

An Investigation of Hair Modelling and Rendering Techniques with Emphasis on African Hairstyles



Submitted in fulfilment of the requirements
for the degree of Master of Science
of Rhodes University

by
Deborah Michelle Patrick
February 2004

Abstract

Many computer graphics applications make use of virtual humans. Methods for modelling and rendering hair are needed so that hairstyles can be added to the virtual humans. Modelling and rendering hair is challenging due to the large number of hair strands and their geometric properties, the complex lighting effects that occur among the strands of hair, and the complexity and large variation of human hairstyles. While methods have been developed for generating hair, no methods exist for generating African hair, which differs from hair of other ethnic groups.

This thesis presents methods for modelling and rendering African hair. Existing hair modelling and rendering techniques are investigated, and the knowledge gained from the investigation is used to develop or enhance hair modelling and rendering techniques to produce three different forms of hair commonly found in African hairstyles. The different forms of hair identified are natural curly hair, straightened hair, and braids or twists of hair.

The hair modelling techniques developed are implemented as plug-ins for the graphics program LightWave 3D. The plug-ins developed not only model the three identified forms of hair, but also add the modelled hair to a model of a head, and can be used to create a variety of African hairstyles. The plug-ins significantly reduce the time spent on hair modelling. Tests performed show that increasing the number of polygons used to model hair increases the quality of the hair produced, but also increases the rendering time. However, there is usually an upper bound to the number of polygons needed to produce a reasonable hairstyle, making it feasible to add African hairstyles to virtual humans.

The rendering aspects investigated include hair illumination, texturing, shadowing and anti-aliasing. An anisotropic illumination model is developed that considers the properties of African hair, including the colouring, opacity and narrow width of the hair strands. Texturing is used in several instances to create the effect of individual strands of hair. Results show that texturing is useful for representing many hair strands because the density of the hair in a texture map does not have an effect on the rendering time. The importance of including a shadowing technique and applying an anti-aliasing method when rendering hair is demonstrated. The rendering techniques are implemented using the RenderMan Interface and Shading Language.

A number of complete African hairstyles are shown, demonstrating that the techniques can be used to model and render African hair successfully.

Acknowledgements

I would like to thank my supervisor, Prof. Shaun Bangay, for his valuable help and guidance over the duration of this research. Furthermore, I would like to thank Prof. George Wells for his support and assistance during the writing up of this thesis.

Many thanks also go to Adele Lobb, Madeleine Wright, Jason Penton and Matthew Gibb for proofreading the drafts of this thesis, to the members of the Rhodes University Computer Science department for their support, to the people who allowed me to use photographs of their hairstyles, and to the National Research Foundation (NRF) for providing funding.

Finally, I would like to thank my family and friends for their support and encouragement.

Many trademarks are mentioned throughout this thesis. All trademarks are the property of their respective owners.

Table of Contents

1 INTRODUCTION.....	9
1.1 Problem Statement.....	9
1.1.1 Hair Modelling.....	10
1.1.2 Hair Rendering.....	11
1.2 Thesis Structure	11
2 RELATED WORK.....	12
2.1 Background	12
2.1.1 The State of Hair Simulation Research	12
2.1.2 Computer Generated Hair in Films.....	12
2.1.3 Hair Simulation Tools	14
2.2 Hair Modelling	15
2.2.1 Introduction.....	15
2.2.2 Implicit Models	16
2.2.3 Explicit Models	19
2.2.4 Cluster Models	24
2.2.5 Interactive Hair Modelling Systems	28
2.3 Hair Rendering	30
2.3.1 Introduction.....	30
2.3.2 Illumination Models.....	31
2.3.3 Shadowing	38
2.3.4 Texturing	43
2.3.5 Anti-aliasing.....	48
2.4 Summary	52
3 MODELLING AFRICAN HAIR.....	54
3.1 Natural Curly Hair.....	54
3.1.1 Concentric Textured Layers.....	55
3.1.2 Placing Layers on a Head Model.....	56
3.1.3 Using Opacity Maps to Define Hair.....	57
3.2 Straight Hair	58
3.2.1 Using Curves to Define a Hairstyle.....	58
3.2.2 Placing Curves on a Head Model.....	59
3.2.3 Representing a Strand of Hair	60
3.3 Braids and Twists of Hair.....	61
3.3.1 Using Curves to Define Braids and Twists in a Hairstyle	62
3.3.2 Placing Curves on a Head Model.....	62
3.3.3 Creating Braids and Twists with Generalized Cylinders.....	63
3.3.4 Representing Strands of Hair.....	70
3.3.5 Additional Texturing.....	70
3.4 Summary	70
4 IMPLEMENTING PLUG-INS.....	72
4.1 LightWave 3D	72
4.2 Hair Modelling Plug-ins.....	73
4.2.1 HPLayer	73
4.2.2 HPCurve.....	77
4.2.3 HPDuplicate.....	81
4.2.4 HPTwist.....	83
4.2.5 HPBraid.....	88
4.2.6 HPPolyline	93
4.2.7 HPGeneralizedCylinder	97
4.3 Summary	103
5 RENDERING AFRICAN HAIR	104
5.1 Introduction	104

5.1.1 Setting Up the Scene	104
5.1.2 RenderMan and LightMan	105
5.2 Illumination Model	107
5.2.1 Introduction	107
5.2.2 Our Illumination Model	109
5.2.3 Implementing the Illumination Model as a Shader	112
5.2.4 Implementing the Illumination Model in Textured Layers	113
5.3 Shadowing	113
5.3.1 Adding Shadows to Explicitly Modelled Hair	114
5.3.2 Self-shadowing in Layers	116
5.4 Texturing	117
5.4.1 Texturing Layers to Create the Effect of Hair	117
5.4.2 Representing Individual Hair Strands in Braids and Twists	119
5.4.3 Adding Additional Texture to the Head	122
5.5 Anti-aliasing	125
5.5.1 Introduction	125
5.5.2 Supersampling with RenderMan	125
5.5.3 Using Textures	125
5.6 Summary	126
6 DISCUSSIONS AND RESULTS	127
6.1 Introduction	127
6.2 Modelling Results	128
6.2.1 Modelling Natural Curly Hair	128
6.2.2 Modelling Straight Hair	131
6.2.3 Modelling Hairstyles with Braids and Twists of Hair	139
6.3 Rendering Results	146
6.3.1 Illumination Model	146
6.3.2 Shadowing	152
6.3.3 Texturing	154
6.3.4 Antialiasing	161
6.4 Final Results	164
6.5 Summary	166
7 CONCLUSION	169
7.1 Thesis Overview	170
7.2 Future Work	174
7.3 Thesis Contributions	175
8 REFERENCES	176
APPENDIX A - TEXTURED LAYERS HAIR SHADER	186
APPENDIX B - DIFFERENT FILTER TYPES	187
APPENDIX C - INTERPOLATION	188
C.1 Determining the Control Hairs to Interpolate Between	188
C.2 Interpolating a Hair Between Two Control Hairs	189

List of Figures

Figure 1.1: Three forms of hair commonly found in African hairstyles.	9
Figure 2.1: Baby Molly, Sid's sister, and Aki.	13
Figure 2.2: Sully with key hairs defined and Sully rendered.	13
Figure 2.3: Different ways to represent a strand of hair modelled explicitly.	21
Figure 2.4: Placing strands of hair onto a surface.	23
Figure 2.5: Calculating a section of a strand of hair.	25
Figure 2.6: A cluster.	26
Figure 2.7: Kajiya and Kay's illumination model.	32
Figure 2.8: Anjyo et al.'s illumination model.	35
Figure 2.9: Obtaining normals.	37
Figure 2.10: Hair rendered with and without self-shadowing.	38
Figure 2.11: Percentage closer filtering.	39
Figure 2.12: The visibility function for a pixel.	40
Figure 2.13: Obtaining a set of opacity maps.	41
Figure 2.14: Modelling and rendering hair using textured strips.	45
Figure 2.15: Texture maps.	46
Figure 2.16: Texturing concentric layers.	47
Figure 2.17: Texture maps.	47
Figure 2.18: Aliasing and anti-aliasing.	49
Figure 2.19: Pixels considered as square boxes.	51
Figure 3.1: Examples of African hairstyles with natural curly hair.	54
Figure 3.2: The result of an opacity map applied to concentric layers.	55
Figure 3.3: Hair represented by textured layers.	56
Figure 3.4: Placing layers over selected polygons.	57
Figure 3.5: Using opacity maps to define the shape, thickness and distribution of hair strands.	57
Figure 3.6: Examples of African hairstyles with straight (relaxed) hair.	58
Figure 3.7: Automatically creating curves using a control curve.	59
Figure 3.8: African hairstyles with hair in the form of a braid and twists.	61
Figure 3.9: Braided (twisted) hair in a variety of styles.	63
Figure 3.10: Cross section of a twist.	64
Figure 3.11: The shape of a curve in a twist along the z-axis.	64
Figure 3.12: A curve in a twist viewed from different angles.	64
Figure 3.13: Different starting values of θ for each curve.	65
Figure 3.14: Cross section of a braid.	66
Figure 3.15: The shape of a curve in a braid along the z-axis.	66
Figure 3.16: Different flatness.	66
Figure 3.17: Cross sections of a cylinder placed along a curve.	67
Figure 3.18: A reference frame.	67
Figure 3.19: Calculating a reference frame.	68
Figure 3.20: A generalized cylinder with and without twisting.	68
Figure 3.21: Aligning a cross section perpendicular to a curve.	69
Figure 4.1: Creating layers, using polygon normals for displacement.	74
Figure 4.2: Creating a layer, using vertex normals for displacement.	74
Figure 4.3: The HPLayer plug-in's interface.	76
Figure 4.4: A message assisting the user.	76
Figure 4.5: Using the HPLayer plug-in to create and place layers on a head model.	76
Figure 4.6: A control hair consisting of four segments and five control points.	78
Figure 4.7: Placing a control hair in the centre of a selected triangle.	78
Figure 4.8: A message assisting the user.	79

Figure 4.9: The HPCurve plug-in's interface.....	80
Figure 4.10: A message assisting the user.....	80
Figure 4.11: Using the HPCurve plug-in to design a hairstyle.....	80
Figure 4.12: The HPDuplicate plug-in's interface.....	82
Figure 4.13: Using the HPDuplicate plug-in.....	83
Figure 4.14: The HPTwist plug-in's interface.....	85
Figure 4.15: A message assisting the user.....	85
Figure 4.16: Different twist values.....	86
Figure 4.17: Twists with the same curves, but different numbers of control points.....	87
Figure 4.18: Converting a curve into a twist.....	88
Figure 4.19: Using the HPTwist plug-in.....	88
Figure 4.20: Viewing a braid from different angles.....	89
Figure 4.21: The HPBraid plug-in's interface.....	90
Figure 4.22: A message assisting the user.....	90
Figure 4.23: Different braid values.....	91
Figure 4.24: Converting a curve into a braid.....	92
Figure 4.25: Using the HPBraid plug-in.....	93
Figure 4.26: Converting a curve defined by control points into a polyline.....	94
Figure 4.27: Using FreezeCurves to convert an open curve into a polygon.....	94
Figure 4.28: The HPPolyline plug-in's interface.....	95
Figure 4.29: Converting curves to polylines so that they can be rendered.....	96
Figure 4.30: Using the HPPolyline plug-in.....	97
Figure 4.31: Creating a generalized cylinder.....	98
Figure 4.32: The HPGeneralizedCylinder plug-in's interface.....	99
Figure 4.33: Changing the properties of a generalized cylinder.....	100
Figure 4.34: Converting a curve into generalized cylinders.....	101
Figure 4.35: Using the HPGeneralizedCylinder plug-in for braids and twists.....	101
Figure 4.36: Using the HPGeneralizedCylinder plug-in.....	102
Figure 5.1: Three point lighting set-up.....	104
Figure 5.2: The effect of the Key, Fill, and Back lights, and the final result.....	105
Figure 5.3: Using a conventional illumination model.....	107
Figure 5.4: Important vectors in a conventional illumination model.....	108
Figure 5.5: Important unit vectors in our illumination model.....	109
Figure 5.6: All strands of hair have the same colour.....	111
Figure 5.7: Rendering hair without shadows and with traditional shadow maps.....	115
Figure 5.8: Rendering hair without self-shadowing and with self-shadowing.....	116
Figure 5.9: An opacity map and a tangent map.....	118
Figure 5.10: Applying textures to concentric layers.....	118
Figure 5.11: Mapping a texture onto a generalized cylinder.....	120
Figure 5.12: Cylinders with and without the seam problem.....	121
Figure 5.13: The texture and U values.....	121
Figure 5.14: Texturing generalized cylinders.....	122
Figure 5.15: Tessellations.....	123
Figure 5.16: A toroidal texture.....	124
Figure 5.17: Adding an additional texture onto the scalp.....	124
Figure 5.18: Adding additional textured layers onto the scalp.....	124
Figure 6.1: Two hairstyles modelled with concentric layers.....	128
Figure 6.2: Improving the shape of a natural curly hairstyle.....	129
Figure 6.3: Rendering times when using different numbers of layers.....	129
Figure 6.4: Hair represented by different numbers of layers.....	130
Figure 6.5: Hair represented by different numbers of layers with the same height.....	131

Figure 6.6: Placing layers too far apart.	131
Figure 6.7: Two explicitly modelled hairstyles.	132
Figure 6.8: Fixing and Removing Stray Hairs.	132
Figure 6.9: Rendering times when using different numbers of curves.....	134
Figure 6.10: The quality of hair produced with different numbers of curves.....	135
Figure 6.11: Rendering times when using different numbers of segments.	137
Figure 6.12: The quality of hair produced with different numbers of segments.	138
Figure 6.13: Hairstyles containing braids and twists.....	139
Figure 6.14: Braids and twists with different numbers of clusters.	140
Figure 6.15: Rendering times when using different numbers of clusters.....	141
Figure 6.16: Overlapping clusters.....	141
Figure 6.17: Rendering times when using different numbers of segments.	142
Figure 6.18: Braids represented by generalized cylinders created with different numbers of segments.	143
Figure 6.19: Rendering times when using different numbers of sides.....	144
Figure 6.20: Twists represented by generalized cylinders that have different numbers of sides.	145
Figure 6.21: Black and brown hair with and without the diffuse component.....	146
Figure 6.22: Using different values for the excess brightness term.	147
Figure 6.23: Using different coloured lights.	148
Figure 6.24: Including the surface colour in the equation for the specular component.	149
Figure 6.25: Using different light reflection and light transmission values.....	150
Figure 6.26: Illuminating hair represented by textured layers.	151
Figure 6.27: Rendering times when using different shadow map resolutions.	152
Figure 6.28: Using different shadow map resolutions.....	153
Figure 6.29: Creating different hair types.	154
Figure 6.30: Rendering times for different hair densities represented with textures.	155
Figure 6.31: Different hair densities.	156
Figure 6.32: Reusing a texture to create the effect of hair.	156
Figure 6.33: Texturing generalized cylinders.	157
Figure 6.34: Concentric generalized cylinders.	158
Figure 6.35: Applying a bump map.....	158
Figure 6.36: Different coloured braids.....	159
Figure 6.37: Colour map.....	160
Figure 6.38: Unwanted artefacts at the edges.	160
Figure 6.39: Supersampling at different sampling rates.....	162
Figure 6.40: Rendering times when using different sampling rates.	163
Figure 6.41: Rendering times when using different filters.....	164
Figure 6.42: Natural curly hair.....	164
Figure 6.43: Straightened hair.....	165
Figure 6.44: Braids of hair.	165
Figure 6.45: Twists of hair.	166
Figure B.1: Using different filter types to render an image.	187
Figure C.1: Calculating the distances between a random point and the three sides of a triangle.	188
Figure C.2: An unrealistic interpolated strand of hair.....	190

1 INTRODUCTION

1.1 Problem Statement

Methods for modelling and rendering hair using computer graphics are needed so that hairstyles can be added to virtual humans. Applications that require virtual humans with hair include virtual worlds, computer games, films and advertisements. Several methods have been developed for generating hair, and although some methods produce very convincing results, most methods only work well for particular types of hair under specific conditions. The use of these methods can be seen in recent films such as *Monsters, Inc.* [Disney 2001], in computer games such as *Final Fantasy X* [Square 2001], and on avatars such as Ananova [Ananova 2003].

As of yet, no methods exist for generating African hair specifically. African hair differs from hair of other ethnic groups. It is curly or kinky, contains melanin that causes it to have a dark colouring, and can be opaque due to large, dense pigment granules [Deedrick 2000]. Due to the difference in hair type and cultural traditions, African hairstyles are different from hairstyles of other ethnic groups. They often contain patterns with braids and twists of hair [Klein 2002].

The goal of this thesis is to present methods for simulating African hair, so that a variety of African hairstyles can be added to virtual humans. To achieve this goal we develop or enhance hair modelling and rendering techniques to produce different forms of hair commonly found in African hairstyles. Our hair modelling techniques are implemented as plug-ins for the graphics program LightWave 3D. Our hair rendering techniques are implemented as shaders compatible with the RenderMan Interface.

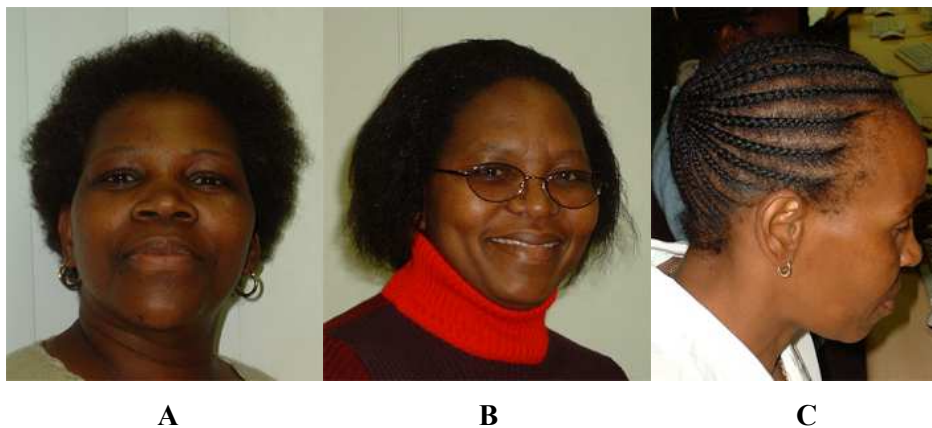


Figure 1.1: Three forms of hair commonly found in African hairstyles.

We identify three forms of hair commonly found in African hairstyles. Figure 1.1 shows examples of African hairstyles containing the three different forms of hair, namely natural curly hair (A), straightened hair (B), and braids or twists of hair (C). We aim to develop modelling and rendering techniques that produce these three forms of hair that may then be used by an artist to create an African hairstyle.

In order to simulate hair three aspects need to be considered. These aspects are hair modelling, hair rendering and hair dynamics. Hair modelling deals with the specification of the hair's geometry, and the density, distribution, orientation and shape of the individual hair strands. Hair rendering involves lighting and shading hair, and is responsible for anti-aliasing techniques. Hair dynamics looks at hair movement and collision detection [Magenat-Thalmann et al. 2000]. Often these three aspects are related to one another. In this thesis we investigate hair modelling and hair rendering techniques. Techniques for hair dynamics are beyond the scope of the investigation. In many African hairstyles, the hair does not move much with respect to the head.

1.1.1 Hair Modelling

Hair modelling is challenging due to the large number of hair strands, their geometric properties, and the complexity and large variation of human hairstyles. The average African scalp contains between 100,000 and 150,000 extremely thin (between 0.05mm and 0.09mm) and very curly strands of hair, which are often straightened (using a process called *relaxing*) or styled into braids or twists [LeBlanc et al. 1991]. Extensions (fake hair usually in the form of braids) can also be added to the hair. Considering the shape of each individual strand of hair when modelling hair is tedious and impractical. With these considerations in mind, our hair modelling techniques for the three different forms of hair must achieve several goals:

- They must be able to add or place the hair onto a model of a head.
- They must be able to represent the shape and geometric properties of the form of hair they are modelling.
- They must overcome the problem of modelling a large number of hair strands.
- They must allow a user to model a variety of hairstyles with ease.

1.1.2 Hair Rendering

Hair rendering is challenging and computationally expensive due to the large number of hair strands, the thin geometric shape of a strand of hair, and the complex lighting effects that occur among the strands of hair. Due to the geometry and orientation of individual strands of hair, anisotropic specular highlights appear in the hair. These highlights form definite patterns across the hair, even when individual strands of hair cannot be seen. The strands of hair and the head cast shadows on each other. The colour of hair differs from person to person, and may also differ from strand to strand, or even within a strand. If the hair has a light colour (which is not the case for African hair) a haloing effect is produced under backlighting conditions due to the hair being partially transparent. In addition, because the width of a strand of hair is very small in comparison to the rendered image, several strands of hair often contribute to the shading of a single pixel, which results in a serious aliasing problem. In order to render the African hair produced with our hair modelling techniques, several rendering aspects must be considered:

- An appropriate anisotropic illumination model must be used.
- Shadowing must be implemented.
- An anti-aliasing technique must be applied.

Instead of modelling individual strands of hair, it is possible to use a texture that represents strands of hair and produces the effect of hair when the image is rendered. Texturing is a useful rendering aspect that solves the problem of modelling the large number of hair strands required, but does not always produce very realistic results. In this thesis, we make use of various texturing techniques for African hair.

1.2 Thesis Structure

Chapter 2 discusses the state of hair simulation within the field of computer graphics and reviews the literature on existing hair simulation techniques, focusing on hair modelling and hair rendering techniques. Chapter 3 introduces the techniques we develop to model the three identified forms of African hair. The implementation of these techniques as plug-ins for the graphics program LightWave 3D is discussed in Chapter 4. In Chapter 5 the rendering of hair modelled with our techniques is discussed. In Chapter 6 our results are presented and a number of issues relating to our techniques are considered. Chapter 7 concludes the thesis and suggests directions for future work.

2 RELATED WORK

2.1 Background

2.1.1 The State of Hair Simulation Research

Research to develop methods for the simulation of hair began just over two decades ago. Csuri et al. were the first to address the subject when they rendered fur-like volumes in the paper “Towards an Interactive High Visual Complexity Animation System” published in 1979 [Csuri et al. 1979]. Until then virtual characters were bald, had hats or similar objects placed on their heads, or were given a hairstyle created using a few polygons and maybe a texture [Magenat-Thalmann et al. 2000]. Since then, much research has been done to develop hair simulation techniques and the quality of computer-generated hair has improved greatly. This is mainly due to the increase in computing power and the graphics industry striving to produce photorealistic images, which has consequently lead to more applications using computer generated humans with hair. In the late 1980s several techniques relating to hair synthesis were published, but most of the techniques were for simulating fur rather than long hair [Anjyo et al. 1992]. In more recent years, work has been published on techniques to simulate long human hair. However, hair simulation research is not complete: hair dynamics is still an area constrained by computing power, there are no simple methods for producing different hairstyles, and the optical properties of hair still need to be explored. The current state of technology and hair simulation research is not at a stage where it is possible to produce any hairstyle in any condition (e.g. wet hair, greasy hair, etc.) that is able to blow in simulated wind in real time with complete rendering and collision detection [Magenat-Thalmann et al. 2000]. In this thesis we aim to further hair simulation research by investigating ways of producing different African hairstyles, which to our knowledge have not yet been considered.

2.1.2 Computer Generated Hair in Films

The results of previous hair simulation research can be seen in the numerous examples of computer-generated hair that occur in applications today. One application where examples of computer-generated hair can be seen is film production. In the past few years several films have been released that include simulated hair. In 1995 Pixar released *Toy Story*, the first full-length digital film. In *Toy Story* computer graphics techniques are not only used for special effects, but to create the entire film, including the characters (which of course need hair!). Baby Molly’s hair

is produced using a procedural texture created by Darwyn Peachey, while the hair for characters with straight hair (such as Sid's sister) is produced using layers of geometry with a few cylinders placed on top, designed by Eliot Smyrl (see Figure 2.1) [Computer Graphics World 1995].



Figure 2.1: *Baby Molly, Sid's sister, and Aki.*

Copyright © Disney / Pixar and © Square.

Since the release of *Toy Story* several other digital films have been produced such as *Monsters, Inc.*, *Ice Age* and *Final Fantasy*. In *Monsters, Inc.*, Sully's hair is modelled as a chain of particles linked by stiff springs. As many as 28,000 key hairs are modelled and a shader called a *builder* is used to generate the in-between hairs (see Figure 2.2) [Robertson 2001]. *Final Fantasy*, released in 2001, has the most realistic simulations of long human hair. In the film Aki's hair is modelled as a deformable surface. For some of the simulations the Maya cloth plug-in is used (see Figure 2.1) [Chang et al. 2002].



Figure 2.2: *Sully with key hairs defined and Sully rendered.*

Copyright © Disney / Pixar.

Other films, where examples of computer-generated fur can be seen, include *Jumanji*, *101 Dalmations* [Lengyel 2000], *Mighty Joe Young* and *Stuart Little*. *Stuart Little*'s fur is modelled in Maya and then rendered using RenderMan [Bredow 2000], while proprietary software by Dream Quest and Industrial Light and Magic (ILM) is used to create *Mighty Joe Young*'s fur [Robertson 1999]. Adding fur was and still is very time consuming. For example, according to Robertson [1999], Rebecca Petrulli-Heskes, the lead Viewpainter at ILM, used 21 texture maps to control the parameters of *Mighty Joe Young*'s fur. These maps took six months to paint.

2.1.3 Hair Simulation Tools

Tools to simulate hair are becoming more widely available to artists. A decade ago techniques to add hair to virtual characters were only available to the large effects studios, but today there are a number of commercially available graphics programs and plug-ins that allow a user to simulate suitable hair or fur [Robertson 1995]. These include Maya Fur, Shag, and Shave, which are plug-ins for the three-dimensional (3D) graphics programs Maya, 3D StudioMAX, and LightWave 3D, respectively. Typically, these plug-ins allow a user to describe the overall shape of hair by defining a few key hairs. Additional hairs are then procedurally created using ray tracing [Lengyel 2000]. The user is also able to control certain attributes such as hair length, colour, transparency, thickness and curliness [Robertson 1999].

The commercial hair simulation tools are large and sophisticated, but they are not always able to produce the effects an artist desires, and they are usually costly. During the production of *Toy Story*, hair simulation tools were just starting to be produced, however, the artists relied on their own creativity using a combination of resources to achieve the hair they were after [Computer Graphics World 1995]. Many artists still do this today, as there is a trade off between effort and artistic freedom. The more automated graphical effects such as hair production become, the less freedom artists have to produce a specific look. Furthermore, each resource has its own strengths and weaknesses, and by using a combination of resources an artist is able to avoid the weaknesses. In this thesis we implement our hair modelling techniques using plug-ins that aim to decrease modelling time without substantially restricting the artist's freedom. Our rendering techniques are implemented using the RenderMan Interface and Shading Language. This allows the hair to be modelled in any modelling program that is able to export the model as a RenderMan Interface Bytestream (RIB) file.

The next sections in this chapter examine hair modelling and rendering in greater detail.

2.2 Hair Modelling

2.2.1 Introduction

Hair modelling deals with the specification of the hair's geometry, and the density, distribution, orientation and shape of the individual hair strands [Daldegan et al. 1993]. Modelling hair is often tedious and time consuming, so the simplest way to generate hair, from the artist's point of view, is not to model it at all, but rather to apply two-dimensional (2D) textures of hair to polygons. This method is not only simple to use, but also requires relatively little rendering time when compared with other methods. It is therefore frequently used to generate the appearance of hair in real-time applications such as computer games. Unfortunately, the complex geometry, shading and dynamics of strands of hair are not reproduced by textured polygons [Kim and Neumann 2000], and other methods, although more expensive and time consuming, produce better results in terms of realism.

Since modelling hair is tedious and time consuming, Grabli et al. [2002] attempt to construct models of hair automatically, by retrieving the geometry of hair strands from photographs. Although their approach to acquiring hair models would be useful, it is still in an early stage of research and does not produce a complete hair model. Currently, therefore, 3D modelling techniques need to be applied in order to obtain hair models for virtual characters. Other than modelling the geometry of strands of hair, researchers have proposed methods that produce the appearance of hair using 3D textures or lighting models [Khouas et al. 1998].

The methods available for modelling hair can be divided into three main categories: those that model hair implicitly, those that model hair explicitly, and those that use a combination of implicit and explicit techniques to model hair as clusters. Each category has advantages and disadvantages, and depending on the circumstances different methods will be more suitable than others. Lengyel [2000] and Ward et al. [2003] use a combination of hair models to generate hair for applications where the distance of the hair from the viewer varies. Different hair models are used depending on the level of detail required due to the hair's distance. Hair modelling techniques proposed by several researchers in each of the three categories are discussed below. To assist a user in modelling hair, some researchers have developed interactive systems with user interfaces. These are also discussed below.

2.2.2 Implicit Models

Implicit hair models define the shape of hair using mathematical functions. The geometry of each individual strand of hair is never explicitly modelled. Instead hair is viewed macroscopically and an illusion of hair and its complexity is created.

Since implicit models do not explicitly model each individual strand of hair, they are able to overcome the problem of modelling the large number of hair strands required. They describe the complexity of hair in a compact way, they are not tedious to use, and it is often possible to create a variety of hair appearances with the same defining functions just by changing the values of a few parameters. A disadvantage of implicit models is that they have high computational costs. When rendering the hair the mathematical functions that define the hair have to be solved for the values of the parameters. It may also be difficult to define appropriate functions that describe the precise appearance of hair or a hairstyle. Additionally, most dynamics models are based on physical laws. Animating and deforming strands of hair that are not explicitly defined using physical laws is difficult [Dischler 1999].

Kowalski et al. [1999] present an algorithm that can be used to render non-photorealistic, stylised hair. Their method uses strokes that suggest complexity of hair without explicitly defining it. The method is based on techniques of depicting complexity that have been used in art and illustration for centuries. Implicit hair models that aim to produce more realistic hair than the stylised hair produced by Kowalski et al. [1999] include volumetric models, models based on fluid flow, and lighting models that create the appearance of hair.

2.2.2.1 Volumetric Models

Volumetric models create the appearance of hair by applying a 3D texture (or *volumetric texture*) of hair onto a surface. Hair is modelled using a *volumetric density function*. The models are able to produce very convincing representations of fur or short hair, but are not appropriate for modelling long hair. Long hair has more complex shapes and interactions than fur or short hair, and because volumetric models do not explicitly define the geometry of individual strands of hair, there is no control over individual hair strands and the complex shapes and interactions of long hair cannot be reproduced [Hadap and Magnenat-Thalmann 2000].

Hypertexture

Perlin and Hoffert [1989] developed a volumetric texturing technique that produces a 3D texture in a region of space. The 3D texture, called *hypertexture*, is produced by evaluating changes in density in the texture space. Perlin and Hoffert show that hypertexture can be used to create the illusion of fur by generating a furry donut and a furry ball. The fur is made curly by applying a noise function that displaces the points in the texture space. Several authors [Sourin et al. 1996; Bidasaria 1995] have modelled hair using volumetric approaches inspired by hypertexture. However, hypertexture is restricted to geometry that can be analytically defined, so the illusion of fur can only be applied to objects such as spheres, cubes or donuts [Magenat-Thalmann et al. 2000].

Texels

Kajiya and Kay [1989] extend Perlin and Hoffert's hypertexture approach so that 3D textures, such as fur, can be applied to objects of arbitrary shape. Their method considers hypertexture as a 3D rectangular volume, called a *texel*, which is tiled onto the complex geometry in a similar manner to 2D textures [Magenat-Thalmann et al. 2000]. In each texel, a density, surface frame and bidirectional light reflectance function parameters are stored. The rectangular shape of the texels may be deformed when the texels are mapped onto a surface in order to prevent gaps or overlapping from occurring between adjacent texels [Kajiya and Kay 1989].

Kajiya and Kay show that texels can be used to create the illusion of fur on complex geometry by modelling fur on a teddy bear. A texel containing parallel cylinders, whose tangent vectors all lie perpendicular to the surface on which the texel is mapped, is used to model fur as straight lines perpendicular to the scalp. Since the texel consists of parallel hairs perpendicular to the object's surface, the frame and bidirectional reflectance function are constant, so only density is stored in the texel [Kajiya and Kay 1989]. The fur rendered on the teddy bear is impressive, showing that texels can be used to model short hair successfully. However, finding the appropriate texel for a desired appearance can be difficult, and rendering times are slow due to the volumetric nature of the texel [Van Gelder and Wilhelms 1997]. Neyret [1998; 1995] extends Kajiya and Kay's texel model so that it can be used for multiresolution, by pre-filtering the texels at different scales.

Concentric Shells

Lengyel [2000] uses a simplified volumetric model to generate short hair. His model represents hair using a set of concentric, partially transparent, textured layers displaced from the surface. The textured layers, called *shells*, are slices of volumetric texture. Since 2D slices of texture are used rather than a 3D texture, the method is able to render hair much faster than hypertexture or texels. However, when the concentric shells are viewed edge-on, the hair appears transparent near the silhouettes and spaces between the shells become apparent. To improve the representation of the hair near silhouettes, Lengyel et al. [2001] place textured *fins* perpendicular to the surfaces near the silhouettes. They also extend the concentric shell method so that fur can be placed on objects with an arbitrary shape. This is done by repeatedly pasting patches of concentric, partially transparent shells over the surface until it is covered.

Papaioannou [2002] presents a report based on implementing Lengyel's concentric shell method [Lengyel 2000]. In the report Papaioannou extends Lengyel's approach to include shadows cast by hairs onto other hairs and onto the skin below.

2.2.2.2 Fluid Flow

Hadap and Magnenat-Thalmann [2001; 2000] model hair shape as streamlines of fluid flow after observing that there are similarities between hair shape and fluid flow. Since fluid is volumetric in nature, volumetric techniques are used to model the hair. However, the hair is rendered explicitly using polylines. Due to the continuum property¹ of fluid, hair-to-hair collision avoidance is automatically implemented and hair-to-body avoidance is described by the flow of fluid around an obstacle. Curls and waves in hair are modelled with vortices and turbulence in flow respectively. Although there are many similarities between hair and fluid flow, there are also differences. Hair strands tend to cluster, while streamlines do not [Hadap and Magnenat-Thalmann 2000]. The fluid-flow method is not useful for producing knots and braids [Magnenat-Thalmann et al. 2000].

¹The continuum property of fluid: "No two streamlines of a fluid flow intersect, however close they may get, with an exception of flow singularities" [Hadap and Magnenat-Thalmann 2000].

2.2.2.3 Fake Fur Lighting Model

Instead of rendering hair directly, Goldman [1997] presents a lighting model he calls the *fakefur* algorithm, which gives the appearance of thin coats of fur on skin. His model is a probabilistic model that calculates the expected value of light reflected within small areas of the surface. The model assumes that the parameters of the fur change slowly across the skin. Due to this assumption lighting calculations can be performed on a reference hair. The reflected light from individual fur strands and the skin below is blended using the opacity of the fur in that area. This is calculated using the probability of a ray hitting a hair in the area. This method has much lower computational costs than other methods, but is only effective in situations where the hair is viewed from a distance and the geometry of the hair strands is not visible [Goldman 1997].

2.2.2.4 Implicit Model Summary

The implicit hair models create the illusion of hair using mathematical functions, and do not explicitly model the geometry of each individual hair strand. They are able to create very convincing representations of fur or short hair, and are not tedious to use when modelling hair. However, defining an appropriate function to describe the appearance of hair required may be difficult, and computation costs when rendering are usually high. With the exception of the fluid flow model, implicit hair models are not appropriate for modelling long hair, and none of the models are useful for modelling braids of hair. Therefore implicit hair models would not be useful for modelling straight African hair or the braids and twists in African hair.

2.2.3 Explicit Models

Explicit hair models are brute-force methods that model the geometry of each individual strand of hair. They are therefore “intuitive and close to reality” [Magnenat-Thalmann et al. 2000]. However, because each individual strand of hair is considered, modelling hair using an explicit method is extremely time consuming (modelling a hairstyle can take five to ten hours [Magnenat-Thalmann et al. 2000]) and may be expensive in terms of computation time and memory usage [Kong and Nakajima 1999]. Unlike implicit models, explicit models view the hair microscopically and have control over the shape and movement of each individual strand of hair. They are therefore suitable for modelling long hair. To decrease the modelling and computation time, only a few key hairs are often modelled, and the shape and dynamics of the other hairs are determined from these key hairs. However, due to even distributions of strands of hair, explicit

models do not model braids in hair effectively [Kim and Neumann 2002]. Explicit models were the first attempts to model long hair [Magenat-Thalmann et al. 2000]. Later models such as cluster models or fluid flow avoid explicitly modelling individual strands of hair to decrease computational costs. However, computing power is increasing and explicit models may now be useful.

2.2.3.1 Representing Strands of Hair

The geometry of individual strands of hair can be defined using lines, curves, cylinders, or surfaces [Kim and Neumann 2000]. Van Gelder and Wilhelms [1997] use an explicit method to model fur. They do not consider long human hair. Using their method, which randomly adds hair to a triangular mesh of an animal, they compare four ways of representing strands of hair. The hair strands are represented as single lines, polylines (connected linear segments), Non-Uniform Rational B-Spline (NURBS) curves, or NURBS surfaces. The results of Van Gelder and Wilhelms' research show that polylines are an appropriate method for representing hair effectively and efficiently. Single lines, although fast, are not able to model the curved shape of a strand of hair. NURBS curves and NURBS surfaces provide a more accurate representation of a strand of hair than a polyline, but render at slower rates than polylines. Polylines provide a reasonable appearance of hair at an acceptable rendering rate and are therefore the preferred method.

Kong and Nakajima [1999] reduce the amount of memory required and the time spent on modelling and rendering hair by modelling fewer, thicker hairs. However, to increase the quality of the hair, thick surface hairs close to the viewer are broken down into many thinner strands of hair. To determine the surface hairs an opacity buffer created from a *visible volume buffer* is used.

Figure 2.3 illustrates different methods that have been used to represent a strand of hair that is modelled explicitly. The different methods include polylines (A), splines (B), particle systems (C) and trigonal prisms (D). These methods are discussed below.

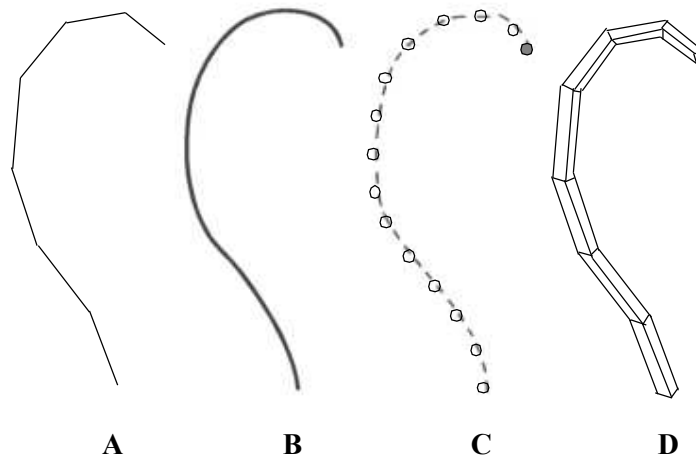


Figure 2.3: *Different ways to represent a strand of hair modelled explicitly.*

Polylines

The use of polylines to represent hair can be seen in work by several researchers [Rankin and Hall 1996; Daldegan et al. 1993; Anjyo et al. 1992; LeBlanc et al. 1991]. In many instances the user is able to specify certain properties of the polyline, such as its length, thickness and colour. Anjyo et al. model the downward bending of a strand of hair due to gravity using a simplified cantilever beam² simulation with the fixed side corresponding to the root of the hair. To create cutting or combing effects they apply external forces [Anjyo et al. 1992].

Splines

Several researchers [Ward et al. 2003; Lao et al. 2002; Lee et al. 2002; Iones et al. 2000; Thalmann et al. 1996] use splines instead of polylines to model hair. Lao et al. [2002] use B-spline curves. Lee et al. [2002] render hair as Cardinal splines, and simulate the downward bending of a strand of hair using Anjyo et al.'s cantilever beam simulation. Although each individual strand of hair is explicitly modelled, the strands of hair are grouped into *wisps* and a key hair in the centre of the wisp determines the shape of each strand. Iones et al. [2000] represent each strand of hair as a spline defined by a number of control points. The number of control points used is determined by the complexity and length of each strand of hair. Thalmann et al. [1996] represent each hair strand as a NURBS. The vertices of the NURBS are considered as control points of a spline and are used to determine and alter the shape of the strand of hair. Ward et al. [2003] model individual strands of hair as subdivision curves using control points.

²A cantilever beam is a beam with one side fixed.

Particle Systems

Heinen and Walter [2001] model each strand of hair (representing fur) as a path of a particle, which is then rendered as a polyline or collection of connected cylinders. Particle systems do not explicitly model the geometry of hair strands (like most explicit hair models), however, the shape of each individual strand of hair is explicitly defined.

Trigonal Prisms

Watanabe and Suenaga [1992] represent a strand of hair using a connected set of trigonal prisms. The trigonal prisms are used as a rough approximation for cylinders to reduce the number of polygons used. To increase the quality of hair produced, smooth shading is applied to the trigonal prisms. Different types of hair are produced by changing the values of various parameters such as thickness, colour, length, amount of twist, and the number of trigonal prisms used for each hair strand. To decrease the amount of time spent on modelling, the hair strands are grouped into wisps, and the values of the parameters are set for each wisp.

2.2.3.2 Placing Strands of Hair on a Head Model

Techniques are required to place explicitly modelled strands of hair onto a model of a head, and researchers have used several different methods to do this.

Control Hairs

Daldegan et al. [1993] present a technique that places strands of hair onto a head model by first creating a number of characteristic strands of hair (called *control hairs*) on the triangular patches representing the scalp of the head model. The control hairs (or key hairs) define a hairstyle, and at most, one control hair is placed on each triangular patch. The rest of the hair strands are then added to the head model by duplicating the control hairs on each patch. The duplicated hairs have the same initial shape, length and orientation as the control hair, and are randomly placed in the same triangular patch as the control hair. The number of hairs duplicated is user defined. Like Daldegan et al., Lao et al. [2002] and Iones et al. [2000] present techniques that use control hairs to define the shape of hair strands in different areas on the head model, and automatically create more strands of hair using the control hairs and interpolation techniques. Iones et al.'s technique allows strands of hair to be created on both NURBS and polygonal surfaces.

Subdividing Triangles

Thalmann et al. [1996] developed a technique that places hair strands onto a surface by placing a strand of hair onto each vertex of the triangles that make up the surface. The triangles are then subdivided into four smaller triangles and strands of hair are placed onto the vertices of the new triangles. The control points of the new hair strands are interpolated from the control points of hair strands placed on the vertices of the two original triangles surrounding the new hair strand's vertex. The process of subdividing triangles and placing strands of hair onto vertices is repeated until enough strands of hair have been created for the required density (see Figure 2.4).

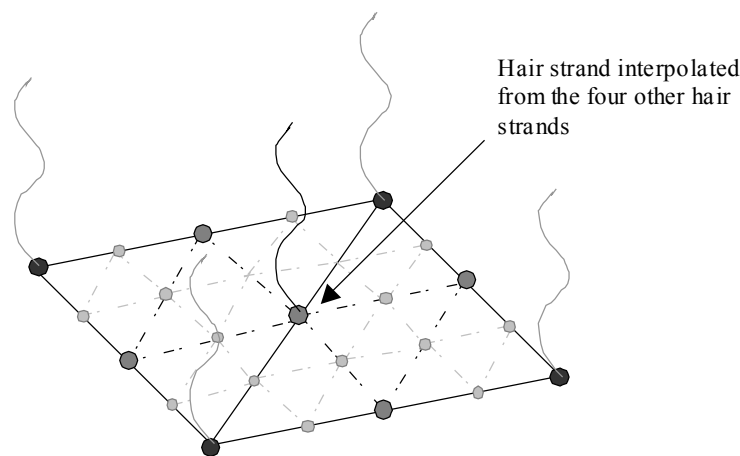


Figure 2.4: *Placing strands of hair onto a surface.*

Other Techniques

Lee et al. [2002] developed a hair modelling system that places strands of hair onto a head model using a growth map determined by the density of hair required. The growth map is placed onto a user-defined area of the head model, where the user would like hair to be placed. Van Gelder and Wilhelms [1997] use a technique that determines the number of hair strands placed onto a surface according to the viewing distance, and Heinen and Walter [2001] uniformly distribute a user-defined number of hairs over the surface.

Anjyo et al. [1992] propose a technique that considers collision detection. To make the collision detection and placement of strands of hair onto a surface of a head model faster, the head model is approximated by an ellipsoidal hull.

2.2.3.3 Explicit Model Summary

The explicit hair models are brute-force methods that model the geometry of each individual strand of hair. Modelling hair using an explicit method is extremely tedious and may be expensive in terms of computation time and memory usage. To decrease the modelling and computation time, key hairs are often used. The explicit hair models are suitable for modelling long and short hair, but are not appropriate for modelling braids of hair. Therefore, an explicit model would be useful for modelling straight African hair, but would not be useful for modelling braids and twists in African hair.

2.2.4 Cluster Models

Cluster (or wisp) hair models make use of the observation that strands of hair tend to form clumps in nature (especially when the hair is wet or oily). Strands of hair are modelled in groups or clumps called *clusters*. The methods explicitly model the shape of each cluster and then implicitly add details of individual strands of hair by procedurally rendering strands of hair (using volume rendering techniques), or by applying hair textures to the surfaces defining the cluster. The clusters are usually added to the head model using similar methods as those used to add explicitly modelled strands of hair. For example, Wang and Yang [2001] and Chen et al. [1999] define a hairstyle with a number of characteristic clusters (similar to control hairs), and then automatically create and place other clusters onto the head model using the characteristic clusters. Kim and Neumann [2002] use a different method, however. A parametric (Catmull-Rom spline) patch is wrapped over the head model where hair should be placed, and clusters are distributed over the patch rather than the head model. Their technique reduces the hair placement problem to a 2D problem rather than a 3D problem.

Cluster models reduce the number of hair strands that need to be modelled (making it easier to model hair), but are often not able to reproduce the variations needed between strands of hair in a cluster. The models are therefore useful for modelling characteristics such as braids, but are not useful for modelling curly hairstyles where the shapes of adjacent hair strands vary greatly [Kim and Neumann 2002]. The cluster models developed can be categorized into two groups: those that create or render individual strands of hair in a cluster using some sort of distribution, and those that apply textures to the surfaces making up each cluster to create the effect of many strands of hair. Several cluster models in each group are discussed below.

2.2.4.1 Clusters with Individual Strands of Hair

Most hair cluster methods model the shape of each cluster explicitly, and then add the details of individual strands of hair by implicitly modelling the individual strands of hair during rendering. Several methods that fall into this category of cluster models are discussed below.

Trigonal Prisms

Chen et al. [1999] model hair in clusters of between 1 and 255 strands of hair. Each cluster is created using a continuous trigonal prism defined by three B-spline curves. To create individual strands of hair, a 2D distribution map is assigned to each cluster. The 2D distribution map is a 2D array that defines the distribution of strands of hair on a cross section of the trigonal prism. The position of a section of each strand of hair is determined by mapping 2D points from the distribution map into 3D space using two pairs of vectors determined by the edges of two triangles (see Figure 2.5). Because the 2D distribution map may define points that are outside the cross section of the trigonal prism, the shape of the cluster is not necessarily triangular. By changing the shape and density of the points in the 2D distribution map, or by linking the calculated points that make up the strands of hair in different ways (e.g. with straight lines or sine curves), a variety of hair types can be created using the same cluster structure (i.e. using the same B-spline curves or continuous trigonal prisms).

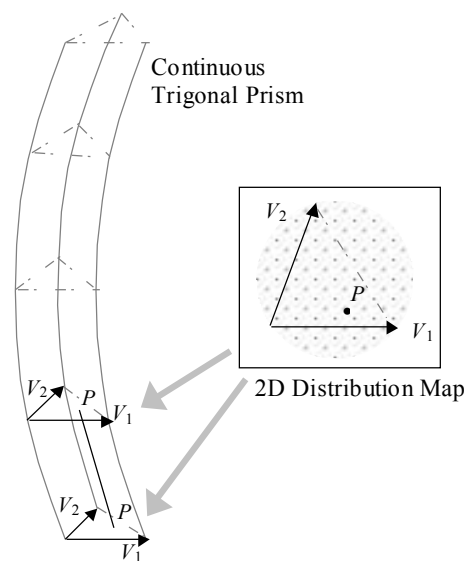


Figure 2.5: Calculating a section of a strand of hair.

The model is able to create hairstyles that contain artificial objects, such as hair bands, by placing restrictions on the clusters, or by requiring the clusters to pass through certain regions in 3D space. Chen et al. [1999] mention that a braid of hair can be created with three hair clusters, but they do not provide details on how to implement a braid. A method for creating a braid with three groups of hair can be found in a paper by Lee et al. [2002], who model hair explicitly, but control the individual strands of hair in clusters.

Generalized Cylinders

Wang and Yang [2001] and Xu and Yang [2001] developed an interactive hair designing system based on an extended cluster hair model that represents each cluster as a generalized cylinder rather than a continuous trigonal prism. A distribution map is used to define the positions of individual strands of hair at cross sections of the generalized cylinder. The individual strands of hair are never explicitly modelled. Instead, they are procedurally created during rendering using volume rendering techniques [Xu and Yang 2001].

Kim and Neumann [2002] developed a multi-resolution hair modelling system, which also uses generalized cylinders to define the boundaries of hair clusters. Using their system, each cluster can be subdivided into smaller clusters for more detail (down to a single strand of hair). The individual strands of hair are assigned to generalized cylinders based on their position on the head model. Each strand of hair is represented by a polyline.

Linked Mass-Points

Plante et al. [2001] model hair in clusters created with mass points linked by damped springs. Each cluster is defined by a curve, and is created by linking a number of cross sections of the cluster along the curve. Each cross section is defined by four mass points (see Figure 2.6).

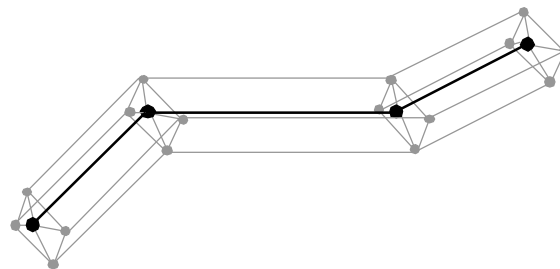


Figure 2.6: *A cluster.*

To create the individual strands of hair in each cluster a set of 2D points are generated according to a given distribution and mapped onto each cross section of the cluster. Corresponding points in the cross sections are then linked to form Catmull-Rom curves representing strands of hair. To create a more natural appearance of hair, jittering is applied to the points defining the individual strands of hair.

Thin Shell Volumes

Kim and Neumann [2000] propose a method for adding the detail of combed hair strands to hairstyles modelled with surfaces, defined parametrically (e.g. by NURBS) or with polygons. The details of hair strands are added by constructing a *thin shell volume* containing strands of hair onto each surface. The thin shell volume is created by displacing the surface in the direction of its normals. To create the individual strands of hair in each volume, sets of particle positions are defined and distributed according to the desired combing effect. Each set of particle positions is then connected to form a curve, which represents a strand of hair.

2.2.4.2 Textured Clusters

Instead of rendering individual strands of hair in a cluster, it is possible to create the illusion of strands of hair by applying textures to the geometry of each cluster. Hair modelled using a textured cluster method is usually less accurately represented than hair modelled using other methods, and the hair's volume is not simulated. However, the method does not require an expensive volumetric rendering technique, and therefore has a higher rendering performance than other methods. Consequently, the method is often used to model hair for real-time applications.

Textured Strips

Koh and Huang [2001] model hair in clusters using 2D strips, represented by NURBS surfaces. To create the illusion of individual strands of hair, texture maps are applied to each strip. We examine the application of the texture maps in more detail in Section 2.3.4.3. Ward et al. [2003] and Guang and Zhiyong [2002] model hair in strips based on Koh and Huang's textured-strip method.

Textured Generalized Cylinders

Ward et al. [2003] model clusters of hair as strips when the hair is viewed from a distance or when fine details cannot be seen. When the hair is viewed more closely, the clusters are represented with generalized cylinders rather than strips.

2.2.4.3 Cluster Model Summary

The cluster hair models make use of the observation that strands of hair tend to form clusters in nature. The methods explicitly model the shapes of clusters of hair, and then implicitly add details of individual strands of hair by rendering them procedurally, or by applying hair textures to the surfaces defining each cluster. Since the strands of hair are modelled in clusters, the time spent on modelling a hairstyle is decreased, but variations between strands of hair in a cluster are not usually reproduced. A cluster technique would be useful for modelling braids and twists in African hair, but it would not be useful for modelling natural curly African hair where the shapes of adjacent hair strands vary greatly.

2.2.5 Interactive Hair Modelling Systems

Creating 3D hairstyles is a time-consuming process and requires a skilled artist [Chen et al. 1999]. There are an infinite number of human hairstyles that can be produced, and the previous sections in this chapter show that many hair-modelling techniques have been developed to model different types of hairstyles. However, only a few researchers present interactive systems based on their techniques that allow artists to design desired hairstyles with ease.

