume transmittance function. The surface transmittance function describes the instantaneous drops in luminance when the light hits a surface. This function can be built from the depth and opacity values of the polygons when rendering. The volume transmittance function describes the continuous drop of luminance when light travels within a complex object such as a cloud or a set of hair. This function is built by sampling at regular intervals along the view vector that the transmittance function is currently being computed for. These functions are illustrated using an example in Figure 9.

When the scene is being rendered from the camera, shadows can be cast by using the

depth of the fragment being rendered and the relevant transmittance function to get the amount of light from a particular light source that reaches the fragment.

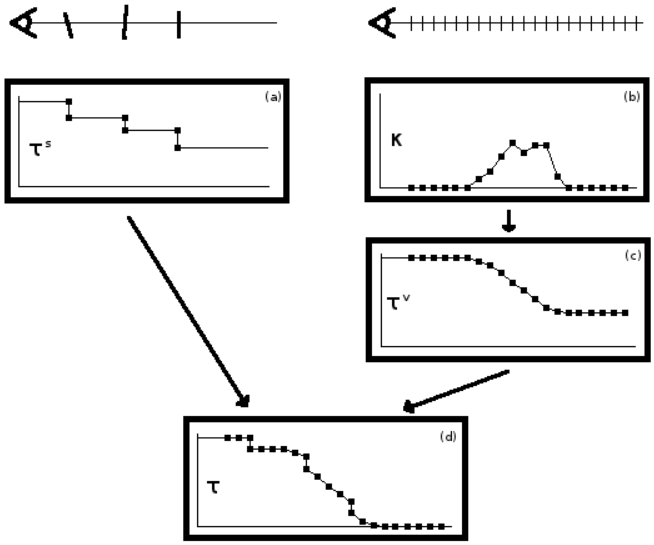



***Figure 9**: Transmittance function. (a) shows the surface transmittance funciton. (b) shows the extinction of light along segments of the ray. (c) shows the volume transmittance function. (d) is the combined transmittance function.*

### 4.2.2. Opacity Shadow Maps

Opacity Shadow Maps assume that the transmittance function as described in the previous section does not change radically over a short distance [8]. This assumption is exploited by rather than storing the transmittance function in a map, using several parallel maps as illustrated in Figure 10 to store the proportion of light that has dropped of at various distances. These maps are produced by rendering samples of the scene from the view of the light, once for each map and clipping the scene for the relevant area in front of the map. When these samples are being rendered, their alpha values accumulate in the maps and the maps ends up having the proportion of light that passes up to each map.

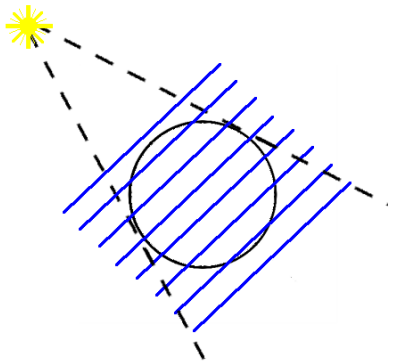


***Figure 10**: Opacity Shadow Maps.*

When the scene is being rendered from the camera, shadows can be cast by looking up the drop in luminance in the two closest maps and interpolating between these to get the amount of light that reaches the fragment.

### 4.2.3. Deep Opacity Maps

Deep Opacity Maps build onto the Opacity Shadow Maps technique by allowing maps that are not planar, as illustrated in Figure 11 [9]. As a first step when producing the maps, the depths are being drawn. It may help to see this depth map as a sheet covering the scene. All maps produced will namely have the same shape as this sheet. This way, the produced depth map will essentially shape the deep opacity maps.

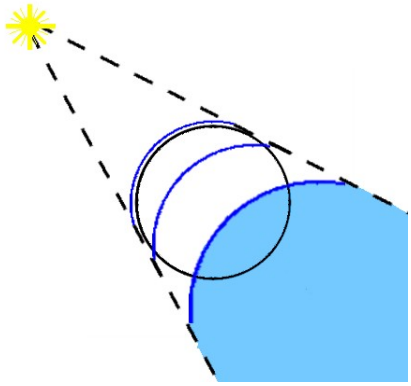


***Figure 11**: Deep Opacity Maps*

When rendering the scene from the view of the light, the depth in the direction of the fragment is used in the fragment shader to calculate the positions of the deep opacity maps. The opacity of the fragment being rendered is then output to all the maps behind it as seen from the point of the light.

## 4.3. Shading

While geometry and even textures can be used to represent real life geometry at a large or macro scale, shading is good at capturing real life geometry at a micro scale. That is, properties of various materials that can not be seen by the naked eye but still makes for instance plastic look different from metal.

