I N F O R M A T I K

Modeling Hair using a Wisp Hair Model

Carina Schmitt   Martin Koster
Jörg Haber   Hans-Peter Seidel

MPI–I–2004–4-001   January 2004

FORSCHUNGSBERICHT   RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

Stuhlsatzenhausweg 85   66123 Saarbrücken   Germany

Author's Address

Carina Schmitt, Martin Koster, Jörg Haber, Hans-Peter Seidel
Computer Graphics Group
Max-Planck-Institut Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
Email: {cschmitt,koster,haberj,hpseidel}@mpi-sb.mpg.de

Abstract

We describe a tool for interactively designing virtual hairstyles. The underlying hair model is efficient enough to render different hairstyles in real time. One of the major problems of modeling human hair is the large number of hair strands in a hairstyle. Despite of increasing hardware and graphics performance, it is still difficult to display about 100,000 hair strands at a sufficient speed for real time applications. Our approach to this problem is to combine single hair strands to hair wisps. Each hair wisp is represented by a 2D triangle strip that is textured with color and alpha maps. To achieve a higher degree of realism, our modeling system takes into account both hair-head collisions and hair-hair collisions.

Keywords

hair modeling, interactive hair design

1 Introduction

In the field of modeling human faces, there is still one challenging problem: the creation and rendering of realistic human hair. Hair play an important role in computer generated images and is one important factor that decisively contributes to the plausibility of a virtual actor. But due to the high number of single hair strands, creating and rendering hairstyles is a difficult and time consuming job for offline rendering systems. Since modeling and rendering human faces becomes a more and more important task in real-time applications, the importance of representing realistic hair in these applications grows.
To reduce the geometric complexity of our hair model, we decide to combine single hair strands to one hair wisp, geometrically represented by planar surfaces. We use photo-realistic textures with color and alpha map to simulate single hair strands. The goal of this project was the development of a hair editing tool using the underlying wisp hair model. With this tool, the user is able to design different kinds of hair dresses, especially hairstyles with long hair. For extremely short hair, the group of fur models as discussed in Section 2 is still better suited for real-time rendering. One important design goal was the development of a system, that is able to render the modeled hair dresses in real-time for future animations.
In the following, we give a short overview of this research report. Section 2 covers previous approaches in the field of modeling human hair. In Section 3, we summarize some important properties and features of our wisp hair model. Beside other facts, we describe, how these patches are represented in our hair module and how they are generated. Section 4 shows up important steps, how different hairstyles can be designed with the hair editor. Different hairstyles generated with our hair editor are presented in Section 5. Finally, Section 7 explains some general problems and limitations of our approach in the field of rendering human hair and problems of wisp hair models in general.

2 Previous Approaches

There are very little satisfying approaches on rendering hair in real-time, that cover a wide range of different hairstyles. Due to the complexity of human hairstyles, most modeling systems are designed to cover certain kinds of hair dresses for special tasks. A solution that covers the entire complexity of modeling and rendering human hair in real-time is not known by the time of writing this paper.
Hair models can be roughly classified into following categories:

- Fur models
- Wisp/Cluster hair models

- Strand hair models
- Hybrid hair models

Fur models for short hairstyles [Gol97, GW97, KK89, Ney98, Len00, LPFH01]: in these models fur-like materials are often represented by three-dimensional volume textures generated via particle systems. Therefore, these models are only suitable for short hairstyles. However, this technique is often used in computer based animation movies like *Shrek, Ice Age, Monster Inc.* for describing animal fur. These models are often very fast due to their simplicity and already find direct application in real-time rendering.
Wisp/Cluster hair models as proposed by [KH01] exploit the observed tendency of adjacent hair strands that form cluster due to static attraction or special styling intentions. These models often describe hair clusters as parametric NURBS patches. Each surface patch represents one hair strip with different shape and different size for different hairstyles (see [KH01]). All hair strands together are presented in layers of strips overlaying each other on top of the scalp. To simulate single hair strands however, these models use texture mapping. Our approach of modeling hair also belongs to the group of wisp hair models.
Strand hair models introduced by [HMT01, DMTKT93, KYT92, CSDI99] allow designing, editing and rendering every single hair strand. These hair models often reach a high degree of realism, because each hair strand can be edited as desired. But on the other hand, they are often not suitable for real-time applications. Furthermore, modeling hairstyles in these systems is a time consuming and tedious task. Some systems claim to present hair in real-time, but they don't often reach the 25 frames per seconds that are necessary for a fluid animation. Additionally, there is no computational capacity left for other tasks like animation or even for putting these characters into an virtual environment.
Hybrid systems try to bridge the gap between wisp and hair strand models and attempt to combine the advantages of both models, the *strand model* and the *wisp model*. [KN02] present a multiresultion hair modeling system that constructs a hierarchy of hair clusters. The user is able to edit every cluster down to a single hair strand. By the aid of this feature it is possible to choose the hierarchy as precise as it is necessary for the current hairstyle. In some cases, the model is useable for simpler hairstyles in real-time.

3 Summary of the Hair Wisp Model

To understand rendering and modeling methods presented in later sections, we first summarize briefly, how hair is represented in our hair module.

3.1 Modeling Hair using 2D Patches

As described above, we combine a wisp of hair to a single two-dimensional strip instead of rendering every single hair strand. The planar surface is denoted as *hair patch* subsequently. This approach decreases the geometric complexity and thus accelerates the rendering process.
In this section, the physics, that found the basis of the modeling algorithms, are discussed. The whole modeling process, that transfers the rectangular surface into a realistic looking hair wisp, can be divided into three single steps:

- Calculating a planar surface that is divided into height and width segments
- Curling the planar surface
- Bending the surface and avoid collisions between hair-head and optionally hair-hair

Generating a Hair Patch

A hair patch is a plane that is subdivided into width and height segments. The number of these width and height segments is variable and depends on the type of the desired hairstyle. Uncurled and unbent hair for example need only one height and one width segment. In order to bend these patches, the number of height segments has to be increased. The number of width segments does not affect this process. On the other hand, a high number of width segments increases the smoothness of a curly hairstyle.
Formally, a hair patch is an array of points, that are assigned to different segments. Every point however has an assigned normal, that is calculated as the average of the surface normals of the neighboring segments to that this point belongs to. This has the effect, that the rendered hair patch looks more smoothly after the shading has been applied.
The last important information, that a point in this array needs is its texture coordinates. These are generated automatically at the creation time of a patch. The point on the bottom left gets the coordinates $(0, 0)$ and the point on the top right gets $(1, 1)$.
Creating the planar surface is an easy task. An interpolation routine calculates every single point on the patch by using two anchor points at the bottom of the patch, a vector that defines the patch direction and a scalar for the length of the patch.

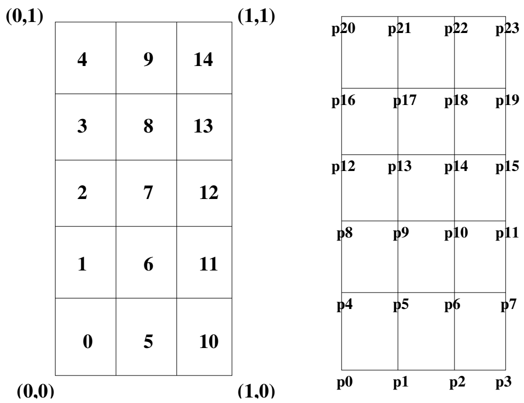

Figure 1: The left side of the picture shows a patch numbered by segments including the texture coordinates. On the right, the hair patch is numbered by points.

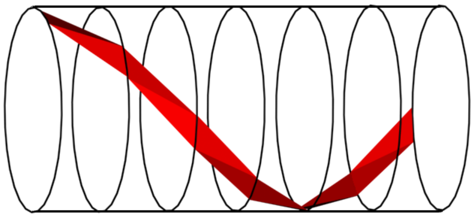

Figure 2: A hair patch projected onto a cylindrical surface. Segments are slightly displaced to achieve a curly effect.