An advantage of explicit and cluster models is that the user has control over the shape of each individual strand of hair or cluster. However, in order to model the shape of a strand of hair or cluster explicitly, design tools are needed. Most interactive hair modelling systems are therefore designed to provide an interface for explicit or cluster hair modelling techniques. Interactive hair modelling systems are discussed below.

2.2.5.1 Systems with Standard 2D Input Devices

Most interactive hair modelling systems allow a user to design a hairstyle by creating, editing and previewing a representation of the 3D hairstyle on a computer screen. The hairstyle can usually be viewed from a perspective and/or three orthogonal viewpoints. Functions are provided by the system to import a head model; create, edit and arrange representations of hair (such as curves or generalized cylinders) on the head model; and to specify rendering properties, such as the position of a light source. These functions are usually activated and controlled using an input device with two degrees of freedom, such as a mouse. To create and edit the hairstyle, the functions include operations such as selection, translation, rotation, scaling, copying and pasting parts of the hair.

Interactive Systems for Explicit Models

Interactive systems that implement explicit hair models include a system by Watanabe and Suenaga [1992] and *HairStyler* by Daldegan et al. [1993].

Watanabe and Suenaga's system [1992] allows a user to create a hairstyle with trigonal prisms. The user is able to define values for 12 parameters (which represent hair characteristics such as length, thickness, density and colour). Unfortunately, the same parameter values are used for every strand of hair, and individual strands of hair cannot be edited separately.

HairStyler allows a user to define a hairstyle interactively using a few control hairs. To create a full head of hair, a user-defined number of hairs are created by the system using the control hairs. Initially these strands of hair have the same shape, length and orientation as the control hairs. However, the user is able to edit individual strands of hair at a later stage [Daldegan et al. 1993].

Interactive Systems for Cluster Models

Interactive systems that implement cluster hair models include a system by Kim and Neumann [2002], *HET* (Hairstyle Editing Tool) by Chen et al. [1999], and *V-HairStudio* developed by Xu and Yang [2001] and improved by Wang and Yang [2001].

Kim and Neumann's system [2002] allows users to model a hairstyle interactively with clusters represented by generalized cylinders. A user is interactively able to subdivide a cluster, at any level, into smaller clusters for more detail.

In HET [Chen et al. 1999] hair is modelled in clusters represented by continuous trigonal prisms. The system is able to combine an image of a face with a hairstyle stored in a database, allowing complicated hairstyles to be reused.

Xu and Yang [2001] developed V-HairStudio based on a cluster model that represents clusters of hair with generalized cylinders. Wang and Yang [2001] extend the system so that clusters can be controlled, created and edited in groups of many clusters.

2.2.5.2 Systems with Specialized Input Devices

Lee et al. [2002] present an interactive hair modelling system that is based on an explicit model. The system creates strands of hair based on a distribution map placed on a user-defined area of a head model. A user is able to edit individual strands of hair interactively using *styling* operations, which change the lengths, positions and curvatures of the hair strands. However, unlike most interactive hair modelling systems that use a standard input device with two degrees of freedom, Lee et al.'s system uses a special hardware rotor platform and 3D trackers. According to Lee et al. their system decreases the amount of time required by a user to create and edit a hairstyle because a user is able to edit the hair using both their hands, imitating a real hair styling process.

2.3 Hair Rendering

2.3.1 Introduction

Hair rendering involves lighting and shading hair, and is responsible for anti-aliasing techniques. Due to the large number of hair strands, their thin geometric shape, and the complex lighting effects that occur among the strands of hair, hair rendering is challenging and computationally expensive. In this section we discuss different techniques that have been developed to render hair.

The rendering techniques used depend on the method employed to model the hair. The easiest way to render hair would be to model each individual strand of hair as a curved cylinder, and

then render the cylinders using conventional rendering techniques. However, the large number, complex shape and narrow width of hair strands have caused researchers to model hair either implicitly, with cluster techniques, or explicitly as 2D polylines or splines.

Hair that has been modelled implicitly is usually ray traced [Kajiya and Kay 1989; Perlin and Hoffert 1989]. However, Hadap and Magnenat-Thalmann [2000] render implicitly modelled strands of hair as explicit polylines, and Lengyel [2000] applies conventional texture-mapping techniques to concentric shells to produce the effect of hair. The results of ray-traced hair are often very realistic, but computation and memory costs are high [Papaioannou 2002]. Hair that is modelled with a cluster technique is usually rendered as explicit polylines or produced using a texturing technique (conventional texture mapping or a volumetric method). A variety of techniques can be used to render hair that has been modelled explicitly.

Different illumination models and shadowing techniques developed for rendering hair are discussed below, as well as texturing and anti-aliasing techniques.

2.3.2 Illumination Models

2.3.2.1 Introduction

Explicitly rendering each individual strand of hair as a cylinder is impractical due to the number and size of the hair strands. An illumination model is therefore needed that is able to reproduce the optical properties of hair. Two important optical properties that affect the appearance of hair are anisotropic highlights (caused by the geometry and orientation of individual strands of hair) and haloing under backlighting conditions, which is prominent in light-coloured hair (due to the hair being partially transparent) [LeBlanc et al. 1991].

2.3.2.2 Kajiya and Kay's Anisotropic Lighting Model

Probably the most well-known illumination model used for rendering hair, which has been used by several researchers [Ward et al. 2003; Kim and Neumann 2002; Iones et al. 2000; Kong and Nakajima 1999; Daldegan et al. 1993; LeBlanc et al. 1991], is Kajiya and Kay's anisotropic lighting model [Kajiya and Kay 1989]. The lighting model calculates the illumination for a thin cylinder, or segment of a single strand of hair.

Figure 2.7 shows a thin cylinder and the important vectors and angles used in the illumination model to calculate the light intensity at a point p . T is the tangent vector that represents the direction of the hair, L is the light vector that points towards the light source, V is the eye vector that points towards the viewer, R is the reflection vector that points in the direction of reflected light, and θ and Φ are the angles between T and L , and T and V respectively. All the vectors are normalized [LeBlanc et al. 1991].

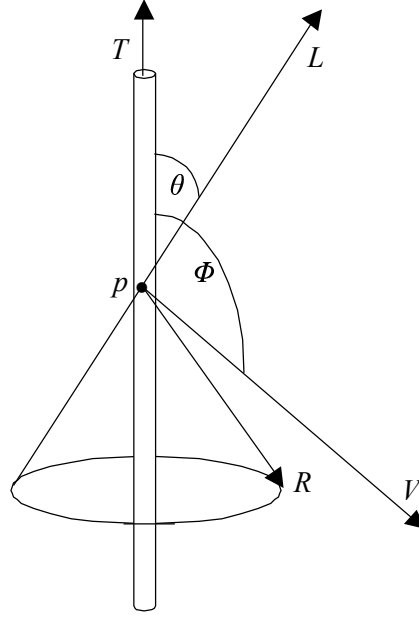


Figure 2.7: Kajiya and Kay's illumination model.

The diffuse component, $\Psi_{diffuse}$, is obtained by integrating Lambert's Cosine Law along the circumference of the half of the cylinder visible from (and illuminated by) the light source:

$$\Psi_{diffuse} = K_d \sin(\theta)$$

where K_d is the diffuse reflectance coefficient. Since the calculation is independent of the eye vector V , self-shadowing is not considered and the hairs are fully lit even when the viewer and the light sources are on opposite sides of the hair. However, since most hair is partially translucent, the model provides an acceptable approximation of the diffuse component. The area in shadow is assumed to transmit a similar amount of light through the hair towards the viewer as that reflected from the illuminated side [LeBlanc et al. 1991].

The specular component, $\Psi_{specular}$, is obtained using a Phong light reflection model modified for a thin cylindrical surface. Because the cylinder representing a strand of hair is so thin (usually a

polyline), the surface normals at each point along the cylinder point in all directions perpendicular to the tangent vector, causing light to be reflected in all directions around the cylinder. The reflection vector is therefore a cone around the cylinder (see Figure 2.7). The angle $(\Phi + \theta - \pi)$ between the eye vector V and the specular reflection vector R (the part of the cone closest to V) is used as the angle in a standard Phong equation to obtain the modified Phong equation for the specular component:

$$\Psi_{specular} = K_s \cos^n(\Phi + \theta - \pi)$$

where K_s is the specular reflection coefficient and n is the Phong exponent specifying the sharpness of the highlight [Kajiya and Kay 1989].

Using the diffuse and specular components, together with an ambient component, the light intensity, Ψ_{hair} , of a point p on a hair strand can be calculated using the equation:

$$\Psi_{hair}(p) = L_a K_a + \sum L_i (\Psi_{diffuse}(p) + \Psi_{specular}(p))$$

where L_a and L_i are the light intensities of the ambient light and each light source i respectively, and K_a is the ambient reflection coefficient (or ambient component) [LeBlanc et al. 1991].

Although Kajiya and Kay's model is a well-known illumination model that considers the anisotropic property of hair and produces reasonable results, it does not implement the effect of backlighting (i.e. haloing), which is prominent in light-coloured hair.

2.3.2.3 Adding Backlighting

Yang and Ouhyoung [1997] extend Kajiya and Kay's model by adding a backlighting effect. Using density maps, the backlighting effect is calculated by computing the intensity of the light passing through overlapped hairs. The extended illumination model is represented by the equation:

$$\Psi_{hair}(p) = L_a K_a + \sum L_i (\Psi_{diffuse}(p) + \Psi_{specular}(p)) + \sum (1-D_i) L_i f(\omega)$$

where D_i is a value obtained from the density map and $(1-D_i)$ represents the fraction of the i^{th} light's intensity reaching the viewer. The function $f(\omega)$ is an ad hoc function that specifies whether or not backlighting should take place depending on the angle ω between the light vector L and the eye vector V . When L and V are on opposite sides of the hair backlighting takes place and $f(\omega)$ has a value of 1. When no backlighting occurs $f(\omega)$ has a value of 0 [Yang and Ouhyoung 1997].

Watanabe and Suenaga [1992] also implement a backlighting effect, using a double z-buffer to compute the thickness of the hair at each pixel. Pixels with low values, representing thin areas of hair, are given higher intensity values. The illumination model they use is not given.

2.3.2.4 Considering Light Reflection and Transmission (Goldman's Illumination Model)

One of the weaknesses of Kajiya and Kay's model is that it does not consider the direction of the viewer, and hairs are fully lit even when the light source is behind the hair. Instead of adding a backlighting effect like Yang and Ouhyoung, Goldman [1997] modifies Kajiya and Kay's model so that both light reflection and light transmission are considered by taking into account the direction of the viewer and the translucency of the hair. An attenuation factor is introduced into Kajiya and Kay's light intensity equation to adjust the relative transmission and reflection of light. The modified illumination model is represented by the equation:

$$\Psi_{hair}(p) = L_a K_a + \sum L_i f_{dir} (\Psi_{diffuse}(p) + \Psi_{specular}(p))$$

where f_{dir} is the directional attenuation factor, and represents the reflection and transmission properties of the hair [Goldman 1997].

The directional attenuation factor f_{dir} is calculated using:

$$f_{dir} = ((1+\kappa)/2)\rho_{reflect} + ((1-\kappa)/2)\rho_{transmit}$$

where $\rho_{transmit}$ and $\rho_{reflect}$ vary in the range [0, 1] and represent the amounts of forward and backward scattering, and κ represents the relative directionality of L , V and T .

The cosine of the angle between the vectors (TxL) and (TxV) , represented by κ , is calculated using:

$$\kappa = ((TxL) \cdot (TxV)) / (|TxL||TxV|).$$

When L and V are on the same side of the hair there is front lighting, and $\kappa > 0$. When L and V lie on opposite sides of the hair there is back lighting, and $\kappa < 0$ [Goldman 1997].

The values of $\rho_{reflect}$ and $\rho_{transmit}$ should be similar for light-coloured hair, while $\rho_{reflect}$ should have a much large value than $\rho_{transmit}$ when the colour of the hair is dark. When $\rho_{reflect}$ and $\rho_{transmit}$ are both 1 there is no difference between Goldman's model and Kajiya and Kay's model [Goldman 1997]. Researchers who use Goldman's illumination model include Grabli et al. [2002] and Chang et al. [2002].

2.3.2.5 Simplifying the Specular Component

Anjyo et al. [1992] introduce an anisotropic lighting model that produces reasonable results for dark-coloured, glossy hair. Like Kajiya and Kay their lighting model calculates the illumination for a thin cylinder, or segment of a single strand of hair, which is usually represented by a line segment in a polyline. Since the illumination model is developed for dark, glossy hair, Anjyo et al. ignore the diffuse component.

The specular component, Ψ_{specular} , based on Blinn's specular model [Blinn 1977], is a simplified computation of the specular term in Kajiya and Kay's model. Blinn's specular model calculates the specular component Ψ_{specular} at a point p on a surface by:

$$\Psi_{\text{specular}}(p) = K_s (N \cdot H)^n$$

where K_s is the specular reflection coefficient, N is the surface normal at p , H is the halfway vector between L and V , and n is an exponent specifying the sharpness of the highlight. Since p lies on the surface of a very thin cylinder when illuminating hair, the surface normal at p lies in all directions perpendicular to the tangent vector T . Anjyo et al. take the normal N to be the normal on the plane *spanned* by the vectors H and T (see Figure 2.8).

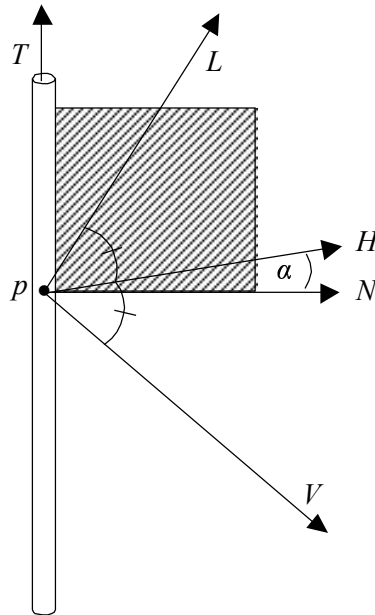


Figure 2.8: *Anjyo et al.'s illumination model.*

Therefore:

$$N \cdot H = \cos \alpha = (1 - (T \cdot H)^2)^{1/2}$$

and the specular component at p is defined as:

$$\Psi_{\text{specular}}(p) = K_s \{1 - (T \cdot H)^2\}^{n/2} \text{ [Anjyo et al. 1992].}$$

The light intensity, Ψ_{hair} , of a point p on a hair strand is calculated using the equation:

$$\Psi_{\text{hair}}(p) = L_a K_a + \sum L_i (\Psi_{\text{specular}}(p))$$

where L_a , L_i and K_a are defined as before and $\Psi_{\text{specular}}(p)$ is calculated using the equation above [Anjyo et al. 1992].

2.3.2.6 Adding an Excess Brightness Exponent

Due to the anisotropic property of hair, the surface normal at each point on a hair strand points in many directions, causing an increased and more uniform intensity of light over the surface. Banks [1994] modifies Kajiya and Kay's model by adding an excess brightness exponent, k , to the diffuse component, Ψ_{diffuse} , in order to compensate for the average increase in light intensity on objects such as hair:

$$\Psi_{\text{diffuse}} = K_d \sin^k(\theta)$$

2.3.2.7 Using Texture Maps and Hardware-Assisted Lighting

Lengyel [2000] and Lengyel et al. [2001] use an illumination model that contains Bank's diffuse component with an excess brightness exponent and Anjyo et al.'s simplified specular component. Since Lengyel and Lengyel et al. model hair with concentric textured layers, the values of the tangent vector T are stored in a texture map, and the light and halfway vectors, L and H , are assigned to a hardware texture matrix. At runtime a lookup table is used to compute the light intensity at each point:

$$\Psi_{\text{hair}}(u, v) = K_a + K_d (1 - u^2)^{k/2} + K_s (1 - v^2)^{n/2}$$

where $u = T \cdot L$ and $v = T \cdot H$ [Lengyel et al. 2001]. Because L and H are encoded using a texture matrix, the technique only works when the viewer and light directions are the same for each point.

Lengyel [2000] presents a second illumination technique that is able to illuminate hair with different viewer directions. The technique uses standard hardware-assisted lighting. However, since a thin cylinder represents a strand of hair, and the surface normals at each point along the

cylinder point in many directions, a new normal is calculated at each point and passed to the graphics hardware to be used as the normal in the standard hardware illumination model.

To calculate the new normal at a point p , the light vector L and halfway vector H are projected onto the plane perpendicular to the tangent vector T at that point (see Figure 2.9):

$$N_L = L - (T \cdot L)T$$

$$N_H = H - (T \cdot H)T, \text{ where } H = (L + V)/2$$

Ideally, the vector N_L (projection of L) should be used as the normal when calculating the diffuse component, and the vector N_H (projection of H) should be used as the normal when calculating the specular component. However, the standard hardware illumination model can only take one normal. To solve the problem Lengyel calculates a new normal as a weighted combination of N_L and N_H , together with the underlying mesh's normal N_M :

$$N = B_L N_L + B_H N_H + B_M N_M$$

where $B_L + B_H + B_M = 1$ [Lengyel 2000].

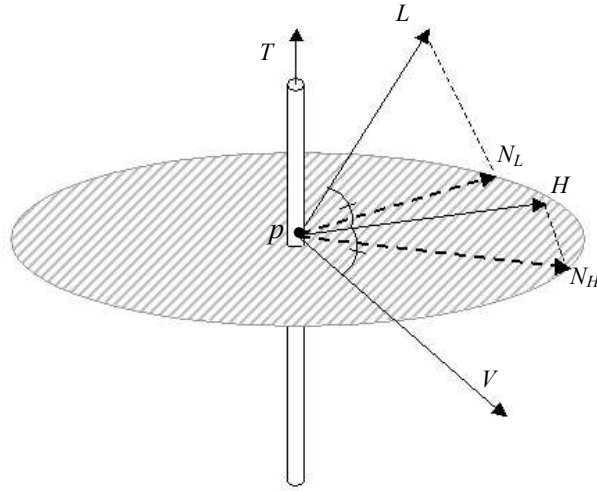


Figure 2.9: *Obtaining normals.*

2.3.2.8 Considering Secondary Scattering of Light

The latest research by Marschner et al. [2003] shows that taking secondary scattering of light into consideration enables more realistic results to be obtained. Marschner et al. examine and measure the scattering of light from strands of human hair. They describe a hair strand as “a transparent elliptical cylinder with an absorbing interior and a surface covered with tilted scales”. Their research shows that due to internal reflection and the tilt of the scales, a multiple specular

highlight occurs and there is variation in scattering with rotation about the hair strand's axis. Using the results from their experiments they present a new shading model that models the multiple specular highlight and approximates the effect caused by the hair strand's rough, elliptical surface. Their model considers not only surface reflection and refractive transmission, but also internal reflection.

2.3.3 Shadowing

2.3.3.1 Introduction

Shadowing is an important factor when rendering realistic hair. Strands of hair cast shadows onto the head and each other. The importance of self-shadowing (hair casting shadows onto hair) can be seen in Figure 2.10 (an image created by Lokovic and Veach [2000]), which shows hair rendered with and without self-shadowing. Unfortunately, adding shadows to hair is difficult and computationally expensive due to the large number of hair strands.



Figure 2.10: *Hair rendered with and without self-shadowing.*

2.3.3.2 Ray Tracing

Ray tracing is a useful technique for generating accurate shadows. Since implicitly modelled hair is usually ray traced, shadows are often generated using ray tracing for implicitly modelled hair. Unfortunately, due to the large number of hair strands, ray tracing hair is expensive in terms of rendering time [LeBlanc et al. 1991], and therefore is often avoided, especially when hair is modelled explicitly or in clusters.

2.3.3.3 Traditional Shadow Maps

Shadow maps are more efficient for producing shadows than ray tracing. Traditional shadow maps store the depth values of the closest points from the point of view of each light source.

When the scene is rendered, shadows are added by checking each point's depth against the depth in the shadow map to see if it is in shadow [Ahokas 2002]. A disadvantage of traditional shadow maps is that aliasing occurs when creating and sampling shadow maps for very small or semi-transparent objects, such as hair. Small objects cause the depth values to change frequently and transparent objects produce partial shadows, which shadow maps cannot handle [Kim and Neumann 2001].

LeBlanc et al. [1991], however, use shadow maps, together with pixel blending, to generate shadows for hair. They explain that stochastic sampling can be used to reduce aliasing when shadow maps are created, and they use percentage closer filtering to reduce aliasing when sampling the shadow maps.

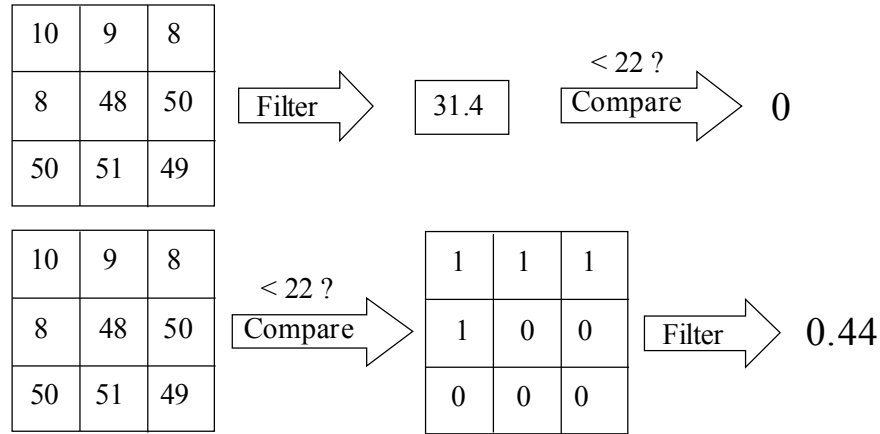


Figure 2.11: *Percentage closer filtering.*

Figure 2.11 [Ahokas 2002] shows the difference between percentage closer filtering (bottom) and ordinary filtering (top). To determine whether a point is in shadow its depth (z value), in this case 22, is compared to a region of several pixels in the shadow map. When ordinary filtering is used the values in the shadow map are filtered, and the point's depth is compared to a single value, resulting in a binary value of 1 (the point is in shadow) or 0 (the point is not in shadow, as in this case). Because the result is a binary value, and each point is either fully in shadow or not, aliasing may occur. Using percentage closer filtering, the depth of the point is first compared to each value in the shadow map, resulting in a set of binary values. The binary values are then filtered to obtain a grey-scale value, in this case 0.44. The grey-scale value specifies the degree of shadow for the point, and solves the aliasing problem [LeBlanc et al. 1991].

LeBlanc et al. render the scene without hair and the hair separately in order to pixel blend the hair. Shadow maps are created separately for the scene and the hair, and are then combined to create a single composite shadow map for each light source. First the scene is rendered without hair using the composite shadow maps. Hair is then rendered with the composite shadow maps and blended with the scene [LeBlanc et al. 1991].

Authors who use LeBlanc et al.'s shadowing method include Iones et al. [2000] and Yang and Ouhyoung [1997].

2.3.3.4 Deep Shadow Maps

Lokovic and Veach [2000] enhance traditional shadow maps by developing *deep shadow maps*. Instead of storing a single depth value for each pixel in an image, deep shadow maps store a function that represents the visibility at all possible depths for each pixel in the image. This function, called the visibility function, records how the intensity of the light decreases as it passes through the pixel to each depth (see Figure 2.12).

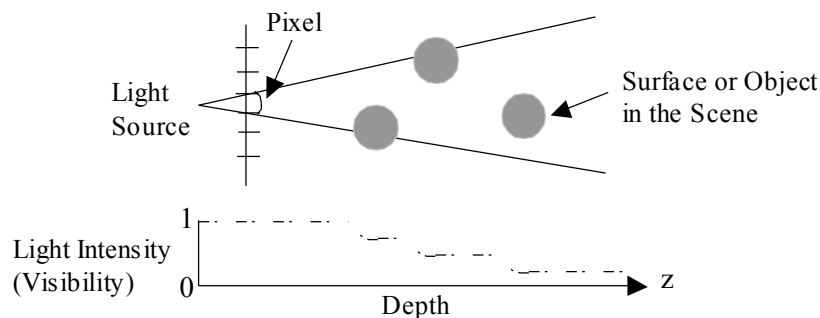


Figure 2.12: *The visibility function for a pixel.*

Deep shadow maps are able to generate shadows for partially transparent and volumetric objects. They also have smaller storage requirements (due to the use of compression) and are faster to access (due to pre-filtering) than traditional shadow maps when high-quality shadows are desired. They also support mip-mapping, which reduces lookup costs when different scales are used.

However, deep shadow maps are more expensive to compute than traditional shadow maps of the same pixel resolution because they require many more samples to be taken for each pixel [Lokovic and Veach 2000].

Lokovic and Veach's deep shadow method was used to render Sully's hair for the film *Monsters, Inc.* [Robertson 2001].

2.3.3.5 Opacity Shadow Maps

Kim and Neumann [2001] generate shadows using a faster approximation of deep shadow maps, which they call *opacity shadow maps* [Kim and Neumann 2001]. Instead of storing a visibility function for each pixel, opacity shadow maps approximate the decrease in light intensity for each pixel using a set of opacity maps.

Each opacity map stores the opacity of the pixels at a certain depth, and can be seen as the opacity map for a slice of the volume being rendered perpendicular to the light's direction (see Figure 2.13). The opacity values in adjacent opacity maps are then sampled and interpolated to obtain an approximation of the decrease in light intensity [Kim and Neumann 2001].

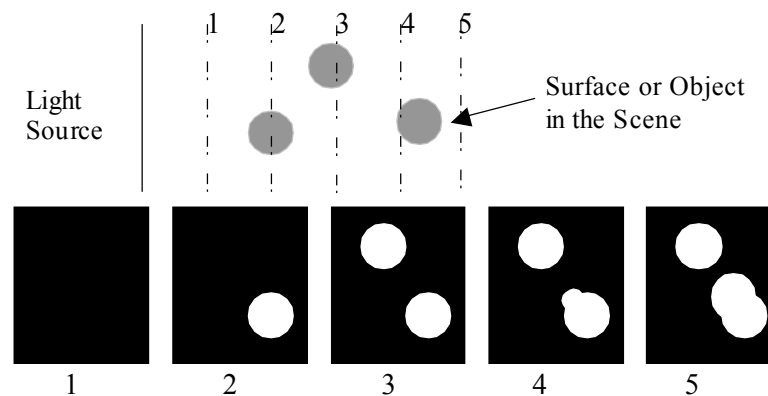


Figure 2.13: Obtaining a set of opacity maps.

Ward et al. [2003] use Kim and Neumann's opacity shadow map technique to shadow hair.

2.3.3.6 Other Shadowing Methods

Chen et al. [1999] add shadowing to hair using a variation of the traditional shadow map method, which they call the *shadow mask method*. Instead of storing depth information (the z value) of the closest points in a shadow map, Chen et al. store the x , y , and z coordinates. A radius is added to the points in the shadow map so that each point represents a sphere in 3D space. To determine if a point is in shadow, its z -value is compared to a range of values in the shadow mask.

Kong and Nakajima [2000; 1999] developed a shadowing technique that uses an opacity buffer constructed from a visible volume buffer to determine shadowing. The volume containing the hair strands is broken up into a number of cubical cells³. An opacity buffer is then constructed which stores the light intensity, or amount of shadow, in each cell. To calculate the values in the opacity buffer, a visible volume buffer is constructed that stores values representing the density of hair strands in each cubical cell. Since shadowing is calculated for cubical cells rather than at every point, the method is not as accurate as other shadowing methods, such as the shadow map methods [Kong and Nakajima 1999]. Kong and Nakajima [2000] believe that the colour of each hair strand should be distinguishable from the colours of adjacent hair strands. To implement the property they calculate the intensity of each hair strand by multiplying the intensity of the cubic cell it is contained in by a factor randomly selected from a range of values.

2.3.3.7 Shadowing Layers

Several researchers model hair with layers. Lengyel [2000], Lengyel et al. [2001], and Papaioannou [2002] model hair with textured shells, while Kim and Neumann [2000] model hair in thin shell volumes. To add the effect of hair self-shadowing, these researchers assume that inner hairs closer to the head are more likely to be in shadow than outer hairs.

Lengyel [2000] and Lengyel et al. [2001] implement the effect of self-shadowing by making the surface colour of the lower layers closer to the head darker than that of the higher layers. Papaioannou [2002] improves their shadowing method for relatively flat layers by rendering the layers in multiple passes. In the first pass, colour and opacity maps are added to each layer. The opacity maps define the distribution and thickness of the hair strands (see Section 2.3.4.4). Then, in subsequent passes the current surface colour of the layer is multiplied with the opacity of the layers above it. However, before the opacity of each layer above is added to the current layer, the layers above are shifted according to the angle of incident light and the distance between the layer and the current layer. Papaioannou also describes a method for approximating self-shadowing by applying noise to the uniform areas of the texture maps to differentiate the individual hairs and create the effect of self-shadowing.

To add the effect of self-shadowing, Kim and Neumann [2000] calculate the amount of shadow in a layer using the amount of shadow in the layer above. For each cell in a layer, the amount of

³ The number and size of the cells is determined by the amount of memory available and the accuracy of the shadows required.

shadow is calculated by adding the amount of shadow in the cell above it to a value that represents the density of the hair in that cell. The value representing the density of hair is proportional to the number of particles in the cell (opacity of the cell) multiplied with a constant representing the thickness of each hair strand.

2.3.4 Texturing

2.3.4.1 Introduction

In addition to illumination and shadowing, texturing may be used to increase the realism of objects in a rendered scene [Watt 1989]. Texturing creates the appearance of surface detail by varying the properties of the surface from point to point [Ebert et al. 1994] when it is not possible, or practical, to model the surface detail explicitly with polygons or other geometric primitives [Foley et al. 1990]. In order to create the appearance of surface detail (i.e. add a texture to a surface) an image, called a *texture map*, is mapped onto the surface. The process is called *texture mapping*, and the image may be a photograph or created by an artist. Alternatively, the texture applied to the surface may be created procedurally [Foley et al. 1990]. A single texture map can be mapped onto the surface, or a smaller texture map may be repeatedly mapped (or tiled) onto the surface [Neyret and Cani 1999]. The subject of texturing is large and there are many methods available for creating and adding textures to a surface. A full assessment of the different texturing techniques available with their advantages and disadvantages is beyond the scope of this project.

Since textures create the effect of detail without the detail having to be modelled explicitly, texturing seems an appropriate method for creating the effect of a large number of hairs on a surface. It is simple to use and requires relatively little rendering time when compared to explicit hair modelling techniques. However, it does not reproduce the complex geometry, shading or dynamics of hair [Kim and Neumann 2000], and therefore does not produce results that are as realistic as those produced by other methods. In this section we discuss techniques that have been developed to create hair or fur textures, and we look at work where researchers have used texturing to help create the appearance of hair or fur.

2.3.4.2 Creating Hair or Fur Textures

Khouas et al. [1998] present a method for creating fur textures. Their method is based on “the modelling of the autocorrelation function of furlike texture and a 2D AutoRegressive synthesis.” The method is able to create textures of fur with different lengths and orientations.

Mao et al. [2000] present a technique to generate hair texture on 3D models of heads. Given a head model, together with a vector field defining the directions of the hair strands, their technique creates a hair texture procedurally on the head model using the *Line Integral Convolution* algorithm. To help a user define and modify a vector field they provide a sketch-based interface.

The Line Integral Convolution algorithm is a flow (or vector field) visualization technique that creates a texture by blurring an image of white noise in the direction of a given vector field. The images produced by line integral convolution contain similarities to photographs of hair. From a distance long fibres in the direction of the vector field can be seen in a line integral convolution image, similar to the strands of hair that can be seen flowing in a certain direction in a photograph of hair. However, when viewed close up, a line integral convolution image contains randomly varying pixel intensities due to the original image being white noise, and a photograph of hair also contains randomly varying pixel intensities due to the complex interaction of light and shadow that occurs between the strands of hair. Due to this similarity, Mao et al. use the line integral convolution algorithm to produce a texture of hair, and suggest that the complex shading calculations that are used to generate realistic hair can be avoided [Mao et al. 2000].

Since the hair texture is generated procedurally at runtime, the texture does not need to be stored or mapped onto the head model. However, it does require a set of vectors representing the direction of hair strands at different points on the head model. The hair textures produced by the technique give the appearance of hair, however, the quality of the hair produced is not very high. The boundary between the hair and skin causes the appearance of hair on the head model to be unrealistic, and the volumetric nature of hair is not captured [Mao et al. 2000].

2.3.4.3 Textured Strips

Textures are used to create the effect of hair, and to avoid explicitly modelling each individual strand of hair. Unfortunately, applying a texture of hair to a head model does not usually produce

very realistic results, even if the hair texture resembles real hair (see the discussion on Mao et al.'s work in the section above). One reason for the unrealistic results is that the overall shape of the hairstyle is not reproduced. Koh and Huang [2001] (see Section 2.2.4.2) use hair textures to create the appearance of clusters of hair strands, so that each individual strand of hair does not need to be modelled, allowing the method to be used to create the appearance of hair in real-time applications. However, instead of applying a hair texture directly to the head model, Koh and Huang [2001] apply hair textures (and alpha maps) to 2D strips or NURBS surfaces that are placed on the head model to create a hairstyle. Several other authors use the method to create hair [Ward et al. 2003; Guang and Zhiyong 2002], and a number of tutorials explaining how to implement the method in various 3D graphics applications using different 2D imaging programs can be found on the web [Lind 2003; Moelzer 2003; Comet 2000; Silas 2000].

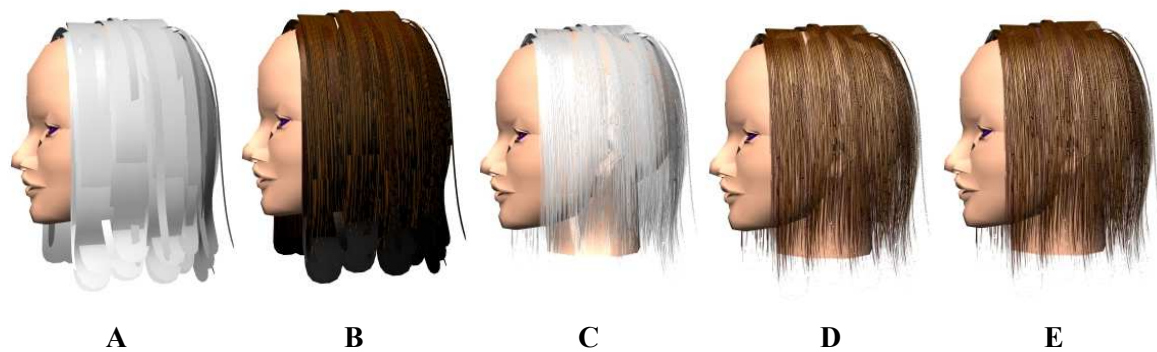


Figure 2.14: *Modelling and rendering hair using textured strips.*

Figure 2.14 illustrates the method. A hairstyle is defined on a 3D head model by positioning a number of 2D strips (in layers) on the head model (A). The 2D strips may have different shapes and sizes [Koh and Huang 2001] and are usually modelled with parametric surfaces (NURBS) because they are easier to work with [Moelzer 2003], but can also be modelled with polygons. Each strip represents a group of hair strands. To create the effect of groups of hair strands when the strips are rendered, texture maps are applied to each strip. Usually two different texture maps are applied to each strip. The first texture map is an image of a hair texture (or colour map) that defines the colour of the hair strands represented by the strip (B). The second texture map is an alpha map, also known as an opacity map, that defines transparency on areas of the strip, and creates the illusion of complex geometry and individual strands of hair (C) [Koh and Huang 2001]. Together the textures create the effect of hair (D). The hair texture and alpha map used to create the diagrams in Figure 2.14 are shown in Figure 2.15. Hair texture maps and alpha maps can be created in a 2D imaging program.

The alpha map can also be used to modify the specularity across the strips [Comet 2000], and as a bump map [Moelzer 2003] to enhance the illusion of individual strands of hair.

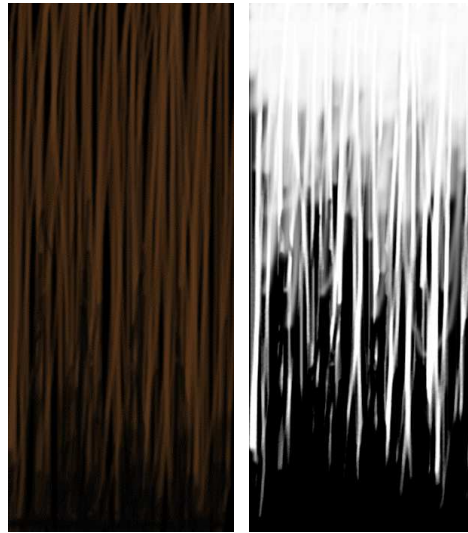


Figure 2.15: *Texture maps.*

Sometimes there may be gaps between the strips of hair, causing parts of the scalp underneath the hair to be seen (see the top of the head in diagram D of Figure 2.14 for an example). To fix the problem, Lind [2003] and Comet [2000] paint (or place) a *cap* onto the top of the head under the hair. The cap is painted (or textured) a dark colour and makes the gaps in the hair less noticeable (see diagram E in Figure 2.14).

The method renders hair faster than other methods that model hair explicitly or with volumetric techniques. However, the hair's volume is not captured by the method [Koh and Huang 2001].

2.3.4.4 Texturing Concentric Layers

Other authors that use textures to create the effect of hair include Lengyel [2000], Lengyel et al. [2001] and Papaioannou [2002]. These authors present techniques to model short hair using concentric shells, which were mentioned in Section 2.2.2.1. To create the effect of hair, texture maps are applied to concentric layers, called shells. However, the texture maps are not images of hair strands, such as those mentioned in the two previous sections, and they do not create the appearance of hair on their own. Instead the texture maps represent slices of 3D space occupied by strands of hair, and when they are applied to a set of layers they create the effect of hair.

The texture maps can be created in a number of ways. Lengyel [2000] and Lengyel et al. [2001] create the texture maps as four-channel (RGBA) images. One image is mapped onto each layer. A set of RGBA images for the layers is created by filtering 3D hair strands into a texel array. The hair strands are created using a particle system and are represented by centrelines with a radius. To filter the hair strands into the texel array, and thus obtain a colour and opacity for each texel, a tall Gaussian filter is used. Each hair strand's centreline is sampled at a high resolution (supersampled) and a weighted contribution is added to each texel [Lengyel 2000].

Lengyel [2000] maps a single texture over the entire surface of each layer. Lengyel et al. [2001] improve the method by repeatedly pasting a smaller texture over each layer, thus reducing the amount of texture memory needed and allowing surfaces with an arbitrary shape to be textured easily. (Lengyel et al.'s textures have a toroidal symmetry so that they can be tiled.) Lengyel et al. [2001] also place textured fins perpendicular to the surface to improve the appearance of the fur near silhouettes. The fin textures are created using the same method that was used for the layer textures. However, the volume containing the 3D hair strands is sampled in a different direction. Alternatively, an artist could also create the textures manually [Lengyel et al. 2001].

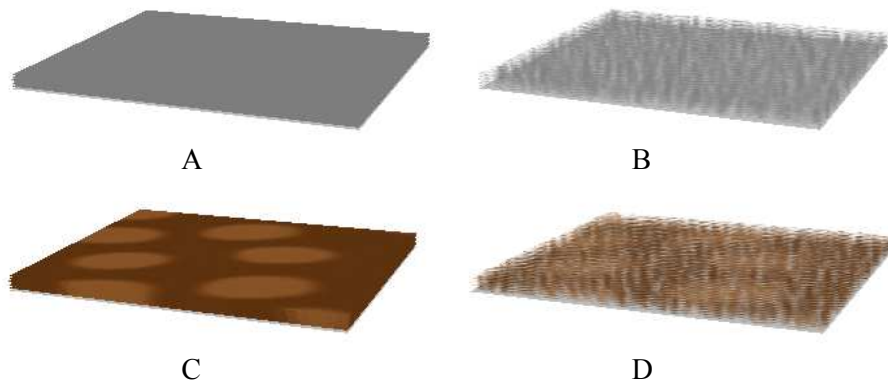


Figure 2.16: *Texturing concentric layers.*

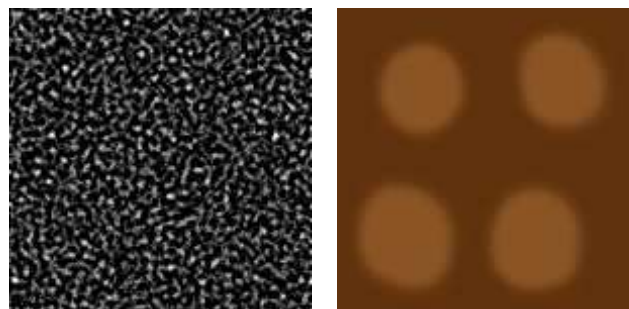


Figure 2.17: *Texture maps.*

Papaioannou [2002] adds two texture maps to each layer. The first, and most important texture map is an opacity (or alpha) map that defines the opacity across a layer, and consequently determines the distribution and thickness of the hair strands. Papaioannou creates the opacity maps by blurring a greyscale image of noise. High intensities in the image represent areas where there is a hair strand. To alter the thickness of the hair strands so that they thin out as they move away from the scalp, Papaioannou creates different opacity maps for the different layers by thresholding the image of noise at different values. The second texture map is a colour map that specifies the colour of the hair strands. Usually, the same colour map is used for each layer, but different colour maps may be applied to the different layers to vary the colour of each hair strand [Papaioannou 2002]. Figure 2.16 illustrates the use of the two texture maps to create the effect of fur. In diagram A, six concentric layers are created to represent fur. Diagram B shows the effect produced when an opacity map is applied to each layer and diagram C shows the effect produced when a colour map is applied to each layer. When both texture maps are applied to the layers the effect of fur is created, as can be seen in diagram D. Figure 2.17 shows the opacity and colour maps used.

2.3.4.5 Examples in Films

Examples of films where texturing is used to create the appearance of hair include *Toy Story* and *Interview with the Vampire*. In *Toy Story*, Darwyn Peachy uses a procedural texture on “solid pieces” to create the effect of curly hair for the baby, Molly; and Eliot Smryl uses painted textures on layers of geometry or cylinders to create the effect of straight hair for characters such as Sid, the neighbourhood bully [Computer Graphics World 1995]. In *Interview with the Vampire*, Kevin Mack and other artists at Digital Domain apply texture maps to modelled hair for the vampire Claudia, whose hair changes from straight to curly in a transformation scene. The texture maps are created in Amazon 3D Paint using paintings and “pieces of the live-action plates” [Robertson 1995].

2.3.5 Anti-aliasing

2.3.5.1 Introduction

Aliasing refers to unwanted artefacts, or image flaws, that occur due to inadequate sampling [Ebert et al. 1994]. Rendered hair often suffers from aliasing, as several thin strands of hair contribute to the shading of each pixel. The left diagram in Figure 2.18 shows an example of a

common aliasing artefact. In the diagram the curves are jagged instead of smooth. Several *anti-aliasing* methods exist to remove or reduce aliasing artefacts. The most common and widely used method is supersampling. Other methods include prefiltering (area sampling) and stochastic sampling [Watt 1989]. The diagram on the right in Figure 2.18 shows the curves with anti-aliasing applied.

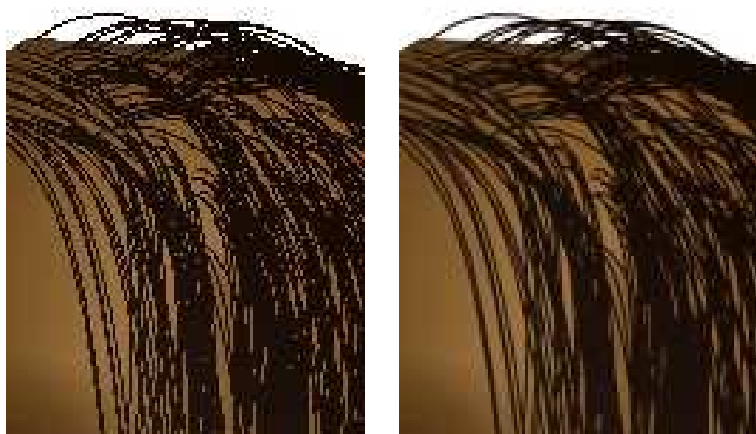


Figure 2.18: *Aliasing and anti-aliasing.*

In this section we examine the anti-aliasing methods that are used by researchers when rendering hair. Normally one of two anti-aliasing techniques is used when rendering hair that has been modelled explicitly, or in clusters where each individual strand of hair is explicitly rendered. The two techniques, namely supersampling and pixel blending, are discussed below. To reduce aliasing in hair that is modelled implicitly or in clusters using 2D or 3D textures, a technique known as mip-mapping is applied. We discuss mip-mapping below, as well as a technique called distance-based culling, which reduces the number of hair strands rendered as the distance between the viewer and the hair increases.

When rendering hair, several researchers rely on graphics hardware for anti-aliasing. For example Ward et al. [2003] use NVIDIA GeForce hardware for multi-sampling and Anjyo et al. [1992] used a VGX graphics board for supersampling. Other researchers, such as Watanabe and Suenaga [1992] who present a trigonal prism method to explicitly model hair, do not consider anti-aliasing at all, while some researchers do not need to consider anti-aliasing as the hair simulation method they use avoids aliasing. For example, Perlin and Hoffert [1989] model hair implicitly using hypertexture, but manually adjust the hypertexture “to be frequency clamped” so that an anti-aliasing technique is not needed.

2.3.5.2 Supersampling

A common, anti-aliasing technique used by several researchers to render hair is supersampling, also known as oversampling or postfiltering. Supersampling involves sampling at a higher resolution than the display resolution. Instead of taking one sample for each pixel, a number of samples are taken and then combined (filtered), usually by averaging [Foley et al. 1990]. The number of samples taken and the shape of the filter used may differ between different supersampling methods [Watt 1989]. Since a number of strands of hair may contribute to each pixel, many samples need to be taken for each pixel (there should be at least as many samples taken for each pixel as the number of hair strands that contribute to each pixel [Neyret 1998]). Supersampling can be computationally expensive when many samples are taken for each pixel, and it has high memory requirements when the samples have to be stored before they are filtered [Watt 1989].

Researchers who use supersampling when rendering hair include Lee et al. [2002], Yu [2001], Bredow [2000], Kim and Neumann [2000], Kong and Nakajima [2000], and Rankin and Hall [1996].

2.3.5.3 Pixel Blending

Since a large number of samples are often needed to effectively render hair with supersampling (making supersampling computationally expensive), LeBlanc et al. [1991] propose another technique for anti-aliasing hair, known as *pixel blending*. The technique is based on an anti-aliasing method known as box filtering (or unweighted area sampling). To render an anti-aliased image, each pixel is considered as a square box (see Figure 2.19), and the intensity of each pixel is computed by blending the intensities of the hair strands and non-hair objects that lie inside the square box.

Using normal box filtering the intensity I of each pixel is calculated using the equation:

$$I = P_B B + \sum P_j H_j$$

where B is the intensity of the background (hairless image), P_B is the portion of the square box not covered by hair, H_j is the intensity of the j^{th} hair strand in the box (calculated using an illumination model), and P_j is the portion of the square box covered by the j^{th} hair strand. Unfortunately, due to strands of hair overlapping, finding the precise value of P_j is extremely expensive [LeBlanc et al. 1991].

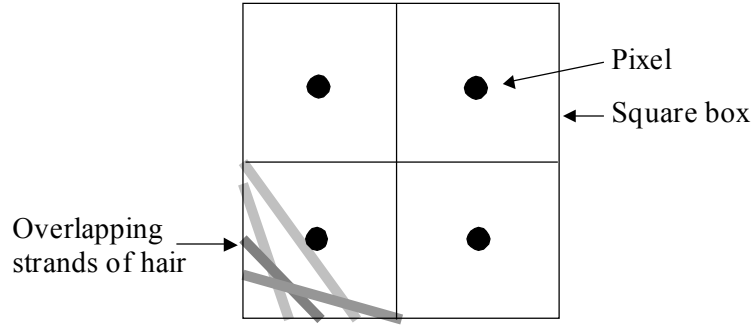


Figure 2.19: *Pixels considered as square boxes.*

LeBlanc et al. [1991] simplify the filtering so that it is less expensive by assuming that “the hair intensity information is distributed evenly” in the square box. The intensity of each pixel is then calculated by blending the intensities of the hair strands in proportion to the area of the square box they cover. Initially the intensity of the pixel is set to the value of the background (hairless image). The intensities of the hair strands are then blended into the pixel in reverse depth order, so that the intensity of the pixel I_n after blending in the intensity of the n^{th} hair strand H_n is:

$$I_n = (1 - P_n)I_{n-1} + P_nH_n$$

where P_n is the proportion of the square box the n^{th} hair strand covers, and I_0 equals the intensity of the background [LeBlanc et al. 1991].

An advantage of the pixel-blending technique is that it can be implemented easily using the alpha channel in existing frame-buffer architectures. The proportion of the square box that each hair strand covers is stored in the alpha channel, and used to blend the hair strand’s intensity with the current intensity of the pixel [LeBlanc et al. 1991].

Several researchers use LeBlanc et al.’s pixel-blending technique to render hair [Chen et al. 1999; Yang and Ouhyoung 1997; Thalmann et al. 1996]. Iones et al. [2000] use a subpixel Z-buffer to render hair, which removes aliasing in a similar manner to LeBlanc et al.’s pixel-blending technique. Kim and Neumann [2002] render anti-aliased strands of hair using a technique similar to pixel blending in which segments of polylines representing strands of hair are ordered using a *visibility ordering algorithm* and then blended using alpha values.

2.3.5.4 Mip-Mapping

Texture maps applied to surfaces in a scene are prone to aliasing artefacts when the resolution of the texture is not well matched to the display resolution (a pixel in the rendered image represents many pixels in the texture image, or a pixel in the texture image is represented by many pixels in the rendered image) [Ebert et al. 1994]. To reduce aliasing artefacts in textures (especially when the distance between the viewer and the textured surface changes), a technique known as mip-mapping is applied. Using mip-mapping, multiple resolutions of the texture are pre-computed by filtering the original texture. The pre-computed texture with a well-matched resolution to the display resolution is then chosen and used during rendering to produce an anti-aliased texture [Neyret 1998].

Researchers who use mip-mapping to render anti-aliased hair include Lengyel et al. [2001] and Lengyel [2000], who create the effect of hair using concentric textured layers. Neyret [1998; 1995] extends Kajiya and Kay's 3D texture (texel) model (see Section 2.2.2.1), which is able to create the illusion of fur, by developing a technique based on mip-mapping that is able to obtain multi-scale representations of a texel.

2.3.5.5 Distance-Based Culling

Unfortunately, when applying an anti-aliasing technique, blurring may appear in the image as a result of filtering [Watt 1989]. Van Gelder and Wilhelms [1997] find that strands of fur tend to become a smooth surface (as a result of blurring) when viewed from a distance. To solve the problem they automatically adjust the number of fur strands rendered, depending on the distance of the fur from the viewer, using a technique known as distance-based culling.

2.4 Summary

In this chapter we discuss techniques that have been developed to simulate hair, focusing on hair modelling and rendering techniques. We find that many techniques have been developed and some impressive results have been produced, which can be seen in several applications including film production. Commercial hair simulation tools are becoming more widely available to artists, and several researchers have developed interactive systems with user interfaces for their techniques. However, there is no one superior technique, and most techniques work well only in

certain situations. We do not find any techniques that are developed specifically for African hair or hairstyles.

The hair modelling techniques can be divided into three categories: those that model hair implicitly, those that model hair explicitly, and those that use a combination of implicit and explicit techniques to model hair as clusters. The hair rendering techniques discussed include illumination models, shadowing techniques, anti-aliasing methods and texturing.

3 MODELLING AFRICAN HAIR

Techniques for simulating African hair are required so that African hairstyles can be added to virtual humans. The aim of this thesis is to present hair modelling and rendering techniques that will assist a user in creating a variety of African hairstyles with ease. We identify three forms of hair commonly found in African hairstyles and develop techniques to model and render these different forms of hair: natural curly hair, straightened hair, and braids or twists of hair. We leave it up to the user of our techniques to create a hairstyle. This gives the user the flexibility to create the exact hairstyle they want, and allows a variety of different hairstyles to be produced with the same technique.

This chapter considers our development or enhancement of hair modelling techniques in order to model the different forms of African hair. Each technique presented provides the user with a way of placing hair onto a head model and of representing the shape and geometric properties of the form of hair being modelled. The techniques are also able to overcome the problem of modelling the large number of hair strands required.

3.1 Natural Curly Hair

Natural African hair is short and curly, and many African people keep their hair in hairstyles consisting of the short, curly form of hair. Figure 3.1 shows examples of African hairstyles with the natural curly hair. In order for a user to create the types of hairstyles shown in Figure 3.1, we need to develop a hair model that is able to produce the characteristic of short, curly hair effectively.