The material and geometrical structure of a strand of hair have some quite unique properties. Commonly used shading models such as Blinn-Phong and Cook-Torrance are unable to capture some of these properties, and so new ones were devised. The most widely known among these are the ones introduced by Kajiya and Kay [3] and Marschner et al. [4].

### 4.3.1. Kajiya and Kay

A strand of hair is mostly cylindrical. For large cylinders, a general purpose shading model could be used successfully. Hair strands however are also incredibly thin, meaning that a specialized shading model is best used to model the light interaction with

the hair. Kajiya and Kay realized this when they devised their shading model for hair [3]. The diffuse term was derived by considering a lit diffuse cylinder and integrating the reflected radiance across the half circumference seen by the light. The specular term consists of a Phong-lobe centered at the incident angle mirrored against the plane with the direction of the hair as its normal.

### 4.3.2. Marschner et al.

Marschner et al. made an extensive study on the geometry and light interaction with individual strands of hair. They discovered that when looking at a strand of hair at a microscopical level, the strand no longer appear very cylindrical. The surface of a strand is rather covered by overlapping cells. This causes small but predictable shifts of the surface normal [4].

They were able to observe two specular highlights, and explained their existence by the shifted surface normal. One highlight comes from the reflected light and is shifted towards the tip of the hair, as the cells on the strand's surface is tilted in that direction. The other highlight comes from light which is transmitted through the strand, reflected against its inside and then refracted towards the light. This highlight is shifted away from the tip. Only the second highlight is colored by the hair's pigment. These components along with a third component for light that passes through the strand are illustrated in Figure 12.

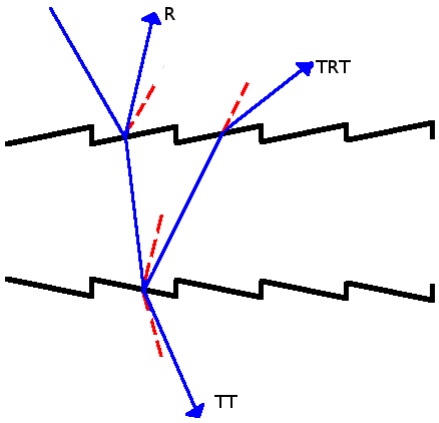



***Figure 12**: Hair strand on microscopical level.*

The math behind light's interaction with hair using this model is fairly complex, requiring a four-dimensional function to be solved. In order to make a practical implementation of their model, they also explained how to approximate their complete model by breaking up this four-dimensional function into two two-dimensional functions.

# 5. Implementation

## 5.1. Model

The implementation needs to run in real-time along with everything else in a game which already requires a lot of processing power by today's standards. It may also be the case that several characters are on the screen simultaneously. Performance is thus a key factor. Because of this, the strips model was used for the geometrical representation of the hair.

The artists also need to be able to create new hair styles easily. Due to the time limitations of this project, it was deemed best to use as much of current technology as possible. This resulted in letting the artists model the hair strips using triangles rather than parametric surfaces.

## 5.2. Simulation

Just as the rest of an animated character in a game, the hair model has a bone chain which is the structure being manipulated to emulate movement. In this implementation, one bone in the hair model's bone chain, called the head bone, is used to match the hair's bone chain to the rest of the character which is being animated by a separate system. Any bone with the head bone as its parent is considered a scalp bone. All of these bones simply follow the character in a rigid motion, with no extra simulation taking place. These differently treated bones are illustrated in Figure 13.

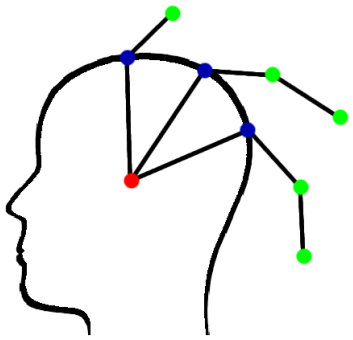


***Figure 13**: Simulated bone-chain. The red bone is the head bone. The blue bones are scalp bones. The green bones are subjected to simulation.*

The chosen simulation model is basically an extensively modified version of the Mass-Springs System. This model was chosen because of its simplicity and suitability to the bone chain being subjected to simulation, since each bone could simply be treated as a particle by the simulation method. Each chain is essentially treated as a hair strand would have been in the original Mass-Springs System.

In addition to the physics simulation, additional functionality was added in order to avoid collision against the head and to sustain a default pose. To do so, each bone has a default distance and direction to its parent. This direction is in the coordinate system of the parent, in order to respect the orientation of the whole set of hair.

The attempted methods that led up to the final solution is described below.

Initial attempt

The first attempt was basically a direct implementation of the Mass-Springs System. Forces for gravity and springs were applied to the particles in order to alter their velocities, which in order altered their positions. This resulted in a highly unstable system, as was expected. Time steps as small as 0.1ms were used with little success.