Curling

The second step of creating a hair patch is the curling procedure. This process takes place before the bending procedure in the modeling pipeline. This ensures a proper intersection test with the head geometry.
The user defines a number of curls per patch and the radius of these curls. The algorithm behind this procedure is quite simple: The method creates a cylinder around the straight, plane patch. This cylinder has the same length than the patch and the diameter of the desired curl size.
Next, we project every height segment onto the surface of this cylinder, whereby the actual patch segments are slightly displaced than the previous projected segments. The strength of this displacement however depends on the number of curls and the number of height segments, the user has defined per patch. This displacement on the cylinder surface has the effect, that the resulting hair patch looks curly. The curling process is depicted in Figure 2.
The intuitive description of this approach can be expressed by a sequence of geometric operations. The exact value for the displacement, measured in radiance, is

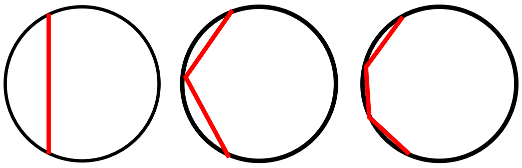

Figure 3: The view shows the cut through the cylinder and the projection of a hair patch with one, two and three width segments.

calculated as follows:

$$disp\_rad = \frac{2 * \pi * n_{curls}}{n_{segments}} \quad ,$$

where n_{curls} is the number of twists and $n_{segments}$ the number of segments of the patch. The first step of the algorithm is the projection of the patch points onto the unit circle by using the sine and cosine functions. This projection can be expressed simplified as follows:

$$P(i) = (\cos(i * disp\_rad), 0, \sin(i * disp\_rad)) \quad ,$$

with $i \in \{0, .., height\_segments - 1\}$.
After this, we transform the projected point back onto the scalp with a sequence of affine transformations. Therfore, we scale the unit circle to the defined curl radius and apply the correct rotation around the three main axes.

$$T = T_{\text{scale}}(r) * T_{\text{rotate}}(xr, yr, zr) * T_{\text{translate}}(x, y, z) \quad ,$$

where r is the curl radius, xr, yr, zr the rotation around the x-,y-,z-axis, x, y, z the center of the cylindrical slice around the patch.
The more height and width segments the user specifies, the better is the visual result of this method. Beside the height segments, the width segments play also an important role. The more width segments a patch posesses, the better looks the projection onto the cylindrical surface, because the curled patch is much more smoothly.
For a smooth result, we need a very high number of these height and width segments. Unfortunately, both decreases the time needed for bending. Furthermore, hairstyles with curly hair are displayed very slowly. It is therefore the designer's task to find a good medium between a visual pleasing hairstyle and its rendering time.

Bending

In this subsection, the bending process of a single hair patch is described. First, we give an explanation of the simple bending routine without considering collision detection between hair-hair and hair-body.

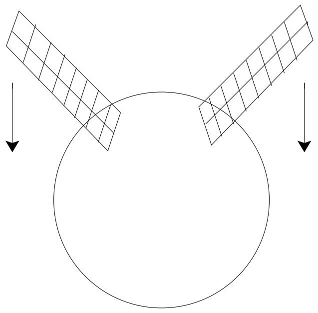

Figure 4: Two generated patches with their patch direction.

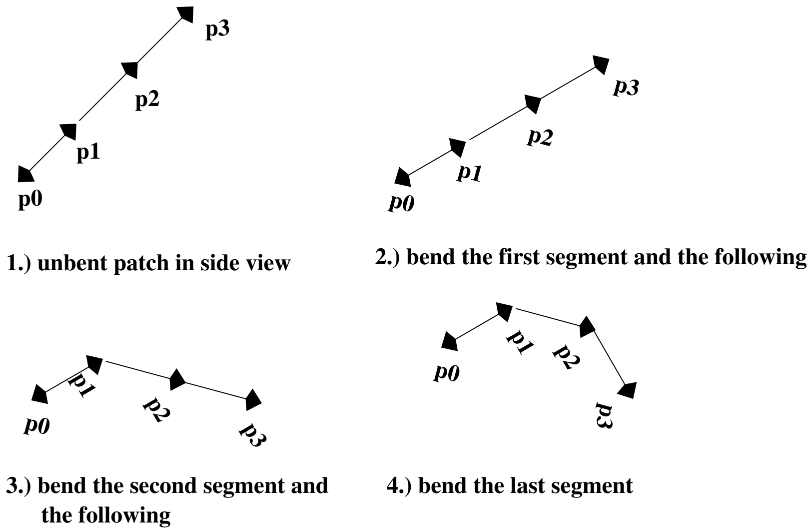

Figure 5: Sideview of a patch with the different states of the bending procedure.

After creating a hair patch on the head's surface, the patch is located orthogonally on the head surface. With the help of this patch position, we know the falling direction of the patch.

For the bending algorithm, width segments do not play a significant role, so in the following, we only consider height segments for simplification.

Our calculation starts with the first height segment. We compute a bending angle depending on several properties. Then, we bend the entire patch beginning with the first segment and the following segments about the computed angle. Next, we compute a new bending angle and bend all height segments step by step. Thus, in the second step, all segments beginning from the second to the last are bent, and so on. This process is depicted in Figure 5.

To clarify the implementation of the proposed bending process, we now explain the single steps of this algorithm. First, we transform the starting point of the current height segment to the origin by using the transformation matrix *T* as illustrated in Figure 6.

T:= Transformation matrix to the origin

Depending on the zenith angle θ and the azimuth angle ϕ, the $bend\_vector$ is rotated onto the positive or the negative z-axis (Figure 7).

B:= Matrix for rotating the bend_vector onto the z-axis

Next, a bending angle β is calculated for the current segment and this segment is rotated around the $bend\_vector$ using the bending direction defined by the starting state (Figure 8).

R:= Rotation matrix around the z-axis

The new vertex locations can now be computed by multiplying the point coordinates by the matrix C.

$$C = T * B * R * B^{-1} * T^{-1}$$

C:= Concatenation of transformation matrices

In the previous specification, we do not take into account collision detection and it is still unclear, how the the bending angle β is computed.
To explain, how to compute the bending angle β, we first look at the algorithm for the hair-head collision detection.
We try to bend our hair patch with the maximal angle. Thus, the patch falls parallel to the gravity vector $g = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$.
This angle α_{max} is the angle between the patch segment and the gravity vector.
We rotate the segments by that angle and test with a number of rays parallel to the modified up_vector. If there is an intersection with the head, we decrease the angle α_{max} and start a new collision test. This procedure is stopped, if there is no collision between the hair patch and the head. So we get the angle β' by that the hair patch lies tangential on top of the scalp. The number of these test rays depends both on the desired precision and the width of the hair patch.
Human hair is a material with physical meaningful properties affecting the bending. For our simple bending model, we use only one of these properties. We introduce a variable for each hair patch, which influences, how strongly the hair is bent.

$$p_{stiffness} \in [0, 100]$$

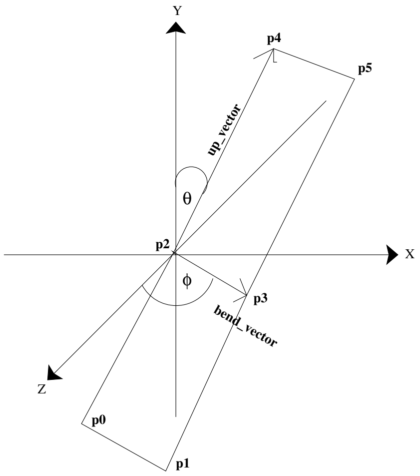

Figure 6: Transform the starting point of the current height segment to the origin.

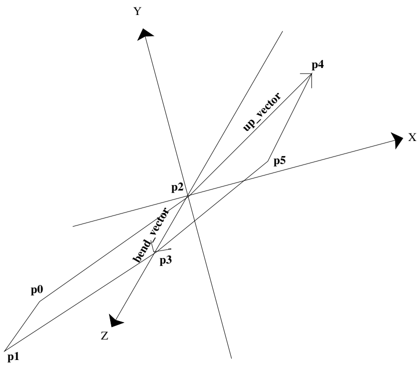

Figure 7: Rotate the bend_vector onto the z-axis.