Figure 3.1: *Examples of African hairstyles with natural curly hair.*

The review of hair modelling techniques in Section 2.3 indicates that there are many different techniques available, and that the techniques can be categorized as implicit, explicit or cluster models depending on the way they represent hair. We choose to use an implicit hair modelling technique to model the natural, curly form of African hair. An implicit hair modelling technique seems the most appropriate technique to use for two reasons. Firstly, the hair is very curly. Modelling straight hair explicitly or in clusters is expensive due to the large number of polygons or line segments required. Curly hair requires many more polygons or segments than straighter hair, making the explicit and cluster models even more expensive. Secondly, a hairstyle comprised of natural curly hair tends to appear as a layer of hair over the scalp. The layer of hair has a constant thickness and pattern, analogous to a layer of fur, and since fur is successfully modelled using implicit hair modelling techniques (see Section 2.2.2), an implicit model seems appropriate for modelling short, curly African hair.

3.1.1 Concentric Textured Layers

Our implicit model is based on Lengyel's concentric shell method [Lengyel 2000] (see Section 2.2.2.1), which creates the illusion of fur. Our model represents hair with concentric textured layers that can be thought of as slices of 3D texture. The layers are created parallel to the scalp one above another. In order to create the effect of hair, an opacity map is applied to each layer. The opacity map controls the opacity across the layer and defines what part of the layer represents hair and what part does not. Opaque areas on the textured layer represent slices of hair strands, while transparent areas represent spaces in between the hair strands. Consequently, the opacity maps control the characteristics of the hair being modelled. Texture maps are also applied to the layers to control the illumination of the hair. When many layers are placed close to one another they produce the effect of hair. The approach of applying opacity maps to layers is illustrated in Figure 3.2. To generate the pre-processed opacity maps we use a different method to Lengyel [2000] (see Section 5.4.1). However, both methods produce equivalent results. Figure 3.3 shows straight hair and curly hair represented by textured layers.

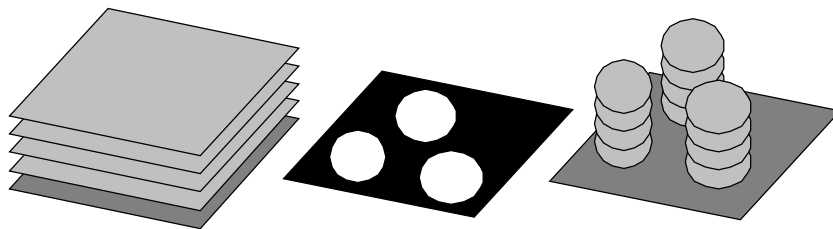


Figure 3.2: *The result of an opacity map applied to concentric layers.*

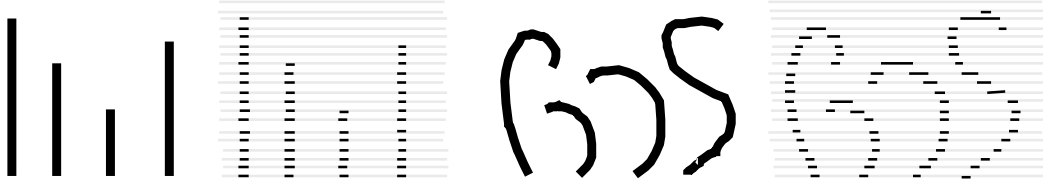


Figure 3.3: *Hair represented by textured layers.*

3.1.2 Placing Layers on a Head Model

One of the requirements for our proposed hair models is to provide the user with a method for placing hair onto a model of a head. We therefore need a way of placing the textured layers that represent short curly African hair onto the scalp of a head model.

We first need to ascertain the region on the head model where the user would like hair to be placed. The user defines the region by selecting the polygons on the head model on which they would like hair to be placed (we assume the model of the head is a polygonal structure). We then create layers on which textures can be placed, above the selected polygons. The number of layers and their height are obtained from the user, allowing the user to control certain properties of the hair. This allows a user to create different hairstyles consisting of natural curly hair using the same modelling technique.

The layers are created by displacing copies of the selected polygons' vertices in the direction of their normal. The displaced vertices are then connected in the same manner as the vertices of the selected polygons on the head model, and each new polygon is given texture coordinates that are the same as the texture coordinates of the selected polygon below it. More precisely, for each vertex i on layer k

$$V_i^k = V_i^0 + (k * h / n) * N_i^0$$

where V_i^0 is the vertex position of the vertex on the head model, N_i^0 is the vertex normal and h and n are the height and number of layers defined by the user (see Figure 3.4) [Lengyel 2000].

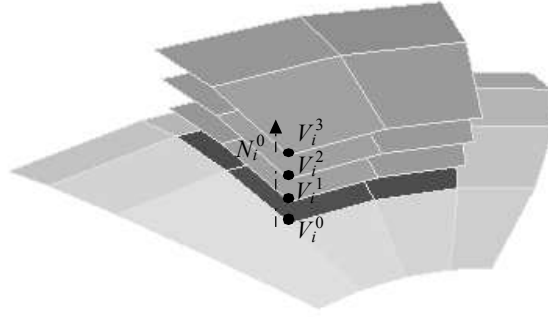


Figure 3.4: *Placing layers over selected polygons.*

3.1.3 Using Opacity Maps to Define Hair

Once layers of geometry have been created in order to place hair on a head model, opacity maps must be applied to the layers in order to define the characteristics of the hair. These opacity maps are able to define the shape and thickness of the strands of hair, as well as specify the distribution of hair on the head model (see Figure 3.5). Since the geometry of each individual strand of hair is not explicitly modelled and an opacity map is able to control the distribution and number of hairs placed on the head, the problem of a large number of hair strands is not an issue in this case.

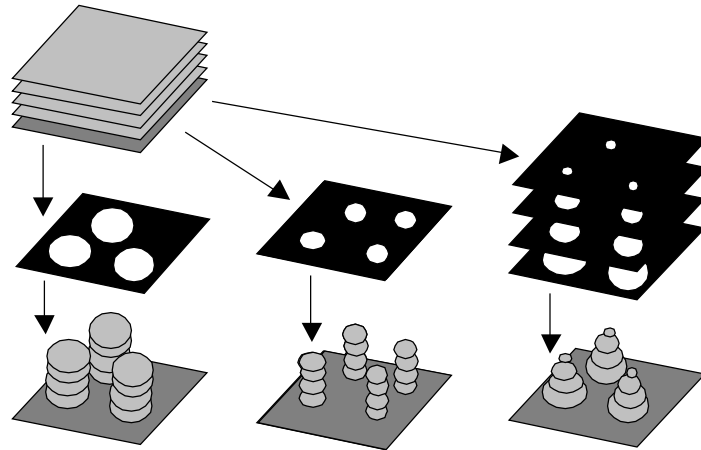


Figure 3.5: *Using opacity maps to define the shape, thickness and distribution of hair strands.*

To produce the opacity maps we generate random curly splines in 3D space, and then take slices of the 3D space at specific heights. The number of slices and the height at which each slice is taken depends on the number and height of the layers used. Each slice is then used as an opacity map. The regions where splines exist are shown as white areas on the opacity map, and represent hair, while regions where no splines exist are represented with black. At the same time we

generate *tangent maps*, which are used in our illumination model (see Sections 5.2.4 and 5.4.1). Our method differs from that of Papaionnou [2002] who creates opacity maps from greyscale images of noise (random intensity values). Papaionnou's method is not able to produce curly hair. Lengyel [2000] creates the opacity maps by growing hair with a particle system. His method is able to produce curly hair, and could have been used instead of our spline method, as could any other equivalent approach.

3.2 Straight Hair

African people often straighten their naturally curly hair using a process called *relaxing*. Figure 3.6 shows examples of African hairstyles with hair that has been straightened. The literature review in Chapter 2 shows that implicit hair modelling techniques are not appropriate for modelling long hair, and cluster methods are better at modelling hairstyles where hair is grouped into clusters rather than smooth hairstyles. We therefore choose to model straight hair in African hairstyles explicitly. However, the literature review also shows that an explicit method may be computationally expensive and modelling hair may be tedious. Computing power is increasing, and techniques such as the use of key hairs can be used to decrease the amount of time spent on hair modelling. We investigate the effectiveness of the explicit model for modelling smooth hairstyles containing straight hair.



Figure 3.6: Examples of African hairstyles with straight (relaxed) hair.

3.2.1 Using Curves to Define a Hairstyle

The problem of a large number of hair strands is accentuated when each individual strand of hair has to be modelled. It is therefore important that we find a way of overcoming this problem if an explicit model is to be used to model straightened African hair.

We overcome the problem by using control hairs to define a hairstyle. Our approach is similar to that taken by Daldegan et al. [1993] in their interactive hair modelling system, called HairStyler (see Section 2.2.3.2). A number of control hairs are placed on the head model. These control hairs are used to describe a hairstyle, and to define the shape and geometric properties of hair strands in their area on the head model. Since we would like a user to be able to use our hair model to define a variety of hairstyles, the control hairs are represented by curves, which are easy to bend and modify, and which are a close representation of strands of hair in reality. To create a full head of hair more curves are automatically defined and placed on the head model using the shape and geometric properties of the control hairs (also called control curves) to define the properties of the new curves (see Figure 3.7). By using a few control hairs that define a hairstyle, many strands of hair can be modelled automatically. This reduces the amount of time a user has to spend on modelling a hairstyle, and allows us to overcome the problem of modelling a large number of hair strands.

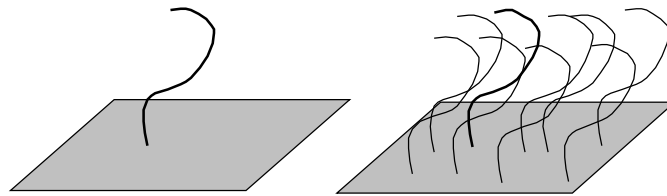


Figure 3.7: *Automatically creating curves using a control curve.*

3.2.2 Placing Curves on a Head Model

To fulfil the requirements of our hair model a way of placing the curves that represent strands of straight hair onto a model of a head is required. This section describes the techniques we use to place control curves onto a head model and to generate additional curves that are required, automatically.

As with our implicit model, we assume that the head model is a polygonal structure, and we ask the user to select the polygons on the head model where they would like hair to be placed. Our method for automatically placing strands of hair onto a head model (discussed later in this section) requires the selected polygons on the head model to have the same number of sides. Since a polygon with any number of sides can be converted into a number of triangles, and triangles always lie in a single plane, we convert all selected polygons that have more than three sides to triangles. Once a region on the head model where the user would like hair to be placed is defined, and the region is made up of triangles, control hairs can be placed onto the region.

To place control hairs onto a selected region of the head model we place a single curve (that represents a control hair) in the centre of each triangle that makes up the region. Since the scalp on a head is not flat, more than one polygon is needed to model the top of a head. However, the scalp does not contain complex geometric detail, and therefore does not require a very large number of polygons to model it. The selected region therefore consists of approximately between 10 and 40 polygons (converted into triangles), which is appropriate for the number of control curves needed. The polygons are also usually of similar size, so the control curves are approximately evenly spaced over the selected region. The curves placed on the head model need an initial direction. We place each curve in the direction of the normal of the triangle on which it is placed. The length of the curve and the number of control points that constitute the curve are obtained from the user. Excluding the end points, at least one control point is needed in order to bend the curve. Increasing the number of control points provides greater control over the bends in the curve, but increases the amount of work needed to model a hairstyle. We allow the user to decide how many control points are needed per curve.

Once control curves have been added to the head model, the user may bend and modify the curves in order to create a desired hairstyle. A technique is needed to create a full head of hair once the user is satisfied with the hairstyle defined by the control curves. The technique should generate curves and place them onto the head model automatically. To model the curves automatically, we make a number of copies of each control curve. The copies are placed on the head model by placing them randomly on the same triangle as the original control curve from which they were copied. They are also placed in the same direction as the original control curve. The number of copies of each control curve is obtained from the user. By changing the number of copies, a user is able to define the density of the hair in the hairstyle.

3.2.3 Representing a Strand of Hair

Curves are a close representation of strands of hair in reality, and are therefore useful for modelling hair. However, it is not always possible to render them (depending on the application being used). We therefore use curves to help a user define a hairstyle, but convert them to alternative geometry before rendering.

Originally we considered representing each strand of hair as a generalized cylinder. A generalized cylinder is an approximation of a curved cylinder and is a more accurate representation of a strand of hair than a chain of straight segments. However, a generalized

cylinder requires an extremely large number of very small polygons in order to represent a strand of hair accurately. The number and size of the polygons required to represent hair using generalized cylinders is impractical, and so an alternative representation is needed.

The results of Van Gelder and Wilhelms' [1997] research show that polylines (a chain of straight segments that approximate a curve) are an appropriate method for representing hair effectively and efficiently. Although they do not provide the most accurate representation of a strand of hair, they provide a reasonable appearance at an acceptable rendering rate. We therefore provide a technique for converting each curve that represents a strand of hair into a polyline.

3.3 Braids and Twists of Hair

Many common African hairstyles contain hair that has been styled into the form of a braid or twist. Figure 3.8 shows two African hairstyles. The first hairstyle contains a braid of hair and the second hairstyle contains twists of hair.



Figure 3.8: *African hairstyles with hair in the form of a braid and twists.*

We develop a cluster hair modelling technique to model hair in the form of braids and twists. Cluster models model clumps of hair rather than each individual strand of hair (see Section 2.2.4). The average scalp contains between 100,000 and 150,000 strands of hair [LeBlanc et al. 1991]. These strands of hair are usually styled into approximately 20 to 30 braids or twists in an African hairstyle (see Figure 3.9). By using a cluster model, only the shape of the wisps of hair that form braids and twists are modelled, rather than each individual strand of hair. This reduces the complexity and time required to model braids and twists of hair.

3.3.1 Using Curves to Define Braids and Twists in a Hairstyle

Our cluster model uses generalized cylinders to model the wisps of hair that make up a braid or twist of hair. Since only a small number of braids or twists are commonly found in an African hairstyle, and a wisp of hair is much thicker than an individual strand of hair, it is possible to represent each wisp of hair in a braid or twist with a generalized cylinder. Curves are placed onto a head model to define the braids or twists in a hairstyle. Curves are easy to bend and modify, and a single curve is able to specify the overall shape of a braid or twist of hair. These curves are then converted into a number of curves where each curve defines a wisp of hair in a braid or twist. Our model allows a user to specify several characteristics of the braid or twist, such as the number of wisps of hair that constitute it. The new curves are then converted to generalized cylinders and texture maps are applied to the generalized cylinders to create the effect of a cluster of individual strands of hair. The user does not have to worry about the path of each cluster, or how a braid or twist is made. Many strands of hair are modelled automatically. This reduces the amount of time a user has to spend on modelling a hairstyle, and allows us to overcome the problem of modelling a large number of hair strands.

3.3.2 Placing Curves on a Head Model

A variety of different hairstyles can be created with braids and twists. We divide the different hairstyles into two groups in order to place curves that represent the braids or twists effectively on a head model. The first group consists of braids that protrude from the scalp (see the first two hairstyles in Figure 3.9). In order to place curves on the head model that represent braids or twists from these hairstyles, we use the same method as the control curve method for straight hair discussed in Section 3.2.2. The user is able to bend the curves in order to define the shape of each braid or twist, and if more curves are needed they can be modelled automatically using the control curves. The curves are initially placed perpendicular to the scalp (in the direction of the scalp polygons' normals). Since the braids and twists in this group of hairstyles tend to protrude from the scalp, the method of initially placing the control curves perpendicular to the scalp works.



Figure 3.9: Braided (twisted) hair in a variety of styles.

The second group of hairstyles are commonly called *cornrow* hairstyles and they contain braids that lie parallel to the scalp (see the last two hairstyles in Figure 3.9). For these hairstyles, using the control curve method in Section 3.2.2 is not an effective way of placing curves on a head model. The braids lie parallel to the scalp, making it hard for the user who has to bend the curves so that they do not appear to be protruding from the scalp. Also, the braids tend to start at the front of the head and end at the back of the head. Therefore, the starting points of each braid are not evenly distributed around the head, so placing a single curve in each triangle that makes up the selected region of the scalp is not an appropriate way of placing the curves on the head model. We therefore use a different method of placing curves on a head for the cornrow hairstyles. Since the hairstyles do not usually consist of a large number of braids, and therefore a large number of curves are not required, we manually place curves across the scalp.

3.3.3 Creating Braids and Twists with Generalized Cylinders

Once curves have been placed on the head model, and the user has bent them so that they represent the overall shapes of individual braids or twists of hair, the curves need to be converted into a user-defined number of curves, where each curve represents a wisp of hair in the braid or twist.

3.3.3.1 Braids and Twists

We define the shape of a curve in a twist by the parametric equation:

$$x = r \cos(\theta)$$

$$y = r \sin(\theta)$$

$$z = k\theta$$

where θ is an angle, $\theta \geq 0$, and $r, k \in \mathbb{R}$, $r \geq 0$.

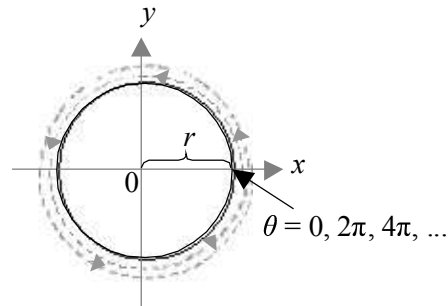


Figure 3.10: *Cross section of a twist.*

The z -axis represents the curve that is being converted into a twist. Figure 3.10 shows a cross section of a twist (the viewer is looking along the z -axis). As the angle θ increases, the curve twists around the z -axis (or original curve defining the twist of hair), and at the same time moves along the z -axis, as shown in Figure 3.11.

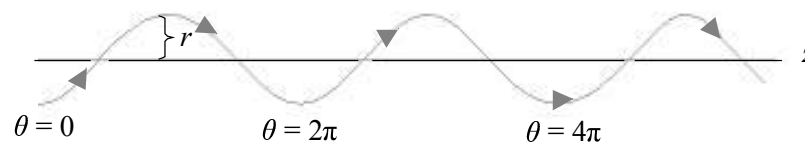


Figure 3.11: *The shape of a curve in a twist along the z -axis.*

The “speed” at which the curve moves along the z -axis determines the amount of twist in the curve. In other words the smaller the increase along the z -axis with respect to the increase in θ , the more times the curve twists around the z -axis, and the larger the increase along the z -axis with respect to the increase in θ , the less twist the curve has. This is controlled by k . Smaller values of k will result in more twist than larger values of k . The radius r determines the thickness of the twist the curve creates. Figure 3.12 shows examples of a curve twisting around another curve from different angles.

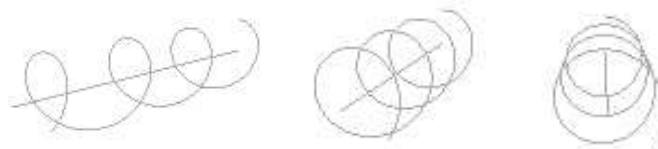


Figure 3.12: *A curve in a twist viewed from different angles.*

The curve is twisted around the z -axis, which represents the original curve defining the twist of hair. However, the z -axis represents a straight line, and a curve defining the twist of hair is unlikely to be a straight line. In order to solve the problem we calculate each point along the twisting curve using a reference frame that consists of three perpendicular vectors. The reference frame represents the x -, y -, and z -axes, and is positioned in 3D space so that the vector representing the z -axis lies in the direction of the tangent of the original curve at each point. We discuss reference frames in more detail in the next section when we look at generalized cylinders.

To create a twist of hair a few wisps of hair are usually twisted together. We represent each wisp of hair as a twisting curve. Each curve has the same radius r and is defined by the same parametric equation. However, we add an offset to the angle θ for each curve, so that the curves twist at different positions along the original curve. Figure 3.13 shows four examples of a twist. The value of θ differs for each of the three curves making up the twist by 0 , $2\pi/9$, $4\pi/9$ and $6\pi/9$ radians (or 0° , 40° , 80° and 120°) in each example respectively.

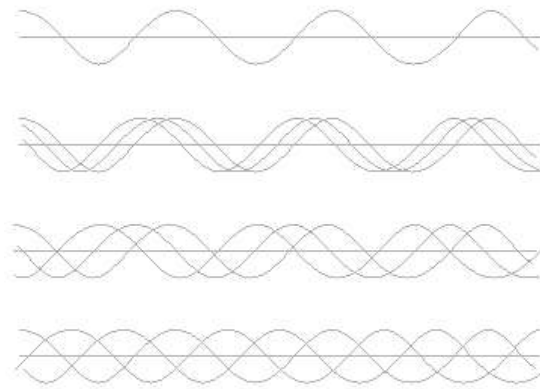


Figure 3.13: *Different starting values of θ for each curve.*

In order for different twists to be created according to the needs of the user, we obtain the number of twisting curves making up the twist from the user. The radius, amount of twist, and difference in the starting angle between curves are also obtained from the user.

The same method is used to define a braid of hair rather than a twist of hair. However, we use a different parametric equation to define the shape of the curves. The shape of each curve is defined by the parametric equation:

$$\begin{aligned}x &= r \cos(\theta) \\ y &= r \sin(2\theta) / f \\ z &= k\theta\end{aligned}$$

where θ is an angle, $\theta \geq 0$, and $f, r, k \in \mathbb{R}$, $r \geq 0$ and $f \geq 1$.

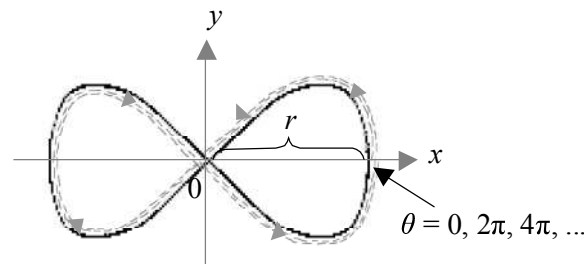


Figure 3.14: *Cross section of a braid.*

Figure 3.14 shows a cross section of a curve in a braid (viewed along the z -axis), and Figure 3.15 shows a curve in a braid twisting around the z -axis.

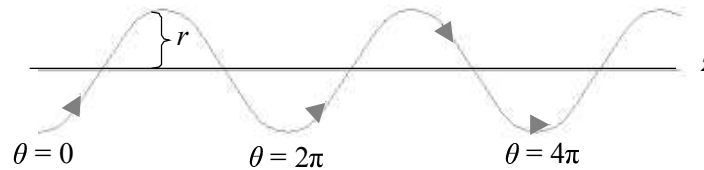


Figure 3.15: *The shape of a curve in a braid along the z -axis.*

We introduce the constant f into the parametric equation for a braid so that a user is able to control the flatness of the braid. A value between 1.0 and 3.0 for f gives the best results. Figure 3.16 shows examples of a braid with the value of f set to 1.0, 2.0 and 3.0 respectively.



Figure 3.16: *Different flatness.*

3.3.3.2 Generalized Cylinders

Each curve that defines a wisp of hair in a braid or twist is converted into a generalized cylinder on which textures are placed to create the effect of a cluster of hair strands.

A generalized cylinder is a cylinder produced by placing cross sections of the cylinder along a 3D curve (see Figure 3.17) [Bloomenthal 1990]. We use a circular cross section, however another shape such as an ellipse could also be used.

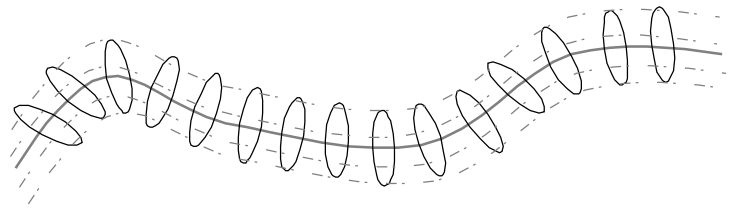


Figure 3.17: *Cross sections of a cylinder placed along a curve.*

To align the cross sections correctly, so that they are perpendicular to the curve, we use reference frames. A reference frame consists of three perpendicular vectors and a point, and is used to define position and orientation. Our reference frame at a point p on the curve consists of a vector T , representing the tangent of the curve at p , a normal N , and a binormal B (see Figure 3.18). The vectors N and B lie on the same plane as the cross section, perpendicular to the curve.

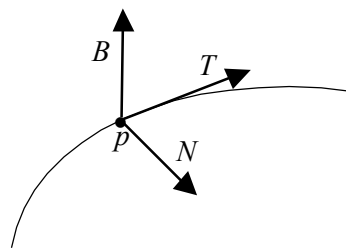


Figure 3.18: *A reference frame.*

We use a reference frame similar to the Frenet frame [PlanetMath 2003]. Given a space curve:

$$p = \gamma(t)$$

where t is the parametric position of a point p on the space curve, the vectors T , N and B making up the Frenet frame at point p are calculated as:

$$T = V/|V|, \text{ where } V = \frac{d}{dt} \gamma(t)$$

$$N = K/|K|, \text{ where } K = \frac{d}{dt} T$$

$$B = T \times N \text{ [PlanetMath 2003].}$$

We use a different method to calculate the vectors, assuming it is not always possible to obtain the equation for the curve. The reference frames are calculated only at known points along the curve (the curve's control points and other calculated points (see Section 4.2.7.1)). The tangent vector T is obtained by normalising the calculated tangent, CT , at point p :

$$T = CT/|CT|, \text{ where } CT = p_{(i+1)} - p_{(i-1)}$$

$p_{(i+1)}$ and $p_{(i-1)}$ are the closest known points on the curve before and after point p . The vectors N and B are calculated by rotating the frame defined by OT , ON , and OB , where:

$$OT = (0.0, 1.0, 0.0)$$

$$ON = (1.0, 0.0, 0.0)$$

$$OB = (0.0, 0.0, 1.0)$$

so that OT is in the same direction as T . The angle α , between T and OT , is defined by:

$$\alpha = \cos^{-1} ((T \cdot OT)/(|T| |OT|))$$

N and B are computed by rotating ON and OB by the same angle α , about the axis:

$$(T \times OT)/(|T| |OT|)$$

The new directions of ON and OB define the vectors N and B (see Figure 3.19).

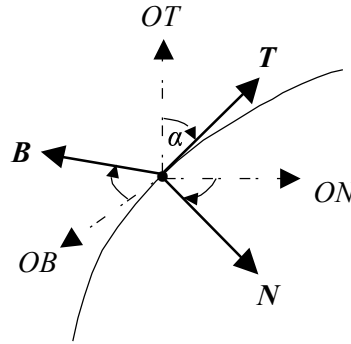


Figure 3.19: Calculating a reference frame.

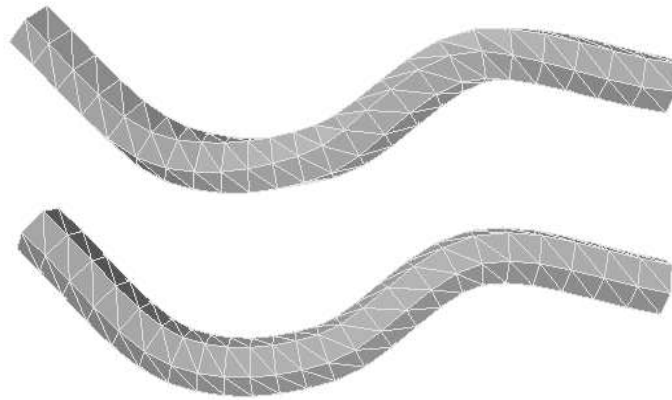


Figure 3.20: A generalized cylinder with and without twisting.

Unfortunately, calculating reference frames using either of the methods above for each control point along the curve causes a twisting problem (see Figure 3.20). To solve the problem, rotation minimizing frames can be used. An initial reference frame is defined at the beginning of the curve and then propagated along the curve using small local rotations [Bloomenthal 1990]. We use our method above to calculate the reference frame for the first control point on the curve, after which we use the reference frame generation method by Ken Sloan, discussed by Bloomenthal [1990]. Instead of using the vectors OT , ON , and OB to calculate each frame, the binormal vector B_0 from the previous frame is used. We calculate the tangent vector T . The vectors N and B are then calculated using the equations below:

$$N = B_0 \times T$$

$$B = T \times N \text{ [Bloomenthal 1990].}$$

Once the reference frame for a point p has been obtained, the cross section of the generalized cylinder at that point can be defined on the plane described by N and B . If a point $c = (c_x, c_y)$ is a point on the 2D cross section of the generalized cylinder, it can be converted to a 3D point on the surface of the generalized cylinder as:

$$(p_x + c_x N_x + c_y B_x, p_y + c_x N_y + c_y B_y, p_z + c_x N_z + c_y B_z) \text{ [Bloomenthal 1990].}$$

Since we use a cylindrical cross section:

$$c_x = r * \cos(\theta)$$

$$c_y = r * \sin(\theta)$$

where r is the radius of the generalized cylinder and θ , dependent on the number of sides of the generalized cylinder, determines the position of the point c on the circumference of a circle. The radius and number of sides making up a generalized cylinder are user defined. Therefore, the user is able to specify the thickness of the generalized cylinder (cluster of hair) and control the number of sides, and therefore the number of polygons, making up the generalized cylinder.

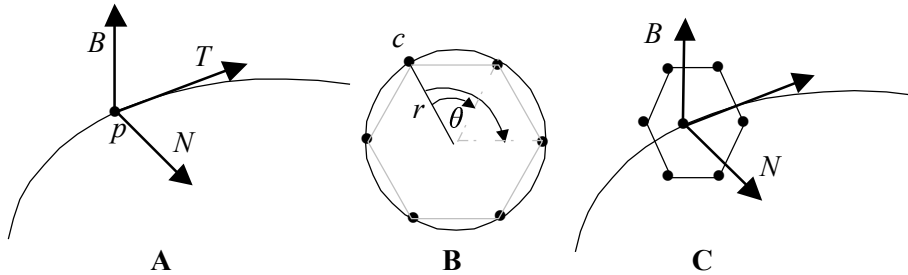


Figure 3.21: Aligning a cross section perpendicular to a curve.

Figure 3.21 shows a cross section of a generalized cylinder (B) aligned perpendicular to a curve (C) using a reference frame (A). The generalized cylinder is approximated by six sides.

3.3.4 Representing Strands of Hair

Texture maps can be applied to the generalized cylinders that represent clusters of hair to create the effect of many strands of hair. The texture maps may be images of hair strands, or they may be generated procedurally. We discuss the texture maps in more detail in Section 5.4.2.

3.3.5 Additional Texturing

Sometimes flatly parted hair on the scalp can be seen in African hairstyles that contain braids and twists. This hair is not part of the braids and twists. For example, parted hair on the scalp can be seen in the first and third hairstyles in Figure 3.9. In the first hairstyle, the hair on the scalp is parted into diamond shapes that tessellate over the scalp, and a single twist of hair protrudes from the centre of each “diamond”. In the third hairstyle, the hair has been parted into parallel strips to create a number of braids, forming a cornrow hairstyle. However parted strips of hair that are not in the braids can be seen on the scalp. In the second and fourth hairstyles, only braids can be seen. To create the effect of parted hair that can be seen between braids and twists in a hairstyle, we place a cap on the scalp. Our method is inspired by the cap technique [Lind 2003; Comet 2000] discussed in Section 2.3.4.3. Lind [2003] and Comet [2000] place dark coloured caps onto the scalp to make gaps between textured strips less noticeable. However, instead of placing darkly coloured caps onto the scalp, we place texture maps representing tessellated patterns of hair. The texture maps are discussed in more detail in Section 5.4.3.

3.4 Summary

In this chapter we discuss the hair modelling techniques we use in order to model the three forms of African hair we identified in Chapter 1. We use an implicit hair modelling technique based on Lengyel’s concentric shell method to model the natural, curly form of African hair, an explicit hair modelling technique that uses control hairs similar to an approach taken by Daldegan et al. [1993] to model straight African hair, and we develop a cluster hair modelling technique, using textured generalized cylinders, to model hair in the form of braids and twists. Our techniques do not create an entire hairstyle. Instead, they are designed so that they can be used by an artist (or user) to create the hairstyle they want, and so that a variety of different hairstyles can be

produced with the same technique. Each technique provides the user with a way of placing hair onto a model of a head and uses an implicit, explicit or cluster hair modelling technique to define the shape and geometric properties of the form of hair it is modelling. Ways of overcoming the problem of modelling the large number of hair strands required are also discussed for each technique. The implementation of the three hair modelling techniques is discussed in the next chapter.

4 IMPLEMENTING PLUG-INS

To implement our hair modelling techniques we develop several plug-ins. Using a combination of these plug-ins, a user is able to model a variety of African hairstyles interactively. In this chapter we describe the plug-ins and some of the implementation issues involved in creating the plug-ins, which are implemented for LightWave 3D.

4.1 LightWave 3D

LightWave 3D (or LightWave) is a graphics program used for modelling, animating and rendering. Many applications today make use of virtual humans that are created and animated in LightWave, or a similar program, and most of these programs have an architecture that supports plug-in development. A plug-in is a dynamically linked library (.dll) that extends the functionality of a program. Plug-ins are imported into a system and become commands that can be used like any other command in the program. Instead of implementing an entire system that imports head models in different formats, has a user interface, and provides tools to perform a variety of operations, we use an existing program (in this case LightWave) and then create plug-ins that add to the functionality of the program so that the three forms of African hair can be modelled with ease to create a variety of hairstyles.

LightWave plug-ins are grouped into different categories, called *classes*. The different classes perform different tasks and are placed in LightWave at different points. The plug-ins created to model hair belong to either the *Modeler Command Sequence* class or the *Mesh Edit Tool* class. Both Modeler Command Sequence plug-ins and Mesh Edit Tool plug-ins create and modify geometry. However, Modeler Command Sequence plug-ins are able to issue commands, which usually correspond to actions the user could execute through the LightWave interface, while Mesh Edit Tool plug-ins allow a user to create and modify geometry interactively. Both classes have access to mesh editing functions, which create, modify and obtain information about the points and polygons of the geometry. LightWave also provides scan functions to enumerate points and polygons allowing information to be obtained or geometry to be modified. For each point or polygon, they reference a callback function, which is supplied by the programmer [Wright 2001].

4.2 Hair Modelling Plug-ins

We implement seven plug-ins (that all have names starting with “HP” for “Hair Plug-in”) in order to model the three forms of hair found in common African hairstyles:

- *HPLayer* and *HPCurve* are plug-ins that place layers or control curves on a head model over the polygons on which a user wishes to place hair. *HPLayer* creates the layers needed to model curly hair implicitly (see Section 3.1.2), while *HPCurve* creates the control curves used to define a hairstyle of straight hair, or braids or twists of hair (see Sections 3.2.2 and 3.3.2 respectively).
- *HPDuplicate* is used to create more curves automatically, based on the control curves that define a hairstyle. This plug-in is used for modelling explicit strands of straight hair (see Section 3.2.2). It can also be used when more control curves are needed to define braids or twists of hair (see Section 3.3.2).
- *HPTwist* and *HPBraid* convert curves into twists and braids of many curves (see Section 3.3.3.1)
- *HPPolyline* and *HPGeneralizedCylinder* are plug-ins that convert curves into polylines and generalized cylinders that can be rendered. *HPPolyline* is used when modelling explicit strands of straight hair (see Section 3.2.3), while *HPGeneralizedCylinder* is used to represent clusters of hair in a braid or twist (see Section 3.3.3.2). *HPGeneralizedCylinder* could also be used to model explicit strands of straight hair. However, we found that representing each strand of hair with a generalized cylinder is expensive in terms of computation time and memory usage.

The details of each plug-in, as well as implementation issues considered when creating each plug-in, are discussed below.

4.2.1 HPLayer

In Section 3.1 we describe a technique for modelling the short curly form of African hair using concentric textured layers. To implement the technique we develop a plug-in, which we call *HPLayer*. The *HPLayer* plug-in creates the concentric layers of geometry on which textures can be placed to produce the effect of hair. The plug-in does not, however, create the texture maps or place them on the layers (the texture maps are discussed in Section 5.4.1). In order for a range of hairstyles to be created using the plug-in, the number and height of the layers are obtained from the user.

4.2.1.1 Implementation

The HPLayer plug-in is designed to create and place layers above selected polygons. In order to create the layers the plug-in must create geometry similar to the selected polygons and then displace the geometry by a certain distance from the surface. Since the thickness and quality of hair required by a user may vary, the number of layers created and their displacement from the surface are user-defined. The user is also asked to provide a surface name, which the plug-in assigns together with successive numbers to each layer.

To create the layers, the plug-in could make copies of each selected polygon and then displace the copies in the direction of the polygon's normal. This method works when the surface is flat. However, when the surface is bent (such as the surface that represents the top of a head), the normals of the selected polygons do not have the same direction, and gaps form between the polygons that make up the layers (see Figure 4.1).

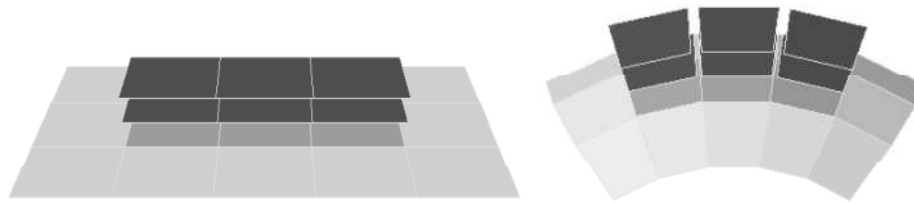


Figure 4.1: *Creating layers, using polygon normals for displacement.*

To solve the problem, the HPLayer plug-in makes copies of the vertices that make up the selected polygons, and then displaces the points (copies of the vertices) in the direction of their normal. Connecting the displaced points in a manner similar to the vertices from which they were copied creates new polygons, and the new polygons form layers without the undesired gaps (see Figure 4.2).

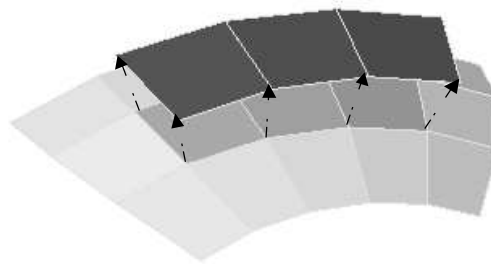


Figure 4.2: *Creating a layer, using vertex normals for displacement.*

It is important to make sure that polygons share vertices (i.e. that there are no duplicate points) in order to apply smooth shading to each layer.

4.2.1.2 LightWave Implementation Issues

The HPLayer plug-in is implemented as a Mesh Edit Tool plug-in. The vertex normals are not available from the Server Development Kit (SDK) [Subrahmanyam 2002], so our plug-in has to obtain them before creating the layers. The vertices that belong to each polygon are available from the SDK. To obtain the normal for a vertex the plug-in finds all the polygons that share that vertex, and then calculates the normal as the average of the polygons' normals [Wright 2000]. The plug-in then performs a polygon scan. If a polygon has been selected the plug-in creates a number of new polygons whose vertices are calculated by displacing the selected polygon's vertices in the direction of their normals. The number n of new polygons created depends on the number of layers specified by the user, and the displacement d_i of each new polygon i is taken as:

$$d_i = (h / n) * i$$

where h is the height of the layers determined by the user. The method creates a number of layers, but the polygons that make up each layer do not share points. To remove duplicate points, so that polygons share a point, LightWave's Merge Points command can be used [Wright 2001].

The plug-in is implemented as a Mesh Edit Tool so that the user can adjust the height of the layers by dragging their mouse up and down. If the plug-in had been implemented as a Command Sequence plug-in, the Merge Points command could have been called automatically by the plug-in to remove duplicate points. Unfortunately, Mesh Edit Tool plug-ins cannot call commands, so the user must execute the Merge Points command manually, by pressing the "merge points" command button on the interface.

The plug-in's interface is programmed so that the plug-in appears to be a tool that is part of LightWave. The user is provided with a numeric panel (see Figure 4.3) in which they may enter the number and height of the layers, as well as a surface name. A number is added to the end of the surface name to identify each layer. For instance, if the user provides the surface name "layersurface", the plug-in will assign the surface name "layersurface0" to all the polygons that make up the first layer, the surface name "layersurface1" to all the polygons that make up the second layer, and so on. This simplifies placing different textures on each layer. The user may also control the height of the layers by holding down the left mouse button and dragging the

mouse up and down to increase and decrease the height. A message is displayed at the bottom of the screen to assist the user (see Figure 4.4).

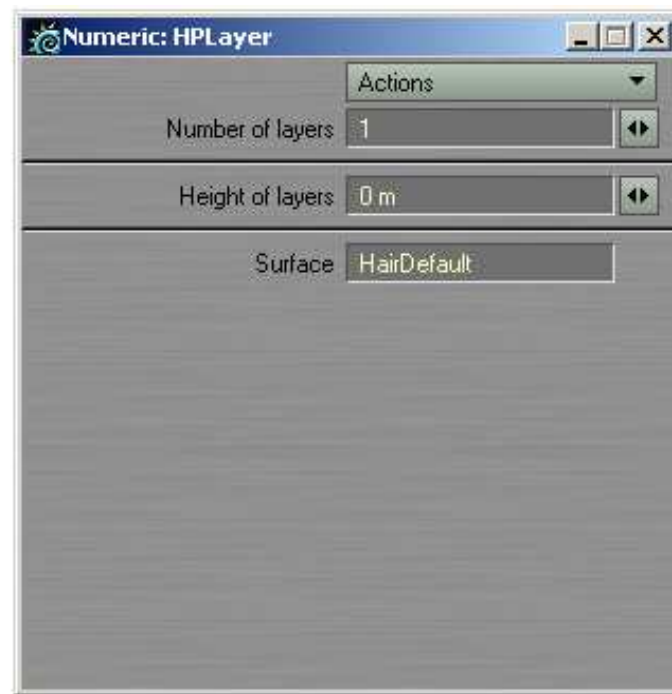


Figure 4.3: *The HPLayer plug-in's interface.*



Figure 4.4: *A message assisting the user.*

4.2.1.3 Results

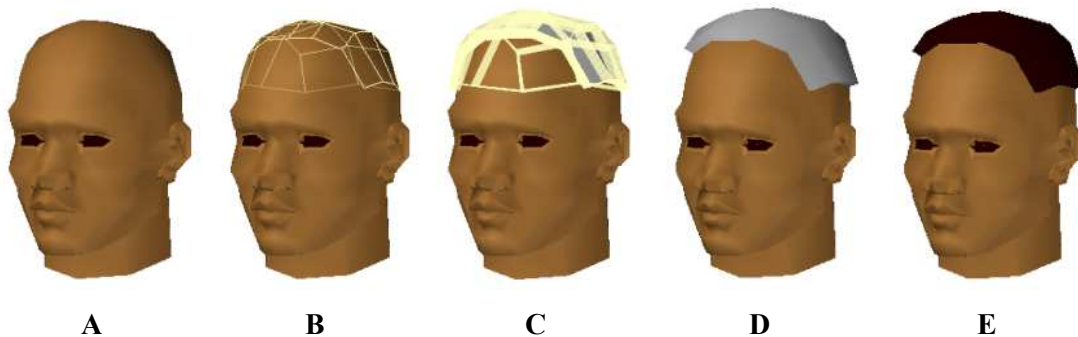


Figure 4.5: *Using the HPLayer plug-in to create and place layers on a head model.*

Figure 4.5 demonstrates the use of the HPLayer plug-in. A head model is created or imported into LightWave's Modeler (A). Polygons on which hair should be placed are selected (B), and then the HPLayer plug-in is run. In the diagram 20 layers are created. The layers have a height of 10cm in total (the head model has dimensions⁴ of approximately 180cm x 250cm), and the surface name is "layersurface" (so the layers have the surface names "layersurface0", "layersurface1", ..., "layersurface19") (C). Although the layers are given different surface names, the surface properties, such as the surface colour, assigned to each layer remain as defaults (D). These can be changed (E), and texture maps can be applied to the layers to produce the effect of hair. The surface properties and textures are discussed in Section 5.4.1.

4.2.2 HPCurve

In Section 3.2 we describe a technique for modelling straight hair explicitly. To overcome the problem of modelling a large number of hair strands, the technique uses control hairs to define a hairstyle, and then creates a full head of hair by automatically multiplying the control hairs. To create the control hairs we implement a plug-in called HPCurve that places control curves on selected polygons of a head model. The plug-in creates one curve for each selected polygon and then adds the curve to the centre of the polygon. A user is then able to bend and modify the curves in order to design a hairstyle.

The HPCurve plug-in can also be used when modelling braids and twists with our cluster technique, described in Section 3.3. The technique uses curves to define the overall shape of a braid or twist of hair. To place the curves on a head model, a user may either add them manually or use our HPCurve plug-in.

4.2.2.1 Implementation

Since the HPCurve plug-in is designed to create control hairs and place them on a head model, implementation of the plug-in involves the creation of control hairs and the placement of control hairs onto a head model. Each control hair is represented by a curve, which can be bent to form the shape of a strand of hair. Since the length and curliness of strands of hair vary greatly, we allow the length of the curve and the number of control points constituting the curve to be user-defined. The colour and other surface properties of strands of hair on a person's head may also

⁴ Note: The dimensions for the hair, head models, and samples of hair used in this thesis are not necessarily life size, but could be given the correct scale if needed.

vary, so the control curves created are assigned a user-defined surface name, which specifies their surface properties.

The curve is created as a straight line consisting of n segments and $(n + 1)$ control points. The control points are evenly spaced so that all the segments have the same length: d / n , where d is the length specified by the user (see Figure 4.6). After the plug-in has created the curve, the user may move the control points to define the shape of a strand of hair. This may cause the segments between the control points to be different lengths. In LightWave a curve cannot be rendered directly. The control hairs must therefore be converted to another form such as a polyline or generalized cylinder, before they are rendered. (Sections 4.2.6 and 4.2.7 discuss plug-ins that convert curves to polylines and generalized cylinders.)



Figure 4.6: *A control hair consisting of four segments and five control points.*

To add control hairs to a head model, the plug-in places a control hair in the centre of each selected triangle (the plug-in could add control hairs to the centre of any selected polygon, but the HPDuplicate plug-in discussed in Section 4.2.3, which makes copies of the control hairs, requires the control hairs to be placed on triangles). Therefore, the first thing the plug-in does is check to see that all selected polygons are triangles. If they are not triangles, the plug-in does nothing but display a message telling the user to convert all polygons to triangles. If they are triangles, the centre point of each triangle is calculated and a control hair is created and placed at the centre point, in the direction of the triangle's normal (see Figure 4.7).

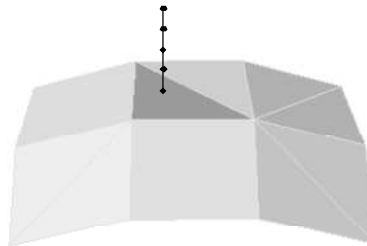


Figure 4.7: *Placing a control hair in the centre of a selected triangle.*

The control hair and triangle it is placed on are grouped together using tags. The plug-in checks to see whether a triangle has already been assigned a tag; if it has, the control hair is given the same tag value; otherwise a tag value is assigned to both the control hair and the triangle it is

placed on. The tags are used by the HPDuplicate plug-in (see Section 4.2.3), which needs to know which triangle each control hair is placed on.

4.2.2.2 LightWave Implementation Issues

The HPCurve plug-in creates geometry (control hairs) and could belong to the Command Sequence or Mesh Edit Tool class. The first thing the plug-in does is check to see that all selected polygons are triangles, which is often not the case. Implementing the plug-in as a Command Sequence class would allow the plug-in to issue the LightWave command TRIPLE, which converts polygons into triangles by subdivision [Wright 2001]. Using the TRIPLE command, the plug-in could automatically convert any selected polygons with more than three sides into triangles. This would allow the user to select any polygons on the head model on which they would like hair to be placed regardless of each polygon's number of sides. Unfortunately, Command Sequence plug-ins do not behave as interactive tools, and it is important that the user is able to adjust the length of the control curves interactively with their mouse. Entering a value for the exact length required is often difficult — it is easier to adjust the length by dragging a mouse up and down.

The HPCurve plug-in is implemented as a Mesh Edit Tool plug-in. Mesh Edit Tool plug-ins cannot issue commands, so the user is required to convert selected polygons into triangles manually (which can be done by clicking the “triple” command button on the interface). To assist the user, a message is displayed telling the user to convert all selected polygons to triangles first (if they are not already triangles), as shown in Figure 4.8.



Convert all the selected ploygons to triangles first (Construct>Subdivide: Triple)

Figure 4.8: *A message assisting the user.*

The plug-in's interface is programmed so that the plug-in appears to be a tool that is part of LightWave. The user is provided with a numeric panel (see Figure 4.9) on which they may enter the length, number of segments and surface name of the control hairs. The user may also control the length of the curves created by holding down the left mouse button and dragging the mouse up and down to increase and decrease the length. Messages are displayed at the bottom of the screen to assist the user (see Figure 4.10).

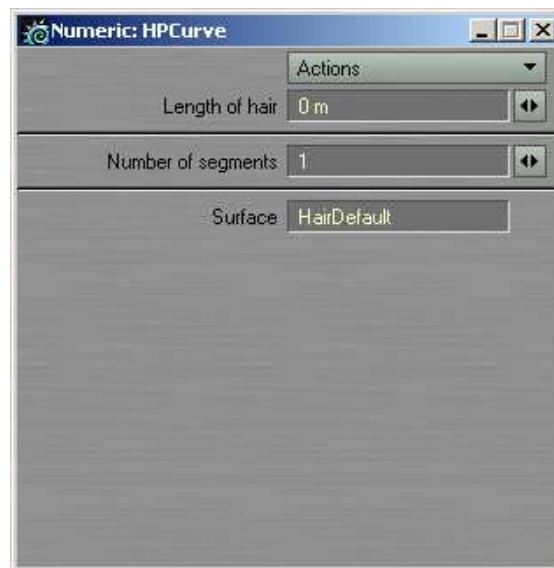


Figure 4.9: *The HPCurve plug-in's interface.*

Drag your mouse up or down to adjust the length

Figure 4.10: *A message assisting the user.*

4.2.2.3 Results

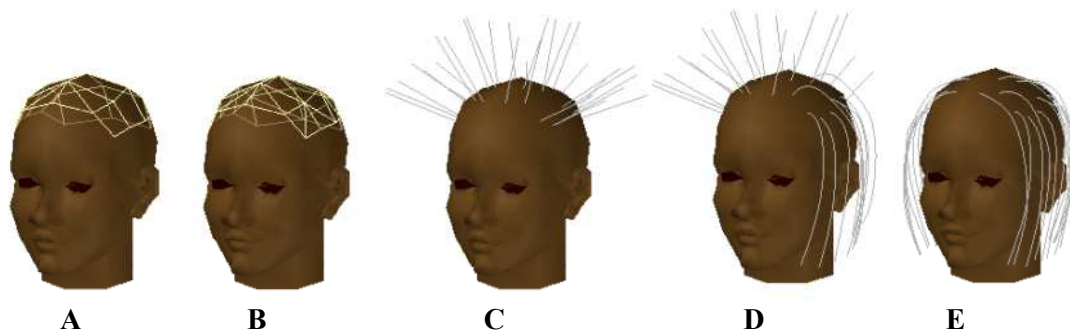


Figure 4.11: *Using the HPCurve plug-in to design a hairstyle.*

The images in Figure 4.11 demonstrate the use of the HPCurve plug-in. Polygons upon which hair should be placed are selected (A) and converted into triangles (B) before the HPCurve plug-in is run. In the diagram 30 control curves are created. The curves consist of four segments and have a length of 9cm (the head model has dimensions of approximately 16cm x 25cm). The surface of each curve is called “HairSurface” and the surface properties are defaults, although they can be changed (C). The user is able to bend and modify the control curves to define a hairstyle using the tools available in LightWave (D and E).

Since the control curves do not define a complete head of hair, and curves cannot be rendered in LightWave, other plug-ins can be used after HPCurve to multiply the control hairs, or to convert them into braids, twists, polylines or generalized cylinders (see Sections 4.2.3–4.2.7 for a description of these plug-ins).

4.2.3 HPDuplicate

When modelling straight hair explicitly we use a technique where curves, representing strands of hair, are automatically created and placed onto a head model to decrease the amount of time a user spends on modelling strands of hair. The curves are based on control curves. The technique is discussed in Section 3.2. To implement the technique we develop a plug-in called HPDuplicate. The HPDuplicate plug-in makes a user-defined number of copies of each selected control hair and randomly places the copies on the same triangle of the head model as the control hair from which they were copied. The plug-in creates curves that represent strands of explicitly modelled straight hair. However, it can also be used to create extra control curves if needed. Control curves are used to define hairstyles for explicitly modelled hair (Section 3.2) or to define the shape of braids and twists of hair (see Section 3.3).

4.2.3.1 Implementation

The HPDuplicate plug-in is designed to reduce the time a user spends on modelling hair, by creating and placing a large number of curves, representing hair strands, onto a head model. In order to reduce the modelling time, the plug-in must automatically model the curves into reasonable shapes so that they represent a hairstyle. To implement the plug-in, we need to consider automatic creation and shaping of curves, and the placement of curves onto a head model. Since the number of hairs on a head varies from person to person, the number of curves should be user defined.

We assume that the user has defined a hairstyle (or braids or twists of hair) using the HPCurve plug-in (see Section 4.2.2). Like the control hairs created with the HPCurve plug-in, each automatically created curve consists of n segments and $(n + 1)$ control points. However, the number of segments is not user defined. Instead, the number of segments, as well as the shape, length and surface properties of the curve, is determined by a control hair. Since we assume that a user has designed a hairstyle (or braids or twists) with control hairs, the plug-in can use

selected control hairs to create, shape and place a user-defined number of curves onto the head model.