Replacing the springs

In an attempt at making the the system more stable, the forces from the springs were replaced with impulses. Instead of allowing the segments between the bones to stretch and thereby affect the velocities through spring forces, stretching was detected and prevented immediately through impulses added to the velocities of a particle and its parent causing them to move closer to each other to keep correct distance. In Figure 14, two bones have been moved according to their velocities, and stretching has occured. Impulses of magnitude $d_a/t$, where t is the time step used during integration, are added to each of the bones to make them keep correct distance $d_d$.

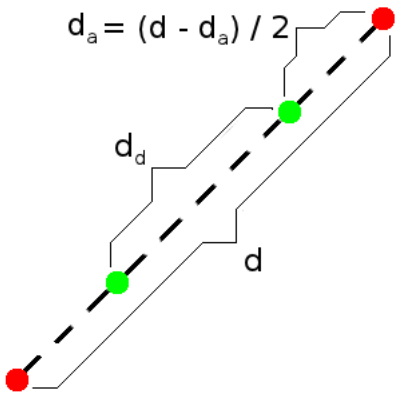



***Figure 14**: Distance correction.*

Head collision and imposing of a default pose were also added at this stage. To not introduce any new forces, collision avoidance was also done by impulses. The default pose was imposed by adding a velocity of constant magnitude in the direction towards the default position of the particle. The default position of a particle was determined by the particle's default direction to its parent and the parent's current position.

Although less physically accurate, this method still gave realistic looking results. Using impulses rather than springs can also be justified by the fact that hair strands do not

stretch. The impulses however had to be calculated from the positions of the particles and the positions they should have to keep the default distance constraints. This resulted in overly complex code which would make it difficult to account for other forces such as wind.

Adjusting positions and velocities separately

The previous attempt only affected positions through velocities, and impulses were applied to the velocities. In order to remove the code complexity this brought, an attempt was made at only using the velocities to move the particles initially, and then move the particles directly to keep constraints. This of course put the velocities and the actual movement out of sync, and so the velocities had to be adjusted to make up for this. The result was an equally performing simulation that was easier to maintain. However, it was not always obvious how to adjust the velocities to keep in sync with the actual movement.

Adjusting positions and velocities indirectly

After examining the previous attempt, a final solution was devised. To avoid all complexities regarding different forces to be accounted for and synchronization of movement and velocities, even less focus was put on the velocities. In short, the particles are first moved according to their velocities, their positions are then altered to satisfy constraints, and then finally the new velocities are calculated according to how the particles moved since the last frame. See Algorithm 1.

***Algorithm 1****: Simulation.*

```
1   for each particle
2       apply forces to velocity (gravity, friction)
3       apply velocity to position
4   end for
5   for each particle
6       use parent to calculate particle's default position
7       move particle towards default position, weighting by resistance
8   end for
9   for each particle
10      calculate distance to parent
11      move particle and parent to keep correct distance
12  end for
13  for each particle
14      calculate distance to head
15      if particle is inside head then move particle to edge of head
16  end for
17  for each particle
18      calculate new velocity to be used next frame
19  end for
```

As long as constraints are not in direct conflict with each other, this method will be stable even when using time steps as large as 0.1 seconds. This is because the last

displacements of the particles are always made to satisfy a constraint, thus forcing them into acceptable positions. In the way that velocities are determined, all possible errors due to inaccuracies carried from one frame to the next are also eliminated.

Depending on the constraints, the hair will behave in a natural way as long as the time step does not grow too large. The reason that problems may occur at larger time steps is that then, the particles will be moved farther by the velocities, thus making it more likely that constraints are violated more severely than if they had moved only a short distance.

Because of the non-critical nature of the problem of simulating hair in a game, optimization theory need not be applied when trying to satisfy the constraints. For this project, it was deemed sufficient to satisfy each constraint separately and sequentially, which may of course cause previously satisfied constraints to be violated. The constraints could also be iterated through any number of times to get a closer approximation or possibly solution to all of the constraints.

Although this solution is a long way from the original one using springs, it still behaves in a natural way. It is also fairly efficient and very easily modified to account for new forces. The relatively crude method of moving particles simply to satisfy constraints also has some physical justifications as it resemble impulses when this movement is fed back to the velocities for the next frame.

Finally, rather than using a velocity of constant magnitude to push particles towards their default positions, a resistance value is used to weight how much each particle should be moved. After a particle has been moved by the velocity, this value between 0 and 1 determines how far the particle should be moved on the line piercing the current and the default position. Figure 15 shows a bone with reistance 0.7, its position $p_b$ before this process and $p_a$ afterwards. The parent has position $p_p$ and $p_d$ is the default position of the bone.