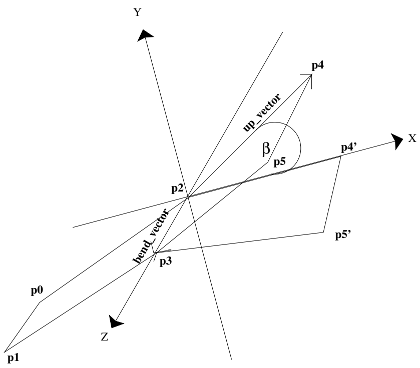

Figure 8: Transform the starting point of the current height segment to the origin.

Figure 9: The left side of the picture shows the bending algorithm without collision detection. The segment intersects the head. On the right side collision detection is used. The patch lies tangential on the head.

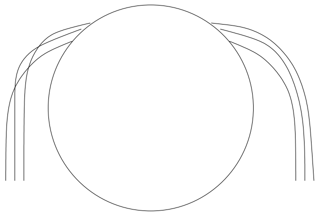

Figure 10: The hair-hair collision detection. On the left side of the picture underlying hair patches penetrate each other. On the right side of the picture hair-hair collision is used.

If we choose $p_{stiffness} = 0$, we have a very flexible hair, which lies directly on top of the scalp. Hair described by a stiffness $p_{stiffness} = 100$ is very rigid and therefore isn't bent at all.
The new bending angle β can now be computed by the following formulae:

$$\beta = (100 - p_{stiffness})/100 * \beta'$$

Our program also has the ability of hair-hair collision detection.
This feature inhibits the penetration of underlying hair patches, which results in more realism (Figure 10).
To do that, we first have to sort the patches according to their y-values. First and foremost, we bend the hair patches with the lowest y-value to this ones with the highest y-value. If there is an intersection with an already bent hair patch, that has a lower y-value, the bending angle is decreased and we start a new collision detection.
The speed of this bending routine depends on the number of collisions as well as the number of segments and the number of patches.

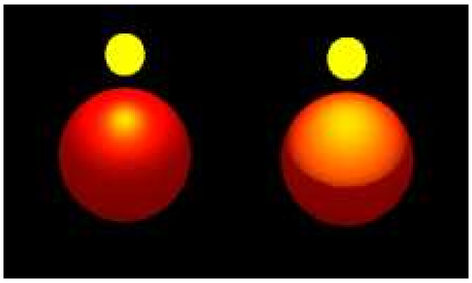

Figure 11: Influence of the shininess: left picture with a small value, right picture with a high value for the shininess

3.2 Rendering Hair Patches

The rendering of the head and also the rendering of hair is done using OpenGL. We use OpenGL *smooth shading*, that corresponds to the *Gouraud Shading* algorithm. In contrast to flat shading, we compute normals for each vertex. This has the advantage, that bent surfaces appear much more smoothly. The disadvantages of this algorithm are the high computational costs, that the computation of all normals requires, and the faulty presentation of the specular reflections caused by a per-vertex shading. But due to the fact that smooth shading improves the visual quality of a hair patch, we do not require a high number of segments per patch. This also accelerates the time consuming bending algorithm enormously.
The user may specify a value for the shininess of the hair, which is one important factor, that contributes to an improved image quality.

$$p_{shininess} \in [0, 128]$$

This value is a measure for the magnitude of the surface which specularly reflects the light.
The maximal surface reflects the light, if we choose $p_{shininess} = 128$, whereas a $p_{shininess} = 0$ means, that there is only a very small surface to reflect as illustrated in Figure 11.
Since we use texture blending with alpha maps for displaying the hair patches, we first have to sort the hair patches with decreasing distance from the viewer. That means that patches, that have the largest distance have to be rendered first in order to receive the correct blending result. The sorting of patches is left to the STL sorting method for lists. In order to use this function, our hair patch class has to provide the comparison operator "$<$" that makes a statement about the position of two patches along the z-axis. Unfortunately, this sorting procedure is not always capable of making a reliable statement about the drawing order of the patches. Refer to Section 7 for more information about this problem. After sorting the patches, the actual rendering takes place. Because texture coordinates are

Figure 12: Hair patches are textured with color and alpha maps. Left: the color map, middle: the alpha map, right: the resultant texture map

Figure 13: Left: A model with untextured 2D patches. Single patches appear rectangular. Right: The same model with alpha mapped texture. After the texturing process, hair patches lose their rectangular shape.

generated at the creation time of the patch, the texturing is quite easy. For texturing, we use a 24bit color map with a photo-realistic hair image and an appropriate grey-scaled alpha map. These two maps together form the actual OpenGL texture. The alpha map defines different degrees of transparency and creates an illusion of complex hair geometry to the otherwise rectangular surface patches (see Figure 13). Alpha mapping is one major factor that contributes to the realism of our hair model.

Another important step in rendering is to disable backface culling, so that patches will be rendered two-sided.

To increase the rendering speed, every single hair patch is stored in its own OpenGL display list. These display lists are created, when the hairstyle is drawn at first time and called every time, if a redraw of the scene is necessary. Fortunately, these dis-

play lists don't need to be updated very often in a normal rendering task, except the geometry of a patch has to be changed. Unfortunately, it doesn't bring any advantage to combine these display lists again in one display list representing the entire hairstyle. Everytime the view of the hair and the face model is changed, this higher-level display list must be recompiled newly. That is because all hair patches have to be sorted according to their distance to the viewer, and therefore the rendering order of the patches changes.
To test the influence of different geometric primitives on the rendering speed, we have implemented two ways of rendering hair patches under OpenGL. The user may choose between rendering the hair patches using quads or triangle strips. The number of used triangle strips for displaying one hair patch corresponds to the number of width segments. As seen in Figure 1, the segments 0 to 4, 5 to 9 and 10 to 15 represent one triangle strip.
It turns out that rendering the patches with triangle strips is about one to four frames per second faster than rendering using quads.

4 The Hair Editor

This section reviews our hair designing tool and shows up how different hairstyles from short to long, from smooth to curly hair can be designed and edited. We give a short overview of the numerous hair parameters and what steps are necessary to create a new hairstyle.
After loading a hairless 3D face model, the actual designing process can start. This process can be split up into four important steps:

- Defining the position of every single hair patch.
- Defining the physical properties of hair.
- Computing the patches with the previous defined properties.
- Reedit the created patches with a number of hair styling tools.

Depending both on the complexity of the desired hairstyle and the user's sense of art, the designing process takes from half an hour up to several hours.

4.1 Positioning of Hair Patches

The first step in our designing chain is the definition of hair patches on top of the human scalp. The location of one hair patch is defined by two points on the head, the so-called *anchor points*. Our hair editor offers two ways of placing these anchor points onto the head, whereby it is advisable to use the automatically generation first for a rough designing of the hairstyle and then to set single patches. This decreases the time needed for the designing process. The further section explains these two possibilities of placing anchor points:

- Placing by clicking onto the head.
- Placing by generating anchor points automatically.

Editing Anchor Points

The user can place anchor points by simply clicking on the head. Two of these points define the position of a patch. After placing an anchor point, the user has the choice of moving or deleting this point again. Every anchor point also has an assigned surface normal, that is important for the later patch generation process. Depending on the favored hairstyle and the desired number of patches, placing every anchor point manually can be a time consuming and boring task. Therefore, our hair editor provides a second and more elegant possibility of placing anchor points: the automatic generation of anchor points.

Automatic Generation of Anchor Points

Instead of placing every single anchor point by hand, the user has the choice of generating these points automatically. For this purpose, the hair editing module provides a function for defining and transforming a 2D plane that consists of four polygons. Each of these four polygons can be selected for calculating intersections between the plane and and the currently loaded face model. Other than that mentioned transforming options, the user can define the number of patches that he wants to generate at the intersection line between head and polygon and additionally the distance between these generated patches.
The algorithm for intersection calculation passes as follows: First of all, we roughly approximate the size of the intersection line between head and the transformed plane. In order to accomplish this, we compute ten points lying on the intersection line and add up the distance between them. After knowing the approximate surface length, we can easily compute the distance between the anchor points that should be placed onto the head surface. After that we send rays that have an appropriate distance from each other beginning from a point lying on the hypotenuse to the opposing vertex. Now every single triangle of the face model will be checked for intersection between itself and the currently generated ray. If there is an intersection, we found a new anchor point. This procedure will be repeated as long as we found enough anchor points for generating the desired number of patches.