For each selected control hair, the HPDuplicate plug-in scans all the polygons to find the triangle on the head model on which the control hair is placed. The plug-in then creates a user-defined number of curves. Each curve is created as a copy of the selected control hair. Using the coordinates of the control points that make up the selected control hair, a curve can be created that has the same number of segments, length and shape as the control hair. The plug-in makes copies of the control hair's control points and then translates them by a random amount. The translated control points are then used to create the new curves. These have the same shape as the control curve, but are positioned at different points in the triangle on which the control curve is placed. The random amounts used to translate the points are calculated by finding a random point in the triangle on which the control curve is placed. Since different algorithms are needed to find random points in polygons with different numbers of sides, the HPCurve plug-in (see Section 4.2.2) makes sure all control curves are placed on triangles, so that the HPDuplicate plug-in can use an algorithm that finds a random point in a triangle [Turk 1990]. The curve is also given the same surface name as the control hair.

4.2.3.2 LightWave Implementation Issues

Since the HPDuplicate plug-in is designed to create, shape and place hair strands on a head model automatically, to reduce the amount of time and effort a user has to spend on modelling hair, it is not designed as an interactive tool, and is therefore implemented as a Modeler Command Sequence plug-in rather than an Mesh Edit Tool plug-in. The plug-in uses only mesh-editing functions and does not issue any commands. It could, therefore, also have been implemented as a Mesh Data Edit class, which is a subset of the Command Sequence class. The Mesh Data Edit class has access to mesh editing functions, but cannot issue commands [Wright 2001].

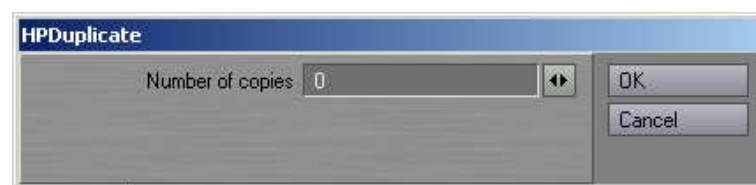


Figure 4.12: *The HPDuplicate plug-in's interface.*

The number of hair strands (copies of each selected control hair) the plug-in must create is obtained from the user. An interface is implemented to obtain the information (see Figure 4.12).

4.2.3.3 Results

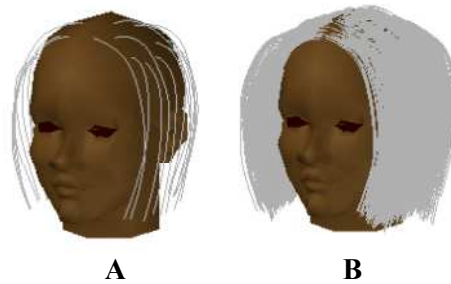


Figure 4.13: *Using the HPDuplicate plug-in.*

Figure 4.13 demonstrates the use of the HPDuplicate plug-in. A hairstyle is defined by 30 control curves (A). All the control curves are selected and the HPDuplicate plug-in is run. A total of 1,800 curves are automatically created (60 duplicates are made of each control curve) resulting in a hairstyle containing many strands of hair (B). The automatically created strands of hair have the same surface properties as the control curves, but the properties can be changed.

4.2.4 HPTwist

We initially define a twist of hair with a curve, which is easy to bend and useful for defining the overall shape of the twist. To convert the curve into a twist we create several curves that twist around the initial curve to form the twist (see Section 3.3.3). We implement the technique as a plug-in, called HPTwist, which converts selected curves into a user-defined number of curves that are twisted together. The user is able to define the thickness of the twist, as well as the number of curves, amount of twist, and surface name of the curves constituting the twist. This allows a variety of twists that meet the needs of the user to be created with the same plug-in.

4.2.4.1 Implementation

The HPTwist plug-in is designed to make modelling twists of hair easy and to reduce the time a user spends on modelling a hairstyle. To create each curve in the twist automatically, we calculate the values of several points for the curves. These points are used as control points and are connected to form the new twisting curves. The values of the points are calculated at each

control point of the original curve using a reference frame and the parametric equation defined in Section 3.3.3.1. The algorithm used is shown below:

```

For each curve  $i$  making up the twist {
     $\theta$  = angle difference between curves *  $i$ 
    For each control point  $k$  along the original curve {
        Calculate point  $p_k$  on the twisting curve
         $\theta = \theta + \text{amount of twist}$ 
    }
    Connect points  $p_0$  to  $p_n$  to create a new curve that twists around the original curve
    ( $n$  is the total number of control points on the original curve)
}

To calculate point  $p_k$  on the twisting curve:
    Obtain the reference frame at  $k$ 
    (i.e. obtain vectors  $N$  and  $B$  that lie perpendicular to the tangent at  $k$ )
    Calculate  $x$  and  $y$  using the parametric equation for a curve in a twist.
    ( $x = r * \cos(\theta)$  and  $y = r * \sin(\theta)$ )
     $p_k[j] = N[j] * x + B[j] * y + k[j]$ , where  $j = 0, 1, 2$ .

```

The amount of twist, obtained from the user, represents the increase in θ as the curve twists around the original curve. The smaller the value of θ with respect to the distance between the control points of the original curve, the less twist the new curve has.

Because the original curve may consist only of a few control points, the user is able to specify the number of samples taken between the control points (i.e. more values are calculated for the new curve). If the number of samples n , specified by the user, is more than one, $(n - 1)$ points on the curve between each set of adjacent control points are calculated and used together with the control points to form the “control points” of the original curve. The method for calculating points on a curve between control points is discussed in Section 4.2.6.2.

The number of samples taken affects the amount of twist in the twisting curve. A large number of samples will result in smaller distances between control points for every increase in θ , and therefore more twist in the curve. The same number of samples is taken between each pair of control points, and the samples are evenly spaced between each pair of control points. However, if the distances between the pairs of control points are not the same, the twist will become more or less pronounced depending on the relative distances. To avoid the problem the user should try to keep the relative spacing between pairs of control points on the original curve as similar as possible when defining the shape of the curve.

4.2.4.2 LightWave Implementation Issues

The HPTwist plug-in is implemented as a Mesh Edit Tool plug-in, allowing the user to adjust the thickness of the twist by dragging their mouse left and right, and the amount of twist by dragging their mouse up and down.

The plug-in also has a numeric panel (see Figure 4.14) that allows the user to enter values for the user-defined parameters, and a message assisting the user (see Figure 4.15).

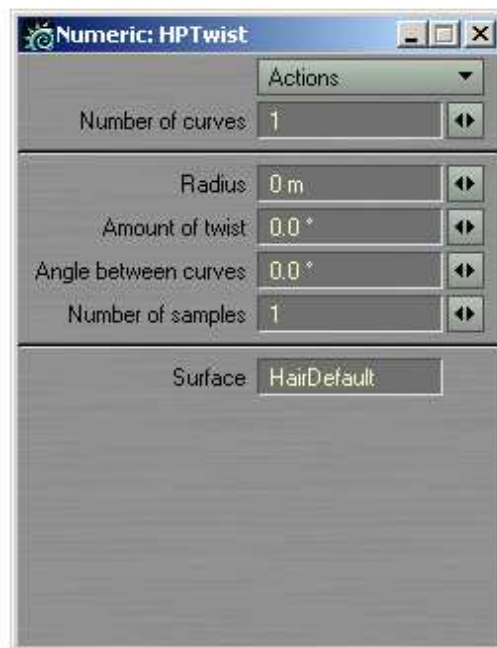


Figure 4.14: *The HPTwist plug-in's interface.*

Drag your mouse: up or down to adjust twist AND left or right to adjust thickness

Figure 4.15: *A message assisting the user.*

The “Number of curves” on the numeric panel represents the number of curves making up the twist; the “Radius” determines the thickness of the twist; the “Amount of twist” is the change in θ each time a new point on the curve is calculated; the “Angle between curves” is the difference in θ for the different curves at each point; the “Number of samples” determines the number of new control points needed between the control points on the original curve in order to create more control points for the new curves; and “Surface” specifies the surface name of the new curves.

4.2.4.3 Results

The examples in Figure 4.16 show how the different user-defined values affect the results of the twist created. All the values are kept constant for each example, except the values of the variable we are testing. The constant values used (unless otherwise stated) are:

Number of curves: 3

Radius: 20cm

Amount of twist: 10.0°

Angle between curves: 120.0°

Number of samples: 10

Surface: “HairDefault”

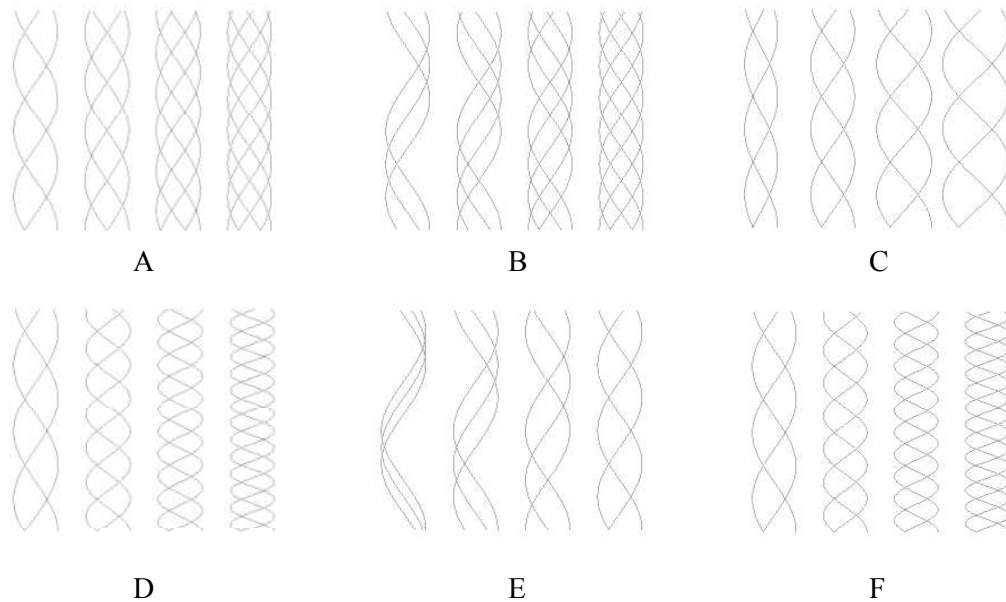


Figure 4.16: *Different twist values.*

In Figure 4.16 the original curve defining each twist has a length of 2m and consists of four segments. Diagrams A and B show twists consisting of different numbers of curves. In each diagram the twists consist of three, four, five and six curves. In Diagram A the angle between the curves has been changed as well in order to space the curves evenly. The angle between the curves is 120°, 90°, 72° and 60° ($360^\circ / \text{number of curves}$). In Diagram B the angle between the curves is 60°. Diagram C shows twists with radii of 15cm, 20cm, 25cm and 30cm. In Diagram D the amount of twist varies. The values for the amount of twist are 10°, 20°, 30° and 40°. In Diagram E the angle between the curves varies. The angle is 30°, 60°, 90° and 120°. In Diagram

If the twists have different numbers of samples. The number of samples for each twist is 10, 20, 30 and 40.

The amount of twist is determined not only by the “Amount of twist”, but also by the “Number of samples”. If the user doubles the amount of twist, it will produce the same curves that would be produced if the user doubled the number of samples. However, the numbers of control points on the curves will double when the number of samples is doubled, but not when the amount of twist is doubled. Therefore, by adjusting the number of samples and (inversely) the amount of twist, it is possible to produce the same curves with different numbers of control points.

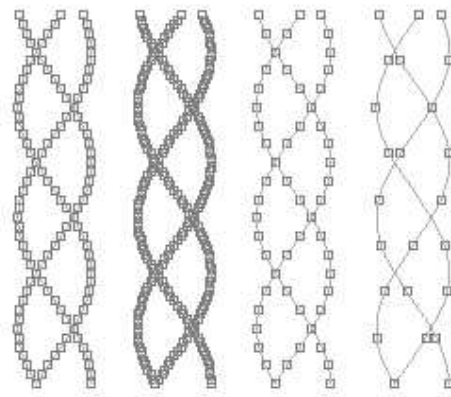


Figure 4.17: *Twists with the same curves, but different numbers of control points.*

In Figure 4.17 the amount of twist and number of samples for each curve are set to 10° , 10; 5° , 20; 20° , 5; and 50° , 2. All four twists have the same shape, but different numbers of control points.

To implement the HPTwist plug-in so that the “Number of samples” determines the number of control points, but does not have an affect on the amount of twist, the plug-in would have to be implemented so that the value for the “Amount of twist” is automatically changed whenever the value for the “Number of samples” is changed:

$$\text{“Amount of twist”} = \text{“Amount of twist”} / (n1 / n2)$$

where $n1$ is the new value for the “Number of samples” and $n2$ is the previous value for the “Number of samples”.

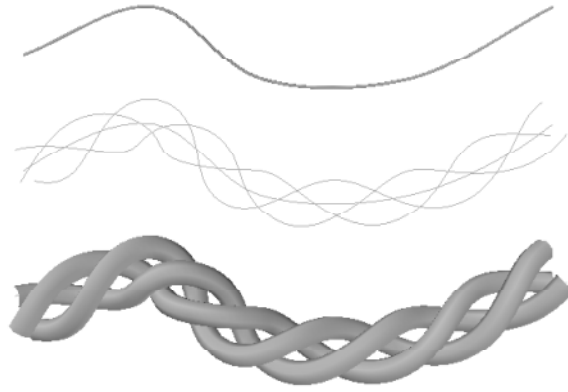


Figure 4.18: *Converting a curve into a twist.*

Figures 4.18 and 4.19 demonstrate the use of the HPTwist plug-in. Figure 4.18 shows a curve converted into a twist represented by three curves. Figure 4.19 shows a hairstyle defined by 100 curves (A). Each curve is converted into three curves that define a twist using the HPTwist plug-in (B). The original curves are defined by four segments. The curves representing each twist are defined by 40 segments, lie 120° apart, and represent a twist with a radius of 2mm (the head model has dimensions of approximately 24cm x 26cm). They have a surface name specified by the user and default surface properties, which can be changed.

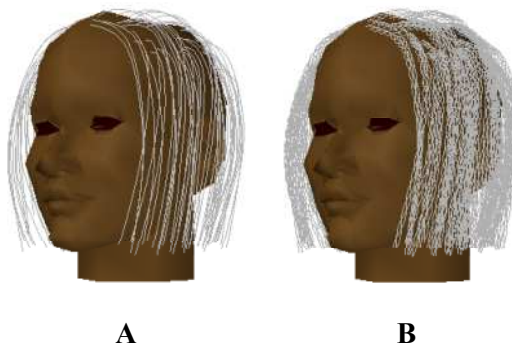


Figure 4.19: *Using the HPTwist plug-in.*

4.2.5 HPBraid

Like a twist, a braid of hair is initially defined by a curve. To convert the curve into a braid we implement a plug-in, called HPBraid, which converts selected curves into a user-defined number of curves that twist around the initial curve to form a braid. The thickness, flatness, number of curves and surface name of the curves the braid consists of are user defined. The user is also able to adjust the amount of twist in the braid.

4.2.5.1 Implementation

To implement the HPBraid plug-in, curves that twist around the original curve are created automatically, using the same method that was used to implement the HPTwist plug-in: the control points of the automatically-created curves are calculated and then connected to define the curve. To calculate the control points, a reference frame is used together with the parametric equation for a braid (presented in Section 3.3.3.1) and the user-defined values of several parameters. The reference frame is calculated at each control point on the original curve. However, the user may specify that more control points are needed. If the value of the number of samples n obtained from the user is greater than 1, $(n-1)$ control points are added between adjacent control points on the original curve.

The shape of a braid viewed along the x -axis towards the y - z plane is not the same as the shape of the braid viewed along the y -axis towards the x - z plane (see Figure 4.20).

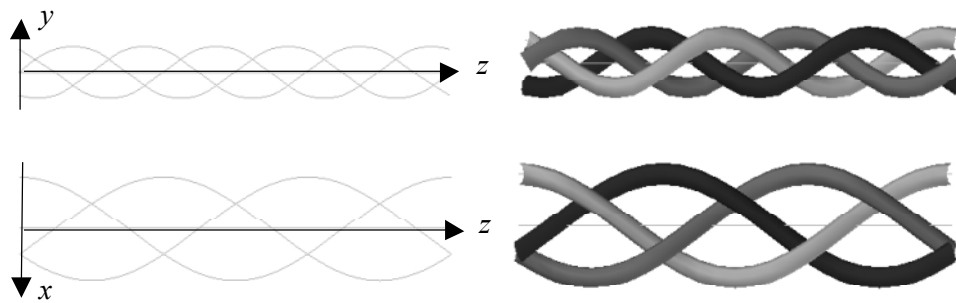


Figure 4.20: *Viewing a braid from different angles.*

Since the braid is automatically created from a curve that lies in any direction in 3D space, the braid may not lie in the desired direction. To solve the problem we allow the user to rotate the braid by $\pi/2$ radians (or 90°). This is implemented by switching the values of x and y in the parametric equation (defined in Section 3.3.3.1).

4.2.5.2 LightWave Implementation Issues

The HPBraid plug-in is implemented as a Mesh Edit Tool plug-in, allowing the user to adjust the thickness of the braid by dragging their mouse left and right, and the amount of twist in the braid by dragging their mouse up and down.

The plug-in also has a numeric panel (see Figure 4.21) that allows the user to enter values for the user-defined parameters, and a message assisting the user (see Figure 4.22).

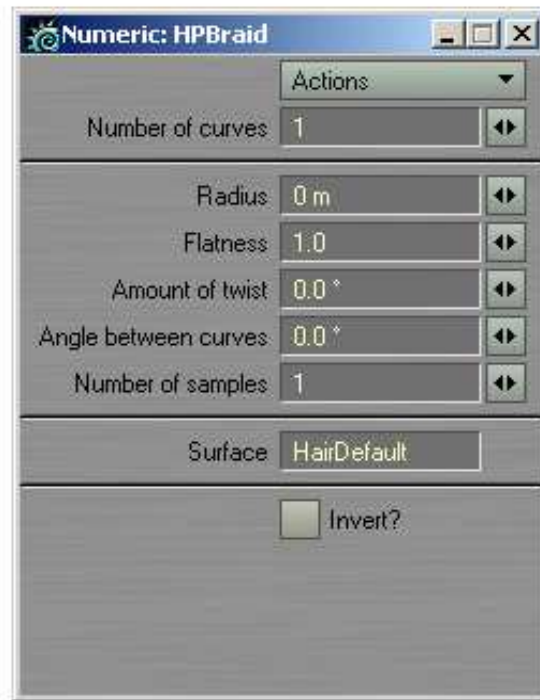


Figure 4.21: *The HPBraid plug-in's interface.*



Figure 4.22: *A message assisting the user.*

The parameters on the numeric panel for the HPBraid plug-in are the same as those for the HPTwist plug-in (see Section 4.2.4.2), except that the user may also enter a value to flatten the braid (“Flatness”) and may select “Invert?” to swap the values of x and y so that the braid is rotated by $\pi/2$ radians (or 90°).

4.2.5.3 Results

The examples in Figure 4.23 below show how the different user-defined values affect the results of a braid created. In each example the top diagram shows a view of the braid on the x - z plane, and the bottom diagram shows a cross section of the braid viewed along the z -axis. All the values are kept constant for each example, except the values of the variable we are testing. The constant values used (unless otherwise stated) are:

Number of curves: 3

Radius: 20cm

Flatness: 2.0

Amount of twist: 10.0°

Angle between curves: 120.0°

Number of samples: 10

Surface: “HairDefault”

Invert?: Not selected

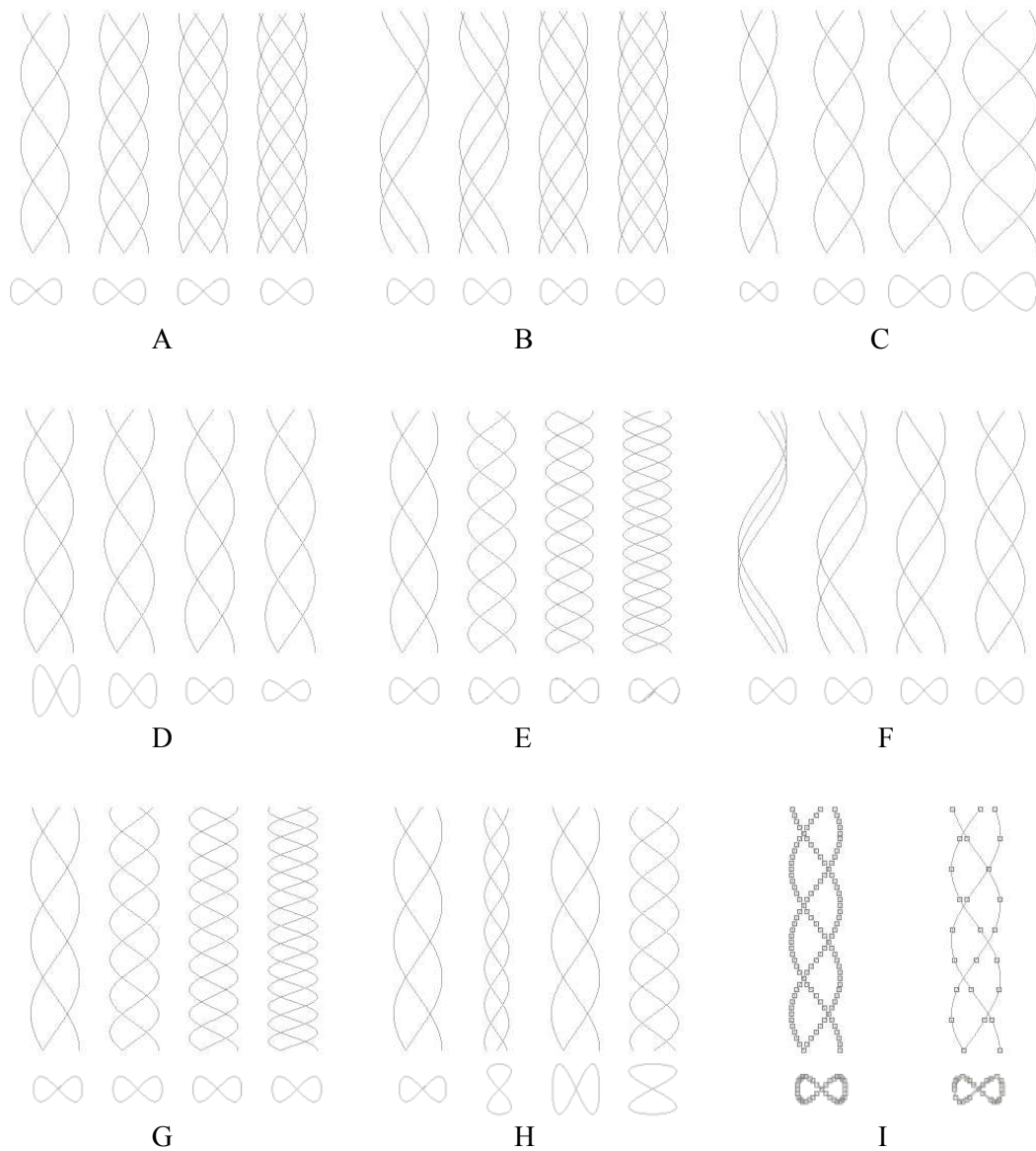


Figure 4.23: *Different braid values.*

The original curve defining each braid in Figure 4.23 has a length of 2m and consists of four segments. Diagrams A and B show braids consisting of different numbers of curves. In each diagram the braids consist of three, four, five and six curves. In Diagram A the angle between the curves has been changed as well, in order to space the curves evenly. The angle between the curves is 120° , 90° , 72° and 60° ($360^\circ / \text{number of curves}$). In Diagram B the angle between the curves is 60° . Diagram C shows twists with radii of 15cm, 20cm, 25cm and 30cm. In Diagram D the flatness of the braids varies. The braids have a flatness of 1, 1.5, 2 and 2.5, respectively. In Diagram E the values for the amount of twist are 10° , 20° , 30° and 40° . In Diagram F the angle between the curves varies. The angle is 30° , 60° , 90° and 120° . In Diagram G the braids have different numbers of samples. The number of samples for each braid is 10, 20, 30 and 40. Diagram H shows the results when “Invert?” is selected. The first two braids have the same user-defined values, with a flatness of 2.0, but “Invert?” has been selected for the second braid. The last two braids also have the same user defined values, but have a flatness of 1.0. “Invert?” has been selected for the last braid. Diagram I shows that like the HPTwist plug-in, the “Amount of twist” and “Number of samples” determine the twist, in this case in a braid, and can be adjusted to produce braids that have curves with the same shape, but different numbers of control points. The first braid’s amount of twist and number of samples are 10° and 10 samples, respectively. The second braid has an amount of twist of 50° and the value of 2 for its number of samples.

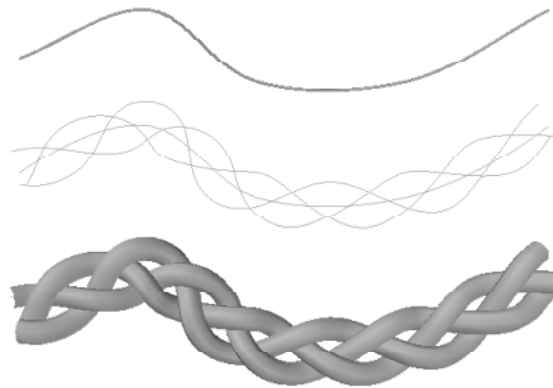


Figure 4.24: *Converting a curve into a braid.*

Figures 4.24 and 4.25 demonstrate the use of the HPBraid plug-in. Figure 4.24 shows a curve converted into a braid represented by three curves. Figure 4.25 shows a cornrow hairstyle defined by 14 curves (A). Each curve is converted into three curves that define a braid using the HPBraid plug-in (B). The original curves are defined by ten segments. The curves representing each braid are defined by 100 segments, lie 120° apart, have a flatness of 2, and represent a braid

with a radius of 2mm (the head model has dimensions of approximately 24cm x 26cm). They have a surface name specified by the user and default surface properties, which can be changed.

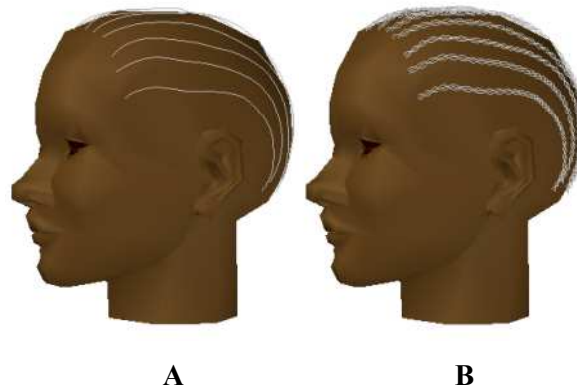


Figure 4.25: Using the *HPBraid* plug-in.

4.2.6 HPPolyline

We use curves to model straight hair explicitly (see Section 3.2, and the *HPCurve* and *HPDuplicate* plug-ins in Sections 4.2.2 and 4.2.3). They are easy to bend, and are a close representation of a strand of hair in reality. However, it is not possible to render curves in LightWave, so the curves need to be converted to other geometry before rendering. We convert the curves into polylines (see Section 3.2.3). To do this, we implement a plug-in, called *HPPolyline*, which converts all selected curves into polylines. The plug-in creates polylines that have a user-defined number of segments.

4.2.6.1 Implementation

To implement the plug-in we need a method for converting a curve, which is defined by a few control points [Nakakihara 2001], into a number of connected line segments, so that the line segments approximate the shape of the curve. A simple approach is to join the adjacent control points that define the curve with straight lines (defined as *two-point polygons* in LightWave [Nakakihara 2001]). In other words, control point i is joined with control point $i + 1$, where first control point $\leq i < \text{last control point}$. Unfortunately, the approach does not produce polylines that represent high-quality approximations of the curve if too few control points are used with respect to the curvature in the shape of the curve. Since the curves are used to define a hairstyle, and a user has to move the control points of each control curve to a desired position manually, most curves will only contain a few control points. To solve the problem our plug-in finds a

user-defined number of points along the curve between the control points and uses them with the control points to create two-point polygons. Figure 4.26 shows a curve defined by control points (A) converted into a polyline (C). The figure also shows that using too few two-point polygons will cause a low quality approximation of the curve (B).

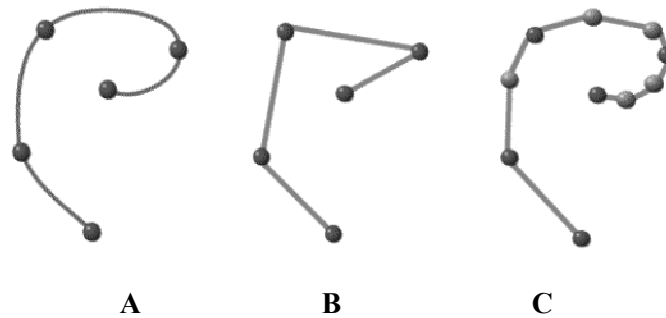


Figure 4.26: *Converting a curve defined by control points into a polyline.*

4.2.6.2 LightWave Implementation Issues

The HPPolyline plug-in converts curves into polylines that can be rendered. LightWave contains a command called FREEZECURVES that is able to convert curves into polygons. Unfortunately, the command converts all open curves into closed curves (by joining the curve's end points) before converting the curve into a polygon (see Figure 4.27) [Nakakihara 2001]. Since strands of hair usually bend and must remain as open curves, the command is not useful. Because we need a command that converts curves into chains of two-point polygons, we develop the HPPolyline plug-in.

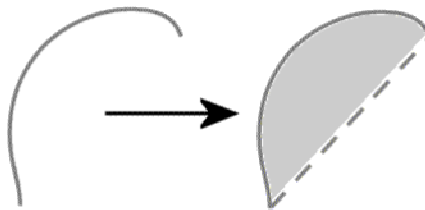


Figure 4.27: *Using FreezeCurves to convert an open curve into a polygon.*

The HPPolyline plug-in is implemented as a Modeler Command Sequence plug-in. It is not implemented as a Mesh Edit Tool plug-in because it does not require the user to determine values interactively. The plug-in has an interface (see Figure 4.28) that is used to obtain the number of two-point polygons (called segments) between consecutive control points that the user wishes the polyline to be made up of. The value is a small positive integer that can be typed in,

and therefore does not need to be determined interactively. Since the plug-in does not issue any commands, it could have been implemented as a Mesh Data Edit plug-in (a plug-in that has access to mesh editing functions like the Modeler Command Sequence plug-ins, but cannot issue commands [Wright 2001]).

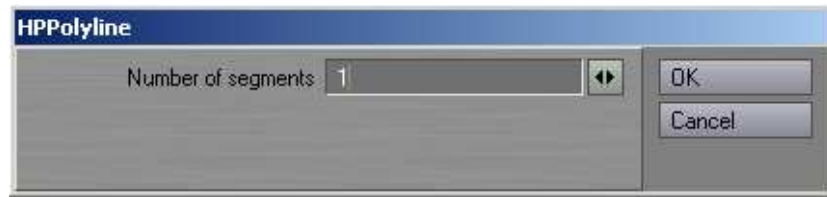


Figure 4.28: *The HPPolyline plug-in's interface.*

To convert each curve into a polyline, the plug-in creates a set of points based on the positions of the control points and a user-defined number of points along the curve between the control points. The created points are then used to create two-point polygons, forming a polyline. Since new points and polygons are created to produce the polyline, the original curve is neither modified, nor part of the new polyline, and it can be deleted. The curve is not modified into a polyline, but rather it is used to define the polyline, so the term “convert” is probably not accurate. However, the curve appears to be converted into a polyline, so we describe the plug-in as one that “converts” curves into polylines.

The positions of the control points are needed to create the new points for the polyline and can be obtained from the SDK [Wright 2000]. To find the positions of the points along the curve between control points (which are needed if the user wants more than one segment between control points) we need to calculate the positions, as they cannot be obtained from the SDK automatically. To find any point $p(t)$ on a curve created in LightWave between position i and position $(i+1)$, where $0 < i < \text{numcontrolpoints} - 1$, we use [Wright 2002]:

$$p(t) = h_1 * p_1 + h_2 * p_2 + h_3 * r_1 + h_4 * r_2$$

where p_1 and p_2 are the control points on either side of $p(t)$, r_1 and r_2 are the tangents of the control points, h_1 , h_2 , h_3 and h_4 are Hermite coefficients, and $0 \leq t \leq 1$. The first and last control points ($i = 0$ and $i = \text{numcontrolpoints} - 1$) are used to calculate the tangents of the first and last points on the curve. These control points are defined using `EDPF_CCSTART` and `EDPF_CCEND` in LightWave. If `EDPF_CCSTART` and `EDPF_CCEND` have not been defined, imaginary points are created, using the closest three existing points. The Hermite coefficients are defined by:

$$\begin{aligned}
t_2 &= t * t; \\
t_3 &= t_2 * t; \\
h_2 &= 3.0 * t_2 - t_3 - t_3; \\
h_1 &= 1.0 - h_2; \\
h_4 &= t_3 - t_2; \\
h_3 &= h_4 - t_2 + t; \text{ [Wright 2002]}
\end{aligned}$$

where t lies between 0 and 1, and is determined by the number of two-point polygons (segments) the user specifies they would like between each set of control points.

Once the positions of the points are determined they can be used to create the points and two-point polygons, using functions from the SDK.

4.2.6.3 Results

Figure 4.29 shows the results when curves are converted to polylines consisting of different numbers of segments. The original curves (shown in diagram A) consist of four segments. In Diagrams B and C the curves have been converted into polylines with one and four segments between the control points of the original curve, respectively. The quality of the polylines in Diagram B is less than the quality of those in Diagram C, but the polylines in Diagram C contain four times as many two-point polygons as those in Diagram B. There is therefore a trade off between using a small number of two-point polygons (which require less storage space and rendering time) and the quality of the polyline produced.

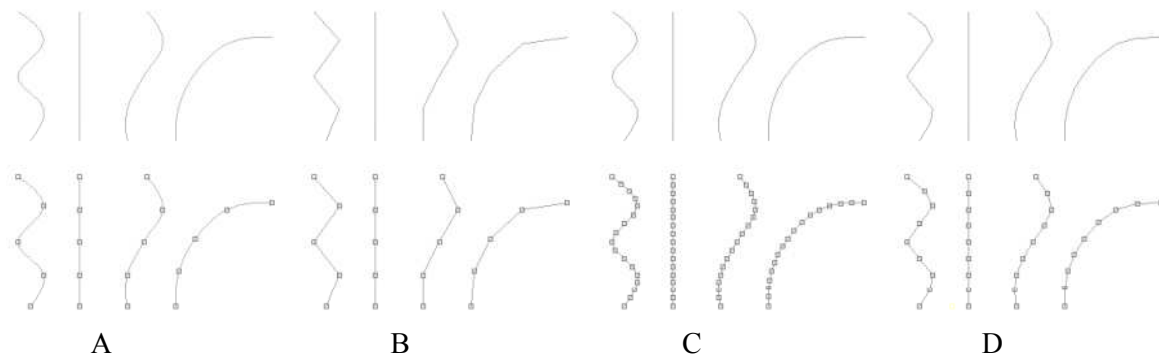


Figure 4.29: Converting curves to polylines so that they can be rendered.

In Diagram D the curves have been converted to polylines consisting of two segments between the control points of the original curve. The quality of the second and fourth polylines is good, but the first and third curves need to be approximated by more segments in order to produce

polylines that have a good quality. Therefore, the number of segments needed for each polyline depends on the shape of the curve. Our plug-in places the same user-defined number of segments between each set of control points, and may therefore create an unnecessary number of segments. For instance, the second curve in each diagram is a straight line and can therefore be approximated using a polyline consisting of one segment. However, 16 segments are used to approximate the curve in Diagram C in order to increase the quality of all the polylines in the diagram. A more advanced approach would be to consider the tangents or angles between control points and create extra segments (two-point polygons) only when needed.

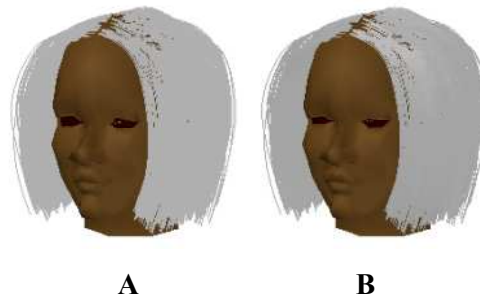


Figure 4.30: *Using the HPPolyline plug-in.*

Figure 4.30 demonstrates the use of the HPPolyline plug-in. A hairstyle is created with 1,830 curves, and each curve consists of four segments (and five control points) (A). All the curves are selected and the HPPolyline plug-in is run. Each curve is automatically converted into a polyline with 16 segments (four segments are created between each pair of adjacent control points on the original curve), resulting in an almost identical hairstyle that can be rendered (B).

4.2.7 HPGeneralizedCylinder

To model braids and twists of hair we use generalized cylinders, which represent wisps of hair making up the braids and twists. The wisps are initially defined by curves and are then later converted into generalized cylinders (see Section 3.3). To implement the technique we develop a plug-in, called HPGeneralizedCylinder, which converts selected curves into generalized cylinders. The plug-in places circular cross sections along each selected curve to form a generalized cylinder. Unlike curves, generalized cylinders can be rendered, and although the plug-in is mainly developed to generate generalized cylinders on which textures can be placed to produce the effect of a cluster of hairs in a braid or twist, the plug-in can also be used to render individual strands of hair. The user may define the thickness, number of sides, and number of segments (between the control points on the curve) that make up each generalized cylinder, as

well as whether the thickness of the generalized cylinders must remain constant, or taper off towards the end (furthest from the scalp). The user can also specify a surface name and *UV* map name. The *UV* map name is used when mapping textures onto the generalized cylinders, and is discussed in more detail in Section 5.4.2. Asking the user to define properties of the generalized cylinders allows a variety of generalized cylinders that have different qualities to be produced with the same plug-in.

4.2.7.1 Implementation

To convert a curve into a generalized cylinder, circular cross sections are placed along the curve and aligned perpendicular to the curve using reference frames (see Section 3.3.3.2). The cross sections are then joined to form the generalized cylinder. Before the cross sections can be calculated or aligned, the points along the curve where the cross sections are to be placed need to be determined. A simple approach would be to calculate and place the cross sections at the control points of the curve (because the positions of the control points can be obtained from the SDK). However, placing cross sections at the control points only, does not provide accurate results when the curve is defined by a small number of control points. Curves are usually defined by only a few control points, because the user has to move the control points manually to shape the curve. The more cross sections calculated along the curve, the higher the quality of the generalized cylinder. In order for our plug-in to produce generalized cylinders with a desired quality, we implement the plug-in using a technique that calculates the cross sections not only at the control points, but also at a user-defined number of points along the curve between the control points. The method for calculating points on a curve between control points is discussed in Section 4.2.6.2.

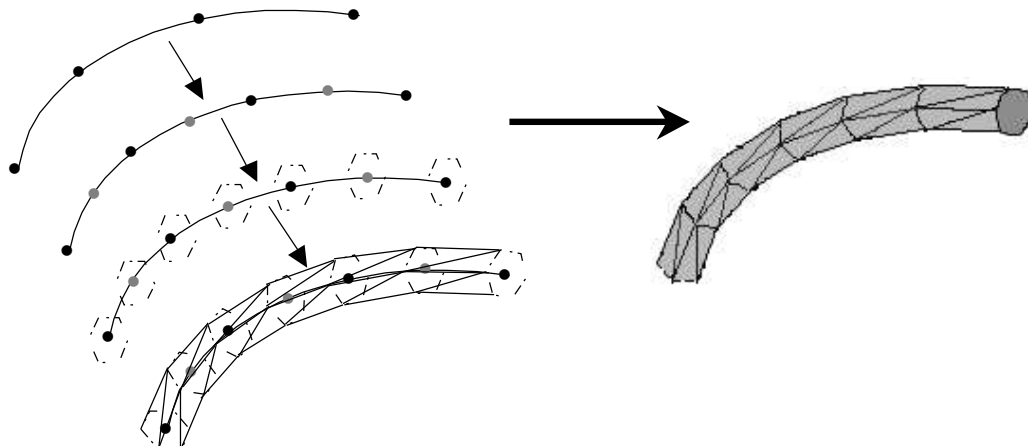


Figure 4.31: *Creating a generalized cylinder.*

The positions of the control points and selected points along the curve between the control points are obtained and stored in an array. The values in the array represent the control points of a “new curve” (although the curve is not created in LightWave) and are used to calculate the cross sections. The points along the circumference of adjacent cross sections are joined to form polygons (triangles) that make up the generalized cylinder (see Figure 4.31).

4.2.7.2 LightWave Implementation Issues

The HPGeneralizedCylinder plug-in is implemented as a Mesh Edit Tool plug-in so that the user can adjust the thickness of the generalized cylinders created by dragging their mouse up and down. The plug-in has a user interface (see Figure 4.32), which allows the user to enter values for the thickness and quality of the generalized cylinders, as well as a surface name, *UV* map name, and whether or not the generalized cylinders taper off towards a point at one end. The quality of a generalized cylinder can be adjusted by changing the number of sides or segments that constitute it. The more sides or segments it contains, the higher its quality. However, more sides or segments result in more polygons, which require more storage space and have longer rendering times. We leave it up to the user to determine these values.

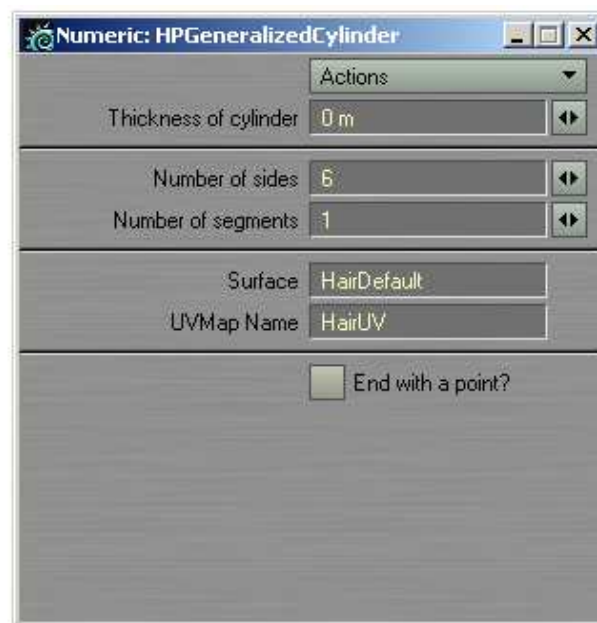


Figure 4.32: The HPGeneralizedCylinder plug-in’s interface.

4.2.7.3 Results

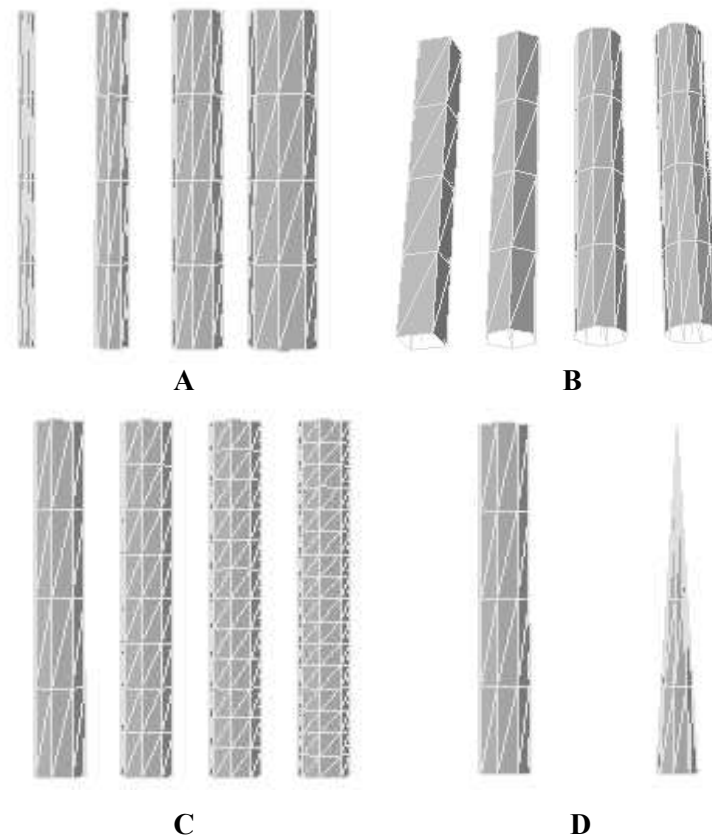


Figure 4.33: *Changing the properties of a generalized cylinder.*

A user is able to specify values for certain properties of the generalized cylinders created with the HPGeneralizedCylinder plug-in. The properties that can be changed by the user include the thickness, number of sides, number of segments, surface name, *UV* map name, and whether the generalized cylinder has a constant thickness or tapers off on one end. Figure 4.33 shows results when values for the properties are changed. All the generalized cylinders in the figure are created from curves made up of four segments (defined by five control points) with a length of 2m. In diagram A the thickness is changed. The generalized cylinders are made up of eight sides, with one segment between the original control points of the curve, and have thicknesses of 50mm, 100mm, 150mm and 200mm. In diagram B the number of sides making up the generalized cylinder is changed. The generalized cylinders have a thickness of 150mm, with one segment between original control points, and are made up of four, six, eight, and ten sides. The diagram shows that generalized cylinders made up of more sides approximate a circular cylinder more accurately than generalized cylinders with fewer sides, however, more sides results in more polygons, higher storage requirements, and slower rendering times. In diagram C the number of

segments calculated between the control points has been changed. The generalized cylinders have a thickness of 150mm, and eight sides. The number of segments that have been created between the control points is one, two, three, and four. The shape and accuracy of the generalized cylinders in the diagram are the same. This is due to the fact that the curves are straight lines and do not bend. Diagram D shows the difference between a generalized cylinder with a constant thickness and one that tapers off on one end. In all the diagrams the generalized cylinders have been given a surface called “HairSurface”, and the *UV* map name “HairUV”. However, the names of the surface and *UV* map, and the surface properties, can be changed.

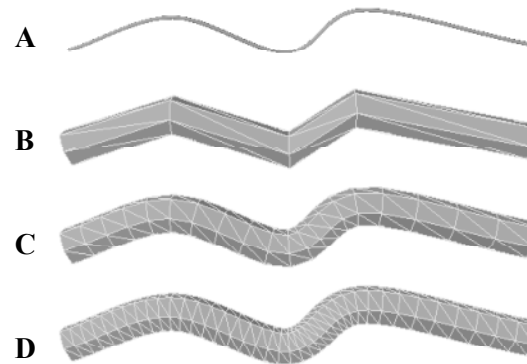


Figure 4.34: *Converting a curve into generalized cylinders.*

Figure 4.34 shows an example of a curve (A) converted into a generalized cylinder with 4 segments (B), 20 segments (C), and 40 segments (D).

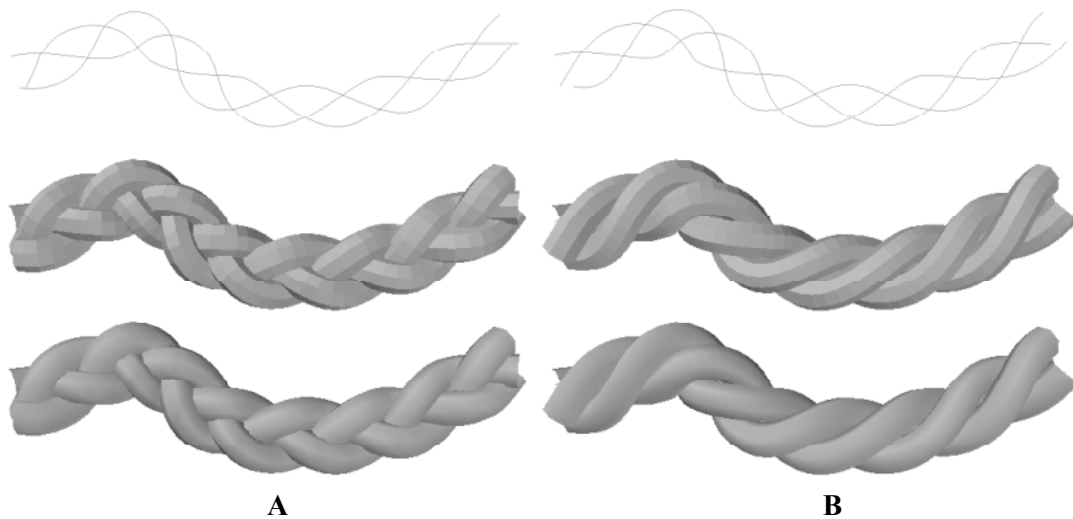


Figure 4.35: *Using the HPGeneralizedCylinder plug-in for braids and twists.*

Figures 4.35 and 4.36 show the use of the `HPGeneralizedCylinder` plug-in. In Figure 4.35 a braid (A) and twist (B) are defined by three curves (top). Using the `HPGeneralizedCylinder` plug-in the curves are converted into generalized cylinders, which can be used to represent a cluster of hair (middle). The generalized cylinders created have eight sides. To create the appearance of a circular generalized cylinder smooth shading is applied (bottom). Figure 4.36 shows two hairstyles. The top hairstyle consists of braids of hair, and the bottom hairstyle consists of twists of hair. Each braid and twist is defined by three curves and has a radius of 2mm (the head models have dimensions of approximately 20 cm x 25 cm) (A). The `HPGeneralizedCylinder` plug-in is used to convert the curves into generalized cylinders that represent clusters of hair (B). The generalized cylinders created for the braids have a radius of 1.4mm, and the generalized cylinders created for the twists have a radius of 1.8mm. The generalized cylinders are given a surface name, default surface properties and a *UV* map. The surface name can be used to change the surface properties of the generalized cylinders, for example the default surface colour can be changed to black (C). The *UV* map name can be used to add textures to the generalized cylinders in order to create the effect of individual strands of hair (see Section 5.4.2).

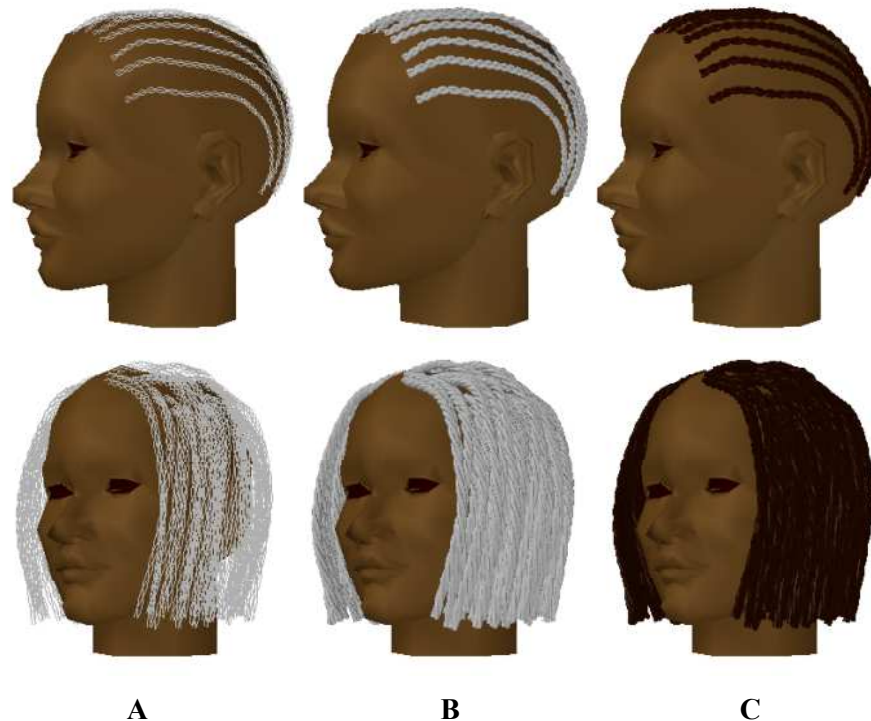


Figure 4.36: Using the `HPGeneralizedCylinder` plug-in.

4.3 Summary

In this chapter we discuss seven plug-ins that we develop in order to implement the three hair modelling techniques discussed in Chapter 3. A description of each plug-in is given together with some of the implementation issues involved in creating the plug-in for LightWave. Using a combination of the plug-ins a user is able to model a variety of African hairstyles containing the three identified forms of African hair: curly hair, straight hair, and braids or twists of hair. The rendering of hair modelled with our plug-ins is discussed in the next chapter.

5 RENDERING AFRICAN HAIR

In order to render modelled African hair, several rendering aspects need to be considered. These aspects include an illumination model, shadowing techniques, anti-aliasing and texturing. In this chapter we discuss the different rendering techniques we use to render African hair that has been modelled using the plug-ins discussed in Chapter 4. The rendering is done using a renderer compatible with the RenderMan Interface, and the African hair modelled with our plug-ins in LightWave is exported as a RIB (RenderMan Interface Bytestream) file using LightMan [Dapper 2001].

5.1 Introduction

5.1.1 Setting Up the Scene

In order to render hair (or any object), a scene containing the modelled hair (or object) together with a camera and lights needs to be set up. We use the classic *three point lighting* set-up [Birn 2000], demonstrated in Figure 5.1.