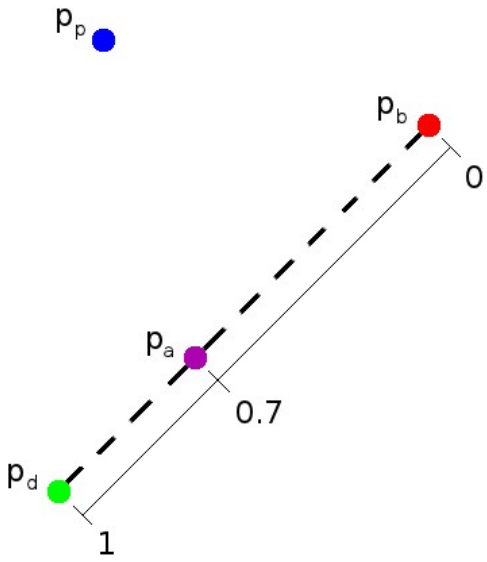



***Figure 15**: Default pose.*

## 5.3. Rendering

Alpha blending was used to achieve transparency, since this works well with the fine geometry of hair. The necessary sorting of the hair is a difficult task, especially if the render order should stay correct even when the hair has moved from its default position. If rendering would occur in an incorrect order, this would be likely to result in severe artifacts. It was argued that the alternative of using alpha to coverage, however, would yield too poor visual quality as the opacity varies radically over a small area of the hair strips, and alpha to coverage tend to produce jittery images in these situations, as can be seen in Figure 16.

***Figure 16**: Jitter by alpha to coverage. (© 1999-2011 Eurogamer Network Ltd.)*

The original intent was to use deep opacity maps for shadowing, since this is currently the fastest method of achieving volumetric shadows. Judging from their experiments, this method would still not be a viable option to be used in an end product, as it would be too slow especially when several heads of hair would be on screen simultaneously. This coupled with the short time frame of this project and the estimated time to implement the solution within CryEngine2 resulted in the decision to not implement any special type of shadowing for the hair.

### 5.3.1. Sorting

A fairly detailed hair model may consists of a few thousand triangles. Sorting these triangles while also calculating their distances to the camera using the triangles central point would be a costly operation. This implementation uses a cruder sorting, by

grouping triangles into patches and sorting them towards the head, similar to the technique used by Scheuermann [5] described in Section 4.1.3.

For the patches to be rendered in roughly back-to-front order from the camera when using this sorting technique, the hair is rendered using four passes; one for the opaque parts, one for the back-facing transparent parts, one for the front-facing transparent parts, and then finally one to fill the Z-buffer.

To make the process of creating a new hair style as easy as possible for the artists, no extra information regarding the patches has to be put into the model. Instead, the implementation breaks up the model into patches at load time. It does so by putting triangles which share common vertices into separate groups, and then creating a patch for each group. Because of this, the artists only need to make sure that the model actually consist of separable groups of triangles. See Algorithm 2.

***Algorithm 2**: Patch-creation.*

```
1   for each triangle
2       for each vertex index of triangle
3           for each group
4               if index is in group then mark triangle to be added to group
5           end for
6       end for
7       if no group held triangle's indices then create new group
8       else merge the groups that the triangle should be added to
9       add triangle's indices to resulting group
10  end for
11  for each group
12      put its triangles in consecutive order on the indices list
13  end for
14  store patch-information for each non-empty group
```

After the patches have been created, each patch has all of its triangles in consecutive order in the index-list. Correct render-order is then easily achieved by determining the correct render-order of the patches and then re-arranging the index-list accordingly. This is illustrated in Figure 17, which shows how the sorting algorithm is not concerned with the actual triangles being rendered.

There are various ways to determine the correct render-order of the patches. Some of them are covered below.

Sorting to head vs. camera

Because of the size of patches, their close proximity to each other and their unpredictable shape, it is difficult to choose a point by which to sort them. The averaged point for the patch's center, various points along its edges as well as random points within the patch were used with little success when sorting to the camera. Sorting the patches towards the center of the head proved much easier and mostly fail safe. Hence, this method was ultimately used.

Sorting patches vs. bones

Since it is the bones that are being manipulated by the simulation, and the geometry is only following through skinning, the bones need to be considered when sorting the geometry. Anything else will result in a sorting of the static model, thus allowing improper render-order when one patch may be thrown on top of another. However, it is the static pose that is most visible, and thus most important to look correct. When the hair is in motion, the fast movement helps hiding any artifacts due to improper render-order.

Sorting the bones alone and then propagating the render-order to the patches based on which bones they were weighted against, gave poor results. A lot of patches stacked on top of each other are more often than not weighted against the same bone. Thus, they are likely to receive incorrect render-order. This problem could probably be resolved by combining the sorting of the bones with a static sorting of the patches belonging to each bone. This was however never attempted due to time restrictions and the fact that the sorting of patches alone gave surprisingly good results.