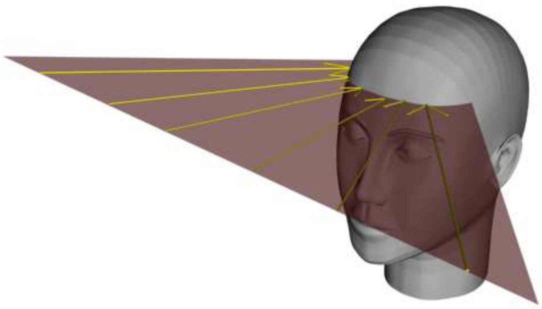

Figure 14: A schematic view of the automatic patch generation. The algorithm sends rays from the rim of the polygon to the center and checks for intersection with the face model. Anchor points are placed at the point of intersection. Two anchor points always define the location of a hair patch.

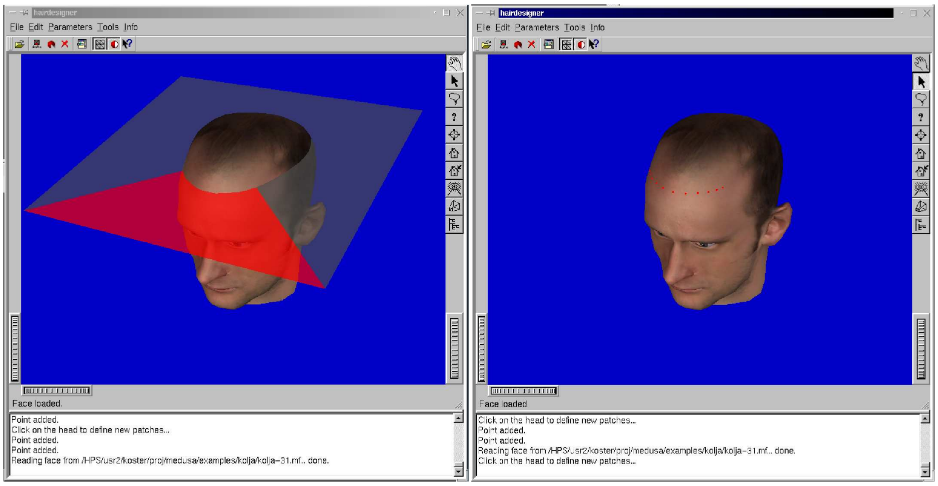

Figure 15: The implementation of the automatic patch generation in the hair editing module. Anchor points are placed at the intersection line (right).

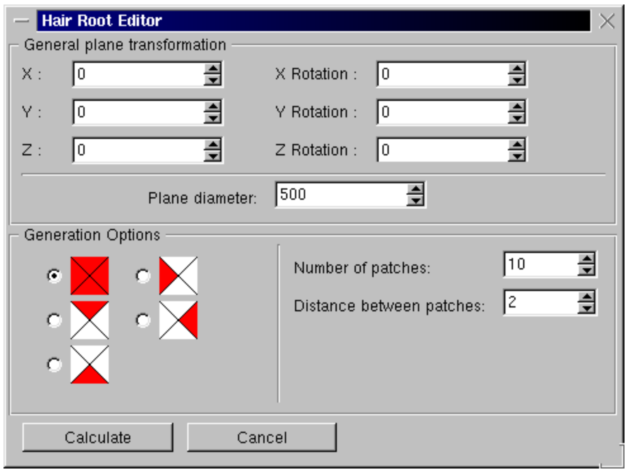

Figure 16: This figure shows the dialog for the automatic generation of anchor points. In the upper part, the user can change the transformation and also the diameter of the plane in object space. Secondarily, he can choose a certain polygon or all four polygons for the intersection calculation with the face model. Furthermore, the designer is expected to enter a desired number of patches that shall be placed at this intersection line.

4.2 Defining the Properties of a Patch

After placing new hair patches on the head, only red anchor points are displayed. To build the hair patches, the user first has to define some properties like the textures or the stiffness already addressed in the sections above. After editing these properties, he can choose, if he wants to change all hair patches by the settings he has just made, or only the last generated hair patches. This feature is a very useful one for creating complex hairstyles, e.g. hairstyles with different sized patches. To change all parameters in a user-friendly fashion, we have designed a parameter dialog shown in Figure 17.
In the following, we will describe these properties and their meaning for the displayed hair patches.

Physical Properties

In this part of the dialog, the user can define the most important physical properties of human hair, which are:

- Length
- Stiffness

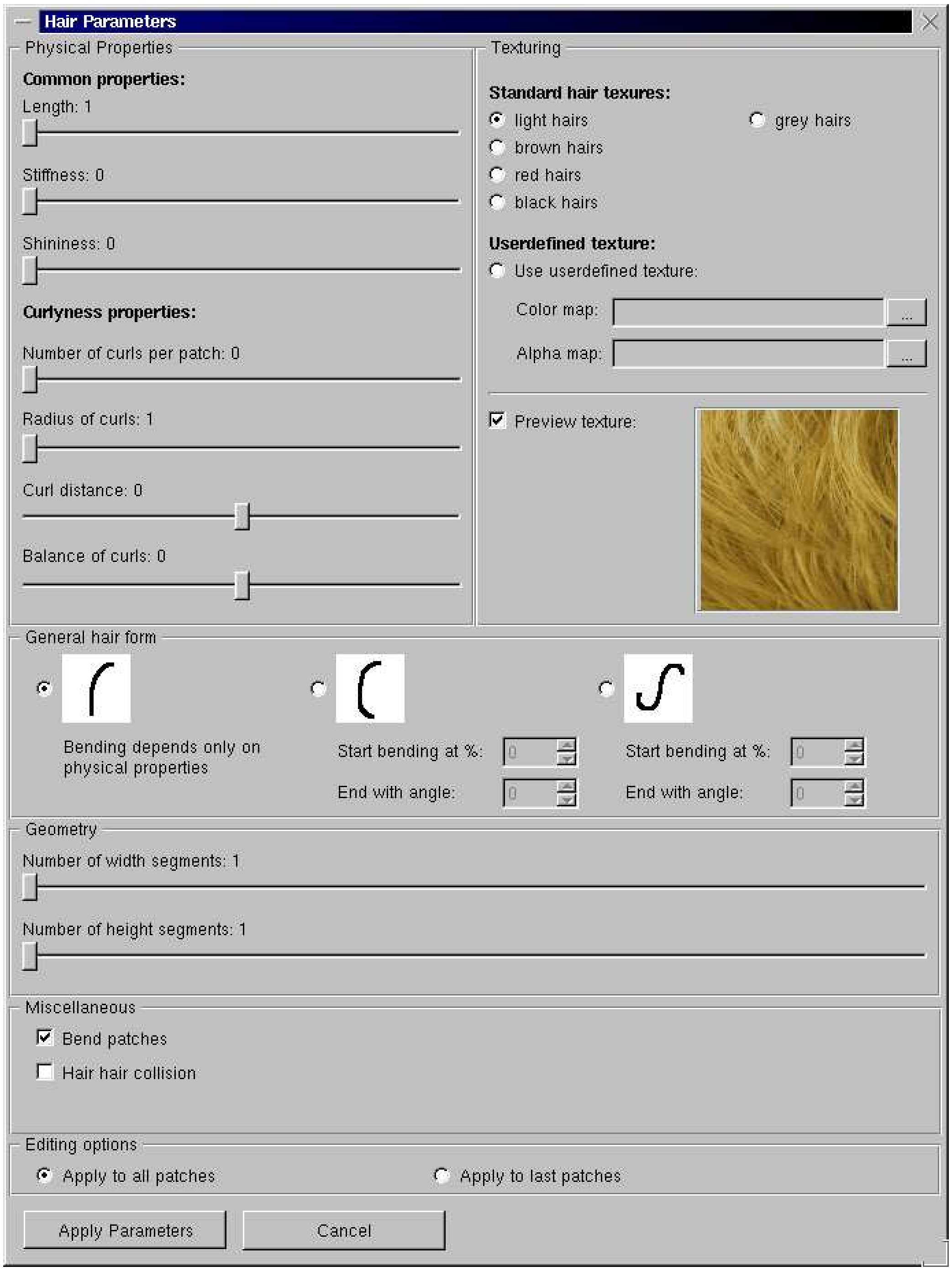

Figure 17: The parameter dialog. These settings influence the resulting hair patch.