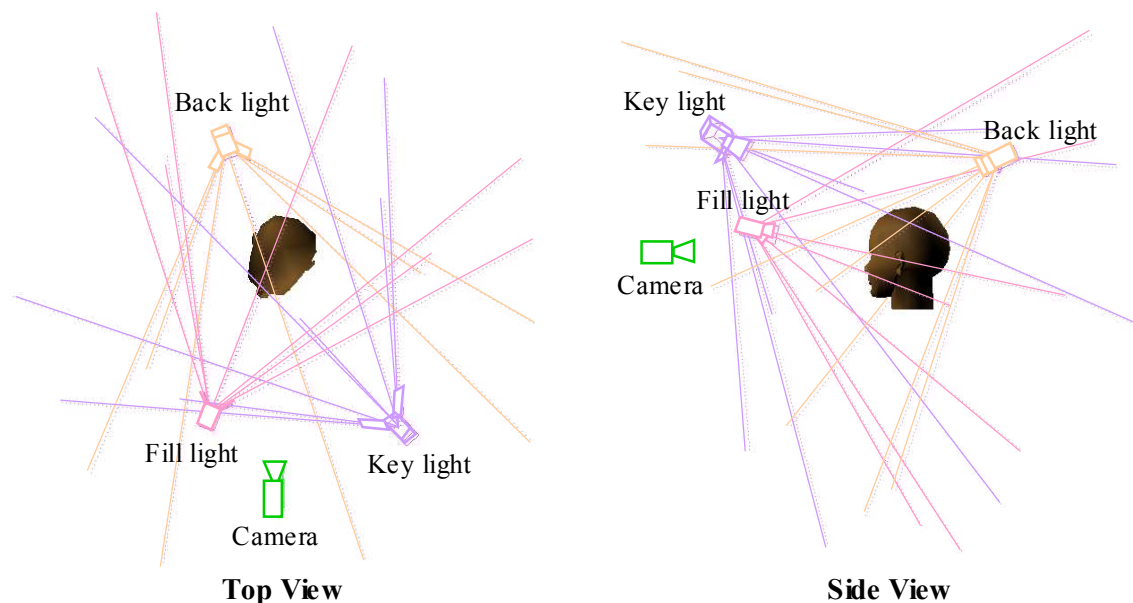


Figure 5.1: *Three point lighting set-up.*

Three spotlights, called a *Key* light, *Fill* light and *Back* light, are placed in the scene. The Key light and Fill light are placed on opposite sides of the camera, and the Back light is placed behind the object. The colours of the lights are set to white. The Key light provides the main lighting in the scene. It has a high intensity (we use 100%), casts shadows and affects specular highlights in

the scene. However, a scene lit with only the key light contains hard shadows and a large contrast between light and dark (see diagram A of Figure 5.2). To soften the shadows and lower the contrast of light and dark, a Fill light is placed in the scene. The Fill light has a much lower intensity than the Key light (we use 40%), and does not cast shadows or affect the specular highlights. The effect of the Fill light can be seen in diagram B of Figure 5.2. The Back light, also called the *Rim* light, is used to create a rim of light around the top or side of the object, which helps to separate the object from the background (especially when they are both dark). The intensity of the Back light is high (we use 80%), and it casts shadows, but it does not affect the specular highlights in the scene. The effect of the Back light can be seen in diagram C of Figure 5.2. Diagram D in Figure 5.2 shows the lighting effect when all three lights are used [Policarpo 2003].

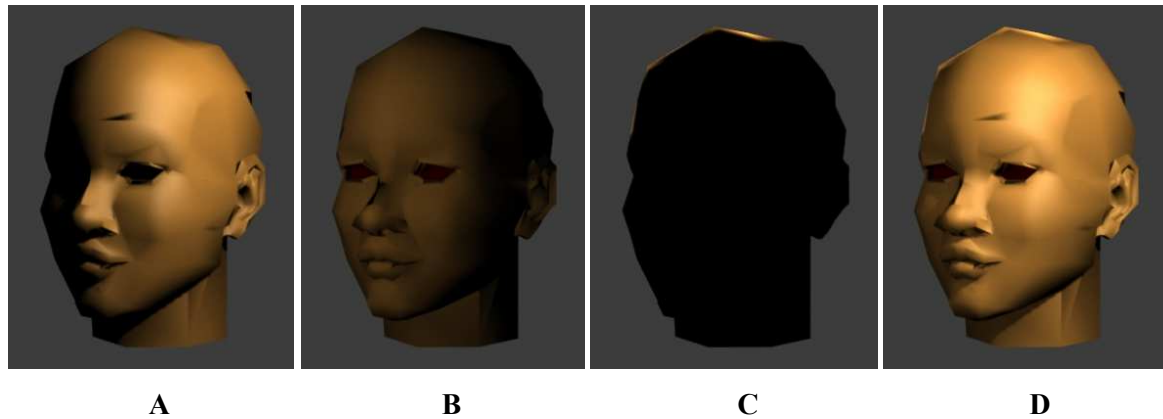


Figure 5.2: *The effect of the Key, Fill, and Back lights, and the final result.*

5.1.2 RenderMan and LightMan

5.1.2.1 RenderMan

We use a renderer compatible with the RenderMan Interface⁵ to render the scene containing the modelled African hair. The RenderMan Interface (or RenderMan) is an industry standard technical specification designed by Pixar [Pixar 2003] to transfer a description of a scene from a modelling program to different rendering programs [Pixar 2000]. The description of the scene is usually passed to the renderer in a file with a RenderMan Interface Bytestream (RIB) file format [Lancaster 2003].

⁵ *RenderMan* is a registered trademark of Pixar.

There are many renderers that adhere to the RenderMan standard. These include Pixar's PhotoRealistic RenderMan (PRMan) [Pixar 2003], Blue Moon Rendering Tools (BMRT) [Gritz 1999], RenderDotC [Dot C Software 2002], 3Delight [3Delight 2002], and Air [SiTex Graphics 2003]. The renderers use different rendering methods to render a scene. For instance RenderDotC and Air use a scanline method, PRMan and 3Delight use the REYES scanline method, and BMRT uses ray tracing [Lancaster 2003].

LightWave has its own renderer, but we choose to use a renderer compatible with RenderMan instead, for a number of reasons:

- RenderMan is an industry standard and is not dependent on the hardware, operating system, or rendering algorithm used [Pixar 2000]. A number of modelling programs have support for exporting a scene as a RIB file [Lancaster 2003]. Using RenderMan our modelled hair can be rendered using any rendering algorithm, or our rendering techniques can be applied to hair that has been modelled in any modelling program that has RenderMan support.
- The RenderMan Interface contains a Shading Language, which is used to write programs called shaders. The shaders are used during rendering to set the material and lighting properties of surfaces [Pixar 2000]. The Shading Language makes developing and implementing an illumination model for hair easy.
- Other reasons, provided by Dapper [2001], include quality (especially when using textures), speed, memory efficiency, and flexibility.

5.1.2.2 LightMan

In order to render the scene containing modelled African hair using a renderer compatible with the RenderMan interface, the scene modelled in LightWave needs to be converted into the RenderMan RIB format. To do this we use LightMan developed by Dapper [2001]. LightMan is a collection of LightWave plug-ins that enable a scene modelled in LightWave to interface with a number of RenderMan compatible Renderers, including PRMan, BMRT, RenderDotC, and Air [Dapper 2001]. The collection of plug-ins include a shader plug-in, which replaces surfaces with RenderMan compatible shaders; an object replacement plug-in, which replaces objects with RIB Archives and custom RIB statements; and a main plug-in, which does the actual RIB export [Dapper 2003].

5.2 Illumination Model

This section looks at the illumination model, and shaders using the RenderMan Interface, that we use to illuminate African hair.

5.2.1 Introduction

5.2.1.1 Using a Conventional Illumination Model

As we pointed out in Section 2.3, the easiest way to render hair is to model each individual strand of hair as a curved cylinder and then render the cylinders using conventional rendering techniques. An example of hair rendered with a conventional illumination model can be seen in Figure 5.3. Each strand of hair is modelled as a simplified cylinder using a thin six-sided generalized cylinder (that has a length of 4m and a radius of 1cm).

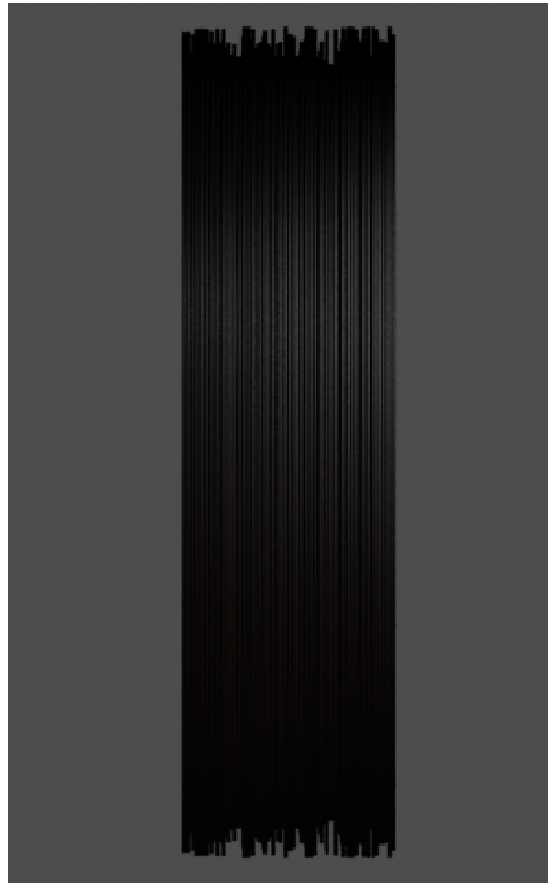


Figure 5.3: *Using a conventional illumination model.*

The illumination model used has an ambient, diffuse and specular component. The diffuse component, $\Psi_{diffuse}$, is calculated using the Lambertian shading model [Foley et al. 1990] and the specular component, $\Psi_{specular}$, is calculated using Blinn's specular model [Blinn 1977], a less expensive version of the Phong illumination model, that uses the halfway vector [Watt and Watt 1992]:

$$\Psi_{diffuse} = K_d (L \cdot N)$$

$$\Psi_{specular} = K_s (N \cdot H)^n, \text{ where } H = (L + V) / 2$$

where K_d and K_s are the diffuse and specular coefficients, n is the Phong exponent specifying the sharpness of the highlight, and L , N , H and V are vectors as shown in Figure 5.4.

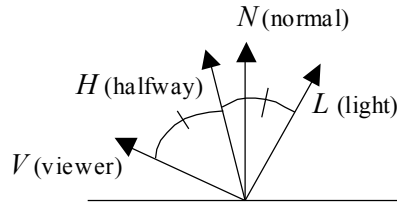


Figure 5.4: Important vectors in a conventional illumination model.

The light intensity, $\Psi_{surface}(p)$, of a point p on a surface of one of the generalized cylinders is calculated using the equation:

$$\Psi_{surface}(p) = L_a K_a + \sum L_i (\Psi_{diffuse}(p) + \Psi_{specular}(p))$$

where L_a and L_i are the light intensities of the ambient light and each light source i , respectively, and K_a is the ambient reflection coefficient.

The result obtained using a conventional illumination model on simplified cylinders (generalized cylinders) is appropriate for African hair. However, due to the large number, complex shape, and narrow width of strands of hair, modelling each strand of hair as a thin, bent cylinder on a model of a head is impractical (because too many polygons are needed). As a result, researchers have developed other methods to model hair (see Section 2.2), and in this thesis we model hair explicitly with polylines (see Sections 3.2, 4.2.2, 4.2.3 and 4.2.6). Unfortunately, the conventional illumination model discussed above is not appropriate for lighting hair modelled with polylines instead of generalized cylinders, because it calculates the light intensity at a point on a surface that has a normal pointing in one direction.

In section 2.3.2 we describe a number of illumination models (or extensions of an illumination model) that have been presented by researchers in order to render hair that is modelled with

polylines. In the section below, we describe the illumination model we use to illuminate African hair.

5.2.2 Our Illumination Model

5.2.2.1 Introduction

Our illumination model consists of an ambient, diffuse and specular component, and like most hair illumination models, it is based on Kajiya and Kay's anisotropic lighting model [1989]. However, a number of researchers have made improvements to different components in Kajiya and Kay's model (see Section 2.3.2), and we combine several of these improvements to develop our model.

5.2.2.2 The Ambient, Diffuse and Specular Components

The ambient component, $\Psi_{ambient}$, is calculated as usual, using:

$$\Psi_{ambient} = L_a K_a$$

where L_a is the total intensity of ambient light in the scene and K_a is the ambient reflection coefficient determined experimentally.

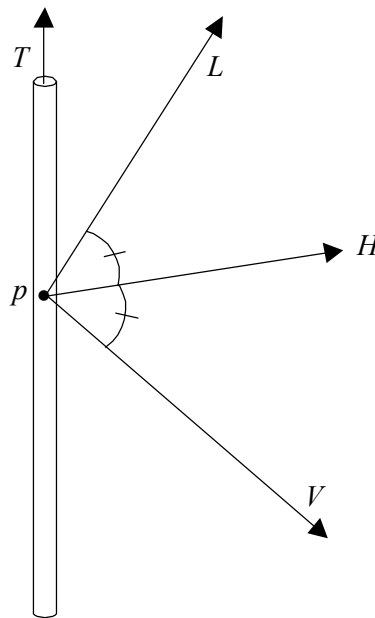


Figure 5.5: Important unit vectors in our illumination model

The diffuse component is calculated using Kajiya and Kay's diffuse term [1989] with the excess brightness exponent presented by Banks [1994] and implemented by Lengyel [2000] and Lengyel et al. [2001]. The equation used is:

$$\Psi_{diffuse} = K_d \{1 - (T \cdot L)^2\}^{k/2}$$

where T and L are the tangent and light vectors respectively (see Figure 5.5), K_d is the diffuse reflection coefficient determined experimentally, and k is the excess brightness exponent. Banks recommends a value of 4.7635 for k . The equation is the same as Kajiya and Kay's diffuse equation when k has a value of 1.

The specular component, $\Psi_{specular}$ (implemented previously by Anjyo et al. [1992], Lengyel [2000] and Lengyel et al. [2001]) is calculated using Blinn's specular model [Blinn 1977], which simplifies the computation for the specular term in Kajiya and Kay's illumination model [1989]. The equation used is:

$$\Psi_{specular}(p) = K_s \{1 - (T \cdot H)^2\}^{n/2}$$

where K_s is the specular reflection coefficient, n is the Phong exponent, H is the halfway vector, and T is the tangent vector along the length of the hair (see Figure 5.5). The values of K_s and n are determined experimentally, and H is calculated as the vector between L (the light vector) and V (the eye vector):

$$H = (L + V) / |L + V|$$

Together the three components can be used to calculate the light intensity, Ψ_{hair} , of a point p on a hair strand using the equation:

$$\Psi_{hair}(p) = L_a K_a + \sum L_i (\Psi_{diffuse}(p) + \Psi_{specular}(p))$$

where L_i is the light intensity of each light source i . However, this equation does not consider the direction of the viewer when calculating the diffuse component, and hairs are fully lit even when the light source is behind the hair.

5.2.2.3 Adjusting the Relative Transmission and Reflection of Light

For light-coloured hair (e.g. blonde hair) our equation above provides an acceptable approximation of the light intensity at a point, even though the direction of the viewer is not considered, because the hair is partially transparent and transmits light through the hair. However, African hair contains dense pigment granules that cause it to be far more opaque than other types of hair [Deedrick 2000]. African hair therefore does not transmit light through the

hair, and the direction of the viewer with respect to the direction of the light needs to be taken into account in the equation above.

To add directionality to our hair illumination model so that both light reflection and light transmission are considered, we use a technique developed by Goldman [1997]. We add a directional attenuation factor, f_{dir} , to our model, so that the light intensity, Ψ_{hair} , at a point p on a hair strand is calculated using the equation:

$$\Psi_{hair}(p) = L_a K_a + \sum L_i f_{dir} (\Psi_{diffuse}(p) + \Psi_{specular}(p))$$

where f_{dir} is the directional attenuation factor. The details of the attenuation factor f_{dir} can be found in Section 2.3.2.4.

5.2.2.4 Extra Features that Could be Added

Other features that could be added to our illumination model include backlighting and a variation in colour between strands of hair or within a strand of hair. However, these features are more important for light-coloured hair. Since African hair is opaque it does not transmit light and haloing is not as prominent under backlighting conditions [LeBlanc et al. 1991]. Also the colour of adjacent strands of hair, or within an individual strand of hair, tends to be the same (see Figure 5.6). However, a user using our plug-ins to create explicitly modelled hair is able to change the surface properties of individual strands of hair manually, if they want different strands of hair to be different colours. To change the colour within a strand of hair they would need to add a vertex colour map to the strands of hair [Nakakihara 2001].



Figure 5.6: *All strands of hair have the same colour.*

5.2.3 Implementing the Illumination Model as a Shader

Since we render hair with a renderer that is compatible with the RenderMan Interface, we implement our illumination model as a shader using the RenderMan Shading Language:

```
surface HairShader (float Ka = 0.0, Kd = 0.5, Ks = 0.5, Kn = 0.1, Kb = 1.0, Fr = 1.0, Ft = 1.0;)
{
    vector T = normalize (dPdV); /* The hair tangent vector */
    vector H, Ln, V = -normalize(I); /* The halfway, light, and eye vectors */
    uniform float nondiff, nonspec; /* The light's contribution */
    float k, Fdir; /* The directional attenuation factor */

    /* The ambient component */
    Ci = Cs * Os * Ka * ambient();

    /* The diffuse and specular components */
    illuminance (P) {
        /* Get the next non-ambient light vector */
        Ln = normalize(L);
        /* Calculate the directional attenuation factor */
        k = ((T^Ln).(T^V))/((length(T^Ln))*(length(T^V)));
        Fdir = (((1 + k)/2) * Fr) + (((1 - k)/2) * Ft);
        /* Calculate the diffuse component */
        nondiff = 0;
        lightsource ("__nondiffuse", nondiff);
        if (nondiff < 1)
            Ci += Ci * Cs * Os * Kd * pow ((1 - pow ((T.Ln), 2)), Kb/2) * Fdir;
        /* Calculate the specular component */
        nonspec = 0;
        lightsource ("__nonspecular", nonspec);
        if (nonspec < 1) {
            H = normalize(Ln+V);
            Ci += Ci * Os * Ks * pow ((1 - pow ((T.H), 2)), 1/(Kn*2)) * Fdir;
        }
    }

    /* The incident ray opacity */
    Oi = Os;
}
```

Not all lights in a scene are required to contribute to the specular and diffuse components. For instance our scene is set up with three lights and only the Key light contributes to the specular component (see Section 5.1.1). The statements:

```
nondiff = 0;                                nonspec = 0;
lightsource ("__nondiffuse", nondiff);      and lightsource ("__nonspecular", nonspec);
if (nondiff < 1) {...}                      if (nonspec < 1) {...}
```

are used to determine whether a light contributes to the diffuse and specular components respectively. When the scene is set up in LightWave the user is able to set the properties of the lights. LightMan exports the scene into a RenderMan RIB file format and sets the value of

`_nondiffuse` and `_nonspecular` to 0 if the light contributes to the component and 1 if the light does not contribute to the component [Dapper 2001].

The value of the Phong exponent, n , is calculated as $1/Kn$, rather than Kn (as done by the default RenderMan shaders). Therefore, the value of Kn will usually be a value between 0 and 1 rather than some large value such as 200 [Pixar 2000].

5.2.4 Implementing the Illumination Model in Textured Layers

The illumination model, implemented as a shader in the section above, is developed for rendering hair that is modelled explicitly with polylines. However the same illumination model can also be used to add lighting effects to hair that is modelled using textured layers (discussed in Section 3.1). To apply the illumination model to textured layers, slight modifications are made to the shader presented in the section above. The modified shader can be found in Appendix A.

Since the hair strands are represented by a texture rather than explicitly, the tangent vector for each hair strand is stored in a texture map (represented by `tangentTexture`), and the line that obtains the tangent vector in the original shader:

is replaced with:

```
vector T = normalize (dPdv);    /* The hair tangent vector */  
  
vector T;  
color t = color texture (tangentTexture);  
setxcomp (T, comp (t, 0));      /* x is represented by the red component */  
setycomp (T, comp (t, 1));      /* y is represented by the green component */  
setzcomp (T, comp (t, 2));      /* z is represented by the blue component */  
normalize (T);
```

More information about the texture map can be found in Section 5.4.1.

5.3 Shadowing

Adding shadows to hair is important in order to create realistic looking hair. In this section we discuss the methods we use to add shadows to our explicitly modelled hair and to create the effect of self-shadowing in textured layers that represent hair.

5.3.1 Adding Shadows to Explicitly Modelled Hair

5.3.1.1 Introduction

In Section 2.3.3 we discuss several methods that can be used to add shadows to hair. In order to add shadows to our explicitly modelled hair we use traditional shadow maps. In Section 2.3.3 we point out that other shadowing methods, such as ray tracing and deep shadow maps, produce higher quality shadows than traditional shadow maps. However, we choose to use traditional shadow maps because they have faster rendering speeds than other shadowing techniques [Tezel and Teo 2002]. We render our hair using a renderer that adheres to the RenderMan Interface. All renderers that implement the RenderMan Interface are required to be able to create depth maps (traditional shadow maps), whereas not all the renderers are able to implement ray tracing [Pixar 2000]. Deep shadow maps would be a better technique to use and are implemented in Release 11 of Pixar's renderer PhotoRealistic RenderMan [Pixar 2002].

5.3.1.2 Implementing Shadow Maps Using RenderMan

To implement shadow maps using RenderMan, a shadow map must first be created for each light source, in the scene, that casts shadows (in our case the Key light and Back light). This can be done using the `RiDisplay` and `RiMakeShadow` procedures:

```
RiDisplay "shadowmap.pic" "zfile" "z"
RiMakeShadow "shadowmap.pic" "shadow.tx"
```

The `RiDisplay` statement creates a file called "shadowmap.pic" that contains depth value (z value) information generated from the point of view of the related light source, and the `RiMakeShadow` statement converts the shadowmap.pic file into a shadow map [Pixar 2000].

The shadow maps are then accessed by light source shaders (using the shadow shading language function), which calculate the amount of light that arrives at a point:

```
RiLightSource "lwspotlight" ... "shadowname" "shadowmap.tx".
shadow (shadowname, Ps)
```

The `RiLightSource` statement specifies the light source shader (e.g. "lwspotlight") and shadow map (e.g. "shadowmap.tx") to be used, and the shadow function is called within the light source shader to access the shadow map and obtain a value for the point Ps .

Since we model hair and place it in a scene with three lights in LightWave, and then use LightMan to export the scene before rendering it with a RenderMan compatible renderer, most of

the shadow map implementation is done for us. In LightWave we specify which of the lights cast shadows, and we set values for the area of the scene affected by the light and the resolution of the shadow map. We then use LightMan to export the hair and lights. LightMan automatically adds the statements required to produce the necessary shadow maps into the RIB file describing the scene, and creates a light source shader for LightWave spotlights (“`lwspotlight.sl`”) that is able to access shadow maps.

5.3.1.3 Results

Figure 5.7 demonstrates the result when hair (modelled explicitly with polylines) is rendered without shadowing (A) and with shadow maps (B). The hair rendered with shadow maps is more realistic than the hair rendered without, and seems to have more depth. The hair rendered without shadows appears very flat.

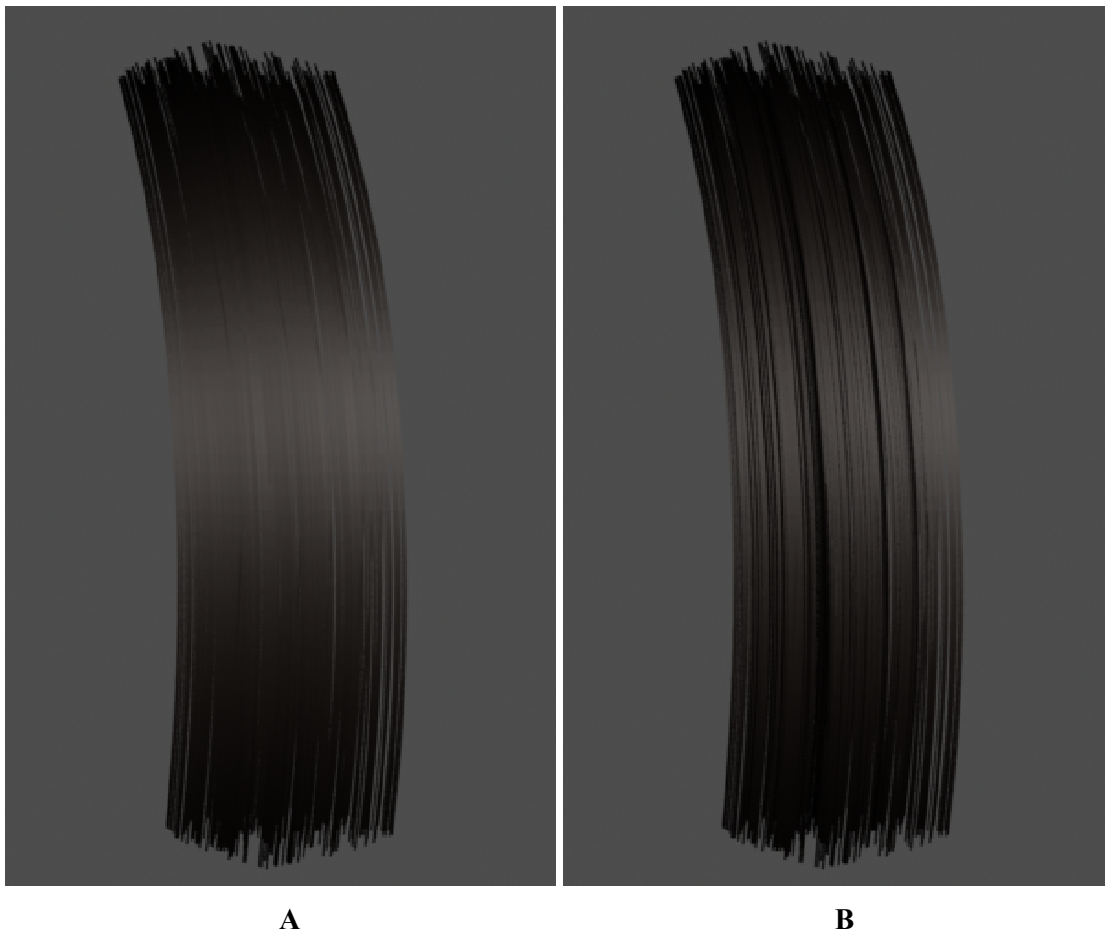


Figure 5.7: *Rendering hair without shadows and with traditional shadow maps.*

5.3.2 Self-shadowing in Layers

We model curly African hair with textured layers (see Section 3.1). To add the effect of self-shadowing in the hair we assume the hair represented by layers closer to the scalp is in more shadow than hair further away from the scalp. To implement this effect we use a technique implemented by Lengyel [2000] and Lengyel et al. [2001]. We make the surface colour of the lower layers closer to the scalp darker than the colour of higher layers further away from the scalp.

5.3.2.1 Results

Figure 5.8 shows the result when hair, represented by textured layers, is rendered without self-shadowing (top) and with self-shadowing using the technique described above (bottom).

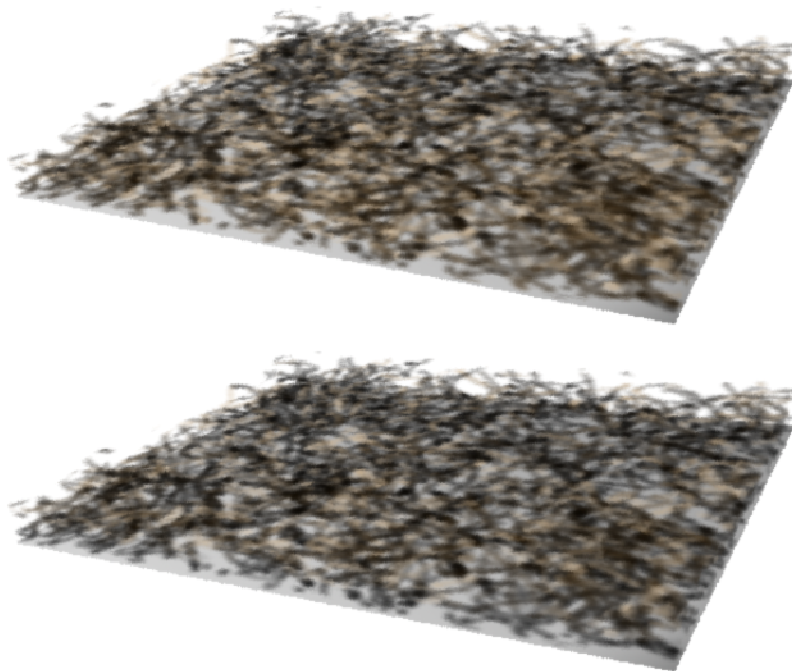


Figure 5.8: *Rendering hair without self-shadowing and with self-shadowing.*

5.4 Texturing

We use texturing in several instances to create the effect of hair or to enhance the realism of a hairstyle created with our plug-ins (discussed in Chapters 3 and 4). We apply texture maps to layers (created with the HPLayer plug-in) to create the effect of curly hair, to generalized cylinders (created with the HPGeneralizedCylinder plug-in) to create the effect of many strands of hair, and to a cap placed on the head model to enhance the realism of a hairstyle. The creation and mapping of the texture maps are discussed below.

5.4.1 Texturing Layers to Create the Effect of Hair

5.4.1.1 Introduction

To create the effect of short, curly African hair we use an implicit model that represents hair with concentric textured layers (see Section 3.1). To create concentric layers we use the HPLayer plug-in (discussed in Section 4.2.1). Once the layers have been created they must be textured to create the effect of hair. We apply an opacity map and a tangent map to each layer. The opacity maps define the shape, thickness and density of the hair strands, and the tangent maps store the tangent vectors of hair strands, which are required when using our illumination model (see Section 5.2.4). The layers are also given appropriate values for their surface properties, which are used in the illumination model when rendering the hair. The surface properties include a surface colour (which determines the colour of the hair and is used for creating the effect of self-shadowing, discussed in Section 5.3.2), diffuse reflection coefficient, specular reflection coefficient, Phong exponent and brightness exponent (see Section 5.2.2).

In order to create the opacity and tangent maps we generate random curly splines in 3D space, and then take slices of the 3D space at specific heights. Each slice is used to create an opacity map and a tangent map. Areas where splines exist are shown as white areas on the opacity map and represent hair, while areas where no spline exists are represented with black. The tangent of each spline at the point where the slice is taken is stored in the tangent map using a colour. The x , y , and z components of each tangent vector are represented by the red, green, and blue components of the colour, respectively. Figure 5.9 shows an example of an opacity map (A) and a tangent map (B).

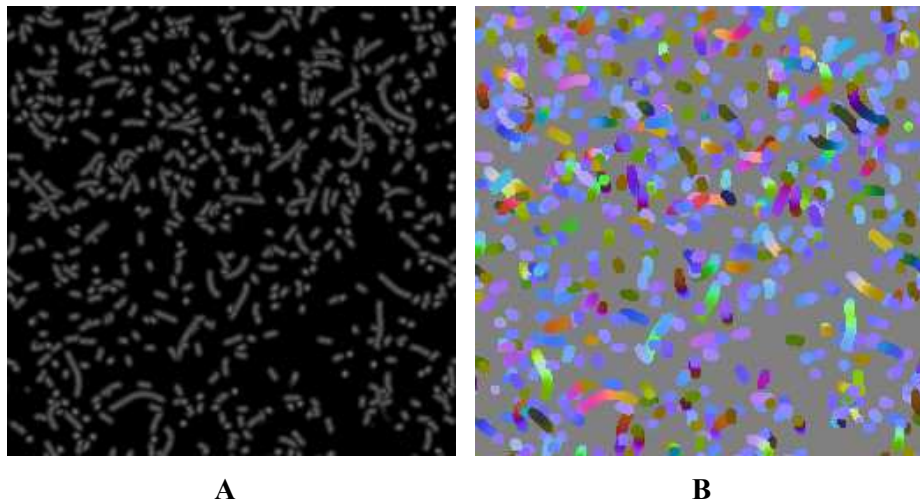


Figure 5.9: *An opacity map and a tangent map.*

5.4.1.2 Implementing the Texturing with a Shader

A set (opacity map and tangent map) of texture maps is assigned to each layer, and the layers are rendered with a shader that uses the texture maps. The shader we use is similar to our shader discussed in Section 5.2.3, except that the tangent vectors are obtained from the tangent map (see Section 5.2.4), and the surface opacity, O_s , is replaced with:

$$O_s * \text{float texture}(\text{opacityTexture})$$

where `opacityTexture` represents the opacity map. A copy of the complete shader can be found in Appendix A.

5.4.1.3 Results

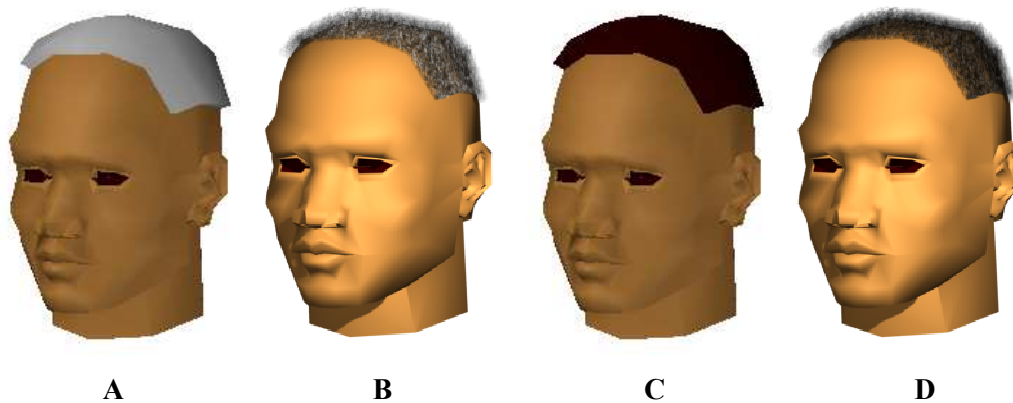


Figure 5.10: *Applying textures to concentric layers.*

Diagrams B and D in Figure 5.10 show the effect when texture maps are applied to concentric layers to create the effect of hair. Diagrams A and C show the hair modelled with concentric layers, but without texture maps applied. In Diagrams A and B the surface colour of the layers is grey. The surface colour of the layers in Diagrams C and D has been set to black.

5.4.2 Representing Individual Hair Strands in Braids and Twists

5.4.2.1 Introduction

To model braids and twists of hair we define wisps of hair using curves, which are later converted into generalized cylinders with the HPGeneralizedCylinder plug-in (see Sections 3.3 and 4.2.7). The generalized cylinders represent wisps of hair. To create the effect of individual strands of hair in each wisp, texture maps can be applied to the surfaces of the generalized cylinders. The texture maps may be photographs of hair or textures created with a paint program. Alternatively, procedural textures could be added to create the effect of individual strands of hair. We use a paint program to create hair texture maps. The method we use is based on a technique presented by Koh and Huang [2001] (see Section 2.3.4.3). We create a texture map (or alpha map) that defines transparent (no hair) and opaque (hair) areas on the surface of the generalized cylinders. The alpha map is used to create the illusion of complex geometry. A second texture map (or colour map) may also be created and mapped onto the generalized cylinders to define the colour of the hair strands. However, the colour map is not essential. The colour of the hair strands will be determined by the surface colour of the generalized cylinder representing them if a colour map is not supplied. The surface colour of the generalized cylinders can be changed to any desired colour.

5.4.2.2 Texturing Generalized Cylinders

In order to apply a texture map to a surface, the texture map must be mapped onto the surface so that each point on the surface is related to a unique point in the texture map (represented by U and V coordinates). Using LightWave, a user may set the UV coordinates for each point manually or the user may apply a projection that automatically determines UV coordinates for each point. The choices of projections include planar, cylindrical, and spherical projections. However, the projections do not always produce accurate results, and some of the UV coordinates often have to be adjusted manually [Nakakihara 2001].

To make texturing generalized cylinders easier, our `HPGeneralizedCylinder` plug-in automatically defines the UV coordinates of the polygon vertices in a generalized cylinder. When the plug-in is run, the user is asked to provide a surface name and a UV map name. The surface name is used to identify the surface of the generalized cylinder. Using LightWave's Surface Editor panel [Nakakihara 2001], the properties of the surface (e.g. colour, transparency, and specularity) can be changed by the user, and a texture can be applied to the surface. Because the UV coordinates are automatically defined, all a user has to do is select "UV" as the type of projection, the name of the generalized cylinder's UV map (which they provided when they ran the plug-in), and an image (for the texture map).

Our `HPGeneralizedCylinder` plug-in assigns UV coordinates, so that an entire image is mapped *vertically* and *once* onto a generalized cylinder: the orientations of the patterns in the image are important and the image is not tiled over the generalized cylinder. The mapping used is shown in Figure 5.11.

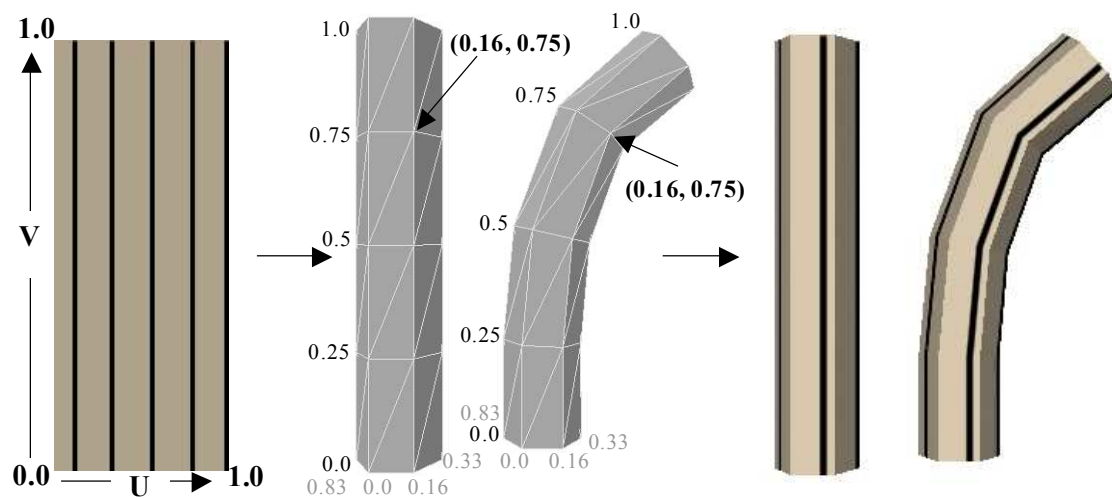


Figure 5.11: Mapping a texture onto a generalized cylinder.

A seam problem occurs when a texture is wrapped around geometry so that some vertices (along the seam) require two UV coordinate values [Nakakihara 2001]. Considering the six-sided generalized cylinder in Figure 5.11, a seam problem will occur along the generalized cylinders because the points with a U value of 0.0 (the left side of the texture map) also need a U value of 1.0 (the right side of the texture map). Figure 5.12 shows an example of a cylinder with and without a seam problem. In the left diagram (with a seam problem) most of the texture has been squashed and mapped backwards onto a polygon (because the U value on the left side of the

polygons should have a value of 0.0 and 1.0. Instead it only has a value of 1.0, causing a mapping with the values 1.0 to 0.16 instead of 0.0 to 0.16) (see Figure 5.13).

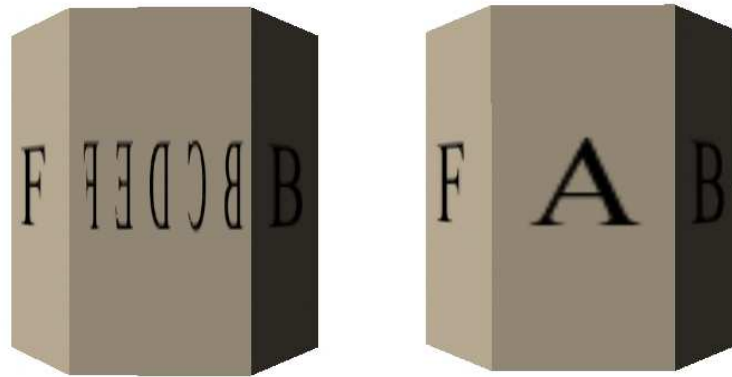


Figure 5.12: *Cylinders with and without the seam problem.*

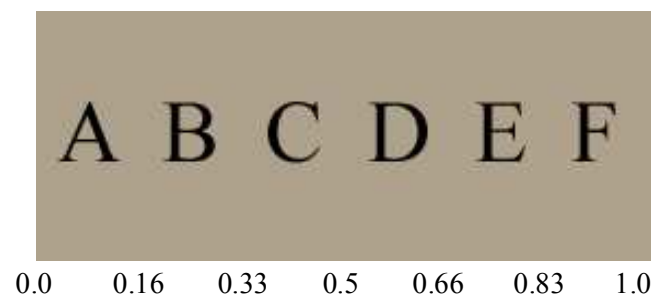


Figure 5.13: *The texture and U values.*

We take the seam problem into consideration when implementing the HPGeneralizedCylinder plug-in. The polygon vertices along the generalized cylinder where the sides of the texture meet ($U = 0.0$ and $U = 1.0$) are given two sets of UV values. This is implemented using the `pntVPMMap` function from the LightWave SDK [Wright 2001].

5.4.2.3 Results

Figure 5.14 shows an example of a texture map applied to three generalized cylinders to create the effect of three clusters of hair in a braid. Diagram A shows a braid represented by three generalized cylinders. In Diagram B the surface colour of the generalized cylinders has been changed to black. Diagram C shows the generalized cylinders with the texture map (shown on the right in Figure 5.14) applied as an alpha map. The texture map defines the opacity across the surface of each generalized cylinder, and is used to create the effect of many strands of hair.

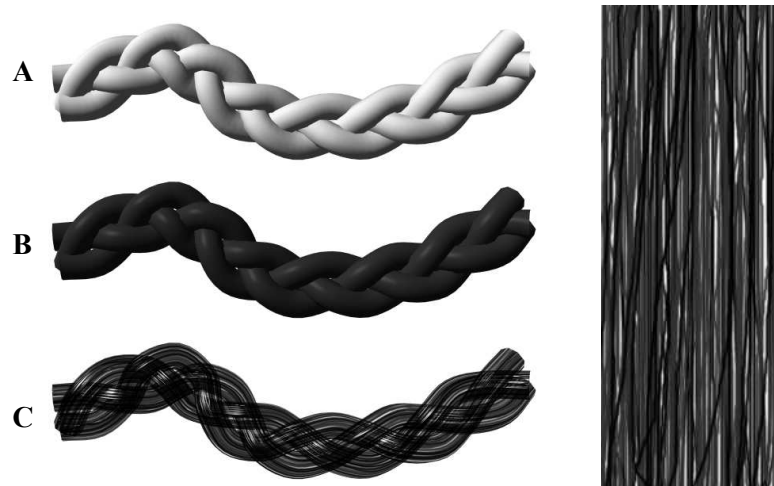


Figure 5.14: *Texturing generalized cylinders.*

5.4.3 Adding Additional Texture to the Head

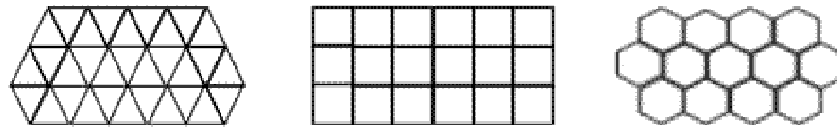
5.4.3.1 Introduction

For some hairstyles that contain braids and twists, flatly parted hair can be seen on the scalp (see Section 3.3.5). This hair is not part of the braids or twists. When modelling and rendering these hairstyles, extra texture maps can be placed onto the scalp to represent the flatly parted hair. Our technique of adding an extra texture map to the scalp is inspired by the cap technique [Comet 2000] (see Section 2.3.4.3).

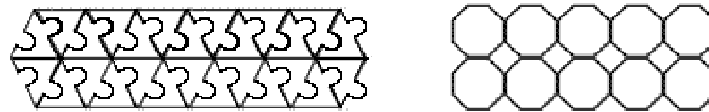
5.4.3.2 Creating the Texture Maps

The parted hair forms a tessellation⁶ over the scalp [Gilmer 1998]. (See Figure 5.15 for more information about tessellations.) Since a tessellation occurs, instead of creating a single texture map for the entire scalp, a smaller texture map, which uses less memory and requires less redundant work to create, can be created and then tiled over the surface [Neyret and Cani 1999].

⁶ A tessellation is a pattern formed when a fundamental shape is tiled in a way so that each shape does not overlap another and no region is left uncovered [Gilmer 1998].



The triangle, square and hexagon are the only regular polygons that are able to form a tessellation [Gilmer 1998].



“Tessellations can be produced from irregular polygons or a combination of polygons” [Gilmer 1998].



Irregular shapes that tessellate can be created by modifying one of the three regular polygons that tessellate [Gilmer 1998].

Figure 5.15: *Tessellations.*

The small texture can be created using 2D painting software. Since the texture is tiled, it must have a toroidal⁷ topology so that it joins correctly with itself when tiled (no noticeable seams must appear between two tiles) [Neyret and Cani 1999].

Instead of tiling a texture on the scalp to represent parted hair, a few textured layers can be added to the scalp to form a 3D cap. The textured layers can be created using our HPLayer plug-in (see Section 4.2.1) and the method discussed in Section 3.1.

In real life, the positions of the shapes in the tessellation are determined by the position of the braids or twists in a hairstyle. When applying an additional texture map to represent flatly parted hair on a scalp, it is important to make sure that the positions of the shapes in the texture representing the flatly parted hair correspond to the positions of the braids or twists on the head. One way to make sure that they correspond is to create an appropriate texture, and map it onto the scalp, before defining the braids or twists in the hairstyle. Using this method, the positions of the braids or twists in a hairstyle are determined by the positions of the shapes in the tessellation. The method produces accurate results when a tessellation that corresponds to the desired positions of the braids or twists is used.

⁷ The left edge of the texture fits next to the right edge of the texture, and the top edge fits with the bottom edge [Neyret and Cani 1999].

5.4.3.3 Results

Figure 5.16 shows an example of an additional texture that can be placed onto a scalp to represent flatly parted hair. Diagram A shows the shape that tessellates over the scalp, and Diagram B shows a texture map with a toroidal topology that can be tiled over a surface to form a tessellation of the shape. Figure 5.17 illustrates the texture tiled onto the scalp of a head model.

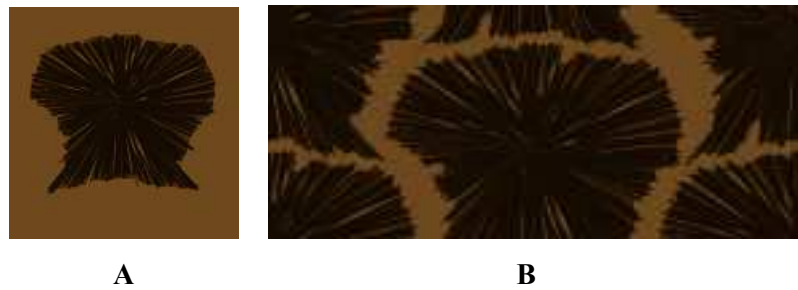


Figure 5.16: *A toroidal texture.*



Figure 5.17: *Adding an additional texture onto the scalp.*

Figure 5.18 shows an example of a texture map that is tiled onto four concentric layers created with the HPLayer plug-in. The textured layers form a 3D cap on the scalp, which represents flatly parted hair.

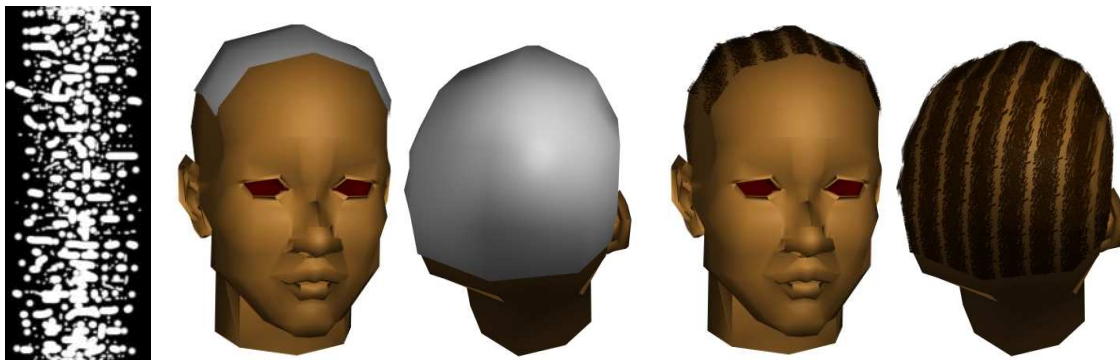


Figure 5.18: *Adding additional textured layers onto the scalp.*

5.5 Anti-aliasing

5.5.1 Introduction

To reduce aliasing in rendered hair we use supersampling, which is supported by RenderMan. Supersampling involves sampling at a higher resolution than the display resolution, and then filtering the samples to produce the final image pixels (see Section 2.3.5.2).

5.5.2 Supersampling with RenderMan

The number of samples taken and the type of filter used are set with the `RiPixelSamples` and `RiPixelFilter` RenderMan commands respectively:

```
RiPixelSamples xsamples ysamples  
RiPixelFilter filterfunc xwidth ywidth
```

When only one sample is taken per pixel (“`RiPixelSamples 1 1`” is called) anti-aliasing is not implemented. Sampling rates greater than one in the horizontal (`xsamples`) and vertical (`ysamples`) directions implement anti-aliasing and reduce aliasing artefacts. Larger sampling rates produce higher quality images than lower sampling rates, but they have slower rendering times (see our results in Section 6.3.4.1) [Pixar 2000].

RenderMan supports a number of standard filter functions, including a box filter, Catmull-Rom filter, Gaussian filter and Sinc filter. The filter type is specified by `filterfunc` in the `RiPixelFilter` command, and `xwidth` and `ywidth` set the size of the filter in pixels [Pixar 2000]. The type and size of the filter we use is obtained through experimentation.

5.5.3 Using Textures

In Section 2.3.5.4 we state that texture maps applied to surfaces are prone to aliasing artefacts when the resolution of the texture is not well matched to the display resolution. To reduce the aliasing artefacts a technique known as mip-mapping can be applied. Mip-mapping is usually needed when the distance between the viewer and the textured surface changes. Since hair animation and dynamics is not the focus of this thesis, we do not consider the effect produced when the distance between the viewer and the textured surface changes, and we do not use mip-mapping when applying textures to represent hair. Instead we apply textures with resolutions that

match the display resolution (through trial and error), for the single distance between the viewer and textured surface that we are using, to avoid aliasing artefacts in the textures.

5.6 Summary

In this chapter we discuss the rendering techniques we use in order to render hair that has been modelled using our plug-ins discussed in Chapter 4. These rendering techniques include several rendering aspects: an illumination model, shadowing techniques, texturing and anti-aliasing. The rendering is done using a renderer compatible with the RenderMan Interface, and the African hair modelled with our plug-ins in LightWave is exported as a RIB file using LightMan. In the next section we discuss several aspects relating to our modelling and rendering techniques and present results.

6 DISCUSSIONS AND RESULTS

6.1 Introduction

This thesis presents hair modelling and rendering techniques that that can be used to create a variety of African hairstyles with ease. In Chapter 3 we discuss the hair modelling techniques we develop in order to model curly hair, straight hair, and braids and twists of hair. The techniques not only model the different forms of hair, but also place the hair modelled onto a model of a head, and are developed so that the problem of modelling a large number of hair strands is overcome. In Chapter 4 we consider the implementation of the techniques using plug-ins for LightWave, allowing a user to model a variety of African hairstyles interactively. In Chapter 5 we discuss the rendering techniques we use in order to render the hair, which include an illumination model, shadowing techniques, texturing, and anti-aliasing. In this chapter we discuss significant results and several issues regarding our hair modelling and rendering techniques.

This Chapter is divided into three parts. In the first part (Section 6.2) we show results and discuss issues relating to our hair modelling techniques. For each technique we show that different hairstyles can be created with relative ease, and examine the effect that certain variables have on rendering time and image quality. The shape of a natural curly hairstyle modelled with our implicit technique may appear unrealistic due to the structure of the underlying mesh. We discuss a method for improving the shape. We also discuss the issue of stray hairs that may appear when modelling hair using our explicit technique and overlapping clusters that may occur when modelling braids with our cluster technique. In the second part (Section 6.3) we show results and discuss issues relating to our hair rendering techniques. We examine the effects produced by different components of our illumination model, with particular reference to black African hair, and show the result of rendering implicitly modelled hair represented by textured layers with a modified version of our illumination model. We examine the effect that different shadow map resolutions and self-shadowing have on rendering time and image quality. We discuss several issues relating to textures, and we investigate the effect of using different sampling rates and filters when implementing supersampling to remove aliasing artefacts. In the third part (Section 6.4) we show that our hair modelling and rendering techniques can be used to create a variety of African hairstyles by presenting a number of hairstyles created with the techniques.