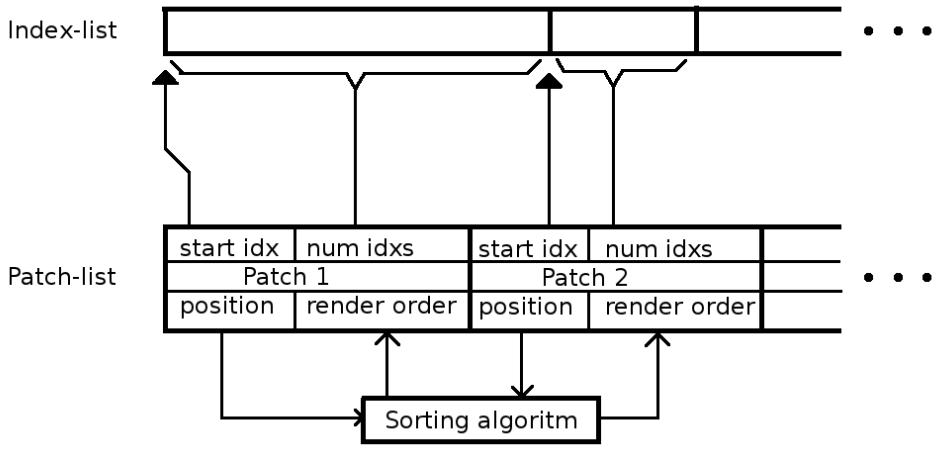



***Figure 17**: Patch- and index-lists.*

### 5.3.2. Shading

The shading model implemented is a phenomenological approximation of what Marschner et al. were able to observe. It is mostly the same model as that proposed by Scheuermann [5], only with a few minor adjustments to fit in well with the rest of the game developed for during this project, such as adding a small ambient term.

Algorithm 3 describes the shading performed for each light source. The accumulated specular and diffuse components from the various light sources are then multiplied with the base texture of the hair and added together. The function ShiftTangent() shifts the tangent using the normal, which is known to be perpendicular to the tangent. StrandSpecular() returns the specular component using a particular tangent. A texture is being used to look up randomized shift-amounts to make a patch look less uniform. Finally, a second texture is being used to look up randomized specularity-amounts for the secondary highlight. This gives the hair a slight sparkling appearance.

***Algorithm 3**: Shading.*

```
1  // primary highlight
2  uniform float prmShift;
3  uniform float3 prmSpecCol;
4  uniform float prmSpecExp;
5
6  // secondary highlight
7  uniform float sndShift;
8  uniform float3 sndSpecCol;
9  uniform float sndSpecExp;
10
11 float3 ShiftTangent(float3 tangent, float3 normal, float shift)
12 {
13    float3 shiftedTangent = tangent + shift * normal;
14    return normalize(shiftedTangent);
15 }
16
17 float StrandSpecular(float3 tangent, float3 view, float3 lightVec, float exponent)
18 {
19    float3 halfVec = normalize(view + lightVec);
20    float TdH = dot(tangent, halfVec);
21    float sinTH = sqrt(1.0 - TdH * TdH);
22    return smoothstep(-1.0, 0.0, TdH) * pow(sinTH, exponent);
23 }
24
25 void apply_light_hair(float3 tangent, float3 normal, float3 view,
26                       float3 lightVec, float2 uv, float3 lightCol,
27                       out fragOut color)
28 {
29    // diffuse component
30    float3 diff = saturate(lerp(0.5, 1.0, dot(normal, lightVec)));
31
32    // shifted tangents
33    float shift = tex2D(spec_shift_lookup_sampler, uv) - 0.5;
34    float3 t1 = ShiftTangent(tangent, normal, prmShift + shift);
35    float3 t2 = ShiftTangent(tangent, normal, sndShift + shift);
36
37    // specular components
38    float specMask = tex2D(spec_mask_lookup_sampler, uv);
39    float3 spec = StrandSpec(t1, view, lightVec, prmSpecExp) * prmSpecCol;
40    spec += StrandSpec(t2, view, lightVec, sndSpecExp) * sndSpecCol * specMask;
41
42    color.diffAcc += diff * lightCol;
43    color.specAcc += spec * lightCol;
44 }
```

## 6. Results

Images of a test-model consisting of 2870 triangles, resulting in 25 patches, being simulated and rendered by the methods presented in this report can be seen in the gallery. The simulated bone chain consists of three separate chains with three bones each. Gallery 1 shows the various components of the shading model and their compositions. Gallery 2 then shows what happens when the specular shifts are being altered. This is followed by Gallery 3 that demonstrates the effect of instead varying the specular exponents and Gally 4 that visualizes the hair in a variety of colors. Finally, Gallery 5 is illustrating some of the simulation aspects.

The same model was used to assess the cost of the simulation and rendering. The numbers shown in Table 1 were gathered when running the game with the resolution 1024x768 on a computer with the following specifications.