- Shininess
- Curlyness Properties

The length is one important factor in order to design different hair dresses. The standard value is 1, but should be changed, because this creates very short hair. A length of 1 corresponds to 10 OpenGL units.

$$p_{length} \in [1, 100]$$
$$preset = 1$$

The stiffness property of a hair patch is described in detail in Section 3.1. Without changing this value, hair patches will lie directly on top of the scalp. In order to obtain a realistic hair dress, it is advisable to try different values for the stiffness.

$$p_{stiffness} \in [0, 100]$$
$$preset = 0$$

Detailed information about the shininess property of a hair patch are given in Section 3.2. The pre-set value for this property is 0, so that each segment reflects the light with the minimal surface. To see larger highlights onto the hair patch, you have to increase this value.

$$p_{shininess} \in [0, 128]$$
$$preset = 0$$

For designing a curly hairstyle, the user can optionally change a value that defines the number of twists that a hair patch shall perform. If the user chooses a value of 0, the curling procedure will not be applied to the single hair patches. At the beginning, curling is disabled.
The next important value, that our curling algorithm needs is the radius of the curls. For a good result of the curling method, it is necessary to choose a radius for the curls that is greater than the width of the patch that has to be curled.
Unfortunately, the next two properties, the user can choose in our parameter dialog, have still no effect on the curling. The basic idea behind the value for the distance between the curls was to push the single twists of the patch closer together to simulate a certain elasticity of the hair.
The property for the balance of the curls should define, where the most part of the curls are situated. A value of 0 means that the twists are equally balanced, a value unequal 0 would have shift the twists either up or down.

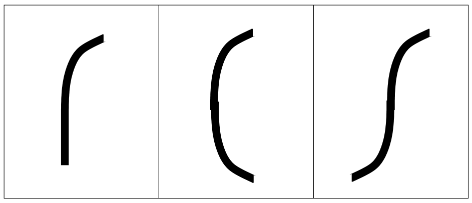

Figure 18: The three general hair forms. left: physical, middle: inward-looking, right: outward-looking

Texturing

In this part you are supposed to set a texture. The colormap of the texture can be displayed via the preview window. Some predefined textures for the most common hair colors can simply be chosen by a checkbox. But it is also allowed to define another texture and its alphamap of the same size. It's advisable to use photo-realistic textures because the degree of realism falls and rises with the quality of the texture.
For texturing we use simple *.ppm files with a power-of-two width and height, because OpenGL only accepts textures of that size.

General Hair Forms

Most modern hairstyles are designed by the help of hair-dryers or hair spray, which results in inward-looking or outward-looking hairstyles. So it is necessary for us to implement different hair forms beside the one, that mirrors the physical behavior of hair.
As you can see in Figure 18, the user may choose between three general hair forms:

- Hair bending depends only on physical properties
- Hair bent inward-looking
- Hair bent outward-looking

If the user chooses the first general hair form, no more settings have to be made. The bending of the hair is automatically computed by our bending algorithm (Section 3.1). In the other two cases, two more settings are necessary:

- Start bending at %
- Bending angle

Figure 19: Three generic hair forms. a) A hairstyle depending only on physical properties. The patches have 10 height segments. b) An inward-looking hairstyle. The patches have 10 height segments, at 60% of the length the segments are bent inwards. The bending angle is 40 degrees. c) An outward-looking hairstyle. The patches have 10 height segments, at 80% of the length the segments are bent outwards. The bending angle is 80 degrees.

By *Start bending at %*, a number between 0 and 100 is expected. This number specifies, when a height segment is bent not using the physical properties, but is bent form-depending subject to the number of height segments of a hair patch.
For example, consider a patch consisting of 10 height segments, then a number of 90 % causes the first 9 segments being bent depending on the physical properties and only the last segment is bent form-depending.
The given *bending angle* is divided by the number of unbent segments of a patch and each of these segments is bent on that part of the angle in the form-depending direction.
Consider a hair patch with 10 segments, bending should start at 80% with a bending angle of 60 degrees. Then the first 8 patches are bent depending only on physical properties and the last 2 segments are bent form-depending. Each of them is bent on an angle of 30 degrees in the given direction.

Geometry

One of the most important properties, the user can select in the parameter dialog, is the setting of the number of width and height segments. These properties decisively determine the quality of the hairstyle.
The more segments the user specifies, the more smoothly appears the patch. In case of uncurled hair, there's no need to increase the number of width segments and also the number of the height segments do not have to be greater than 10 in general. But for curly hairstyles, it is absolutely necessary to set a high number of width and height segments.

It is advisable to use as less segments as possible, because the number of the used segments increases the rendering time of the hairstyle enormously.

Miscellaneous Options

For testing and debugging purposes the user can optionally turn the bending process on and off. Also the collision detection between hair-hair can be disabled. Especially the last feature is important, because there are cases, when hair-hair collision is not desired. If patches are penetrating each other right from the beginning or if other degenerated cases occur, a hair-hair collision leads to an undesired, unnaturally looking hairstyle. In addition, the user may apply the made settings either to the last generated patches or to all yet placed hair patches.

4.3 Hairstyling Tools

After the creation of numerous hair patches with different properties, it is still inevitable to edit single patches for achieving a desirable hairstyle.
Like in an actual hair styling process, many artificial techniques for changing the current hairstyle are required. Professional hairdresser therefore provide numerous tools. We have implemented some of them in our hair designing tool.
These tools provide functions for:

- cutting hair
- combing hair
- creating a parting of the hair

Others like the use of hair sprays are not implemented yet. But hair sprays can be simulated by the aid of the stiffness parameter.

Virtual Scissors

Probably the most famous tool of a hairdresser is the scissor. With the help of this, he is able to shorten human hair. Unfortunately, he is not capable of elongating it, but we are. With our virtual scissor, it is possible to either cut hair or even lengthen it to a desired size. The user can therefore enter the desired length of a hair wisp via a dialog window and by clicking on a patch the length shall be applied to.

Hair Combing Made Easy

When a plane patch passes through the bending process, the falling direction of the patch only depends on the exact position on the scalp. This falling direction can be easily changed with our virtual comb. This tool also enables us to create certain hair parting, but for this purpose, we have implemented another interesting instrument.

Creating a Parting of Hair

Right before the creation process of hair patches, the user can define a parting of the hair with our parting editor (Figure 20). Therefore, he can define a region on the head with two planes. One of these planes is fixed in the middle of the head, marking the point, where normally patches change the falling direction (the blue plane in Figure 20). The other plane is freely relocatable along the x-axis (the green one in Figure 20). By shifting this plane left or right, we define a region on the scalp, where patches shall fall to the opposing direction. By this procedure, a parting of the hair at the position of the relocatable plane is reached. The standard setting of this parting editor doesn't influence the falling direction of the patches at all and should therefore create a center parting of the hair, since now the falling direction only depends on the form of the head.

Other hairstyling tools

Beside the mentioned hairstyling tools described earlier, there are some other tools that make life easier for the creation of a hairstyle. So, the user is able to delete a patch, that has been created with wrong parameters, just by clicking on it. The selection of hair patches for deletion and editing is described more exactly in Section 9. Furthermore, the entire parameters of a patch can be changed. With the help of the hair parameter dialog, the user can change every property of a generated patch. After finishing however, the patch has to pass again through the modeling pipeline. This means, it has to be created, curled and bent newly.