6.2 Modelling Results

6.2.1 Modelling Natural Curly Hair

6.2.1.1 Modelling Different hairstyles with Our Implicit Technique

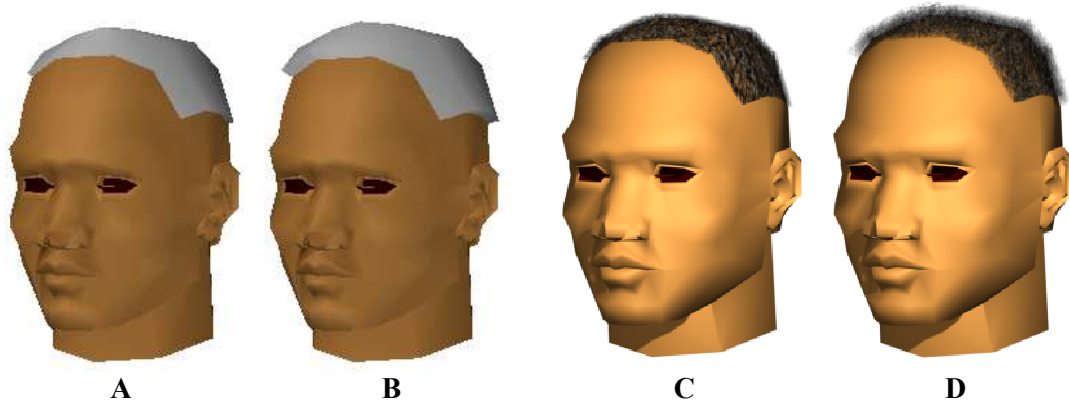


Figure 6.1: *Two hairstyles modelled with concentric layers.*

Figure 6.1 shows two short, curly hairstyles modelled with our implicit hair modelling technique. The first hairstyle (shown in diagrams A and C) consists of very short hair, while the second hairstyle (shown in diagrams B and D) consists of slightly longer hair. Both hairstyles are represented by 20 concentric layers modelled with the HPLayer plug-in. The total height of the layers in the first hairstyle is 5cm, while the total height of the layers in the second hairstyle is 10cm (the head model has dimensions of approximately 180cm x 250cm). Diagrams A and B show the modelled hairstyles, while diagrams C and D show the final rendered results. It takes approximately 5–10 minutes to model the hairstyles, apply the pre-processed texture maps and set the surface properties. More examples of different hairstyles created with our implicit hair modelling technique can be found in Section 6.3.3.

6.2.1.2 The Shape of a Natural Curly Hairstyle

Hair modelled with our implicit hair modelling technique is represented by concentric textured layers that are created parallel to the scalp above selected polygons. The shape of the layers is determined by the selected polygons, causing the boundary of the hairstyle represented by the layers to be dictated by the underlying mesh. This may cause the shape of the hairstyle to be unrealistic. To improve the shape of a hairstyle an extra opacity map that defines the overall shape of the hairstyle may be applied to each layer.

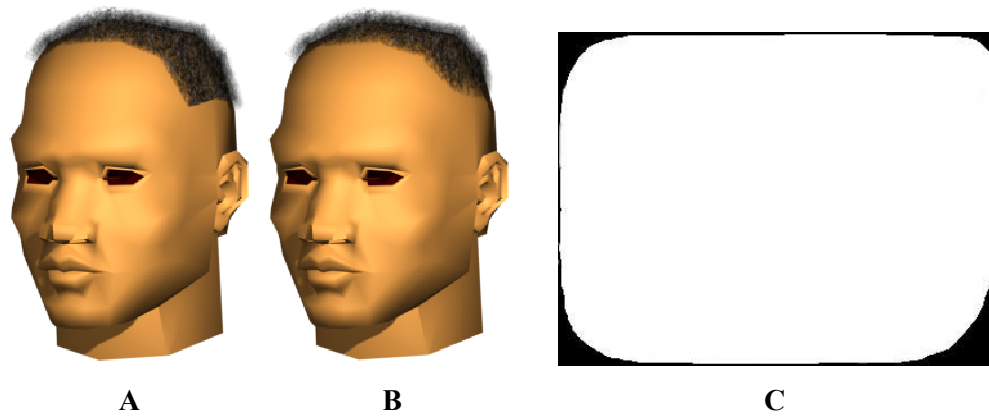


Figure 6.2: *Improving the shape of a natural curly hairstyle.*

Diagram A in Figure 6.2 shows a hairstyle with a shape dictated by the underlying mesh. Diagram B shows the hairstyle with a more realistic shape, and Diagram C shows the extra opacity map that was applied to each layer to define the shape of the hairstyle in Diagram B.

6.2.1.3 The Number of Layers Used

Figure 6.3 shows the rendering times of hair samples created with different numbers of layers. The graph shows that the rendering time increases linearly, in proportion to the number of layers used.

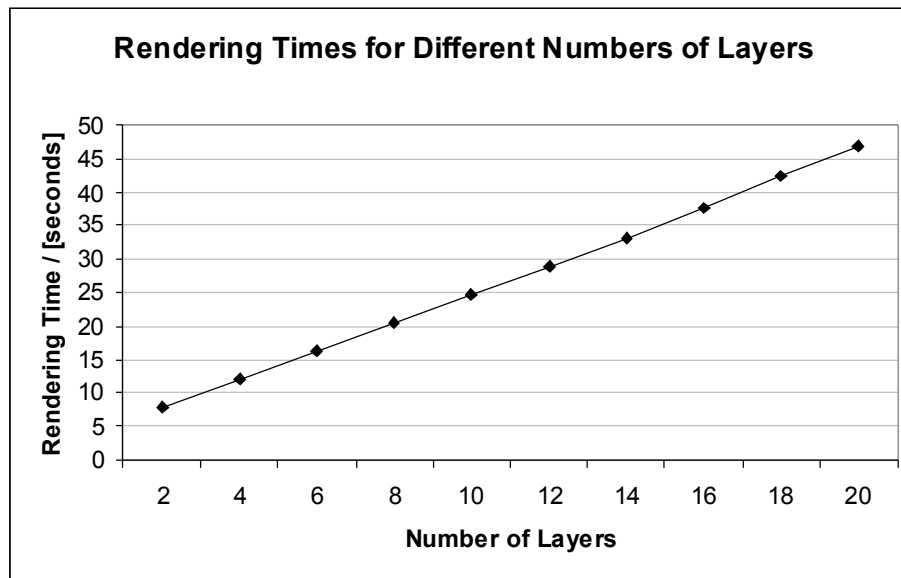


Figure 6.3: *Rendering times when using different numbers of layers.*

The rendered hair samples, created with different numbers of layers, can be seen in Figure 6.4. The samples show that larger numbers of layers produce better results, and hair represented by too few layers does not look realistic. In this example, at least 12 layers are needed to produce reasonably realistic-looking hair. However, because the number of layers affects the rendering time, we would not suggest using more layers than are needed.

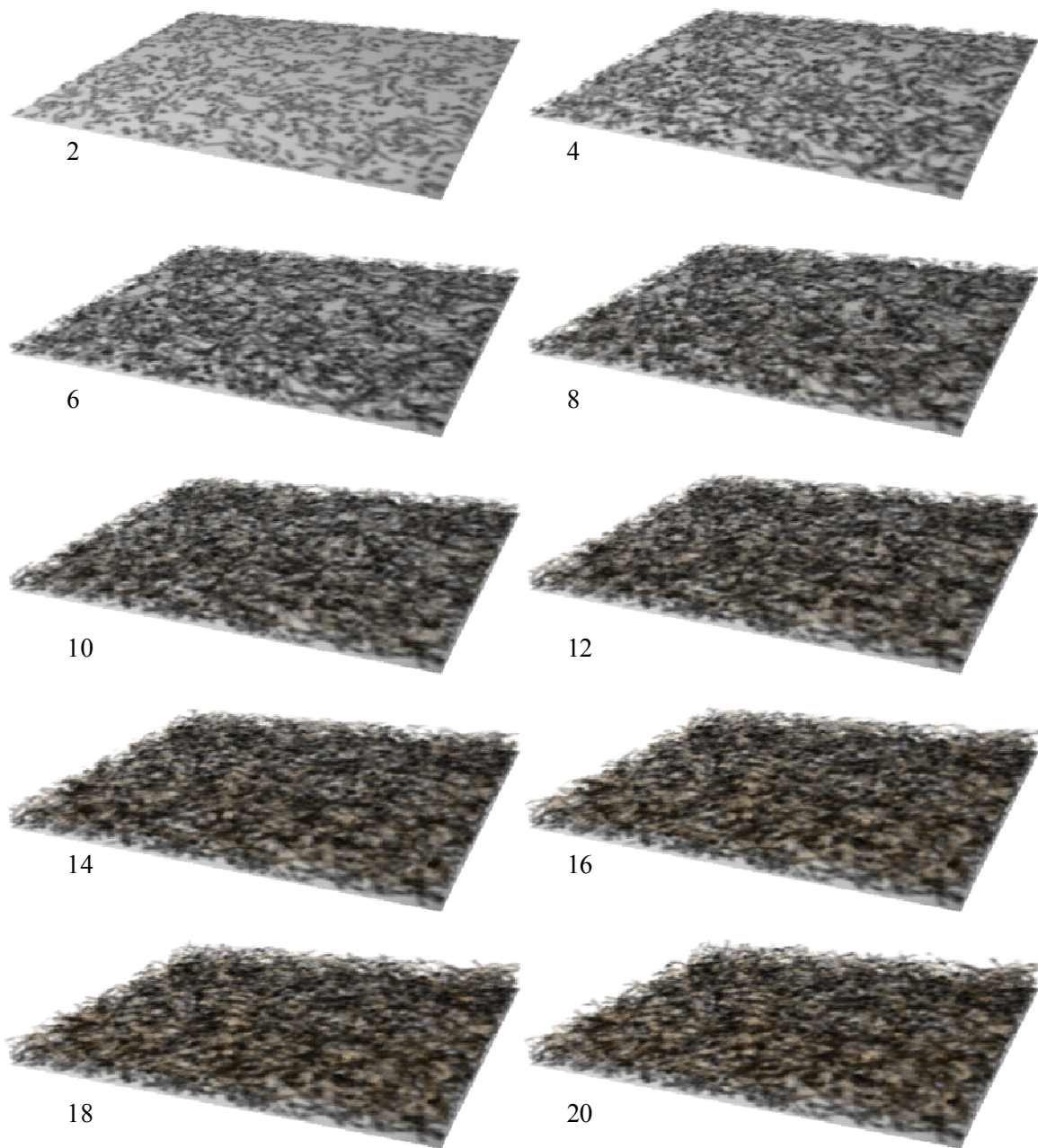


Figure 6.4: Hair represented by different numbers of layers.

The distance between the layers in each sample in Figure 6.4 is the same. Therefore the total height of the layers differs for each sample. In Figure 6.5 we show the results when 20 layers and 10 layers are used to represent hair, where the distance between the layers in the two samples differs, but the total height of the layers is the same. The hair represented by 10 layers appears thinner than the hair represented by 20 layers.

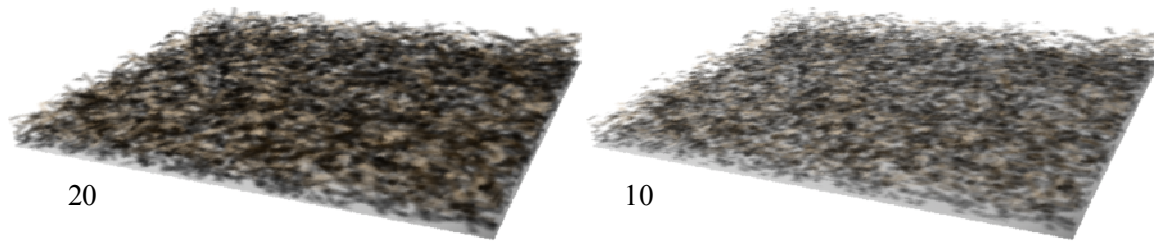


Figure 6.5: *Hair represented by different numbers of layers with the same height.*

Figure 6.6 illustrates that unrealistic results are obtained when the layers are placed too far apart. In the figure, the hair is represented by five layers that have the same height as the layers in Figure 6.5. However, the distance between the layers is too large, causing gaps to appear between the layers so that specks of hair are seen instead of the effect of strands of hair.

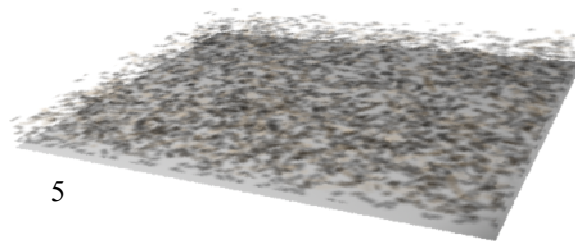


Figure 6.6: *Placing layers too far apart.*

6.2.2 Modelling Straight Hair

6.2.2.1 Modelling Different Hairstyles with Our Explicit Technique

Figure 6.7 shows two different hairstyles modelled with our explicit hair modelling technique. The top hairstyle is defined by 30 control hairs initially created with the HPCurve plug-in. Each control curve consists of four segments and has five control points. It takes approximately one hour to design the hairstyle by modifying the control curves (moving the control points). The control point closest to the scalp is not moved as it represents the root of the hair strand. The final hairstyle is created by randomly placing 80 copies of each control curve onto the head

model. The copies are automatically created and placed on the head model using the HPDuplicate plug-in. The bottom hairstyle is defined by 68 control hairs and takes approximately 2½ hours to design. The final hairstyle is produced by automatically creating 40 copies of each control hair. The figure shows that using our explicit hair modelling technique different hairstyles can be modelled.



Figure 6.7: *Two explicitly modelled hairstyles.*

6.2.2.2 Stray Hairs

When explicitly modelling hair, we overcome the problem of modelling the large number of hair strands required by automatically creating and randomly placing most of the hair strands on the head model using our HPDuplicate plug-in. Sometimes a few hair strands are created that have an unwanted shape or appear in the wrong place. For example, in the left image in Figure 6.8 a few hairs on top of the head have an unwanted shape and three of the hairs at the bottom are in the wrong place.

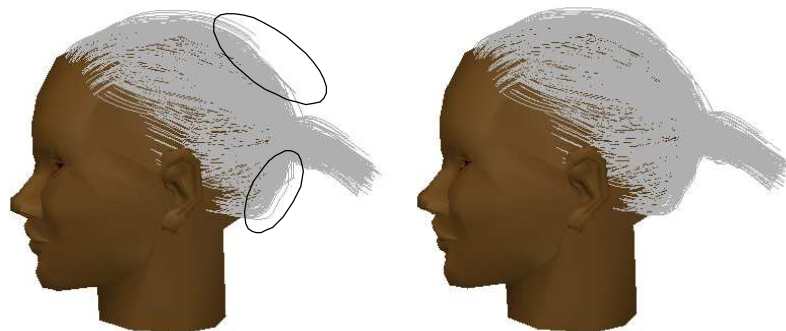


Figure 6.8: *Fixing and Removing Stray Hairs.*

Fortunately, the problem of unwanted stray hairs is easily solved because the hair strands are explicitly modelled. By selecting and moving control points of the stray hairs, the shape of the hairs can be modified, and unwanted hairs can be selected and deleted. In Figure 6.8 the shapes of the stray hairs on top of the head that appear to end in mid air are changed by selecting the control points at the end and moving them down towards the head. The three hair strands at the bottom that have been placed in an unwanted position are not needed, and may be deleted, producing the image on the right.

It is possible to design a hairstyle with control hairs and then procedurally render the rest of the hair strands without ever explicitly modelling them. The LightWave plug-in *SasLite*, a simplified version of Worley Laboratories' commercial plug-in *Sasquatch*, uses this method [Nakakihara 2001]. An advantage of the method is that the storage requirements are much lower than our explicit technique. However, the technique does not provide a way for stray hairs to be modified or deleted, and the users have to keep rendering the hair and adjusting values until they have the desired image.

6.2.2.3 The Number of Curves Used

Figure 6.9 shows rendering times when different numbers of curves are used to model a hairstyle. The hairstyle tested is designed with 30 control hairs, each consisting of five control points. Using the HPDuplicate plug-in, n copies (duplicates) of each control hair are made and randomly placed on the head model to create a full hairstyle. The curves are converted into polylines with five segments between control points. The hairstyle is then rendered in LightWave without anti-aliasing or shadowing (NN), without anti-aliasing but with shadow mapping (NS), with anti-aliasing but no shadowing (AN), and with anti-aliasing and shadow mapping (AS).

The scene consists of one spotlight and LightWave's default illumination model is used. We render the hair on its own without the head geometry and then with the head geometry. The results in Figure 6.9 show that the rendering time increases linearly in proportion to the number of curves used. Adding a shadow map does not have a large effect on the rendering time, but when anti-aliasing is applied rendering is much slower, and the rendering times increase significantly with respect to the number of curves used. Adding the head model (consisting of 728 polygons) does not have a large effect on the rendering time. This is due to hidden surface removal. Although more polygons are added to the scene, the strands of hair that appear behind

the head do not have to be rendered. The rendering times are satisfactory when real-time rendering is not required.

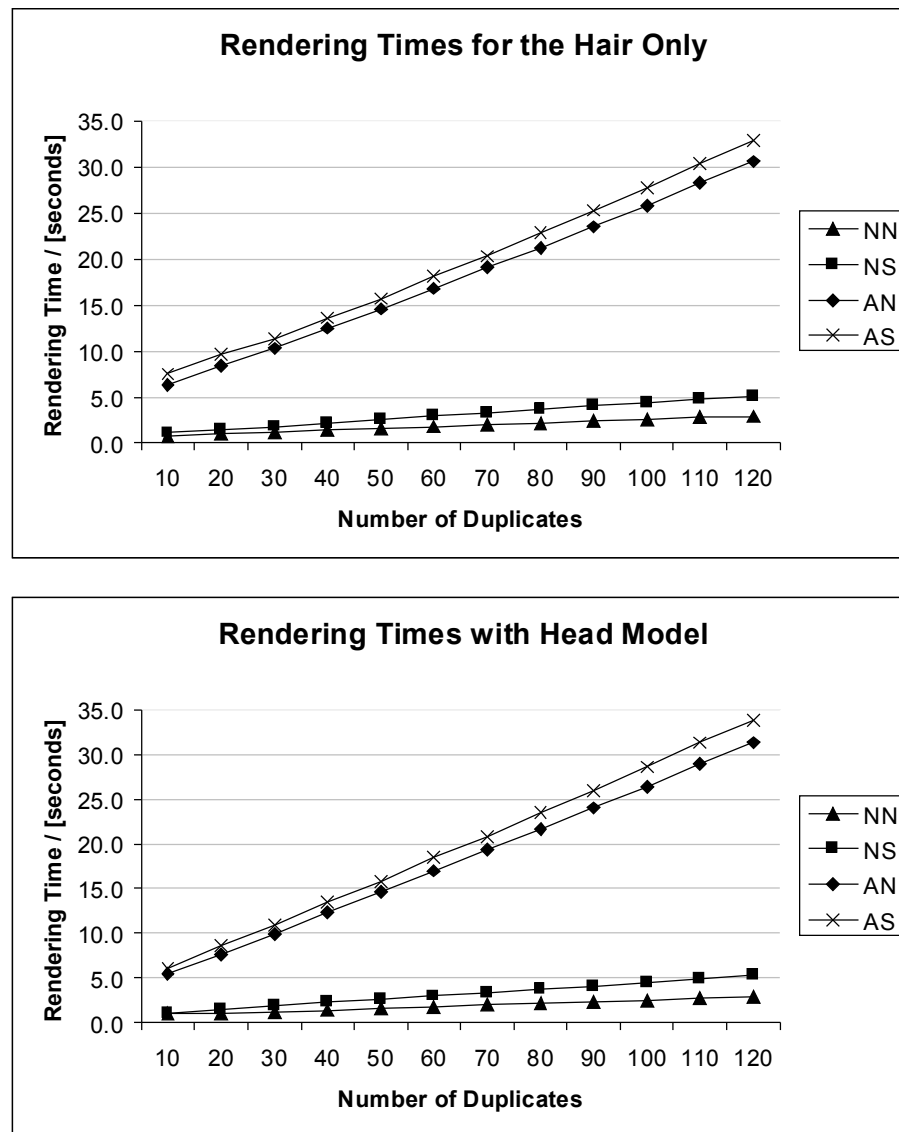


Figure 6.9: Rendering times when using different numbers of curves.

Images of the hair rendered can be seen in Figure 6.10. The labels under the images represent the number of duplicate curves created for each control curve, and, in brackets, the total number of curves making up the hairstyle. The hairstyles in the images become similar in appearance and quality as the number of curves is increased. Thus, although a large number of polygons are needed to represent hair explicitly, there is an upper bound on the number of polygons needed to produce a reasonable hairstyle. Generating more hair strands than needed will increase the time

taken to render the hair without improving the visual quality of the hair. The number of hairs needed is well below the number actually present on a human head [Patrick and Bangay 2003].



Figure 6.10: The quality of hair produced with different numbers of curves.

6.2.2.4 The Number of Segments making up Polylines

Because curves cannot be rendered in LightWave, we convert curves representing strands of hair into polylines before rendering. Polylines approximate a curve and our HPPolyline plug-in allows the user to specify the quality of the approximation by entering the number of segments between control points. We investigate the effect the number of segments making up a polyline has on the quality of the hair and the rendering time. Figure 6.11 shows rendering times when polylines representing strands of hair in a hairstyle consist of different numbers of segments. The hairstyle tested consists of 1,230 curves, and each curve has five control points. Using the HPDuplicate plug-in, the curves are converted into polylines consisting of n segments between the control points, and therefore a total of $n \times 4$ segments. The hairstyle is rendered in LightWave without anti-aliasing or shadowing (NN), without anti-aliasing but with shadow mapping (NS), with anti-aliasing but no shadowing (AN), and with anti-aliasing and shadow mapping (AS). The scene consists of one spotlight and LightWave's default illumination model is used. We render the hair on its own without the head geometry and then with the head geometry.

The results in Figure 6.11 show that the rendering time increases linearly in proportion to the number of segments used. Shadow maps do not have a large effect on the rendering time, but anti-aliasing increases the rendering times significantly. Adding the head model (consisting of 728 polygons) decreases the rendering time by less than a second. The results are similar to the results for different numbers of curves (Section 6.2.2.3), so we conclude that increasing the number of segments will increase the rendering time. Figure 6.12 shows the quality of hair produced when the polylines, representing strands of hair, consist of different numbers of segments.

The labels under the images represent the number of segments between control points, and, in brackets, the total number of segments making up each polyline. The quality of the hair strands increases and approximates a curve more accurately as the number of segments used increases. However, the difference in quality becomes negligible after a certain number of segments have been used (for the hairstyle shown in Figure 6.12 at approximately 20 segments per hair strand). Therefore, polylines can be used to represent strands of hair even though they are only approximations, and there is an upper bound on the number of segments needed to produce a reasonable strand of hair. Using more segments than needed increases the time taken to render

the hair without improving the visual quality of the hair. For the simple hairstyle shown in Figure 6.12, 16–20 segments per hair strand are appropriate.

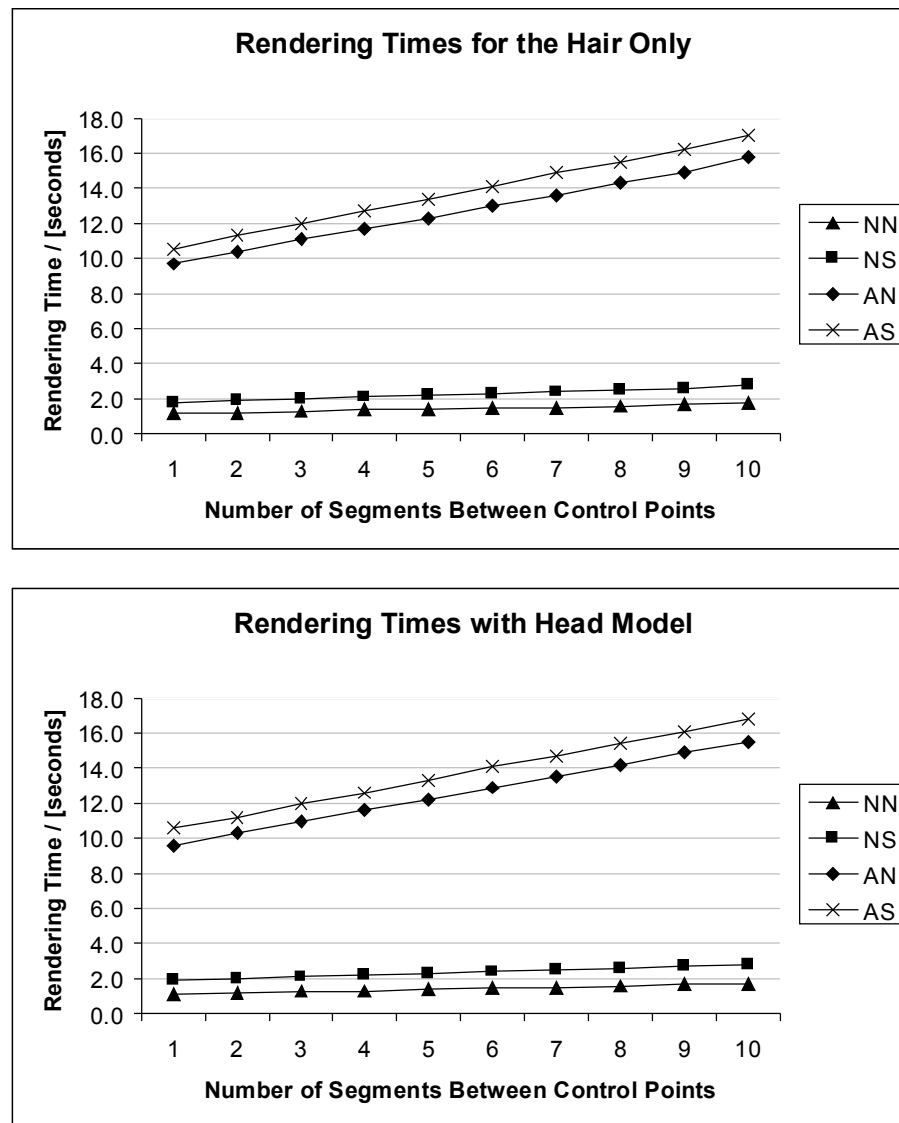


Figure 6.11: *Rendering times when using different numbers of segments.*

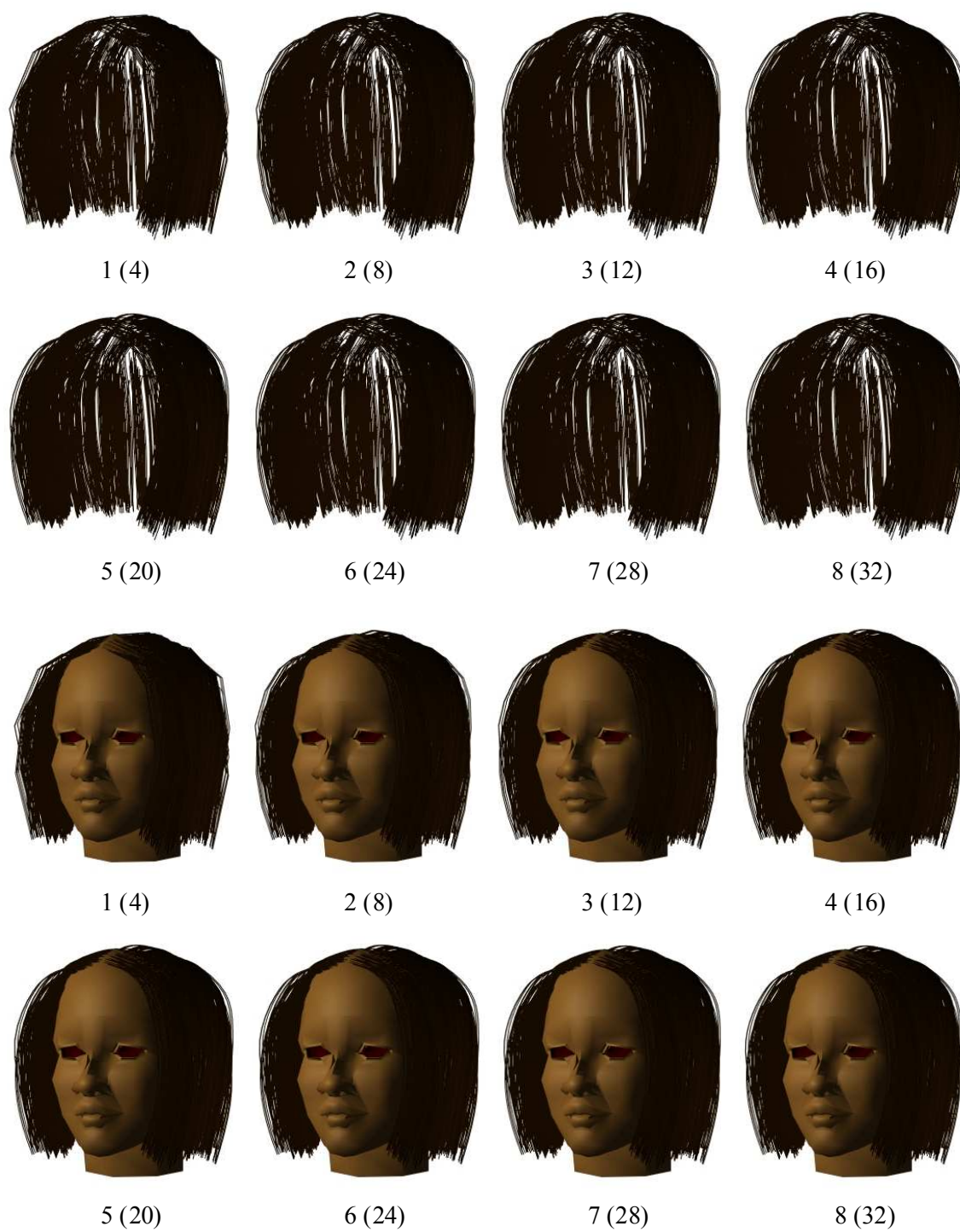


Figure 6.12: *The quality of hair produced with different numbers of segments.*

6.2.3 Modelling Hairstyles with Braids and Twists of Hair

6.2.3.1 Modelling Different Hairstyles with our Cluster Technique



Figure 6.13: *Hairstyles containing braids and twists.*

Figure 6.13 shows two different hairstyles modelled with our cluster hair modelling technique. The top hairstyle is a cornrow hairstyle initially defined by 14 curves that are manually placed across the scalp. Each curve consists of ten segments. It takes approximately 40 minutes to modify the curves so that they represent the desired hairstyle. The curves are converted into braids using the HPBraid and HPGeneralizedCylinder plug-ins. The HPBraid plug-in is used to convert each curve into three curves that define a braid, and the HPGeneralizedCylinder plug-in is used to convert the curves into generalized cylinders that represent clusters of hair. The bottom hairstyle consists of twists of hair. The hairstyle is initially defined by 30 control curves that are placed on the head using the HPCurve plug-in. It takes approximately one hour to modify the curves so that they represent the desired hairstyle. More curves are created from the control curves using the HPDuplicate plug-in so that the final design for the hairstyle consists of 100 curves. The 100 curves are converted into twists using the HPTwist and HPGeneralizedCylinder plug-ins. The HPTwist plug-in is used to convert each curve into three curves that define a twist, and the HPGeneralizedCylinder plug-in is used to convert the curves into generalized cylinders that represent clusters of hair. The figure shows that using our cluster hair modelling technique different hairstyles can be modelled.

6.2.3.2 The Number of Clusters in a Braid or Twist

Figure 6.14 shows braids (left) and twists (right) created with different numbers of clusters. The figure shows that at least three clusters should be used in order to create a realistic braid or twist. The braids and twists consisting of more than five clusters do not appear significantly different.

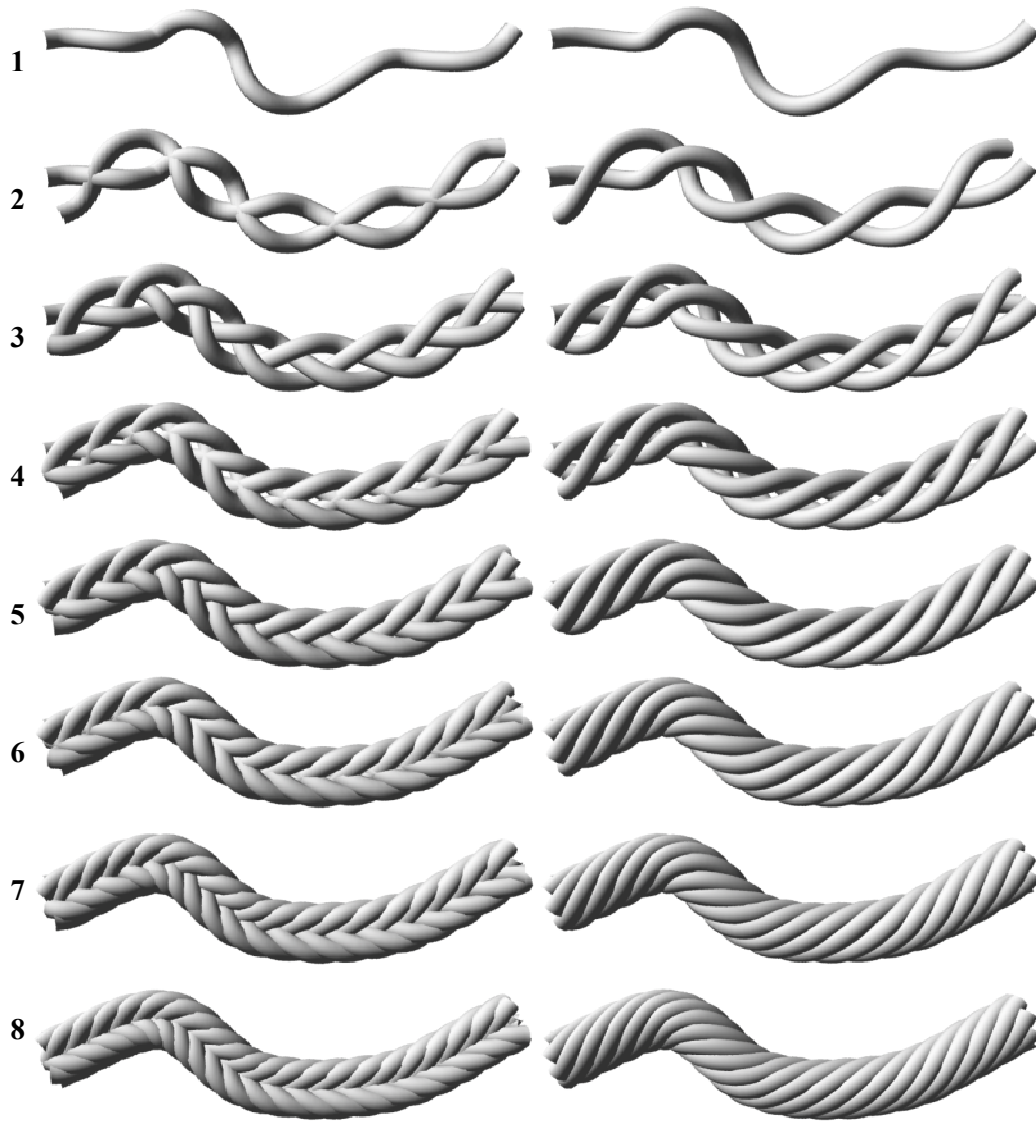


Figure 6.14: *Braids and twists with different numbers of clusters.*

Figure 6.15 shows the times taken to render the braid and twist samples shown in Figure 6.14. The rendering times increase as the numbers of clusters, representing the braids and twists, increase. Each cluster is made up of 1,536 polygons.

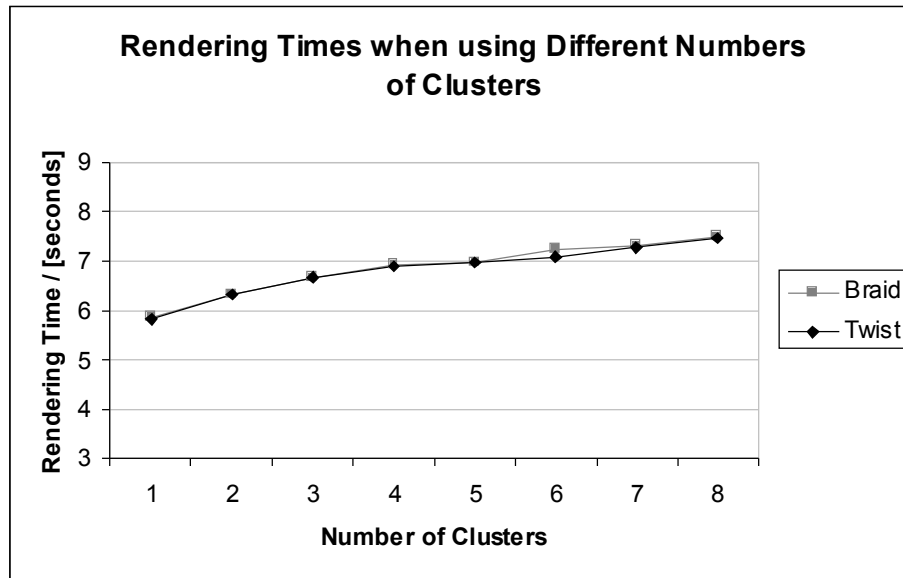


Figure 6.15: *Rendering times when using different numbers of clusters.*

6.2.3.3 Overlapping Clusters

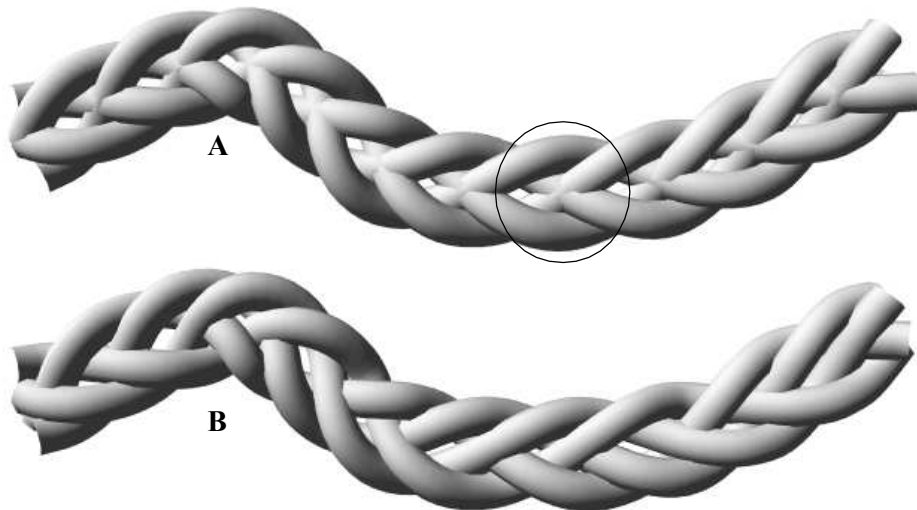


Figure 6.16: *Overlapping clusters.*

Depending on the number of clusters used and the angle between the clusters, some of the clusters representing a braid may overlap. Diagram A in Figure 6.16 shows an example of a braid in which clusters overlap. The braid consists of four clusters, and there is a difference of 90° between the starting angles of the clusters. The problem of overlapping clusters can be solved by changing the value of the angle between the clusters slightly. This can be done by using a slightly different value for “Angle between curves” when creating the curves that define

the clusters in the braid using the HPBraid plug-in. In Diagram B, most of the overlapping has been removed by setting the difference in starting angles between the clusters to 80° rather than 90° . The disadvantage of setting the difference in starting angles between the clusters to 80° rather than 90° for four clusters is that the clusters are not evenly spaced (although the angle between the clusters is set to 80° , the angle between the first and last cluster will be 120° , because $4 \times 80^\circ = 320^\circ$ and not 360°).

6.2.3.4 The Number of Segments Making up a Generalized Cylinder

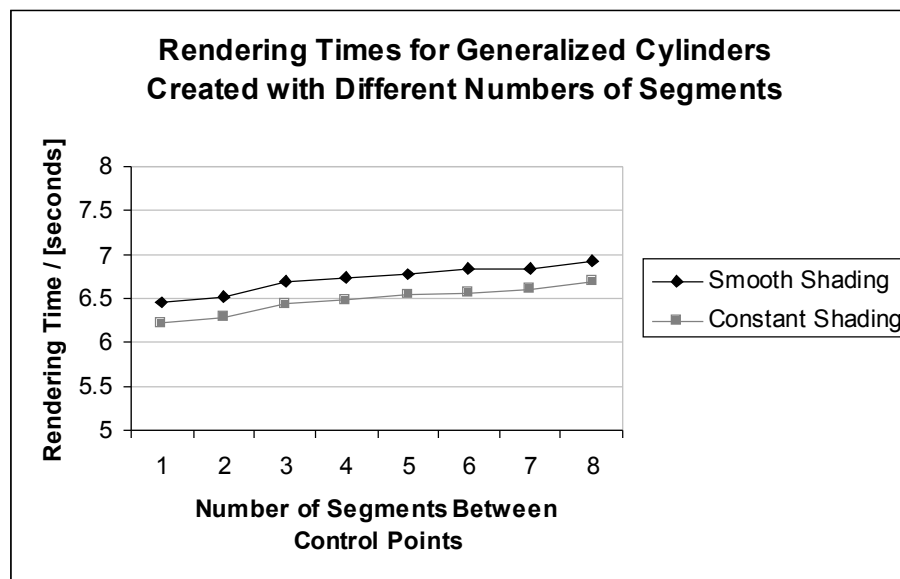


Figure 6.17: *Rendering times when using different numbers of segments.*

Figure 6.17 shows the times taken to render braid samples consisting of three clusters. The clusters in each sample are represented by generalized cylinders created with different numbers of segments. The original curves defining the clusters have 16 segments, and each generalized cylinder has eight sides. Therefore a generalized cylinder created with n segments between each set of control points on the original curve will have a total of $n \times 16$ segments and $n \times 16 \times 8 \times 2$ polygons (each segment along the eight sides is represented by two triangles). Each braid sample consists of $n \times 16 \times 8 \times 2 \times 3$ (or $n \times 768$) polygons (the braid is made up of three clusters represented by three generalized cylinders). Figure 6.17 shows that the rendering time increases as the number of segments making up each generalized cylinder increases. The figure also shows that the rendering time is slightly longer (by approximately 3%) when smooth shading is used instead of constant shading. Figure 6.18 shows the braid samples rendered with constant shading (left) and with smooth shading (right). The quality of the braid samples produced increases as the

number of segments used increase. However, when smooth shading is applied the difference in quality when different numbers of segments are used is negligible after a certain number of segments (at about four segments between control points for the samples in Figure 6.18). The quality of the braid samples rendered with smooth shading is also substantially better than the quality of the braid samples rendered with constant shading. Therefore, the least number of segments that produce an appropriate quality should be used to create the generalized cylinders, and although smooth shading has slightly longer rendering times than constant shading, it produces substantially better results, especially when lower numbers of segments are used, and should therefore be considered.

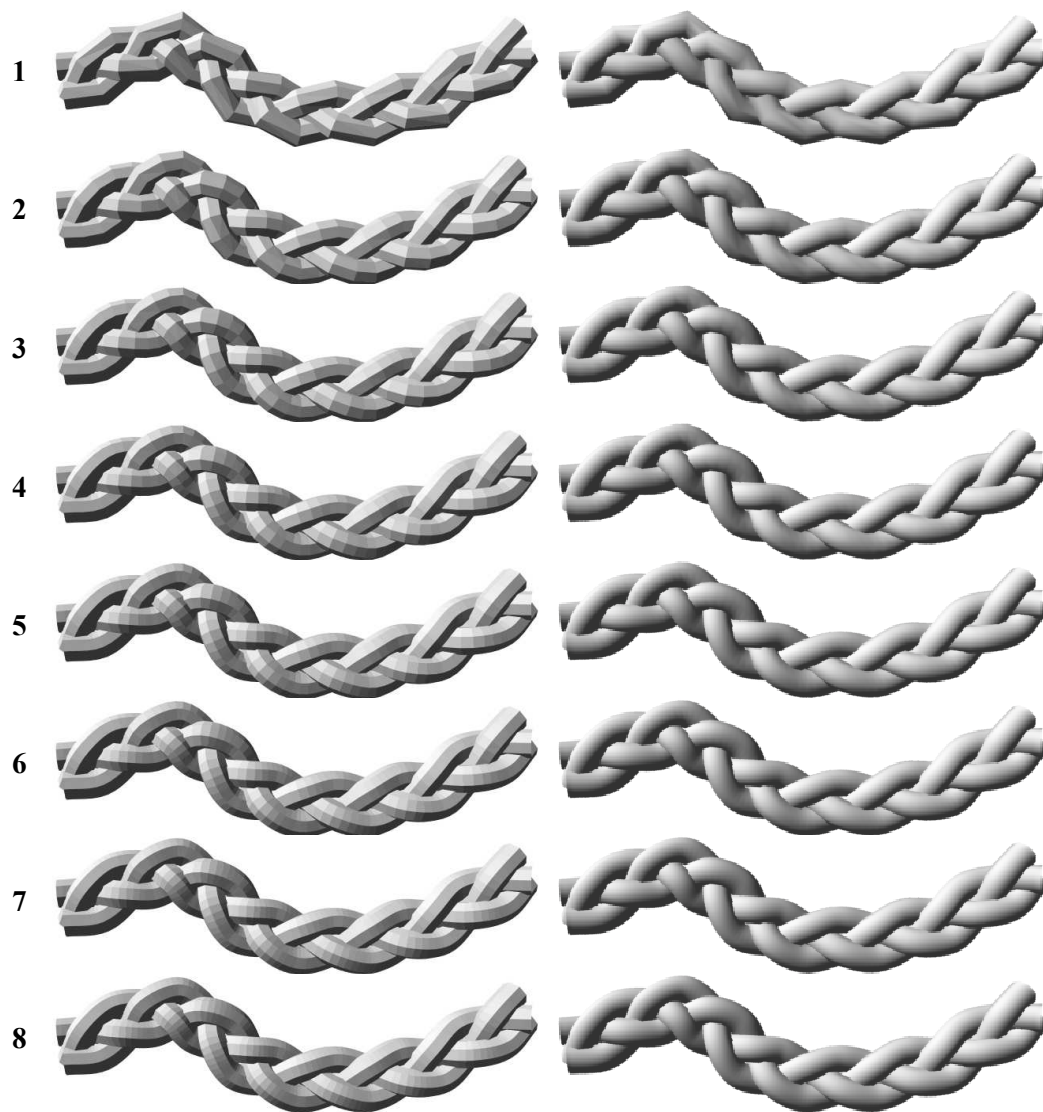


Figure 6.18: Braids represented by generalized cylinders created with different numbers of segments.

For the braid sample of hair shown in Figure 6.18 we would choose to create the generalized cylinders with four segments between the control points of the defining curves, and would render the generalized cylinders with smooth shading.

6.2.3.5 The Number of Sides Making up a Generalized Cylinder

Figure 6.19 shows the times taken to render twist samples consisting of three clusters. The clusters in each sample are represented by generalized cylinders that have different numbers of sides. The original curves defining the clusters have 16 segments, and each generalized cylinder is created with four segments between each pair of control points on the original curve. Therefore, a generalized cylinder with n sides will have a total of $n \times 16 \times 4 \times 2$ (or $n \times 192$) polygons (each segment is represented by two triangles), and each twist sample (represented by three generalized cylinders) will have a total of $n \times 192 \times 3$ (or $n \times 576$) polygons. Figure 6.19 shows that the rendering time increases as the number of sides making up each generalized cylinder increases. Figure 6.20 shows the twist samples rendered with constant shading (left) and with smooth shading (right). The quality of the twist samples produced increases as the number of sides making up the twists increase. However, when smooth shading is applied the difference in quality when different numbers of sides are used is negligible from a certain number of sides (at five sides for the samples in Figure 6.20). Therefore the least number of sides that produce an appropriate quality should be used to create the generalized cylinders. For the sample of a twist of hair shown in Figure 6.20 we would chose to give the generalized cylinders five sides.

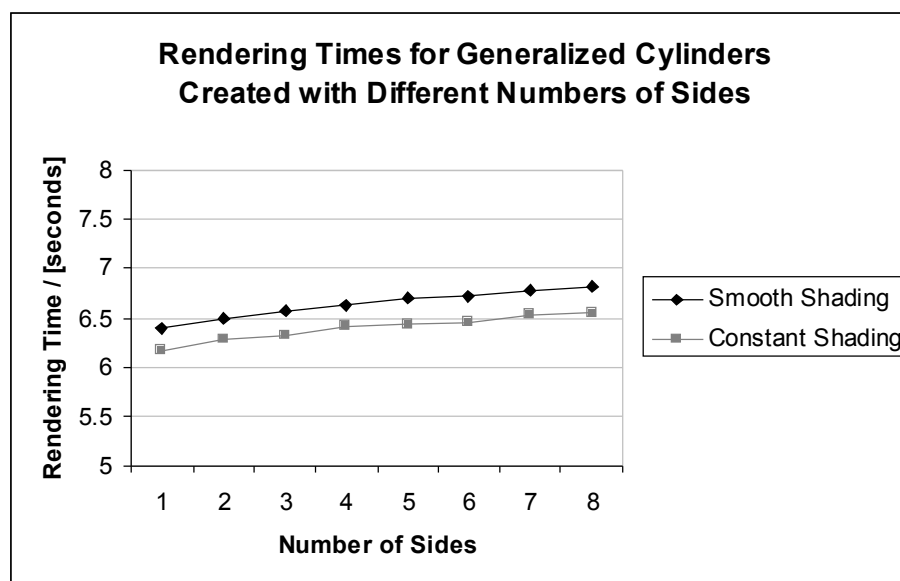


Figure 6.19: Rendering times when using different numbers of sides.

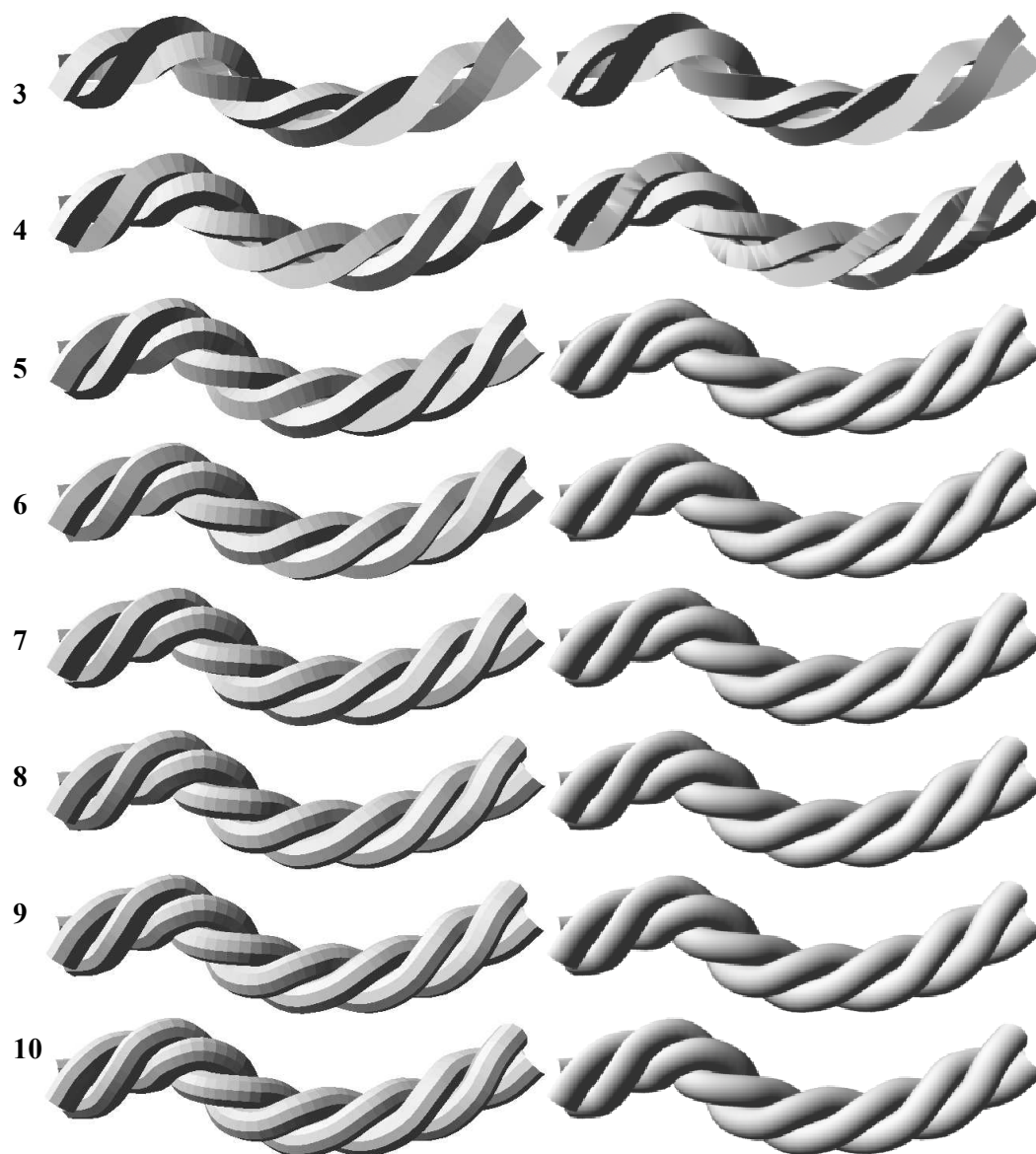


Figure 6.20: *Twists represented by generalized cylinders that have different numbers of sides.*

6.3 Rendering Results

6.3.1 Illumination Model

6.3.1.1 The Effect of the Diffuse Component

Our illumination model includes a diffuse component, which we implemented in a shader (written with the RenderMan Shading Language) using the equation:

$$C_i += C_l * C_s * O_s * K_d * \text{pow}((1 - \text{pow}((T \cdot L_n), 2)), K_b/2) * F_{dir}; \text{ (see Section 5.2.3)}$$

The variables C_i , C_l , C_s , and O_s represent the final colour, light's colour, hair surface's colour and hair surface's opacity, respectively [Pixar 2000]. African hair usually has a very dark colour and is often black. If the colour of the hair is set to black, C_s will be black (i.e. 0) and the final colour C_i will be black (because the result of any colour multiplied with black (the value 0) is black). Therefore the diffuse component has no effect and is not needed in the illumination model when the hair surface colour is set to black. Figure 6.21 shows the results of black hair (represented by polylines) rendered with (A) and without (B) the diffuse component and brown hair (represented by polylines) rendered with (C) and without (D) the diffuse component. The figure shows that using the illumination model without the diffuse component will render black hair no matter what the surface colour of the hair is.