Intel Xeon W350 @ 2.8 GHz
5,98 GB RAM
NVIDIA GeForce GTX 470

Windows XP x64 SP2
DirectX 9.0c

***Table 1**: Performance results.*

| Heads | Simulation Transparency | Simulation No transparency | No simulation Transparency | No simulation No transparency |
|---|---|---|---|---|
| 20 | 53 | 58 | 53 | 58 |
| 50 | 27 | 30 | 27 | 30 |
| 100 | 14 | 16 | 14 | 16 |
| 200 | 7 | 8 | 7 | 8 |

The first column indicates the number of heads in the scene when gathering the numbers. The four subsequent columns show the number of frames per second (fps) when turning on and off simulation and transparency.

As can be read from Table 1, the simulation is virtually costless. Allowing transparency, however, shows an expected drop in performance. When transparency is permitted, the frame buffer must be read from in order to perform the alpha blending. More render calls will also be made as the hair requires four passes to be rendered. In contrast to when using an opaque material, no early Z-culling[1] can be made, causing the fill-rate[2] to increase. All of these factors play a part in reducing the perfomance when using transparency.

Comparing the hair's shading model against the generic shading model otherwise used for most parts of the game gave no measurable difference in performance.

1 A method used to shade only what is visible in the final image.
2 A measurement of overlapping shaded fragments.

## 7. Discussion

The final solution was only tested on one hair model. It is difficult to predict how it will perform when creating models of different styles and types of hair. Problems are likely to arise when making long hair for instance, especially if the patches are sorted only according to the static pose. The longer hair would then be more likely to overlap in undesirable ways, causing improper alpha blending to occur. Long hair would also twist in a way that is difficult to capture particularly when skinning the model to a skeleton.

It is also difficult to predict how well alpha blending using the sorting of the patches towards the head will perform on shorter hair. With a spiky hair style, there will be relatively dense volumes of hair on top of the head which will be partially transparent all the way through. This is in contrast to the transparent parts of the hair around the neck, where all hair is relatively far away from the center of the head. Hopefully, the artists would find a way to work around this, if problems would occur.

With multisampling becoming increasingly cheaper, it may be worth using some other technique than alpha blending for transparency. With a sufficiently high number of samples, the visual quality would not necessarily need to suffer a lot. This would likely be a more reasonable solution than trying to sort the hair patches, as this proved to be a difficult task when there were a lot of overlapping patches.

# 8. Conclusion

The presented techniques are well suited for implementation in modern real-time games. Still, the visual quality of hair may be of little importance to some games, in which case the decreased performance may not be justified. Over the years, game designers have devised several strategies for hiding the hair, which may still be an option. For other games where human hair is frequently visible, the presented techniques would be a viable option for producing livelier hair for the in-game characters.

## 8.1. Future Work

A proper shadowing technique will become increasingly important as the geometrical model gets more detailed. If the methods discussed in this report still would be too expensive, exploring stochastic methods may also be an option. The shadows would not be of the same high quality, but they would undoubtably look better than the current shadows which are cast by a method meant for fully opaque objects.

To be certain that the patches are rendered in the correct order even in shaking motions, the bones must be sorted as well as the patches. The bones could be sorted to give a primary render order of the patches. Each bone could then have a set of patches associated to it, with a static render order determined much like the patches for the entire hair model is now.

Currently, bones have an orientation which allows hair patches to bend. This model will always have the same side of a patch facing the same direction, with the exception of when a patch is bent over itself. It would be interesting to see if torsion could be incorporated into this model. That way, a hair patch could twist along its own direction, rather than just bend. This may look odd when using patches to represent the geometry of the hair. The simulation method may, however, be used for other types of geometrical models.

## References


[1] C. K. Koh, Z. Huang. *A simple physics model to animate human hair modeled in 2d strips in real time*, Eurographics Workshop of Computer Animation and Simulation, September 2001, pp. 127–138.

[2] A. Daldegan, N. M. Thalmann, T. Kurihara, D. Thalmann, *An integrated system for modeling, animating and rendering hair*, Computer Graphics Forum, vol. 12, no. 3, 1993, pp. 211–221.

[3] J. Kajiya, T. Kay, *Rendering fur with three dimensional textures*, Proceedings of ACM SIGGRAPH, 1989, pp. 271–280.

[4] S. Marschner, H. W. Jensen, M. Cammarano, S. Worley, P. Hanrahan, *Light scattering from human hair fibers*, ACM Transactions on Graphics, vol. 22, no. 3, July 2003, pp. 780–791.

[5] T. P. Scheuermann, *Practical Real-Time Hair Rendering and Shading*, Proceeding of ACM SIGGRAPH, 2004.