5 Results

In this section, we present the resulting hairstyles created with our hair editor. The procedure of creating realistic 3D hairstyles is a time-consuming task. Furthermore, the designer's art sense and technique are also required. Roughly speaking, we can categorize hairstyles on the basis of two criteria: on the one hand due to their actual style, that means either short or long, smooth or curled, or on the other hand due to their complexity depending on their number of patches and the

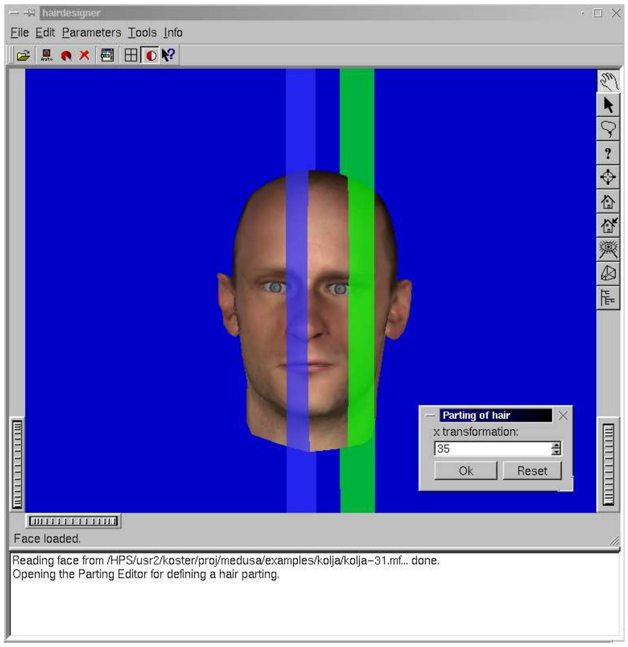

Figure 20: The parting editor enables the user of defining a region in that the falling direction of a patch will be influenced. Thus, we are able to create a parting of the hair.

number of segments. We start our presentation with some basic hairstyles using different number of hair patches and number of segments. After that, we continue with some more complex hairstyles, that have a large number of segments per patch. We will see, that the more complex models, to which curly hairstyles belong to, will consume a great deal of computational capacity. To compare the performance of our hair model, we put some frame rates on different systems together in Section 9.

5.1 Short Hairstyles

Unfortunately, short hairstyles still represent one challenging problem for our hair model. Due to the lack of volumetric properties of our 2D strips, it is very hard to create a feeling of dense and voluminous hair. To accomplish this goal in some degree, a large number of hair patches is needed. Figure 21 and 22 present one short hairstyle, that is similar to a “hedgehog”. This hairstyle looks good from certain points of view. Due to the above mentioned lack of volume, it is possible for the viewer to see the human scalp through the hair patches (Figure 23). For these kinds of hairstyles other hair models seem to be better suited, for example fur models, that try to represent short hair with 3D volume textures. Also hair strand models could be suitable for these styles, but with high computational costs. Other, maybe not so ordinary, but also simple hairstyles like mohawks can also be realized in our hair module as shown in Figure 26 and 27. The complexity of these hairstyles is quite low. Although they use a large number of patches, the patches themselves have low complexity, that means they only have a few number of height and width segments. In the case of the mohawk and the short hairstyle, the hair patch even consists of only one segment.

5.2 Hairstyles with Medium Length

After short hairstyles, there follow hairstyles with a medium length. As a representative of this group, we present a hairstyle with a center parting of the hair (Figure 28 to 30). This hairstyle also has a medium degree of complexity, since it only consists of about 100 and 150 hair patches and only a few segments, that are necessary for the bending process. These kinds of styles are best suited for real-time presentations, because a lot of computational capacity will be saved, that could be invested in other rendering tasks.

5.3 Long Hairstyles

As representatives of the group of long hairstyles, we have chosen two hairstyles with absolutely different complexities. First, we have created a hairstyle with a low number of patches and segments as seen in Figure 31 to 32. This hairdress only needs one width segment and ten height segments that are necessary for a smooth bending process. As a representative of the class of hairstyles having a very high complexity, we have designed a curly hairstyle as presented in Figure 33 to 34. A curly hair cluster, that is represented by one hair patch must have a large number of height segments and at least a number of width segment, that is greater or equal three for a good curling result. The number of height segments depends on the number of twists the patch shall perform. For twists between 5 and 10 a hair patch needs from 40 to 60 height segments. Thus, the curling algorithm described in Section 3.1 guarantees good looking curls with a certain smoothness.

Figure 21: A short hairstyle with low complexity. Although this hairstyle consists of about 370 patches, one patch itself consists only of one segment.

Figure 22: A sideview of the short hairstyle

Figure 23: Depending on the point of view, the user can see the human scalp due to the lack of volume of 2D strips. The contour however seems dense, because there are enough patches overlapping each other.

Figure 24: Another short hairstyle that demonstrates difficulties with the blending of small, directly neighboring hair patches. A decision which patches have to be rendered first is not always possible.

Figure 25: A monk hairstyle.

Figure 26: A mohawk hairstyle with a low degree of complexity.

Figure 27: A mohawk hairstyle with a different texture.

Figure 28: A hairstyle of a center parting of the hair. This figure shows the effects of our collision detection. All hair patches lie properly on the ears.

Figure 29: A center parting of hair with a photo-realistic light hair texture.

Figure 30: The center parting hairstyle with a dark hair texture.

Figure 31: A long hair style. Patches are bent outwards.

Figure 32: A long hair style.

6 Integration into the Medusa System

We have integrated our hair editing tool with the Medusa facial modeling and animation system to combine users' 3D face models with our designed hairstyles. These hairstyles can be used and viewed in all Medusa applications that read out the appropriate Medusa scull file (*.ms) correctly, for example the muscle editor. When the user saves a created hairstyle with our hair editor, a file with the extension **.mhg* (means *Medusa Hair Geometry*) is generated containing all designed hair patches, and the current Medusa scull file is updated.
Therefore, the hair editor creates or updates following entries:

- *hair.texture*: this property defines the used color map for the hair model.
- *hair.alphamap*: this is the used alpha map for texture blending.
- *hair.geometry*: this property contains the path to the geometry file in that all hair geometry is stored. This file is actual a database containing all hair patches generated with our hair editor.

With these three properties an entire hairstyle can be defined in Medusa.
The geometry file itself is stored in plain ASCII and has the following structure: The first line is an identifier for the file and reads "PATCHGEOMETRY". In the

Figure 33: A curly hairstyle with a grey hair texture. This hairstyle has 290 patches and 38370 segments, that means 76740 triangles.

Figure 34: The curly hairstyle from a different point of view.

next line, there is the number of stored patches the file contains. After that, two sections follow for every single patch: one for the hair patch geometry, for instance vertex position, normals, etc., and one for its physical properties. Physical properties are for example the stiffness of a patch and all information needed for the curling process like the radius. Due to the fact that our wisps in our hair representation can have different physical properties for example different stiffness values, it is necessary to store these properties for every single hair patch. The exact file structure is shown in Table 1.

7 Limitations

Wisp models in general have one major disadvantage: the lack of volume. Fortunately, this is irrelevant for long hairstyles. Long hair patches overlapping each other on top of the scalp create an illusion of volume and density. But short hair represent a problem. Because of the size of the patches, there is just a small overlapping between the patches, that is not capable of hiding the human scalp. By a certain orientation of the head to the viewer, the user remarques locations of very thin hair and is able to see straight onto the scalp. For these hairstyles, fur rendering methods or strand models as mentioned in Section 2 seem to be better suited.

Another major problem of our model is the sorting of the patches due to their depth position. As seen in Figure 35 and 36, it is hard to decide, which patches have to be rendered first for receiving a correct blending result. This problem also applies to patches penetrating each other. One solution could be, that every single segment should be tested for their z-position. According to that position, a database of segments with the correct rendering sequence could be created. But this approach also has one major disadvantage. First of all, it doesn't give us a solution for patches penetrating each other and second, we would lose a great deal of rendering speed. Each time the user changes the viewing position of the hairstyle, the database and all OpenGL display lists have to be updated. This would consume an unnecessary amount of time. In our approach, all hair patches have their own display list, that is independent from changing the point of view.

One promising solution to this problem is the use of special extensions of today's 3D graphics cards. NVidia proposes a new approach to this issue, which is called "Order-independent transparency". This technique makes it possible of rendering and blending surfaces correctly, independent from their drawing order. For further information on this technique, NVidia provides special tutorials at `http://developer.nvidia.com`.