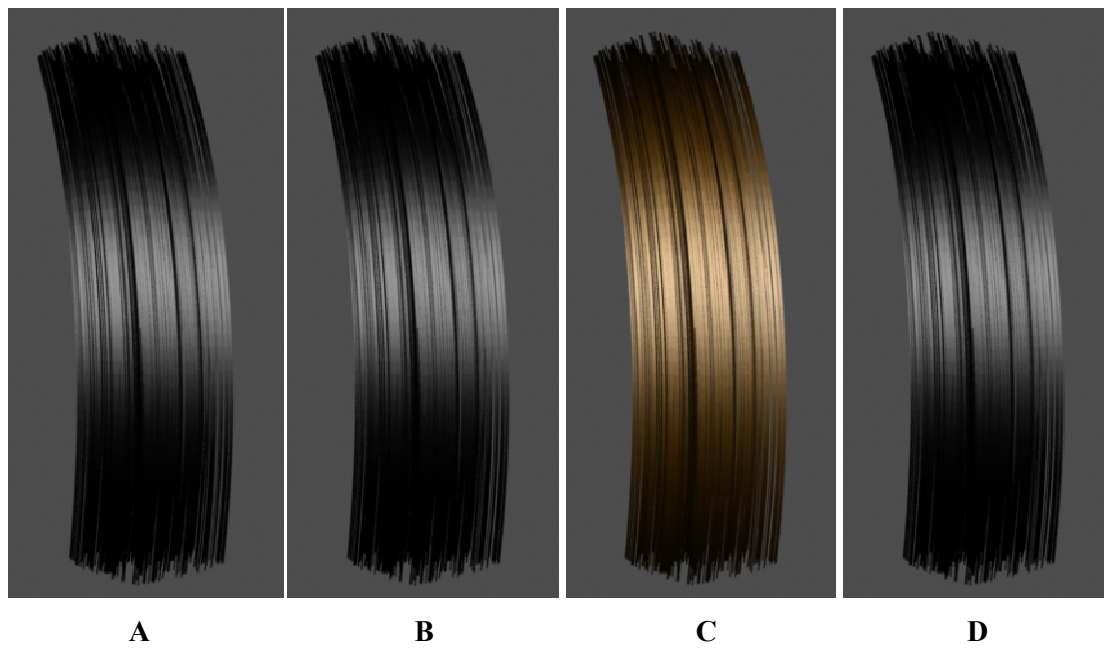


Figure 6.21: Black and brown hair with and without the diffuse component.

6.3.1.2 The Excess Brightness Exponent

The diffuse term in our hair illumination model contains an excess brightness exponent presented by Banks [1994] to compensate for an average increase in light intensity on objects such as hair. Banks recommends a value of 4.7635. When the value is 1 the excess brightness term has no effect on the results of illumination model. Figure 6.22 shows hair rendered with an excess brightness term value of 1 (A), 4.7635 (B), 10 (C), and 20 (D). We use brown hair to show the effect of the term clearly, because as we concluded in Section 6.3.1.1 above, the diffuse component has no effect when the surface colour of the hair is black. By including the excess brightness exponent in our illumination model, the user has the option of adjusting the light intensity if they choose to give their African hair a colour other than black.

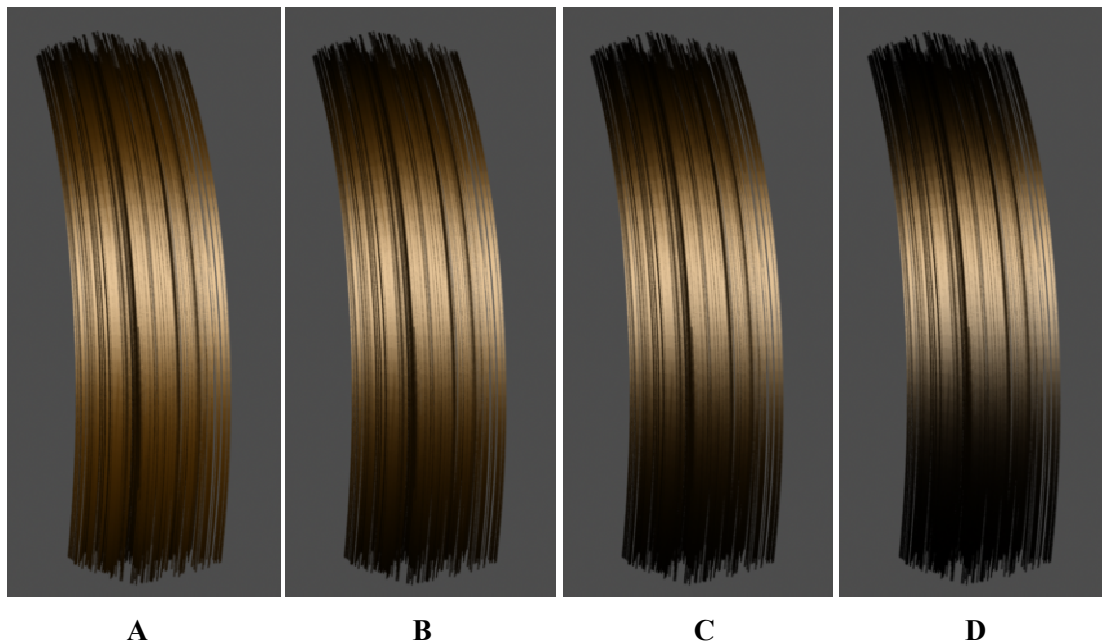


Figure 6.22: Using different values for the excess brightness term.

6.3.1.3 The Effect of the Specular Component

The specular component in our illumination model is implemented in a shader (written with the RenderMan shading language [Pixar 2000]) using the equation:

$$C_i += C_l * O_s * K_s * \text{pow}((1 - \text{pow}((T \cdot H), 2)), 1/(K_n * 2)) * F_{dir}; \text{ (see Section 5.2.3).}$$

As in the equation for the diffuse component (see Section 6.3.1.1), the variables C_i , C_l , and O_s represent the final colour, light's colour and hair surface's opacity respectively [Pixar 2000]. However, unlike the equation for the diffuse component, we do not include C_s (the hair surface's

colour) in the equation for the specular component. The colour of the hair therefore has no effect on the colour of the specular highlights rendered in the hair.

Using our equation, the colour of the specular highlights is influenced by the colours of the lights illuminating the hair. Figure 6.23 shows an example of black and brown hair illuminated with three white lights (A), an off-white (cream) Key light (B), a slightly blue Key light (C), and a cream Key light with a blue Fill light (D). Since the Fill light is set with a much lower intensity than the Key light, and is set not to affect the specular component (see Section 5.1.1), diagram D is very similar to diagram B.

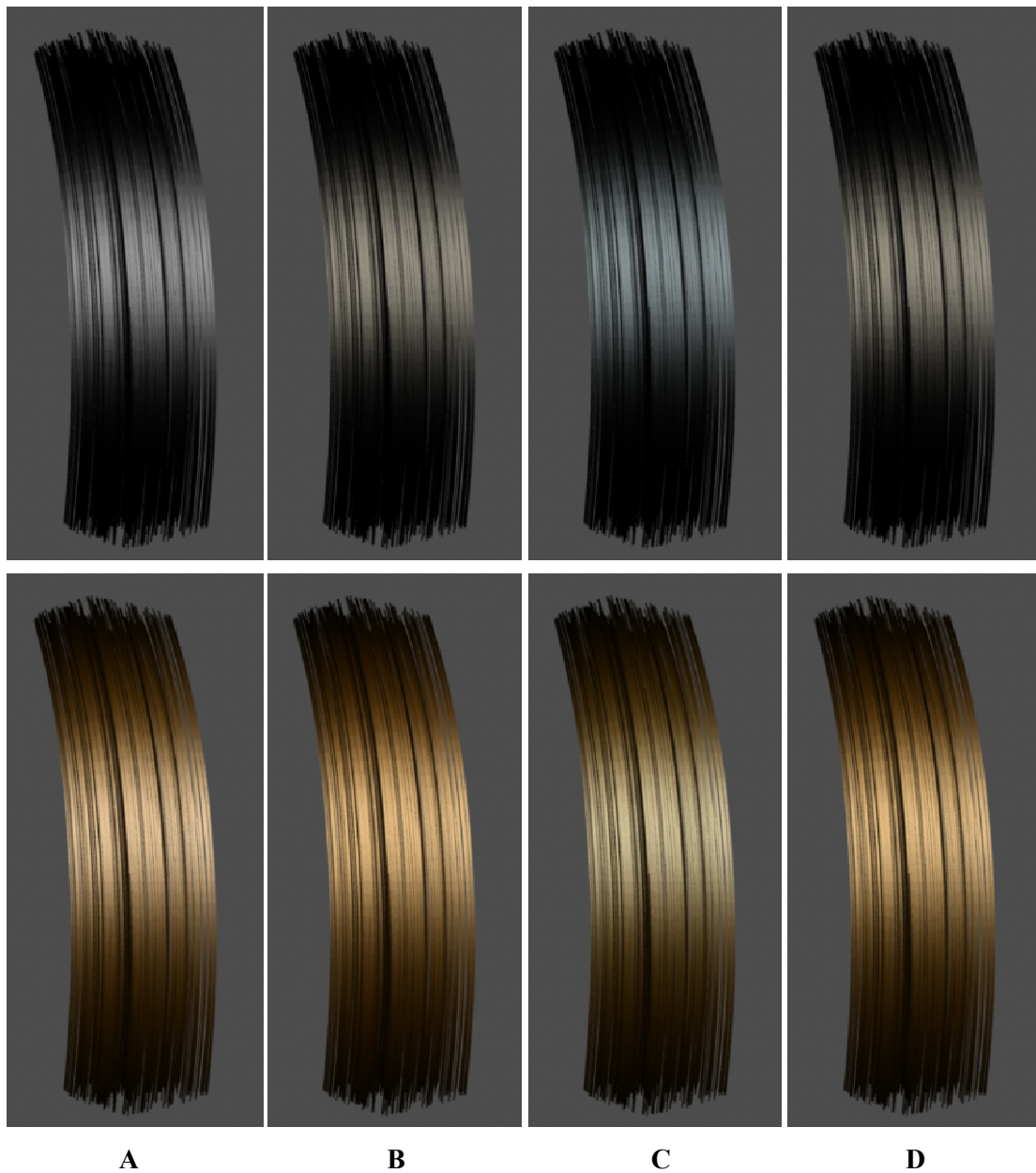


Figure 6.23: *Using different coloured lights.*

If C_s is added to the equation for the specular component the surface colour will influence the specular highlights on the hair. The equation is:

$$C_i += C_l * C_s * O_s * K_s * \text{pow}((1 - \text{pow}((T.H), 2)), 1/(K_n * 2)) * F_{dir};$$

The result produced is fine for coloured hair, but the specular highlights “disappear” when the equation is used in our shading model to illuminate black hair. Figure 6.24 shows black hair and brown hair illuminated with our shader that does not contain C_s in the equation for the specular component (A and C) and with a shader that does contain C_s in the equation for the specular component (B and D).

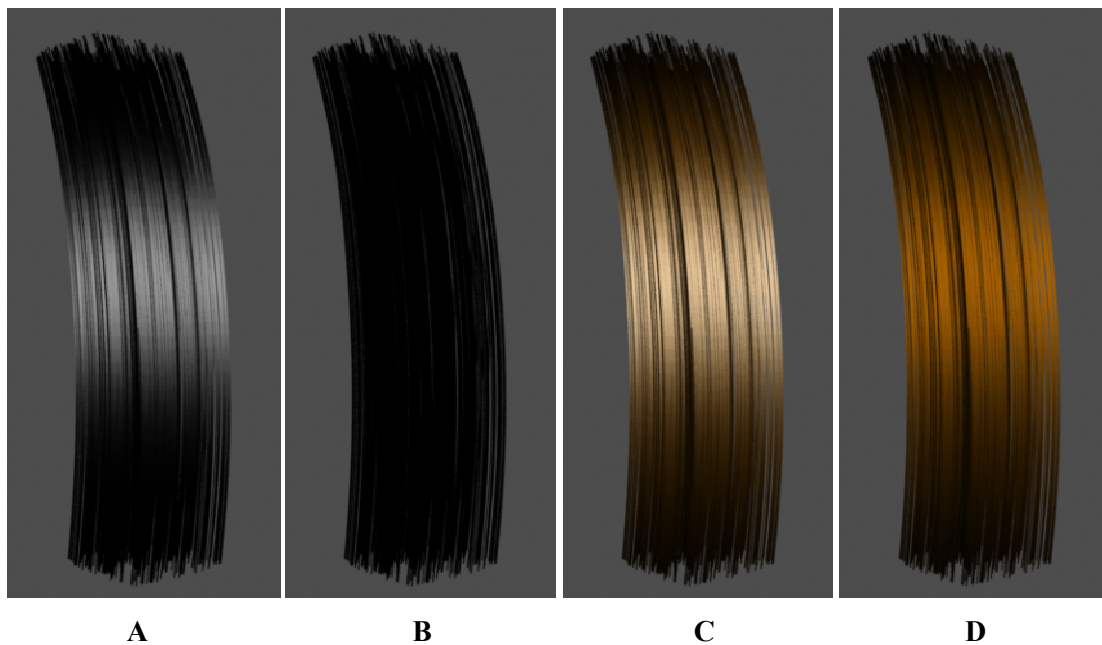


Figure 6.24: Including the surface colour in the equation for the specular component.

A variable representing the colour of the specular highlights, *specularcolour*, could have been added to the specular equation in our shader [Pixar 2000], which would allow the user to determine the colour of the specular highlights. The equation for the specular component would then become:

$$C_i += C_l * O_s * \text{specularcolour} * K_s * \text{pow}((1 - \text{pow}((T.H), 2)), 1/(K_n * 2)) * F_{dir};$$

6.3.1.4 Light Reflection and Transmission

We added a directional attenuation factor to our illumination model (a technique developed by Goldman [1997]) so that both light reflection and light transmission are considered. Figure 6.25 shows the results when different values of light reflection (F_r) and transmission (F_t) are used in

our illumination model to render brown hair with only the Back light in the scene. We use brown hair for the example because the Back light only contributes to the diffuse component (see Section 5.1.1) and the diffuse component has no effect when the surface colour of the hair is black (which was concluded in Section 6.3.1.1). The values of F_r and F_t used to render each diagram are:

Diagram A: $F_r = 1.0$, $F_t = 1.0$

Diagram B: $F_r = 1.0$, $F_t = 0.0$

Diagram C: $F_r = 1.0$, $F_t = 0.5$

Diagram D: $F_r = 0.5$, $F_t = 0.5$

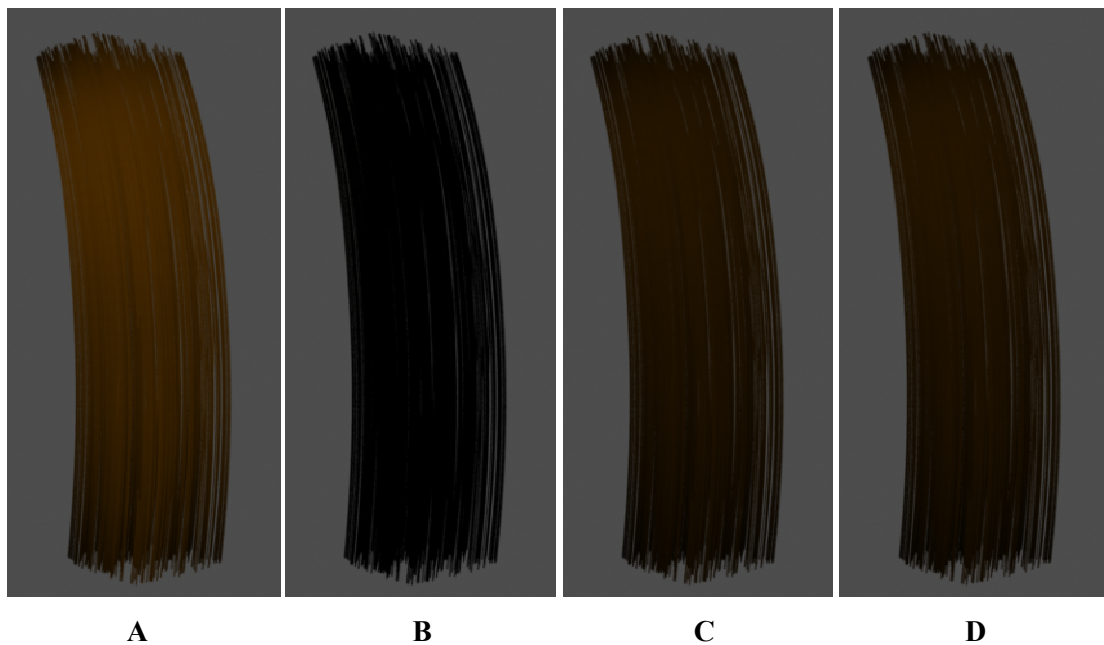


Figure 6.25: *Using different light reflection and light transmission values.*

When both F_r and F_t equal 1.0, the directional attenuation factor has no effect in the illumination model, and the hair is illuminated even though the Back light is on the opposite side of the hair to the viewer (A). This effect is appropriate for light coloured hair that is slightly transparent. However, African hair is usually opaque [Deedrick 2000]. African hair can be illuminated correctly, so that light is not transmitted through the hair by setting F_t equal to 0.0 (B). The amount of light transmitted can be adjusted (to correspond to the opacity of the hair being rendered) by setting F_t to different values (C and D). Since the Back light and the viewer are on opposite sides of the hair, the value of F_r has no effect on the hair rendered in Figure 6.25.

6.3.1.5 Black Hair Conclusion

When rendering black, opaque African hair, our illumination model does not need a diffuse component and the directional attenuation factor could be implemented as a constant value (with F_r equal to 1.0 and F_t equal to 0.0). However, African hair does not always have a totally black surface colour, and a user may want a certain amount of light to be transmitted through the hair. Our illumination model with a diffuse component and directional attenuation factor allows African hair, with different colours and opacities, to be illuminated, and can also be used to illuminate other types of hair.

6.3.1.6 Including our Illumination Model in Textured Layers

Figure 6.26 shows the result when hair represented by textured layers is rendered with (top) and without (bottom) our illumination model. The light intensity varies in the hair rendered with our illumination model, while the hair rendered without our illumination model is a constant intensity.

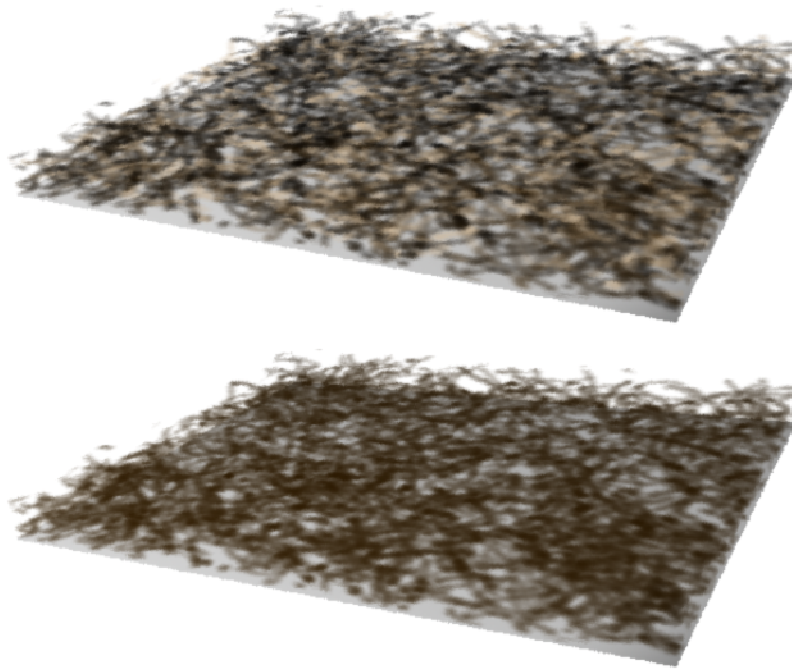


Figure 6.26: *Illuminating hair represented by textured layers.*

6.3.2 Shadowing

6.3.2.1 Using Different Shadow Map Resolutions

We use traditional shadow maps to add shadows to our explicitly modelled hair (see Section 5.3.1). Figure 6.27 shows the rendering times of a sample of hair rendered using shadow maps with different resolutions. The shadow maps are created from the point of view of the Key light (see Section 5.1.1). The results in Figure 6.27 show that using a shadow map with a higher resolution results in a higher rendering time. Figure 6.28 shows the sample of hair rendered with the shadow maps of different resolutions. If the resolution of the shadow map is too low, the quality of the image produced is poor. The results in Figure 6.28 also show that the difference in quality for different shadow map resolutions is negligible after a certain resolution (in this example at a resolution of approximately 2048 x 2048 pixels). Therefore the lowest resolution that produces an appropriate quality should be chosen. For the sample of hair shown in Figure 6.28 we would chose a shadow map resolution of 2048 x 2048 pixels.

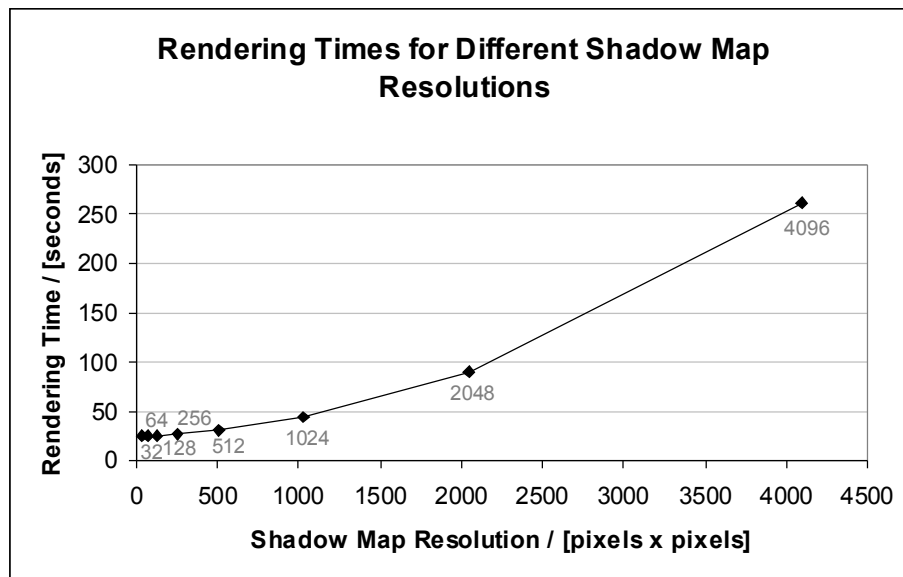


Figure 6.27: *Rendering times when using different shadow map resolutions.*

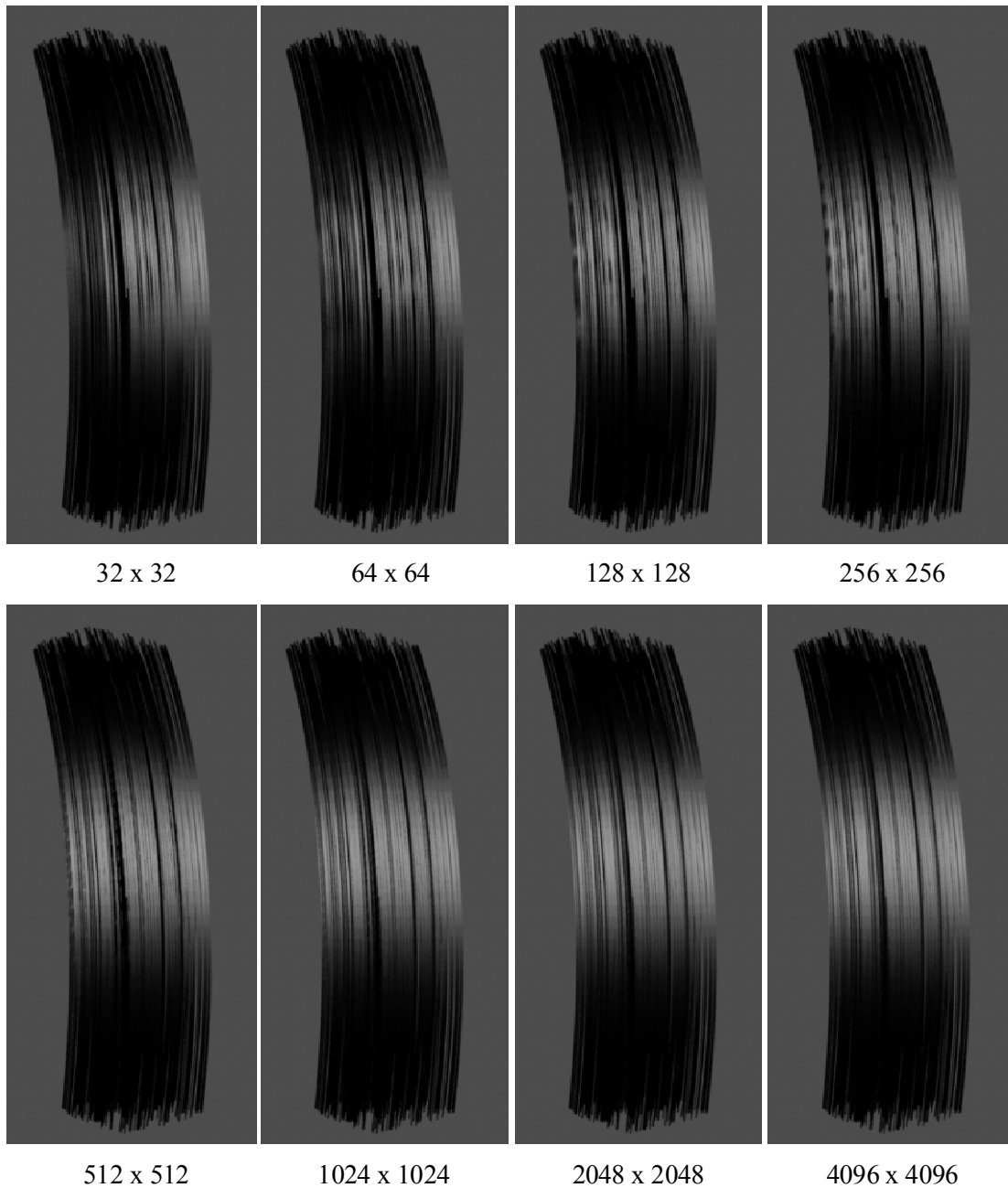


Figure 6.28: *Using different shadow map resolutions.*

6.3.2.2 Self-shadowing in Layers

To add the effect of self-shadowing to hair represented by textured layers, we make the surface colour of the lower layers closer to the scalp darker than the colour of higher layers further away from the scalp (see Section 5.3.2). Since only the surface colour of the layers is changed, adding self-shadowing to hair represented by textured layers has no effect on the rendering time.

6.3.3 Texturing

6.3.3.1 Using Different Texture Maps and Surface Colours to Render Different Types of Hair with Our Implicit Technique

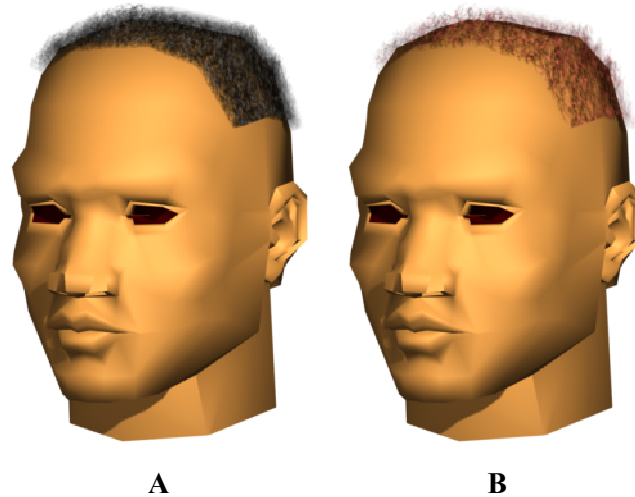


Figure 6.29: *Creating different hair types.*

Figure 6.29 shows that different types of hair can be created for the same hairstyle by using different texture maps and surface properties. In diagram A the texture maps are created from 840 splines and the surface colour is set to black. In diagram B the texture maps are created from 240 splines and the surface colour is set to red. In both diagrams the texture maps have been tiled 16 times over each layer (because smaller texture maps are less expensive to create and store).

6.3.3.2 Using Texture Maps that Represent Different Hair Densities

By using texture maps in our implicit hair modelling technique, the problem of modelling a large number of hair strands is overcome. The texture maps are automatically generated, so a user does not have to model strands of hair explicitly. Unfortunately, as a result the user does not have control over the shape of each individual strand of hair. However, a user is able to specify the density and curliness of the hair strands represented by the textures. Because the texture maps are pre-processed, the density of the hair (number of hair strands represented by the textures) does not affect the rendering time. Figure 6.30 shows the rendering times for samples of hair with different densities. Each hair sample is modelled with 20 layers. The texture maps applied to the layers in each sample are created using different numbers of splines to create different hair densities. The graph shows that the samples took roughly the same time to render (each sample

took approximately 47 seconds). The different samples of hair with different densities are shown in Figure 6.31.

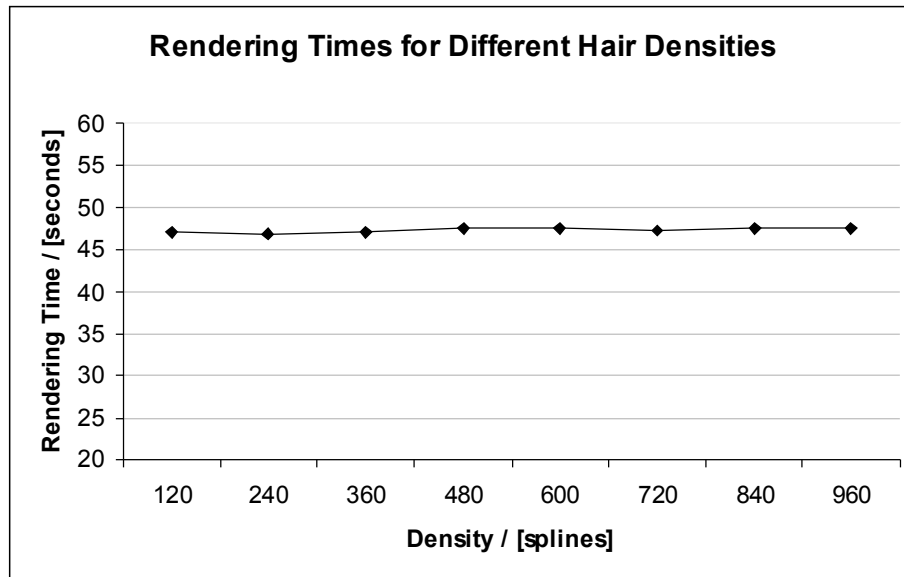


Figure 6.30: *Rendering times for different hair densities represented with textures.*

Since the textures have the same resolution, they have the same storage requirements (unless compressed). Therefore the density of the hair represented by textures does not affect the storage requirements of the hair. Smaller texture maps that represent hair can be tiled over a surface if they have a toroidal topology. The smaller texture maps have smaller storage requirements and require less time to process.

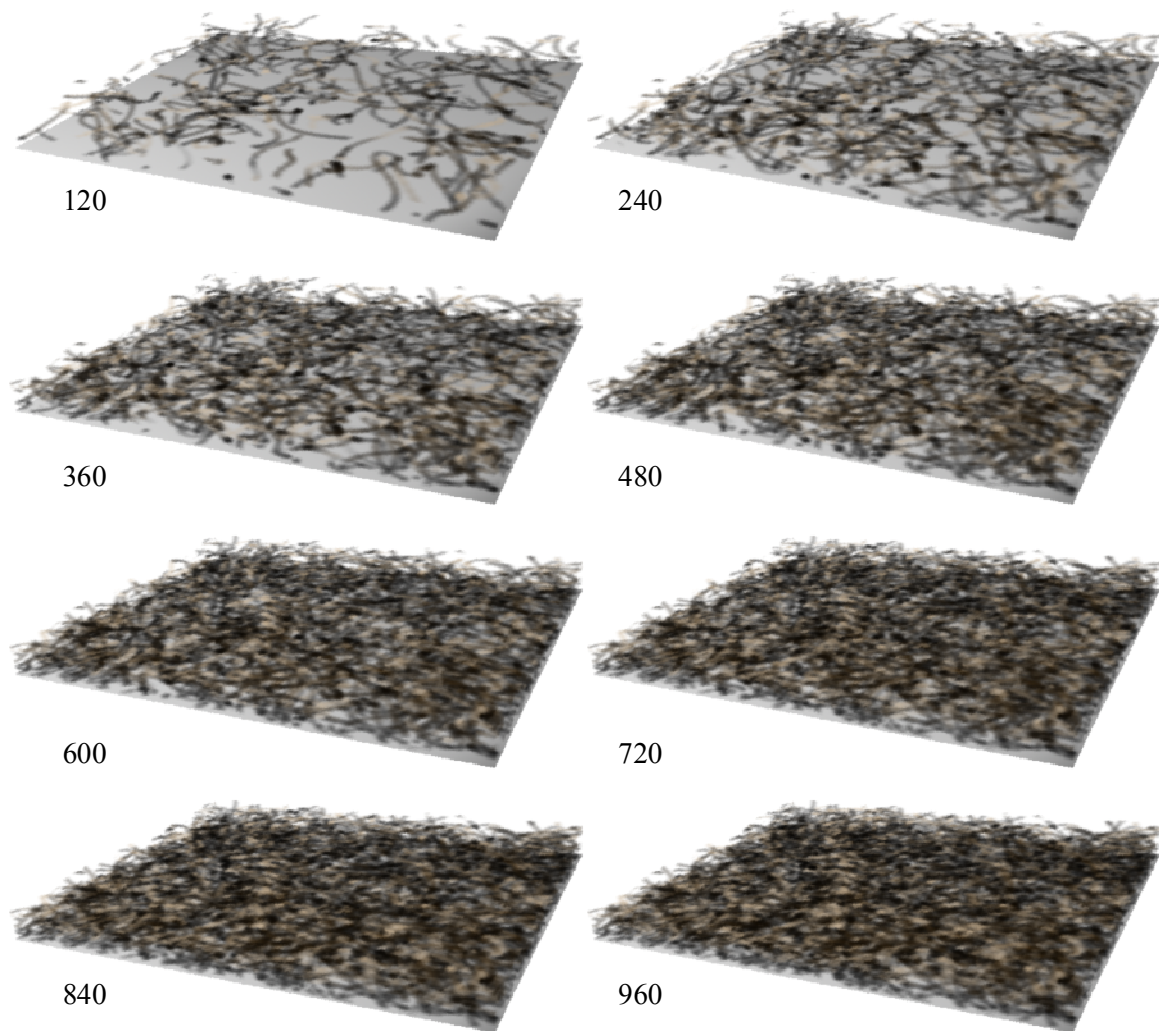


Figure 6.31: *Different hair densities.*

6.3.3.3 Reusing Texture Maps



Figure 6.32: *Reusing a texture to create the effect of hair.*

Another advantage of using texturing is that the textures are pre-processed and can be reused. Figure 6.32 shows two hairstyles created on different head models using the same texture maps and a braid and twist that have been textured with the same texture map to create the effect of many strands of hair.

6.3.3.4 Texturing Braids to Create the Effect of Many Strands of Hair

Texture maps are applied to generalized cylinders with our cluster hair modelling technique in order to create the effect of many strands of hair. A texture map (or opacity map) is used to define hair and no hair areas across the generalized cylinders. Areas where no hair exists are rendered as transparent areas. However, we find that the technique does not produce realistic results when used to create a braid or twist, because clusters of hair behind other clusters can be seen through the transparent areas (see diagram A in Figure 6.33).

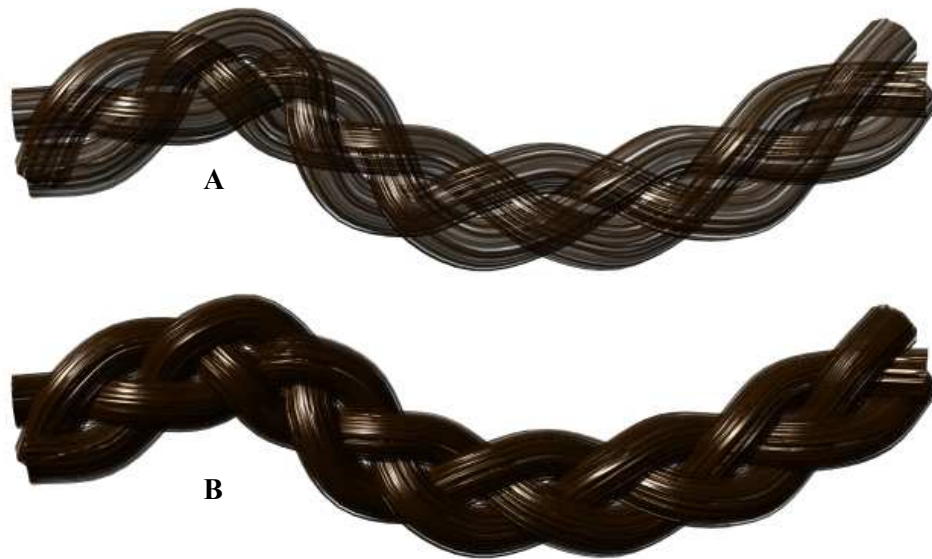


Figure 6.33: *Texturing generalized cylinders.*

To solve the problem we use three concentric generalized cylinders for each cluster instead of a single generalized cylinder. The three generalized cylinders are created using the same defining curve, but have slightly different radii (see Figure 6.34). Each generalized cylinder is textured and given a surface colour. To create the effect of self-shadowing, the generalized cylinders closer to the centre can be given darker surface colours than the outer generalized cylinders. Diagram B in Figure 6.33 shows the result produced when three generalized cylinders are used. The generalized cylinders have a radius of 5cm, 6cm, and 7cm respectively. A disadvantage of

using three generalized cylinders for each cluster is that number of polygons representing each cluster is tripled, and increasing the number of polygons that represent a cluster causes an increase in rendering time (see the previous results in Sections 6.2.3.4 and 6.2.3.5). For the rest of the results in this section we use three generalized cylinders to create each cluster.

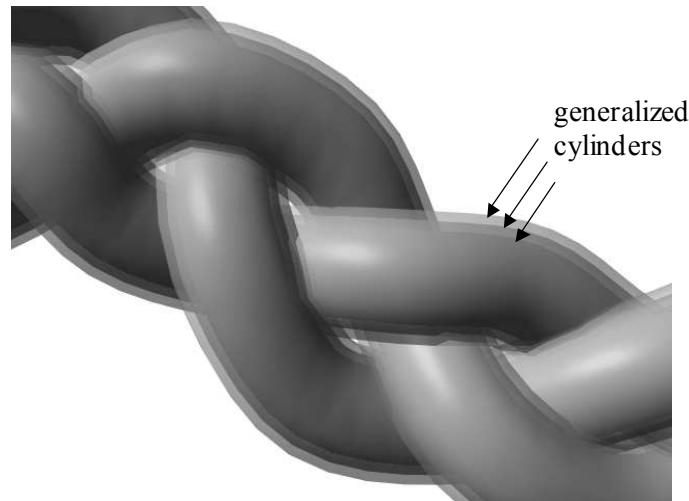


Figure 6.34: *Concentric generalized cylinders.*

Diagram A in Figure 6.35 shows the result when only an opacity map is applied to each generalized cylinder. We find that using a texture map as a bump map as well as an opacity map enhances the effect of individual strands of hair. Diagram B in Figure 6.35 shows the result when a texture map is applied as an opacity map and a bump map.

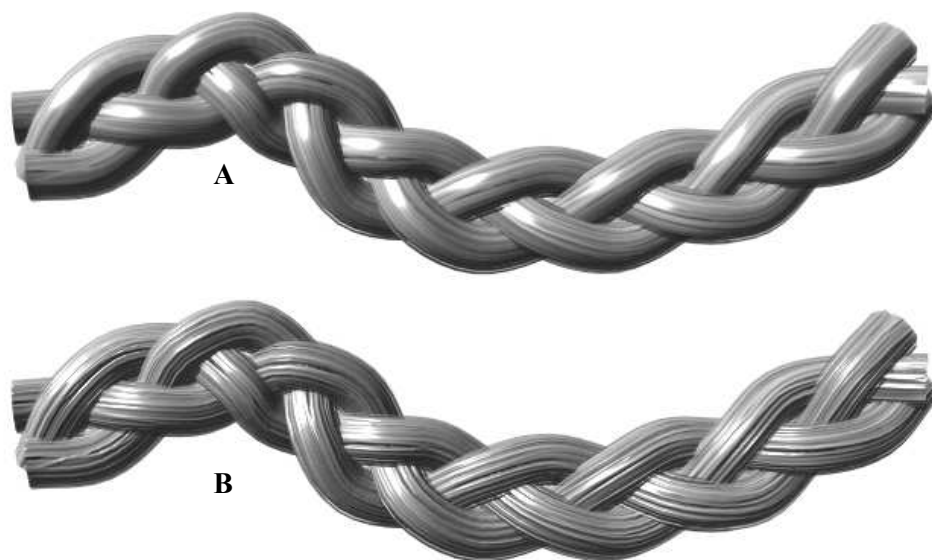


Figure 6.35: *Applying a bump map.*

Different coloured hair can be created using our cluster hair modelling technique by changing the surface colour of the generalized cylinders representing clusters of hair, or by applying another texture map that represents colour to the generalized cylinders. Figure 6.36 shows a number of different coloured braids of hair that are represented by generalized cylinders created with our cluster hair modelling technique.

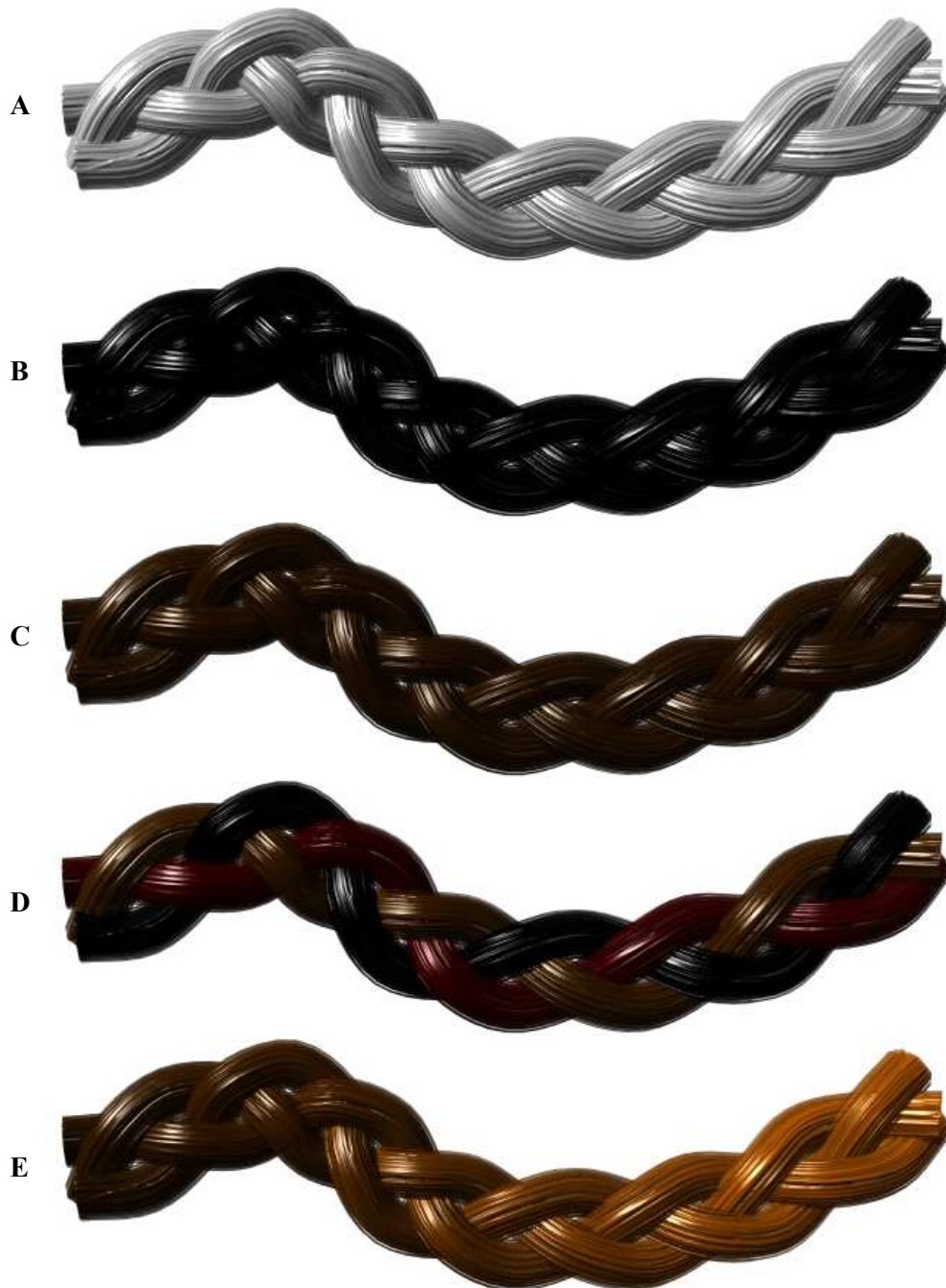


Figure 6.36: *Different coloured braids.*

Diagram A in Figure 6.36 shows a braid of hair with the default surface properties. The braid can be used to represent black hair (B) or any other colour of hair (C) by changing the surface colour of the generalized cylinders making up the braid. Each cluster in the braid can be given a different colour (D) if the curves defining the braid are converted into generalized cylinders separately and each given a different surface name. The colour in each cluster of hair can vary by applying a texture map to the generalized cylinders representing the clusters (E). Figure 6.37 shows the texture map that was used to specify the colour of the braid in Diagram E (in Figure 6.36).

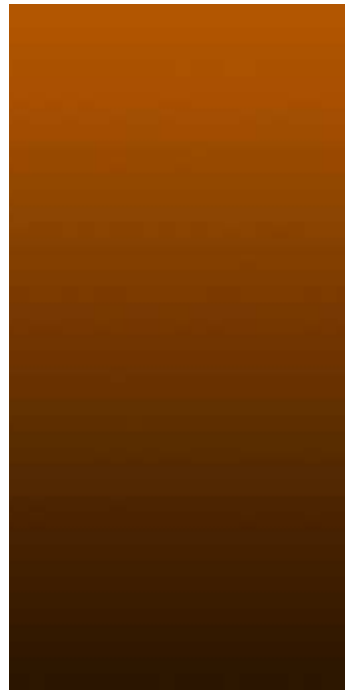


Figure 6.37: *Colour map.*

6.3.3.5 Unwanted Boundary Effects



Figure 6.38: *Unwanted artefacts at the edges.*

Additional textures are added to the head to create the effect of flatly parted hair (see Section 5.4.3). Since the parted hair forms a tessellation over the scalp, a single texture map does not have to be created for the entire scalp. Instead a smaller texture map that has a toroidal topology can be created and tiled over the scalp. Depending on the texture map used, unwanted boundary effects caused by the representation of incomplete shapes may appear at the edges due to the texture map's toroidal topology. Figure 6.38 highlights incomplete shapes (or unwanted artefacts) that occur at the edges due to the tiled texture map having a toroidal topology. Therefore, there is a trade off between creating a single texture map, which requires more memory and is more time consuming to create, and tiling a texture that may result in a few unwanted artefacts.

6.3.4 Antialiasing

6.3.4.1 Supersampling at Different Sampling Rates

To reduce aliasing in rendered hair we use supersampling, which involves sampling at a higher resolution than the display resolution and then filtering the samples to produce the final image pixels (see Section 5.5). Figure 6.39 shows the results when different sampling rates are used to render a sample of hair. The labels under the images represent the sampling rate in the horizontal and vertical directions, and, in brackets, the total number of samples taken for each pixel. The images show that the quality of the rendered hair increases as the sampling rate increases, but the difference in quality becomes negligible after a certain sampling rate (in this example at a rate of approximately 16 samples per pixel). Figure 6.40 displays the time taken to render the sample of hair with different sampling rates, and shows that the rendering time is higher for larger sampling rates. Therefore, there is a trade off between image quality and rendering time when using supersampling when rendering an image. We suggest using the lowest sampling rate that produces images with a reasonable quality, or the sampling rate at the point where increasing the sampling rate further will have an insignificant effect on the improvement of image quality.