[6] T. Williams, *Casting Curved Shadows on Curved Surfaces*, Computer Graphics (SIGGRAPH Proceedings), August 1978, pp. 270-274.

[7] T. Lokovic, E. Veach, *Deep shadow maps*, Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., 2000, pp.385–392.

[8] T.-Y. Kim, U. Neumann, *Opacity shadow maps*, Rendering Techniques, ser. Springer, July 2001, pp. 177–182.

[9] C. Yuksel, J. Keyser, *Deep Opacity Maps*, Computer Graphics Forum (Proceedings of EUROGRAPHICS), 2008.

[10] E. Sintorn, U. Assarsson, *Hair Self Shadowing and Transparency Depth Ordering Using Occupancy maps*, Proceedings of the 2009 symposium on Interactive 3D graphics and games, 2009, pp. 67–74.

[11] R. Rosenblum, W. Carlson, E. Tripp, *Simulating the structure and dynamics of human hair: Modeling, rendering, and animation*, The Journal of Visualization and Computer Animation, vol. 2, no. 4, 1991, pp. 141–148.

[12] D. Baraff and, A. Witkin, *Large steps in cloth simulation*, Proceedings of ACM SIGGRAPH, 1998, pp. 43–54.

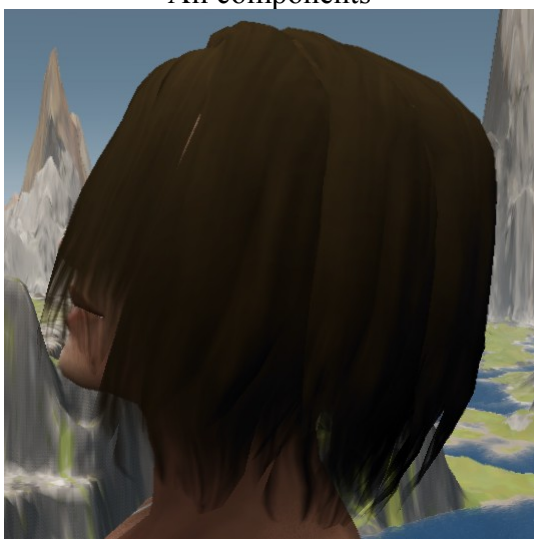
[13] K. Anjyo, Y. Usami, T. Kurihara, *A simple method for extracting the natural beauty of hair*, Proceedings of ACM SIGGRAPH, August 1992, pp. 111–120.

[14] T. Porter, T. Duff, *Compositing Digital Images*, ACM Siggraph Computer Graphics, vol. 18, no. 3, 1984, pp. 253-259.