PATCHGEOMETRY
int *number_of_patches*
HAIRPATCH
int *number_of_points*
int *number_of_segments*
int *number_of_width_segments*
int *number_of_height_segments*
int *patch_height*
int *patch_width*
Vec3f *patch_direction*
Vec3f *point_1*
Vec2f *texture_coordinate_1*
Vec3f *point_normal_1*
.
Vec3f *point_n*
Vec2f *texture_coordinate_n*
Vec3f *point_normal_n*
int *index_of_P1_in_segment_1*
int *index_of_P2_in_segment_1*
int *index_of_P3_in_segment_1*
int *index_of_P4_in_segment_1*
Vec3f *normal_of_segment_1*
.
int *index_of_P1_in_segment_m*
int *index_of_P2_in_segment_m*
int *index_of_P3_in_segment_m*
int *index_of_P4_in_segment_m*
Vec3f *normal_of_segment_m*
PHYSICS
int *stiffness*
int *curlyness*
int *curlradius*
int *curlbalance*
int *curldistance*
int *patchlength*
int *number_of_width_segments*
int *number_of_height_segments*
int *shininess*
int *hairform*
int *hairform_percent*
int *hairform_angle*
bool *combed*
bool *bend*
bool *haircollision_enabled*
HAIRPATCH
.
PHYSICS
.

Table 1: File structure of hair geometry.

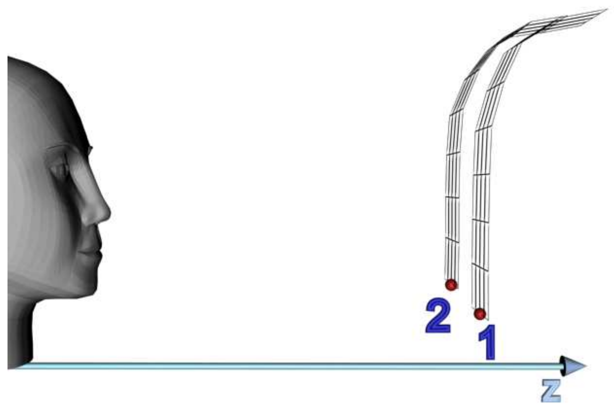

Figure 35: Patches were rendered and blended correctly. First, the farthest and then the nearest patch is rendered. The red point marks the position after that the depth position is decided.

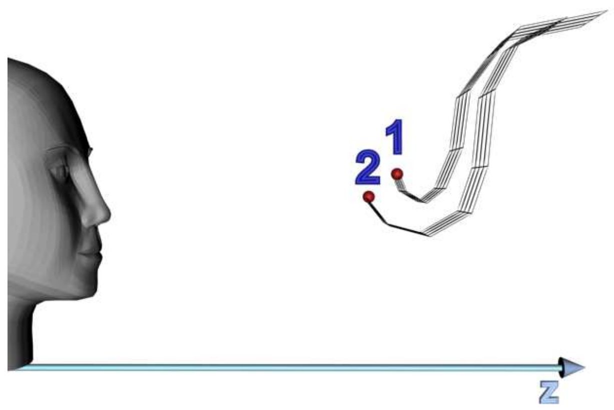

Figure 36: In this case a decision of which patch should be rendered first is impossible, because they partially overlap each other. The “Order-independent transparency” technique as proposed by NVidia could be a solution to this problem.

8 Conclusions

We have presented a fast and efficient method for modeling and rendering human hair, that can be used in real-time applications. One remarkable fact of our hair model compared to brute-force approaches, that try to render every single hair strand, is the rendering speed. This speed can still be increased by using special hardware extensions of state-of-the-art graphics cards. Hair strand models use most of the system resources including processor and memory on drawing hair strands, leaving no computational capacity left for other kinds of facial animation or animation in general. With our model it is possible to integrate a realistic human image into a virtual environment, that again requires resources.
Although we do not provide functions for realistic hair animation, these can be applied in future versions of our hair model. This hair animation again can be applied in real-time applications leaving resources left for other important facial animation tasks. Beside the already implemented features, our hair model can be easily extended by other features, that increase the realism of human hair. Among these features are:

- *Shadow mapping:* The technique of shadow mapping can be used to compute shadows on the face model. Furthermore, the self-shadowing of the hair has also to be taken into account.

- *Bump- and/or Displacement Mapping:* Bump Mapping increases the realism by simulating the lightning situation of single hair strands. By using this technique hair patches will not look like plane surfaces. By applying Displacement Mapping the geometry of these plane surfaces could be actually changed, so the viewer has the impression that he can see single hair strands.

- *Specularity mapping:* Specularity maps would contribute to a more realistic lighting situation and is also an important factor that critically influences the realism of human hair.

Maybe by adding these features, we could reach a degree of realism, that would be similar to strand models, but by spending fewer computational costs.

	Dual 600 MHz Intel PIII, 500 MB RAM, GeForce 256		Dual 1.7 GHz Intel Xeon, 1 GB RAM, GeForce 3	
	Quads	Triangle Strips	Quads	Triangle Strips
center parting hair, 135 patches, 700 segments, 20 segments per patch	23 fps	25 fps	108 fps	112 fps
curly hair, 290 patches, 38370 segments, 132,31 segments per patch	4 fps	5 fps	18 fps	20 fps
long hair, 110 patches, 1900 segments, 17,27 segments per patch	27 fps	29 fps	137 fps	142 fps

Table 2: *Frame rates of different hairstyles on different systems.*

9 Appendix

9.1 Performance

Table 2 presents the rendering speed of three different hairstyles with different complexities on two different computer systems. It shows up, that even on system that are not up-to-date anymore, the rendering speed of hairstyles having medium complexity is about the magical barrier of 25 frames per second. On high-performance systems, the rendering speed lies above 100 frames per second. This leaves enough resources left for other rendering or animation tasks.

9.2 Selecting Single Hair Patches

Instead of using the OpenGL selection buffer for choosing and editing single patches, we decide to make a different approach. Since it is possible to transform mouse coordinates into object coordinates, we transform the 2D mouse coordinates via *gluUnProject* onto the corresponding point on the near clipping plane and next, we transform them onto the opposing far clipping plane. By computing these two points in world space, we have created a 3D vector out of 2D mouse coordinates. Now, we can check every single segment of the hair patches that logically lie between the near and far clipping planes for intersection with our generated line (as shown in Figure 37). Are there more than one intersections

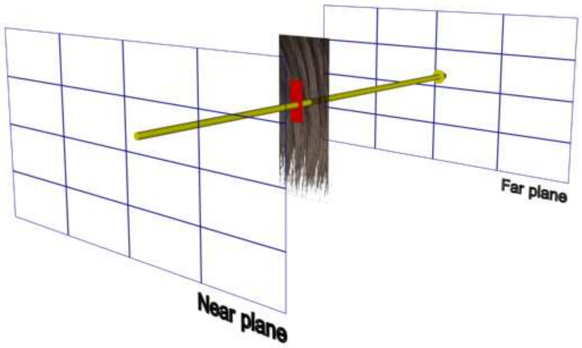

Figure 37: The selection algorithm sends a ray beginning from a point on the near plane to the far plane and checks every hair segment for intersection.

with the ray, the patch with the smallest distance to the viewer will be selected for further editing.

References

[CSDI99] Lieu-Hen Chen, Santi Saeyor, Hiroshi Dohi, and Mitsuru Ishizuka. A System of 3D Hair Style Synthesis based on the Wisp Model. *The Visual Computer*, 15(4):159–170, 1999.

[DMTKT93] Agnes Daldegan, Nadia M. Magnenat-Thalmann, Tsuneya Kurihara, and Daniel Thalmann. An Integrated System for Modeling, Animating and Rendering Hair. In Roger J. Hubbold and Robert Juan, editors, *Computer Graphics Forum (Proceedings of Eurographics '93)*, volume 12, pages 211–221, September 1993.

[Gol97] Dan B. Goldman. Fake fur rendering. *Computer Graphics*, 31(Annual Conference Series):127–134, August 1997.

[GW97] Allen Van Gelder and Jane Wilhelms. An interactive fur modeling technique. In *Graphics Interface '97*, pages 181–188, May 1997.

[HMT01] Sunil Hadap and Nadia Magnenat-Thalmann. Modeling Dynamic Hair as a Continuum. In *Computer Graphics Forum (Proceed-*

ings Eurographics 2001), volume 20, pages C329–C338, September 2001.

[KH01] Chuan Koon Koh and Zhiyong Huang. A Simple Physics Model to Animate Human Hair Modeled in 2D Strips in Real Time. In *Proceedings Computer Animation and Simulation (CAS 2001)*, pages 127–138, September 2001.

[KK89] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):271–280, July 1989.