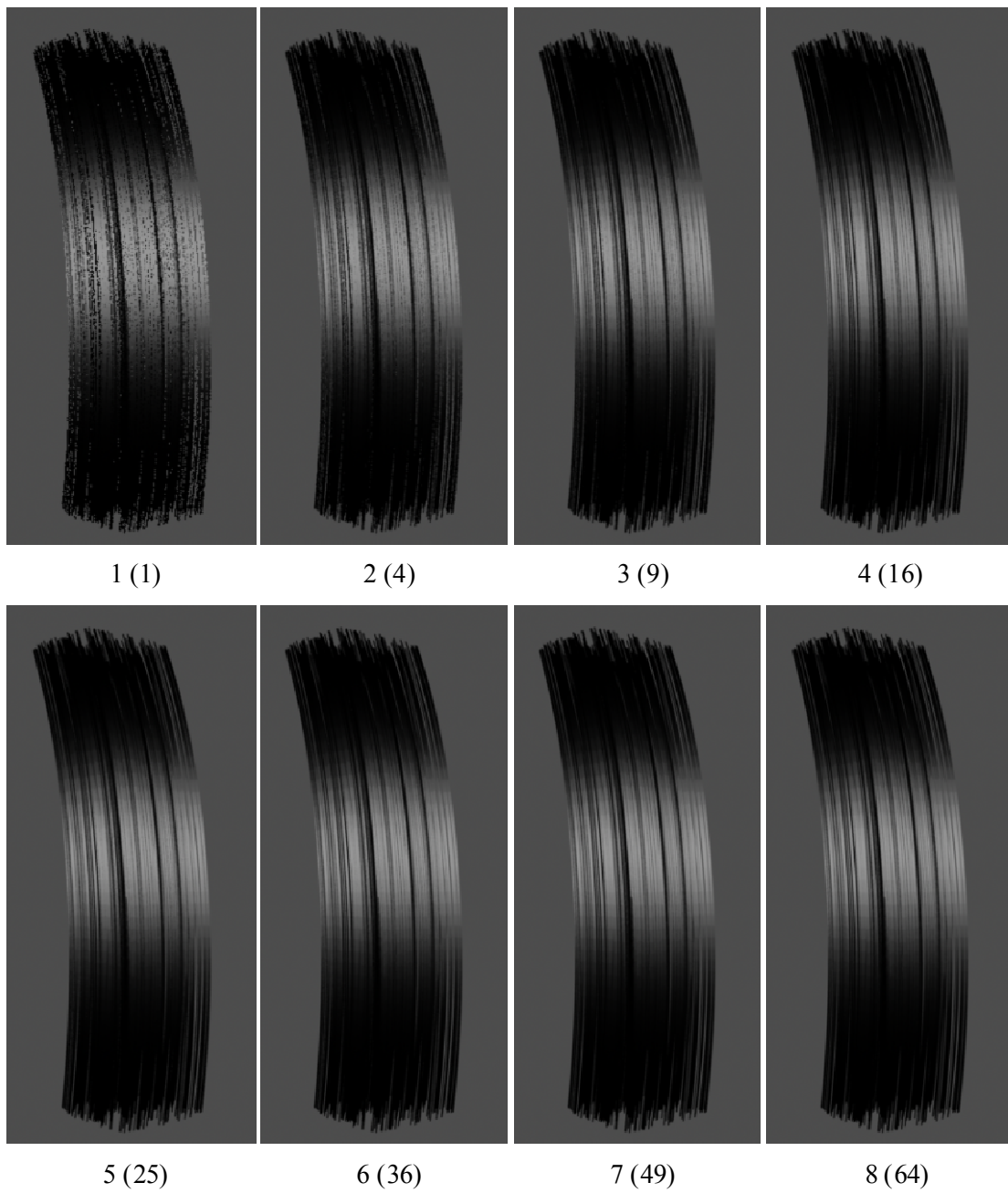


Figure 6.39: *Supersampling at different sampling rates.*

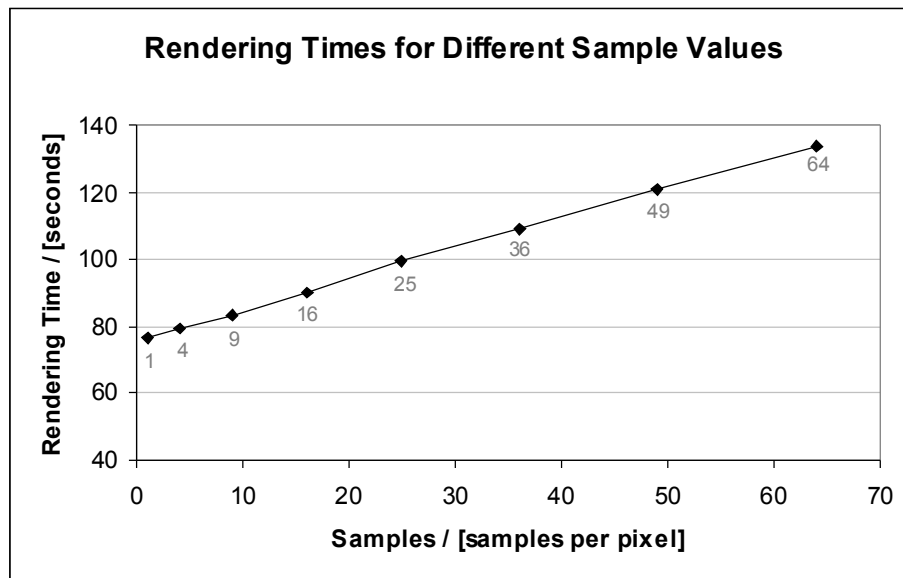


Figure 6.40: *Rendering times when using different sampling rates.*

6.3.4.2 Different Filters

Because supersampling involves sampling at a higher resolution than the display resolution, the samples taken need to be filtered before producing the final image. RenderMan supports a number of standard filter functions, including a box filter, Catmull-Rom filter, Gaussian filter and Sinc filter [Pixar 2000] (see Section 5.5). Figure 6.41 shows the rendering times when different types of filters are used with supersampling to render a sample of hair. The sample of hair was rendered using each filter with a supersampling rate of 4 samples per pixel and 16 samples per pixel, and a filter width of two pixels. We did not find any significant difference in rendering time or image quality between the different types of filters. The images rendered using the different filters can be seen in Appendix B. For this example the type of filter used does not matter, and any filter could be used. We suggest choosing the type and size of the filter through experimentation.

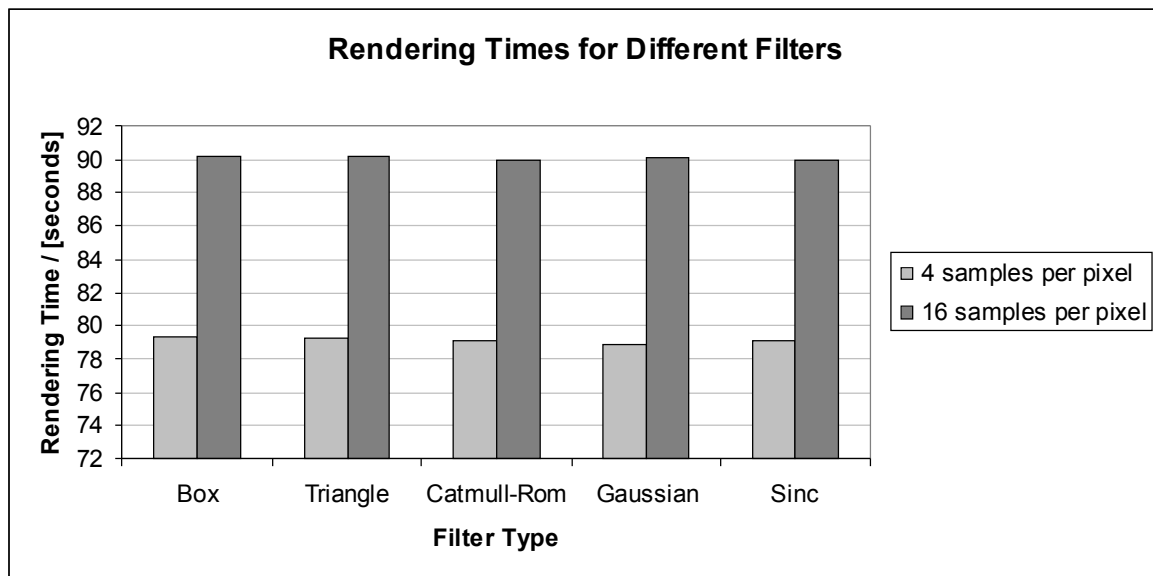


Figure 6.41: *Rendering times when using different filters.*

6.4 Final Results

Figures 6.42 – 6.45 show photographs of African hairstyles containing different forms of hair together with a similar hairstyle that has been generated using our hair modelling and rendering techniques.



Figure 6.42: *Natural curly hair.*

Figure 6.42 shows natural curly hair. The computer-generated hair created with our implicit hair modelling technique is less dense than the hair in the photograph, and appears slightly fuzzy.

However, it is still convincing, and may be made denser by applying texture maps created with more splines.



Figure 6.43: *Straightened hair.*

Figure 6.43 shows straightened hair. The computer-generated hair created with our explicit hair modelling technique is convincing. However, it appears smoother than the hair in the photograph, which has a slightly wavy texture. Possible ways of creating straightened hair with a slightly wavy texture would be to apply a texture map along the length of the hair, or to explicitly model wavy strands of hair. As a result of our control hair technique (discussed in Sections 3.2.1 and 3.2.2) the computer-generated hairstyle appears to consist of clumps of hair. The hair in the photograph also appears to clump, but not as much. To create straightened hair without clumps of hair, an interpolation technique should be used to create the hair strands in between the control hairs. Interpolation is discussed in more detail in Appendix C.



Figure 6.44: *Braids of hair.*



Figure 6.45: *Twists of hair.*

Figures 6.44 and 6.45 show hairstyles with braids and twists of hair. Figure 6.44 shows a cornrow hairstyle containing braids of hair, while Figure 6.45 shows a hairstyle containing twists of hair. In each figure, the computer-generated hair created with our cluster hair modelling technique appears visually convincing.

The African hairstyles created with our hair modelling and rendering techniques in figures 6.42 – 6.45 are convincing, and demonstrate that our techniques can be used to model and render African hair successfully.

6.5 Summary

In this chapter we provide several results, and discuss a number of issues, relating to our hair modelling and rendering techniques.

We show that our implicit, explicit and cluster hair modelling techniques can be used to create a variety of different African hairstyles with relative ease. Tests are performed on samples of hair to examine the effect that certain variables have on rendering time and image quality. The variables tested include the number of layers used to produce curly hair, polylines used to produce straight hair, and clusters used to represent braids and twists of hair; the number of segments making up a polyline or generalized cylinder; and the number of sides representing a generalized cylinder. Examining the results of the tests, we conclude that increasing the number of polygons used when modelling hair will increase the quality of the hair produced, but will also increase the rendering time. A large number of polygons are needed to represent hair explicitly

or with clusters. However, for most of the variables there is an upper bound on the number of polygons needed to produce a reasonable hairstyle. Generating more polygons than needed will increase the time taken to render the hair without substantially improving the visual quality of the hair. We suggest setting the variables to values that create the least number of polygons needed to provide an appropriate quality. The shape of a hairstyle modelled with our implicit hair modelling technique may appear unrealistic due to the structure of the underlying mesh. We show that the shape can be improved by applying an extra opacity map. We also discuss the issue of stray hairs that may appear when modelling hair using our explicit technique, and overlapping clusters that may occur when modelling braids with our cluster technique. We provide solutions to overcome these problems.

We examine the effects different components of our illumination model have on the rendered image of a sample of hair, and conclude that the diffuse component has no effect and is not needed, and the directional attenuation factor could be implemented as a constant value, when black, opaque African hair is rendered. However, we leave them in the illumination model so that African hair with different colours and opacities can be rendered. The result of rendering implicitly modelled hair represented by textured layers with a modified version of our illumination model is shown.

We examine the effect that different shadow map resolutions and self-shadowing have on rendering time and image quality. We conclude that the lowest resolution that produces hair with an appropriate quality should be chosen for a shadow map, because higher resolution shadow maps have higher rendering times. Adding self-shading to hair represented by layers does not have an effect on the rendering time.

We discuss several issues relating to textures. We show that different types of hair can be created for the same hairstyle by applying different texture maps, and different hair densities can be created with texture maps without affecting the rendering time, because the texture maps are pre-processed. We demonstrate that texture maps can be reused. We discuss the issue of texturing braids and twists to create the effect of many strands of hair in a cluster, and show that texturing three concentric generalized cylinders produces a better result than texturing only one. We find that using a texture map as a bump map as well as an opacity map enhances the effect of individual strands of hair, and we show that different coloured hair can be created using our

cluster hair modelling technique. The issue of incomplete shapes (or unwanted artefacts) that occur when tiling a texture onto a head model is also discussed.

Tests are performed on samples of hair to examine the effect that different sampling rates and filters have on rendering time and image quality when supersampling is implemented to remove aliasing artefacts. Examining the results of the tests, we conclude that increasing the sampling rate increases the quality of the hair produced, but also increases the rendering time. However, the difference in quality becomes negligible after a certain sampling rate, so we suggest using the lowest sampling rate that produces images with a reasonable quality, or the sampling rate at the point where further increases will have an insignificant effect on the improvement of image quality. We did not find any significant difference in the rendering time or image quality of hair rendered with different filters, so we suggest choosing the type and size of the filter through experimentation.

Finally, different African hairstyles created with our hair modelling and rendering techniques are presented in the chapter. The hairstyles presented demonstrate that our techniques can be used to simulate African hairstyles that contain curly hair, straight hair, or braids or twists of hair. In the next chapter we conclude the thesis and suggest directions for future work.

7 CONCLUSION

The aim of this research is to provide hair modelling and rendering techniques that can be used to add African hairstyles to virtual humans. In the Introduction we identify three forms of hair commonly found in African hairstyles, namely natural curly hair, straightened hair, and braids or twists of hair. Our goal is to develop hair modelling and rendering techniques that produce these three forms of hair, so that a user employing our techniques can design an African hairstyle.

Our hair modelling techniques are developed with several goals in mind:

- They need to be able to add or place hair produced onto a model of a head.
- They need to be able to represent the shape and geometric properties of the form of hair they are modelling.
- They need to overcome the problem of modelling a large number of hair strands.
- They need to be developed so that a user can model a variety of hairstyles with ease.

The hair modelling techniques presented satisfy the goals above and are implemented as plug-ins for the graphics program LightWave. We find that plug-in development for LightWave is not easy, and several issues encountered when developing plug-ins to implement our hair modelling techniques are discussed.

In order to render the African hair produced with our hair modelling techniques, several rendering aspects are investigated, and techniques to implement each aspect are presented. In particular:

- An appropriate anisotropic illumination model is developed.
- Shadowing techniques are presented.
- Anti-aliasing is described.
- Texturing techniques that create the effect of individual strands of hair are presented.

Our hair rendering techniques are implemented using shaders and a renderer compatible with the RenderMan Interface. Although LightWave has its own renderer, we use RenderMan for several reasons. RenderMan is an industry standard and the RenderMan Interface contains a shading language that is useful for developing and implementing rendering techniques.

7.1 Thesis Overview

In Chapter 2 we present a thorough review of existing hair modelling and rendering techniques. In the chapter we find that many techniques have been developed and some impressive results have been produced, which can be seen in several applications including film production. Commercial hair simulation tools are becoming more widely available to artists and several researchers have developed interactive systems with user interfaces for their techniques. However, there is no one superior technique, and most techniques only work well for particular types of hair under specific conditions. No previous techniques exist that are developed specifically for African hair or hairstyles.

The hair modelling techniques are divided into three categories: those that model hair implicitly, those that model hair explicitly, and those that use a combination of implicit and explicit techniques to model hair as clusters. Each category has advantages and disadvantages. Examining the advantages and disadvantages we conclude that an implicit model would be the most useful technique for modelling natural curly African hair, an explicit model would be the most appropriate method for modelling straightened African hair, and a cluster technique would be the best method for modelling braids and twists of African hair.

The rendering techniques investigated in Chapter 2 include illumination models, shadowing techniques, anti-aliasing methods and texturing. We find that an anisotropic illumination model is needed to render hair accurately, due to the narrow width of each strand of hair; shadowing and anti-aliasing techniques must be implemented in order to render realistic hair; and texturing methods can be used to create the effect of a large number of individual strands of hair.

In Chapter 3 we discuss the hair modelling techniques we use in order to model the three identified forms of African hair. Each technique is able to place hair onto a model of a head using a selection of polygons or a few control curves.

An implicit model, based on a technique that uses concentric textured layers, is used to model natural curly African hair. The shape, thickness, density and distribution of the individual strands of hair are defined by texture maps (or opacity maps) that control the opacity across each concentric layer. Since the individual strands of hair are represented by texture maps and do not have to be explicitly modelled, modelling a large number of hair strands is not an issue.

To model straightened African hair we use an explicit hair modelling technique. Each individual strand of hair is represented by a curve. To overcome the problem of modelling a large number of hair strands, we implement a technique that uses control hairs. The user defines a hairstyle with a few control curves, and these are then used to create and define the shape of many more strands of hair automatically.

We develop a cluster hair modelling technique to model African hair in the form of braids and twists. Initially the overall shape of each braid or twist is represented by a curve defined by a user. The curve is then converted into a number of curves that define the shape of the wisps of hair making up the braid or twist, and each wisp of hair is modelled as a cluster of hair using a textured generalized cylinder. Since the strands of hair in each braid are modelled as clusters represented by textured generalized cylinders, a large number of individual strands of hair do not have to be modelled. Additional textures may be placed onto the scalp to create the effect of parted hair that is not part of braids or twists.

In Chapter 4 we discuss seven plug-ins for LightWave that we develop in order to implement our hair modelling techniques. Using a combination of the plug-ins a user can interactively model hairstyles that contain curly African hair, straightened hair or braids or twists of hair. By implementing our techniques as plug-ins for an existing modelling program we do not have to implement an entire system that imports head models in different formats, has a user interface, or provides tools to perform a variety of operations. Our plug-ins add to the functionality of LightWave so that the three forms of African hair can be modelled with ease to create a variety of hairstyles. However, we find that plug-in development for LightWave is not easy. In the chapter we give a description of each plug-in together with some of the implementation issues involved in creating the plug-in for LightWave.

In Chapter 5 we discuss the rendering techniques we use in order to render hair that has been modelled using our plug-ins. The rendering techniques discussed include an illumination model, shadowing, texturing and anti-aliasing. We develop an illumination model for rendering African hair created with our implicit and explicit hair modelling techniques, because conventional illumination models are not appropriate. We apply shadowing to explicitly modelled African hair using traditional shadow maps and show that hair rendered with shadow maps appears more realistic and seems to have more depth than hair rendered without shadow maps. We create the effect of self-shadowing in implicitly modelled hair represented by textured layers by making the

surface colour of the lower layers closer to the scalp darker than the colour of the higher layers further away from the scalp. We use textures to create the effect of hair and to enhance the realism of a hairstyle, by applying texture maps to layers to create the effect of curly hair, to generalized cylinders to create the effect of many strands of hair, and to a cap placed on the head model to enhance the realism of a hairstyle. We reduce aliasing in rendered hair by using supersampling.

The rendering is done using a renderer compatible with the RenderMan Interface, and the African hair modelled with our plug-ins in LightWave is exported as a RIB file using LightMan. In Chapter 5 we also discuss the classic three point lighting set-up, which we use to test our rendering techniques. Because a renderer compatible with the RenderMan Interface is used, our illumination model is implemented as a shader using the RenderMan Shading Language.

In Chapter 6 we present several results, and discuss a number of issues relating to our hair modelling and rendering techniques. We show that our implicit, explicit and cluster hair modelling techniques can be used to create a variety of different African hairstyles with relative ease. Tests are performed on samples of hair to examine the effect that different variables have on rendering time and image quality. Examining the results of the tests, we conclude that increasing the number of polygons used when modelling hair will increase the quality of the hair produced, but also increase the rendering time. A large number of polygons are needed to represent hair explicitly or with clusters. However, for most of the variables there is an upper bound on the number of polygons needed to produce a reasonable hairstyle. Generating more polygons than needed will increase the time taken to render the hair without substantially improving the visual quality of the hair. We suggest setting the variables to values that create the least number of polygons needed to provide images of an appropriate quality. The shape of a hairstyle modelled with our implicit hair modelling technique may appear unrealistic due to the structure of the underlying mesh. We show that the shape can be improved by applying an extra opacity map. We also discuss the issue of stray hairs that may appear when modelling hair using our explicit technique and overlapping clusters that may occur when modelling braids with our cluster technique. We provide solutions to overcome these problems: the stray hairs can be modified or deleted because they are explicitly modelled with our explicit hair modelling technique, and the overlapping among clusters can be removed by changing the angle between the curves that originally defined the clusters.

We examine the effects different components of our illumination model have on the rendered image of a sample of hair. Results show that a diffuse component is not needed in the lighting model used to illuminate black, opaque African hair, and the colour of the hair should not affect the specular highlights. However, we leave the diffuse component in the illumination model so that African hair with different colours and opacities can be rendered. Due to the opacity of African hair, light reflection and transmission must be considered. The result of rendering implicitly modelled hair represented by textured layers with a modified version of our illumination model is shown.

We examine the effect that different shadow map resolutions and self-shadowing have on rendering time and image quality. We conclude that the lowest resolution that produces hair with an appropriate quality should be chosen for a shadow map, because higher resolution shadow maps have higher rendering times. Adding self-shading to hair represented by layers does not have an effect on the rendering time.

We discuss several issues relating to textures. We show that different types of hair can be created for the same hairstyle by applying different texture maps, and different hair densities can be created with texture maps without affecting the rendering time because the texture maps are pre-processed. We demonstrate that texture maps can be reused. We discuss texturing braids and twists to create the effect of many strands of hair in a cluster, and show that texturing three concentric generalized cylinders produces a better result than texturing only one. We find that using a texture map as a bump map as well as an opacity map enhances the effect of individual strands of hair, and we show that different coloured hair can be created using our cluster hair modelling technique. The issue of unwanted boundary effects that occur when tiling a texture onto a head model is also discussed.

Tests are performed on samples of hair to examine the effect that different sampling rates and filters have on rendering time and image quality when supersampling is implemented to remove aliasing artefacts. Examining the results of the tests, we conclude that increasing the sampling rate increases the quality of the hair produced, but also increases the rendering time. However, the difference in quality becomes negligible after a certain sampling rate, so we suggest using the lowest sampling rate that produces images with a reasonable quality, or the sampling rate at the point where increasing the sampling rate any more will have an insignificant effect on the improvement of image quality. We do not find any significant difference in the rendering time or

image quality of hair rendered with different filters, so we suggest choosing the type and size of the filter through experimentation.

Finally we present different African hairstyles created with our hair modelling and rendering techniques. The hairstyles presented demonstrate that the techniques presented in the thesis can be used to simulate African hairstyles that contain curly hair, straight hair, or braids or twists of hair.

7.2 Future Work

Methods for simulating hair using computer graphics are needed so that hairstyles can be added to virtual humans. In this thesis we contribute to hair simulation research by investigating hair modelling and rendering techniques in order to produce different African hairstyles. When simulating hair three aspects need to be considered: hair modelling, hair rendering, and hair dynamics. However, due to time constraints and the maximum length permitted for a thesis, techniques for hair dynamics (which involves the movement of hair and its interaction with objects, the air and itself [Magenat-Thalmann et al. 2000]) are beyond the scope of the investigation. Therefore, future work could concentrate on investigating hair dynamics techniques that could be used in conjunction with our hair modelling and rendering techniques in order to simulate African hair for virtual humans.

More specifically we recommend:

- An investigation of African hair movement and the development of a dynamics model, which could be used to animate African hair modelled with the techniques discussed in this thesis. Due to the curliness and types of hairstyles produced, the dynamics of African hair differs from that of other hair, and in many African hairstyles the hair does not move much (if at all) with respect to the head.
- The development of methods for collision detection and response, which are needed when animating hair, and which can also be useful when modelling hair to ensure that strands of hair do not intersect with other objects such as the head.

- An investigation into improving the performance of our hair modelling and rendering techniques so that African hair can be placed on virtual humans used in applications that require real-time performance. The investigation should include the application of multi-resolution techniques (e.g. mip-mapping and distance-based culling) to improve performance and image quality when the distance between the viewer and the hair changes.

7.3 Thesis Contributions

As outlined in the introduction to this chapter, the goal of this research is to provide hair modelling and rendering techniques that can be used to add African hairstyles to virtual humans. As demonstrated throughout this thesis, this goal has been achieved, making the following contributions:

- A comprehensive review of the literature on existing hair modelling and rendering techniques.
- The development of three hair modelling techniques, which are designed to model different forms of hair commonly found in African hairstyles. The techniques are developed to model natural curly hair, straightened hair and braids or twists of hair, and can be used to produce a variety of African hairstyles, for which no specific methods existed previously.
- The enhancement of an illumination model that can be used to render African hair.
- An investigation of several rendering aspects, emphasizing the importance of including a shadowing technique and applying an anti-aliasing method when rendering hair, and demonstrating the usefulness of using texturing to represent many strands of hair.

8 REFERENCES

- [3Delight 2002] THE 3DELIGHT TEAM. 2002. 3Delight. Available online at: <http://www.3delight.com/>.
- [Ahokas 2002] AHOKAS, T. Shadow Maps. Seminar on Computer Graphics, Helsinki University of Technology Telecommunications Software and Multimedia Laboratory. Available online at: http://www.tml.hut.fi/Opinnot/Tik-111.500/2002/paperit/timo_ahokas.pdf.
- [Ananova 2003] ANANOVA LTD. 2003. Available online at: <http://www.ananova.com/video>.
- [Anjyo et al. 1992] ANJYO, K., USAMI, Y., AND KURIHARA, T. 1992. A Simple Method for Extracting the Natural Beauty of Hair. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26(2), ACM, 111–120.
- [Banks 1994] BANKS, D. 1994. Illumination in Diverse Codimensions. In *Proceedings of ACM SIGGRAPH 94*, ACM Press / ACM SIGGRAPH, 327–334.
- [Bidasaria 1995] BIDASARIA, H. 1995. A New Method for Modeling of Hair-Grass Type Textures. *ACM Conference on Computer Science*, 109–113.
- [Birn 2000] BIRN, J. 2000. Three-Point Lighting for 3D Renderings. Available online at: <http://www.3drender.com/light/3point.html>. (Adapted from *[Digital] Lighting & Rendering*, by J. Birn, published by New Riders, 2000).
- [Blinn 1977] BLINN, J. 1977. Models of Light Reflection for Computer Synthesized Pictures. *Computer Graphics (Proceedings of SIGGRAPH 77)*, 11(2), 192–198.
- [Bloomenthal 1990] BLOOMENTHAL, J. 1990. Calculation of Reference Frames along a Space Curve. In *A. Glassner, editor, Graphics Gems, vol. 1*, Academic Press, 567–571.
- [Bredow 2000] BREDOW, R. 2000. Chapter 5: Fur in Stuart Little. *Advanced RenderMan 2: To RI_INFINITY and Beyond*, SIGGRAPH 2000 Course 40, 89-117. Available online at: http://www.185vfx.com/stuart_fur.html.

- [Chang et al. 2002] CHANG, J., JIN, J., AND YU, Y. 2002. A Practical Model for Hair Mutual Interactions. *ACM SIGGRAPH Symposium on Computer Animation, San Antonio*, 73–80.
- [Chen et al. 1999] CHEN, L., SAEYOR, S., DOHI, H., AND ISHIZUKA, M. 1999. A System of 3D Hair Style Synthesis Based on the Wisp Model. *The Visual Computer*, 15 (4), 159–170.
- [Comet 2000] COMET, M. 2000. 3D Hair Tutorial. Available online at:
<http://www.comet-cartoons.com/toons/3ddocs/hair/>.
- [Computer Graphics World 1995] COMPUTER GRAPHICS WORLD. 1995. Believable Hair in a Fantasy World. *Computer Graphics World*, November 1995. Available online at:
http://cgw.pennnet.com/Articles/Article_Display.cfm?Section=Archives&Subsection=Display&ARTICLE_ID=47683&KEYWORD=hair%20raising&x=y.
- [Csuri et al. 1979] CSURI, C., HACKATHORN, R., PARENT, R., CARLSON, W., AND HOWARD, M. Towards an Interactive High Visual Complexity Animation System. In *Computer Graphics (Proceedings of ACM SIGGRAPH 79)*, 13(2), ACM, 289-299.
- [Daldegan et al. 1993] DALDEGAN, A., THALMANN, N., KURIHARA, T., AND THALMANN, D. 1993. An Integrated System for Modeling, Animating and Rendering Hair. *Computer Graphics Forum (Eurographics 93)*, 12(3), Blackwell Publishers, 211–221.
- [Dapper 2001] DAPPER, T. 2001. LightMan 1.1. Available online at:
<http://www.td-grafik.de/softw/lightman.php>.
- [Dapper 2003] DAPPER, T. 2003. LightMan Version 1.1 Documentation. Available online at:
<http://www.td-grafik.de/softw/lightman/doc/index.html>.
- [Deedrick 2000] DEEDRICK, D. Hairs, Fibers, Crime, and Evidence. 2000. *Forensic Science Communications*, 2(3). Available online at:
[http://www.fbi.gov/hq/lab/fsc/backissu/july2000/deedric1.htm#Racial Determination](http://www.fbi.gov/hq/lab/fsc/backissu/july2000/deedric1.htm#Racial%20Determination).

- [Dischler 1999] DISCHLER, J. 1999. A General Model of Animated Shape Perturbation. *Graphics Interface '99*, 140–147.
- [Disney 2001] ©DISNEY ENTERPRISES, INC./PIXAR ANIMATION STUDIOS. 2001. Monster's Inc. Available online at:
<http://disney.go.com/disneyvideos/animatedfilms/monstersinc/index2.html>.
- [Dot C Software 2002] DOT C SOFTWARE. 2002. RenderDotC v3.3.5. Available online at:
<http://www.dotcsw.com/>.
- [Ebert et al. 1994] EBERT, D., MUSGRAVE, F., PEACHEY, D., PERLIN, K., AND WORLEY, S. 1994. *Texturing and Modeling: A Procedural Approach*, Second Edition. AP Professional, Academic Press, 1994.
- [Foley et al. 1990] FOLEY, J., VAN DAM, A., FEINER, S., AND HUGHES, J. 1990. *Computer Graphics: Principles and Practice*, Second Edition in C. Addison-Wesley, 1990.
- [Gilmer1998] GILMER, G. 1998. Mathematical Patterns in African American Hairstyles. Available online at:
http://www.math.buffalo.edu/mad/special/gilmer-gloria_HAIRSTYLES.html.
- [Goldman 1997] GOLDMAN, D. 1997. Fake Fur Rendering. In *Proceedings of ACM SIGGRAPH 97*, ACM Press / ACM SIGGRAPH, 127–134.
- [Grabli et al. 2002] GRABLI, S., SILLION, F., MARSCHNER, S., AND LENGYEL, J. 2002. Image-Based Hair Capture by Inverse Lighting. In *Proceedings of Graphics Interface 2002*, 51–58.
- [Gritz 1999] GRITZ, L. 1999. Blue Moon Rendering Tools, User Manual, Release 2.5. Available online at: <http://www.bmrt.org/bmrtdoc/bmrtdoc.html>.
- [Guang and Zhiyong 2002] GUANG, Y., AND ZHIYONG, H. 2002. A Method of Human Short Hair Modeling and Real Time Animation. *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications (PG'02)*, 435–438.

- [Hadap and Magnenat-Thalmann 2000] HADAP, S. AND MAGNENAT-THALMANN, N. 2000. Interactive Hair Styler based on Fluid Flow. *Proceedings of Eurographics Workshop on Computer Animation and Simulation 2000*, 87–100.
- [Hadap and Magnenat-Thalmann 2001] HADAP, S. AND MAGNENAT-THALMANN, N. 2001. Modeling Dynamic Hair as a Continuum. In *Eurographics Proceedings 2001*, 20(3).
- [Heinen and Walter 2001] HEINEN, F. AND WALTER, M. 2001. GLFurry - An Interactive 3D Fur Modeling System. *Proceedings of the XIV Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'01)*, 379.
- [Iones et al. 2000] IONES, A., KRUPKIN, A., VOLODARSKY, S., AND ZHUKOV, S. 2000. Fur and Hair: practical modeling and tendering techniques. *International Conference on Information Visualisation (IV2000)*, 145–151.
- [Kajiya and Kay 1989] KAJIYA, J. AND KAY, T. 1989. Rendering Fur with Three Dimensional Textures. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, 23(3), ACM, 271–280.
- [Khouas et al. 1998] KHOUAS, L., ODET, C., AND FRIBOULET, D. 1998. 3D Furlike Texture Generation by a 2D Autoregressive Synthesis. *Winter School in Computer Graphics, The Sixth International Conference on Computer Graphics and Visualization '98*, 171–177.
- [Kim and Neumann 2000] KIM, T. AND NEUMANN, U. 2000. A Thin Shell Volume for Modeling Human Hair. *IEEE Computer Animation*, 104–111.
- [Kim and Neumann 2001] KIM, T. AND NEUMANN, U. 2001. Opacity Shadow Maps. *Eurographics Workshop on Rendering*, 177–182.
- [Kim and Neumann 2002] KIM, T. AND NEUMANN, U. 2002. Interactive Multiresolution Hair Modeling and Editing. *ACM Transactions on Graphics*, 21(3), 620–629.

- [Klein 2002] KLEIN, B. 2002. African American Hairstyle Exhibit. *American Mosaic: Radio Magazine*, Broadcast 15 February 2002. Available online at: http://www.manythings.org/voa/02/020215am_t.htm.
- [Koh and Huang 2001] KOH, C., AND HUANG, Z. 2001. A Simple Physics Model to Animate Human Hair in 2D Strips in Real Time. In *Proceedings of Eurographics Workshop on Computer Animation and Simulation*, 127–138.
- [Kong and Nakajima 1999] KONG, W., AND NAKAJIMA, M. 1999. Visible Volume Buffer for Efficient Hair Expression and Shadow Generation. *Computer Animation 1999*, 58–65.
- [Kong and Nakajima 2000] KONG, W., AND NAKAJIMA, M. 2000. Hair Rendering by Jittering and Pseudo Shadow. *Computer Graphics International (CGI'00)*, 287–291.
- [Kowalski et al. 1999] KOWALSKI, M., MARKOSIAN, L., NORTHRUP, J., BOURDEV, L., BARZEL, R., HOLDEN, L., AND HUGHES, J. 1999. Art-Based Rendering of Fur, Grass, and Trees. In *Proceedings of ACM SIGGRAPH 99*, ACM Press / ACM SIGGRAPH, 433–438.
- [Lancaster 2003] LANCASTER, T. 2003. RenderMan® Repository. Available online at: <http://www.renderman.org/RMR/>.
- [Lao et al. 2002] LAO, W., KONG, D., AND YIN, B. 2002. An Improved Algorithm for Hairstyle Dynamics. *Fourth IEEE International Conference on Multimodal Interfaces (ICMI'02)*, 535–540.
- [LeBlanc et al. 1991] LEBLANC, A., TURNER, R., AND THALMANN, D. 1991. Rendering Hair using Pixel Blending and Shadow Buffers. In *The Journal of Visualization and Computer Animation, Vol 2*, 92–97.
- [Lee et al. 2002] LEE, C., CHEN, W., LEU, E., AND OUHYOUNG, M. 2001/2002. A Rotor Platform Assisted System for 3D Hairstyles. *Journal of Winter School in Computer Graphics*, 10, 271–278.

- [Lengyel et al. 2001] LENGYEL, J., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2001. Real-Time Fur over Arbitrary Surfaces. In *Proceedings of Interactive Symposium on 3D Graphics*, 227–232.
- [Lengyel 2000] LENGYEL, J. 2000. Real-Time Fur. *Eurographics Rendering Workshop 2000*, 243–256.
- [Lind 2003] LIND, C. 2003. Creating a Head with Hair in Maya. Available online at: <http://www.caligraphics.dk/Maya/hair/hair.htm>.
- [Lokovic and Veach 2000] LOKOVIC, T. AND VEACH, E. 2000. Deep Shadow Maps. In *Proceedings of SIGGRAPH 2000*, ACM, New York, 385–392.
- [Magnenat-Thalmann et al. 2000] MAGNENAT-THALMANN, N., HADAP, S., AND KALRA, P. 2000. State of the Art in Hair Simulation. *International Workshop on Human Modeling and Animation*, 3–9.
- [Mao et al. 2000] MAO, X., KIKUKAWA, M., KASHIO, K., AND IMAMIYA, A. 2000. Automatic Generation of Hair Texture with Line Integral Convolution. *International Conference on Information Visualisation (IV2000)*, 303–308.
- [Marschner et al. 2003] MARSCHNER, S., WANN JENSEN, H., CAMMARANO, M., WORLEY, S., AND HANRAHAN, P. 2003. Light Scattering from Human Hair Fibers. In *Proceedings of SIGGRAPH 2003*, 780–791.
- [Moelzer 2003] MOELZER, E. 2003. Creating Polygon Hair. Available online at: <http://www.newtek.com/products/lightwave/tutorials/modeling/hair/>.
- [Nakakihara 2001] NAKAKIHARA, D., BARTON, S., AND MARSHALL, B. 2001. *LightWave 3D Seven Reference Guide*. NewTek.
- [Neyret 1995] NEYRET, F. 1995. A General and Multiscale Model for Volumetric Textures. In *Graphics Interface '95 Proceedings*, 83–91.

- [Neyret 1998] NEYRET, F. 1998. Modeling Animating and Rendering Complex Scenes using Volumetric Textures. *IEEE Transactions on Visualization and Computer Graphics* 1(4), 55–70.
- [Neyret and Cani 1999] NEYRET, F. AND CANI, M. 1999. Pattern-Based Texturing Revisited. In *Proceedings of ACM SIGGRAPH 99*, ACM Press / Addison-Wesley Publishing Co., 235–242.
- [Papaioannou 2002] PAPAIOANNOU, G. 2002. A Simple and Fast Technique for Fur Rendering. Technical Report, University of Athens. Available online at:
<http://cgi.di.uoa.gr/~georgep/fur/fur2002.pdf>
- [Patrick and Bangay 2003] PATRICK, D. AND BANGAY, S. 2003. A LightWave 3D Plug-in for Modeling Long Hair on Virtual Humans. In *Proceedings of AFRIGRAPH 2003*, ACM SIGGRAPH, 161–166.
- [Perlin and Hoffert 1989] PERLIN, K. AND HOFFERT, E. 1989. Hypertexture. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, 23(3), ACM, 253–262.
- [Pixar 2000] PIXAR. 2000. *The RenderMan Interface, Version 3.2*. Available online at:
<https://renderman.pixar.com/>.
- [Pixar 2002] PIXAR ANIMATION STUDIOS. 2002. Pixar Releases RenderMan 11. Available online at: https://renderman.pixar.com/products/news/prman11_release.html.
- [Pixar 2003] PIXAR ANIMATION STUDIOS. 2003. Pixar’s RenderMan®. Available online at:
<https://renderman.pixar.com/>.
- [PlanetMath 2003] PLANETMATH. 2003. Frenet Frame Definition. Available online at:
<http://planetmath.org/encyclopedia/TNBFrame.html>.
- [Plante et al. 2001] PLANTE, E., CANI, M., AND POULIN, P. 2001. A Layered Wisp Model for Simulating Interactions inside Long Hair. *Eurographics Workshop on Computer Animation and Simulation 2001*, 139–148.

- [Policarpo 2003] POLICARPO. 2003. 3 Point Lighting Setup and Light Layer Compositing. Available online at: <http://www.newtek.com/products/lightwave/tutorials/animation/3point-light/>.
- [Rankin and Hall 1996] RANKIN, J. AND HALL, R. 1996. A Simple Naturalistic Hair Model. *ACM SIGGRAPH Computer Graphics* 30(1), ACM Press, 5–9.
- [Robertson 1995] ROBERTSON, B. 1995. Hair-Raising Effects. *Computer Graphics World*, October 1995. Available online at: http://cgw.pennnet.com/Articles/Article_Display.cfm?Section=Archives&Subsection=Display&ARTICLE_ID=47604&KEYWORD=hair%20raising.
- [Robertson 1999] ROBERTSON, B. 1999. PuttinOn Hairs. *Computer Graphics World*, January 1999. Available online: http://cgw.pennnet.com/Articles/Article_Display.cfm?Section=Archives&Subsection=Display&ARTICLE_ID=50214&KEYWORD=hair%20raising.
- [Robertson 2001] ROBERTSON, B. 2001. Monster Mash. *Computer Graphics World*, October 2001. Available online at: http://cgw.pennnet.com/Articles/Article_Display.cfm?Section=Archives&Subsection=Display&ARTICLE_ID=120494&KEYWORD=hair%20raising.
- [Silas 2000] SILAS, F. 2000. Creating Photo-Realistic Hair in Animation Master. Available online at: <http://www.franksilas.com/Hair.htm>.
- [SiTex Graphics 2003] SITEX GRAPHICS, INC. 2003. Air 2.6. Available online at: <http://www.sitexgraphics.com/html/air.html>.
- [Sourin et al. 1996] SOURIN, A., PASKO, A., AND SAVCHENKO, V. 1996. Using Real Functions with Application to Hair Modelling. *Computers and Graphics*, 20(1), 11–19.
- [Square 2001] SQUARE CO., LTD. 2001. Final Fantasy X. Available online at: <http://www.square-enix-usa.com/index.html>.

- [Subrahmanyam 2002] SUBRAHMANYAM, P. 2002. Re: Static vertex normals (was [LW-P]). *LightWave Plugin Developers Mailing List*. Available online at: <http://groups.yahoo.com/group/lw-plugin/message/7691>.
- [Tezel and Teo 2002] TEZEL, A., AND TEO, L. 2002. On RenderMan 11 - Interview with Dana Batali from Pixar. *CGNetworks*. Available online at: http://www.cgnetworks.com/story_custom.php?story_id=96&page=.
- [Thalmann et al. 1996] THALMANN, N., CARION, S., COURCHESNE, M., VOLINO, P., AND WU, Y. 1996. Virtual Clothes, Hair and Skin for Beautiful Top Models. *Computer Graphics International*, 132–141.
- [Turk 1990] TURK, G. 1990. Generating Random Points in Triangles. In *A. S. Glassner, editor, Graphics Gems I*. Academic Press, 24–28.
- [Van Gelder and Wilhelms 1997] VAN GELDER, A. AND WILHELMS, J. 1997. An Interactive Fur Modeling Technique. *Graphics Interface '97*, 181–188.
- [Wang and Yang 2001] WANG, T., AND YANG, X. 2001. A Design Tool for the Hierarchical Hair Model. *Proceedings of the Fifth International Conference on Information Visualisation (IV'01)*, 186–191.
- [Ward et al. 2003] WARD, K., LIN, M., LEE, J., FISCHER, S., AND MACRI, D. 2003. Modeling Hair Using Level-of-Detail Representations. *Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA'03)*, 41–47.
- [Watanabe and Suenaga 1992] WATANABE, Y., AND SUENAGA, Y. 1992. A Trigonal Prism-Based Method for Hair Image Generation. *IEEE Computer Graphics and Applications*, 47–53.
- [Watt 1989] WATT, A. 1989. *Fundamentals of Three-Dimensional Computer Graphics*. Addison-Wesley, 1989.

- [Watt and Watt 1992] WATT, A., AND WATT, M. 1992. *Advanced Animation and Rendering Techniques: Theory and Practice*. ACM Press, 1992.
- [Wright 2000] WRIGHT, E. 2000. Scenscan Sample. Available online at:
<http://www.lightwave3d.com/developer>.
- [Wright 2001] WRIGHT, E. 2001. *LightWave Development Documentation*. Available online at:
<http://www.lightwave3d.com/developer>.
- [Wright 2002] WRIGHT, E. 2002. Re:[LW-P] value on curve in Modeler. LightWave Plugin Developers Mailing List. Available online at:
<http://groups.yahoo.com/group/lw-plugin/message/6781>.
- [Xu and Yang 2001] XU, Z., AND YANG, X. 2001. V-HairStudio: An Interactive Tool for Hair Design. *IEEE Computer Graphics and Applications*, 36–43.
- [Yang and Ouhyoung 1997] YANG, T. AND OUHYOUNG, M. 1997. Rendering Hair with Back-lighting. *Proceedings of CAD/Graphics '97*, 291–296.
- [Yu 2001] YU, Y. 2001. Modeling Realistic Virtual Hairstyles. *Proceedings of the 9th Pacific Conference on Computer Graphics and Applications 2001*, Tokyo, 295–304.

APPENDIX A - Textured Layers Hair Shader

We implement an anisotropic illumination model as a shader, using the RenderMan Shading Language [Pixar 2000]. The modified shader used to illuminate African hair that is modelled as textured layers (and discussed in Section 5.2.4) is given below:

```
surface HairShader (float Ka = 0.0, Kd = 0.5, Ks = 0.5, Kn = 0.1, Kb = 1.0, Fr = 1.0, Ft = 1.0;
                    string opacityTexture = "", tangentTexture = "");
{
    /* The hair tangent vector */
    vector T;
    color tt = color texture (tangentTexture);
    setxcomp (T, comp (tt, 0));
    setycomp (T, comp (tt, 1));
    setzcomp (T, comp (tt, 2));
    normalize (T);
    vector H, Ln, V = -normalize(I); /* The halfway, light, and eye vectors */
    uniform float nondiff, nonspec; /* The light's contribution */
    float k, Fdir; /* The directional attenuation factor */

    /* The ambient component */
    Ci = Cs * Os * float texture (opacityTexture) * Ka * ambient();

    /* The diffuse and specular components */
    illuminance (P) {
        /* Get the next non-ambient light vector */
        Ln = normalize(L);
        /* Calculate the directional attenuation factor */
        k = ((T^Ln).(T^V))/((length(T^Ln))*(length(T^V)));
        Fdir = (((1 + k)/2) * Fr) + (((1 - k)/2) * Ft);
        /* Calculate the diffuse component */
        nondiff = 0;
        lightsource ("__nondiffuse", nondiff);
        if (nondiff < 1)
            Ci += Ci * Cs * Os * float texture (opacityTexture) * Kd *
                pow ((1 - pow ((T.Ln), 2)), Kb/2) * Fdir;
        /* Calculate the specular component */
        nonspec = 0;
        lightsource ("__nonspecular", nonspec);
        if (nonspec < 1) {
            H = normalize(Ln+V);
            Ci += Ci * Os * float texture (opacityTexture) * Ks *
                pow ((1 - pow ((T.H), 2)), 1/(Kn*2)) * Fdir;
        }
    }

    /* The incident ray opacity */
    Oi = Os * float texture (opacityTexture);
}
```

APPENDIX B - Different Filter Types

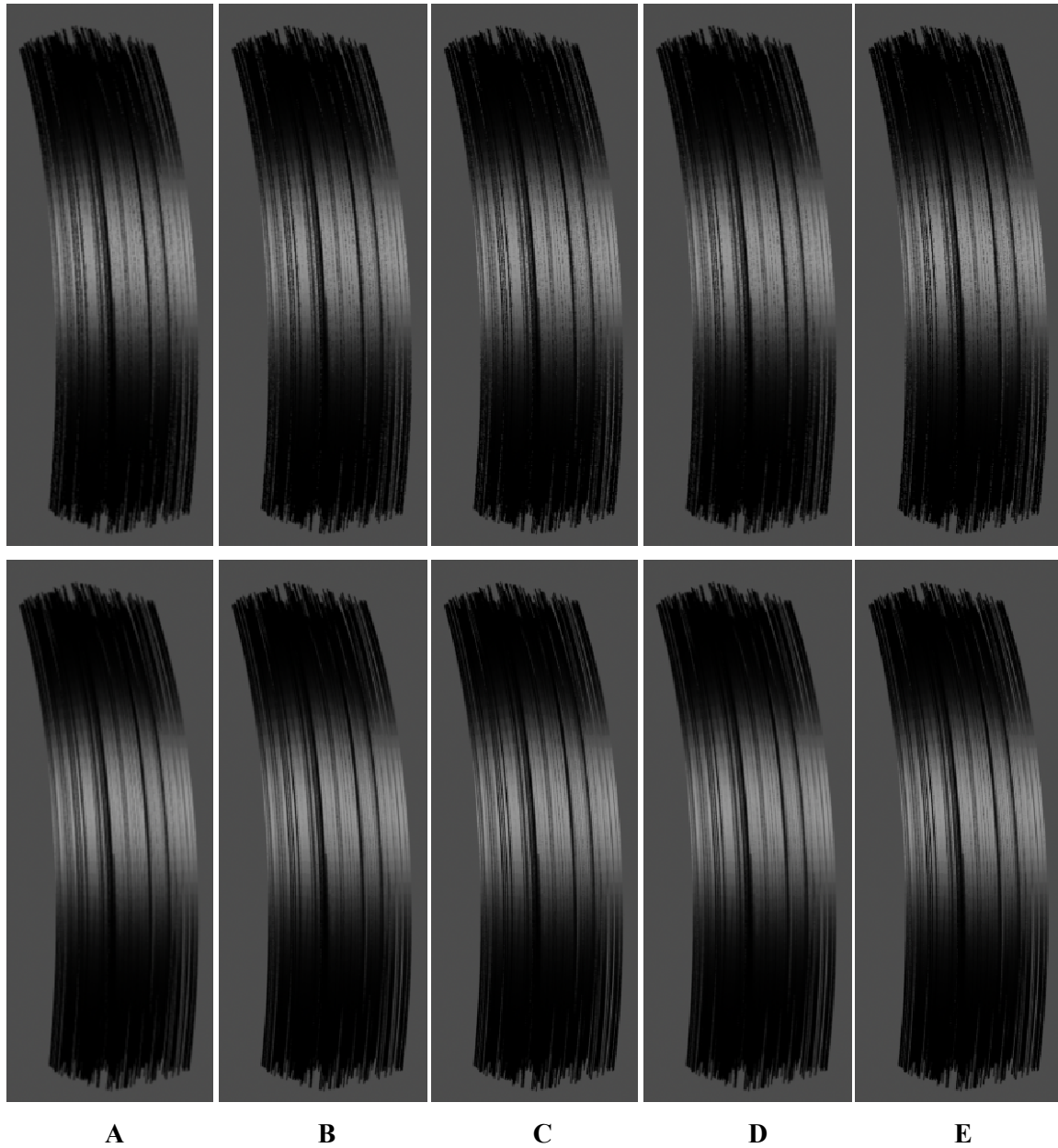


Figure B.1: *Using different filter types to render an image.*

Figure B.1 shows a sample of hair rendered with different filters: Box filter (A), Triangle filter (B), Catmull-Rom filter (C), Gaussian filter (D), and Sinc filter (E). The top images are supersampled by taking 4 samples per pixel, and the bottom images are supersampled by taking 16 samples per pixel.

APPENDIX C - Interpolation

We use an explicit hair modelling technique (discussed in Section 3.2) to model straightened African hair. To reduce the amount of time spent on hair modelling, a hairstyle is defined by a few control hairs, which are placed in the centre of triangles making up the region of the head model where hair should be placed. To create a full head of hair, other hairs are automatically created by making a number of copies of each control hair. The copies are placed on the head model by placing them randomly on the same triangle as the original control hair from which they were copied. Simply making a number of copies of each control hair and placing them at random points on the same triangle tends to create clumps of hair. To produce a hairstyle without clumps of hair, an interpolation technique should be used when automatically creating strands of hair using control hairs. Correctly interpolating hairs is a complex problem. Several issues are discussed below.

C.1 Determining the Control Hairs to Interpolate Between

Before a new hair can be created, the two control hairs from which it is to be interpolated need to be determined. Our explicit hair modelling technique creates a new hair by making a copy of the control hair in the centre of a triangle, and randomly placing the copy in the same triangle. Using an interpolation technique, the random point in the triangle where the new hair should be placed is determined. The first control hair is then the control hair in the centre of the triangle. The second control hair should be the control hair in the centre of the neighbouring triangle closest to the random point. To find the closest neighbouring triangle, the shortest distances between the random point and the three sides of the triangle should be calculated. Figure C.1 shows a random point p in a triangle consisting of vertices t_1 , t_2 , and t_3 , and the shortest distances d_1 , d_2 and d_3 between the random point and the three sides of the triangle.

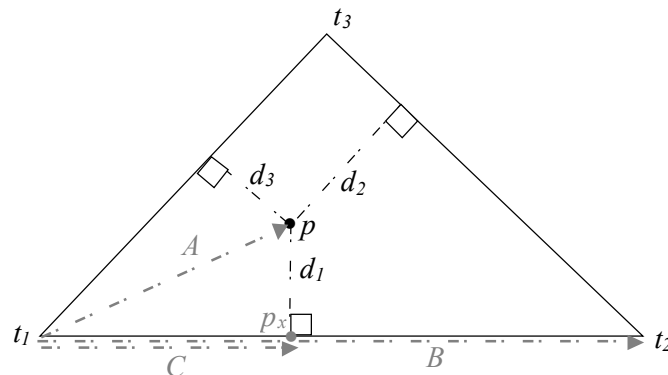


Figure C.1: Calculating the distances between a random point and the three sides of a triangle.

The distance d_1 can be calculated using:

$$d_1 = |p - p_x|$$

The point p_x is calculated as:

$$p_x = t_1 + C$$

where

$$C = ((B \cdot A) / |B|^2) B,$$

$$A = p - t_1, \text{ and}$$

$$B = t_2 - t_1.$$

The distances d_2 and d_3 can be calculated in a similar manner.

The closest neighbouring triangle is the triangle that shares the side closest to the random point. In Figure C.1 the distance d_3 is the shortest distance of the distances d_1 , d_2 , and d_3 . Therefore the side determined by vertices t_1 and t_3 is the closest side, and the control hair in the neighbouring triangle that shares the side determined by t_1 and t_3 is used.

There may not be a control hair in the closest neighbouring triangle. This occurs when creating strands of hair near the edges of a hairstyle. One solution is to interpolate the new hair from a single control hair (i.e. make a copy of the first control hair, using the same method as our explicit hair modelling technique).

C.2 Interpolating a Hair Between Two Control Hairs

Once two control hairs have been determined, a new hair can be interpolated using the two control hairs. The points along the new hair should be calculated as a weighted average of the points along the two control hairs using the equation:

$$C_1 W_2 + C_2 W_1$$

where C_1 and C_2 are corresponding points on the first and second control hairs respectively. W_1 and W_2 are calculated using:

$$W_1 = n_1 / (n_1 + n_2) \text{ and } W_2 = n_2 / (n_1 + n_2)$$

where n_1 and n_2 are the distances between the random point and the first and second control hairs respectively.

A method is needed to determine the corresponding points on the two control hairs. If the control hairs have the same number of control points, the control points could be used. However, the

control points are not usually evenly spaced along the control hairs, and the control hairs may not have the same number of control points. It may be useful to parameterise the control hairs by arc length first, and give them the same number of control points. Problems may arise when the two control hairs bend in different directions. For instance an interpolated hair will pass through the centre of the head when interpolated from two control hairs on opposite sides of a parting. The length of an interpolated hair may also be too short or an unrealistic shape, as shown in Figure C.2. There may also be places in a hairstyle where interpolation is not wanted, for example in a hairstyle consisting of straight long hair and a fringe, interpolation is not wanted between the short hair in the fringe and the long hair. The fringe is a clump of short hair separate from the long hair, and interpolation would result in strands of hair varying in length between the fringe and the long hair. The problems discussed show that interpolating strands of hair is a complex problem, and an investigation that is beyond the scope of this thesis is needed in order to interpolate between two control hairs in a realistic manner.

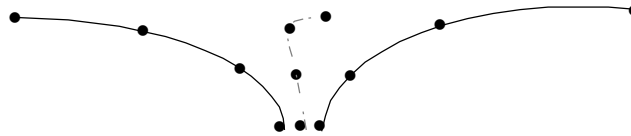


Figure C.2: *An unrealistic interpolated strand of hair.*