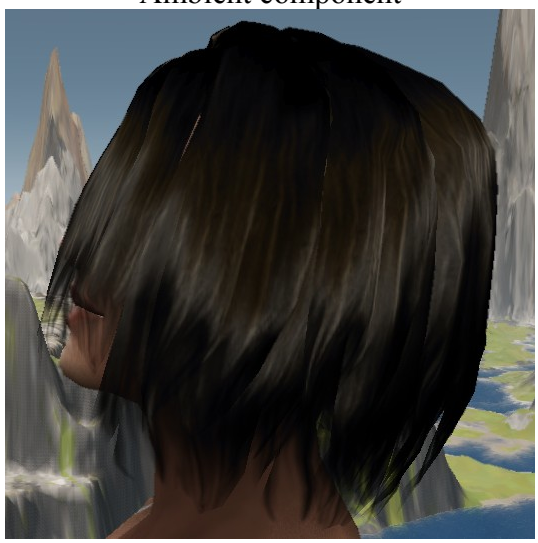
**Gallery 1**

All components

Ambient component

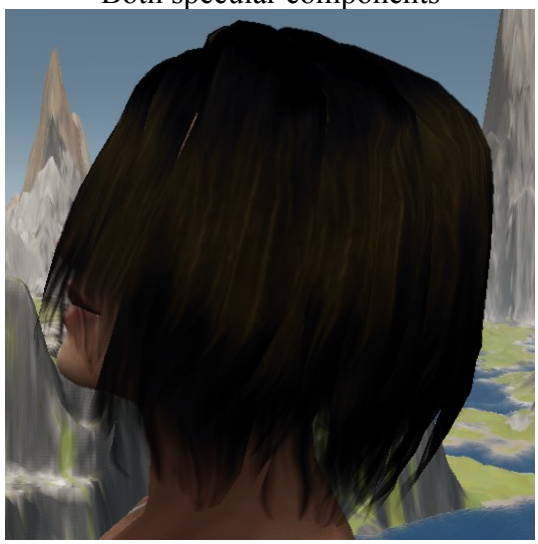


Diffuse component



Both specular components

Primary specular component

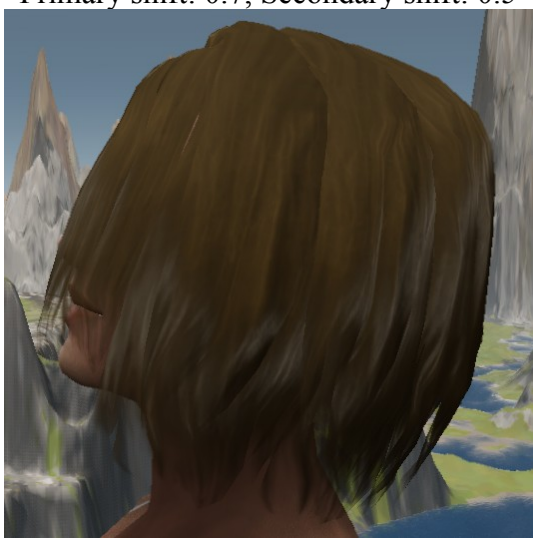


Secondary specular component

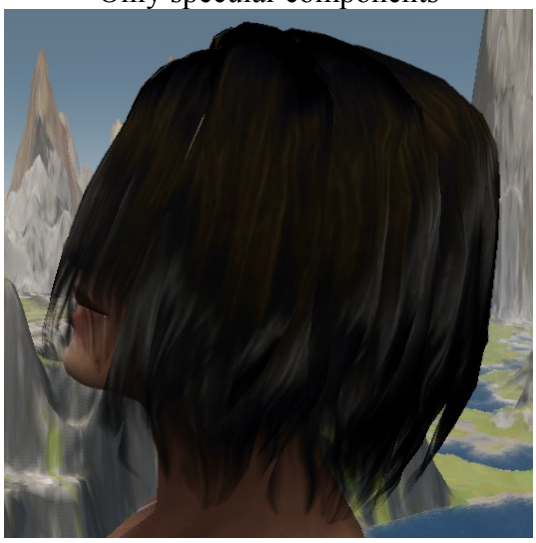
## Gallery 2

Primary shift: 0.7, Secondary shift: 0.5

Only specular components



Primary shift: 0.9, Secondary shift: 0.3



Only specular components

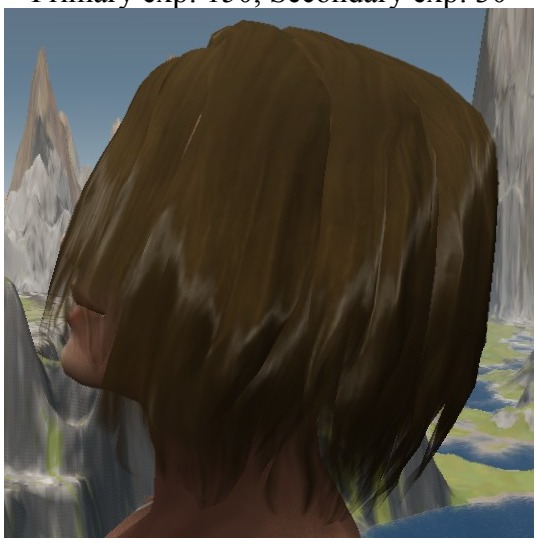
Primary shift: 0.5, Secondary shift: 0.7

Only specular components

## Gallery 3

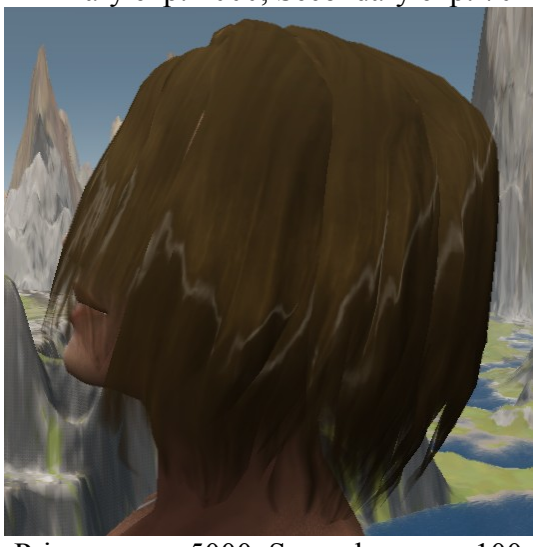
Primary exp: 150, Secondary exp: 30

Only specular components



Primary exp: 1000, Secondary exp: 70

Only specular components



Primary exp: 5000, Secondary exp: 100

Only specular components

**Gallery 4**

Brown

Green

Purple

Red

Black

Yellow

## Gallery 5

Looking down. The head collision prevents the hair from penetrating the skull.

Making a rapid turn. The hair is thrown to the side of the head as this was the previous heading.

Looking up. The resistance is stronger closer to the skull, causing the bending seen by the neck.

Charging forward. The hair is draged forward by the movement originating at the top of the strands.