[KN02] Tae-Yong Kim and Ulrich Neumann. Interactive multiresolution hair modeling and editing. *ACM Transactions on Graphics*, 21(3):620–629, July 2002.

[KYT92] Ken-ichi Anjyo, Yoshiaki Usami, and Tsuneya Kurihara. A Simple Method for Extracting the Natural Beauty of Hair. In *Computer Graphics (SIGGRAPH '92 Conference Proceedings)*, volume 26, pages 111–120, July 1992.

[Len00] Jerome Edward Lengyel. *Real-Time Fur*. Microsoft Research, 2000.

[LPFH01] Jerome E. Lengyel, Emil Praun, Adam Finkelstein, and Hugues Hoppe. Real-time fur over arbitrary surfaces. In *2001 ACM Symposium on Interactive 3D Graphics*, pages 227–232, March 2001.

[Ney98] Fabrice Neyret. Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, January 1998.

I N F O R M A T I K

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-2004-1-003	I. Katriel	On Algorithms for Online Topological Ordering and Sorting
MPI-I-2004-1-002	P. Sanders, S. Pettie	A Simpler Linear Time 2/3 - epsilon Approximation for Maximum Weight Matching
MPI-I-2004-1-001	N. Beldiceanu, I. Katriel, S. Thiel	Filtering algorithms for the Same and UsedBy constraints
MPI-I-2003-NWG2-002	F. Eisenbrand	Fast integer programming in fixed dimension
MPI-I-2003-NWG2-001	L.S. Chandran, C.R. Subramanian	Girth and Treewidth
MPI-I-2003-4-009	N. Zakaria	FaceSketch: An Interface for Sketching and Coloring Cartoon Faces
MPI-I-2003-4-008	C. Roessl, I. Ivrissimtzis, H. Seidel	Tree-based triangle mesh connectivity encoding
MPI-I-2003-4-007	I. Ivrissimtzis, W. Jeong, H. Seidel	Neural Meshes: Statistical Learning Methods in Surface Reconstruction
MPI-I-2003-4-006	C. Roessl, F. Zeilfelder, G. Nrnberger, H. Seidel	Visualization of Volume Data with Quadratic Super Splines
MPI-I-2003-4-005	T. Hangelbroek, G. Nrnberger, C. Roessl, H.S. Seidel, F. Zeilfelder	The Dimension of C^1 Splines of Arbitrary Degree on a Tetrahedral Partition
MPI-I-2003-4-004	P. Bekaert, P. Slusallek, R. Cools, V. Havran, H. Seidel	A custom designed density estimation method for light transport
MPI-I-2003-4-003	R. Zayer, C. Roessl, H. Seidel	Convex Boundary Angle Based Flattening
MPI-I-2003-4-002	C. Theobalt, M. Li, M. Magnor, H. Seidel	A Flexible and Versatile Studio for Synchronized Multi-view Video Recording
MPI-I-2003-4-001	M. Tarini, H.P.A. Lensch, M. Goesele, H. Seidel	3D Acquisition of Mirroring Objects
MPI-I-2003-2-004	A. Podelski, A. Rybalchenko	Software Model Checking of Liveness Properties via Transition Invariants
MPI-I-2003-2-003	Y. Kazakov, H. Nivelle	Subsumption of concepts in $DL\ \mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete
MPI-I-2003-2-002	M. Jaeger	A Representation Theorem and Applications to Measure Selection and Noninformative Priors
MPI-I-2003-2-001	P. Maier	Compositional Circular Assume-Guarantee Rules Cannot Be Sound And Complete
MPI-I-2003-1-018	G. Schaefer	A Note on the Smoothed Complexity of the Single-Source Shortest Path Problem
MPI-I-2003-1-017	G. Schfer, S. Leonardi	Cross-Monotonic Cost Sharing Methods for Connected Facility Location Games

MPI-I-2003-1-016	G. Schfer, N. Sivadasan	Topology Matters: Smoothed Competitive Analysis of Metrical Task Systems
MPI-I-2003-1-015	A. Kovcs	Sum-Multicoloring on Paths
MPI-I-2003-1-014	G. Schfer, L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, T. Vredeveld	Average Case and Smoothed Competitive Analysis of the Multi-Level Feedback Algorithm
MPI-I-2003-1-013	I. Katriel, S. Thiel	Fast Bound Consistency for the Global Cardinality Constraint
MPI-I-2003-1-012		- not published -
MPI-I-2003-1-011	P. Krysta, A. Czumaj, B. Voecking	Selfish Traffic Allocation for Server Farms
MPI-I-2003-1-010	H. Tamaki	A linear time heuristic for the branch-decomposition of planar graphs
MPI-I-2003-1-009	B. Csaba	On the Bollobás – Eldridge conjecture for bipartite graphs
MPI-I-2003-1-008	P. Sanders	Polynomial Time Algorithms for Network Information Flow
MPI-I-2003-1-007	H. Tamaki	Alternating cycles contribution: a strategy of tour-merging for the traveling salesman problem
MPI-I-2003-1-006	M. Dietzfelbinger, H. Tamaki	On the probability of rendezvous in graphs
MPI-I-2003-1-005	M. Dietzfelbinger, P. Woelfel	Almost Random Graphs with Simple Hash Functions
MPI-I-2003-1-004	E. Althaus, T. Polzin, S.V. Daneshmand	Improving Linear Programming Approaches for the Steiner Tree Problem
MPI-I-2003-1-003	R. Beier, B. Vcking	Random Knapsack in Expected Polynomial Time
MPI-I-2003-1-002	P. Krysta, P. Sanders, B. Vcking	Scheduling and Traffic Allocation for Tasks with Bounded Splittability
MPI-I-2003-1-001	P. Sanders, R. Dementiev	Asynchronous Parallel Disk Sorting
MPI-I-2002-4-002	F. Drago, W. Martens, K. Myszkowski, H. Seidel	Perceptual Evaluation of Tone Mapping Operators with Regard to Similarity and Preference
MPI-I-2002-4-001	M. Goesele, J. Kautz, J. Lang, H.P.A. Lensch, H. Seidel	Tutorial Notes ACM SM 02 A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models
MPI-I-2002-2-008	W. Charatonik, J. Talbot	Atomic Set Constraints with Projection
MPI-I-2002-2-007	W. Charatonik, H. Ganzinger	Symposium on the Effectiveness of Logic in Computer Science in Honour of Moshe Vardi
MPI-I-2002-1-008	P. Sanders, J.L. Trff	The Factor Algorithm for All-to-all Communication on Clusters of SMP Nodes
MPI-I-2002-1-005	M. Hoefer	Performance of heuristic and approximation algorithms for the uncapacitated facility location problem
MPI-I-2002-1-004	S. Hert, T. Polzin, L. Kettner, G. Schfer	Exp Lab A Tool Set for Computational Experiments
MPI-I-2002-1-003	I. Katriel, P. Sanders, J.L. Trff	A Practical Minimum Scanning Tree Algorithm Using the Cycle Property
MPI-I-2002-1-002	F. Grandoni	Incrementally maintaining the number of l-cliques
MPI-I-2002-1-001	T. Polzin, S. Vahdati	Using (sub)graphs of small width for solving the Steiner problem
MPI-I-2001-4-005	H.P.A. Lensch, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models
MPI-I-2001-4-004	S.W. Choi, H. Seidel	Linear One-sided Stability of MAT for Weakly Injective Domain
MPI-I-2001-4-003	K. Daubert, W. Heidrich, J. Kautz, J. Dischler, H. Seidel	Efficient Light Transport Using Precomputed Visibility
MPI-I-2001-4-002	H.P.A. Lensch, J. Kautz, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing, Transmission, and Interactive Display of High Quality 3D Models on the Web
MPI-I-2001-4-001	H.P.A. Lensch, J. Kautz, M. Goesele, W. Heidrich, H. Seidel	Image-Based Reconstruction of Spatially Varying Materials
MPI-I-2001-2-006	H. Nivelle, S. Schulz	Proceeding of the Second International Workshop of the Implementation of Logics
MPI-I-2001-2-005	V. Sofronie-Stokkermans	Resolution-based decision procedures for the universal theory of some classes of distributive lattices with operators
MPI-I-2001-2-004	H. de Nivelle	Translation of Resolution Proofs into Higher Order Natural Deduction using Type Theory