REAL-TIME FUR MODELING WITH SIMULATION OF PHYSICAL EFFECTS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Sinan Arıyürek

September, 2012

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Bülent Özgüç (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Uğur Güdükbay

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Veysi İşler

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

REAL-TIME FUR MODELING WITH SIMULATION OF PHYSICAL EFFECTS

Sinan Arıyürek
M.S. in Computer Engineering
Supervisor: Prof. Dr. Bülent Özgüç
September, 2012

Fur is one of the important visual aspects of animals and it is quite challenging to model it in computer graphics. This is due to rendering and animating high amounts of geometry taking excessive time in our personal computers. Thus in computer games most of the animals are without fur or covered with a single layer of texture. But these current methods do not provide the reality and even if the rendering in the game is realistic the fur is omitted.

There have been several models to render a fur, but the methods that incorporate rendering are not in real-time, on the other hand most of the real-time methods omit many of the natural aspects , such as; texture lighting, shadow and animation. Thus the outcome is not sufficient for realistic gaming experience.

In this thesis we propose a real-time fur represantation that can be used on 3D objects. Moreover, we demonstrate how to; render, animate and burn this real-time fur.

Keywords: Real-time, Fur, Rendering, Animation, Burning.

ÖZET

GERÇEK ZAMANLI HAYVAN KÜRKÜ MODELLEMESİNİN FİZİKSEL EFEKTLERLE SİMÜLASYONU

Sinan Arıyürek
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Bülent Özgüç
Eylül, 2012

Kürk, hayvanların önemli görsel unsurlarından biridir ve bilgisayar grafiklerinde bunu modellemek oldukça zordur. Bunun en büyük nedeni ise, kürkün çok fazla geometrik tüyden oluşuyor olması ve kişisel bilgisayarlarımızda bu kadar fazla geometrik öğeyi gerçek zamanda hem imgeye dönüştürmek, hem de hareket ettirmemizin zor olmasıdır. Bundan dolayı güncel bilgisayar oyunlarında, hayvanların çok basit bir kürk yapısı vardır ama bu mevcut yöntemler yeterli gerçekliği sağlamamaktadır.

Araştırmacılar hayvan kürkünü modellemek için bir çok farklı yöntem sunmuş olmasına rağmen, çoğu tüyün doğal özelliklerini yansıtmayı başaramamışlardır. Bunlara tüyün aydınlatılması, gölgelendirmesi, ve animasyonu örnek olarak gösterilebilir. Bu yüzden ortaya çıkan sonuç gerçekçi bir oyun deneyimi için yeterli olmamıştır.

Bu tezde 3B nesneler üzerinde kullanılabilir gerçek zamanlı bir kürk modelleme yöntemi gösterilmektedir. Ayrıca tezimizde bu yönteme aydınlatmanın, gölgelendirmenin, animasyonun ve hatta yakmanın nasıl entegre edilebileceğini gösteriyoruz.

Anahtar sözcükler: Gerçek Zamanlı, Animasyon, Hayvan Kürkü, Yanma.

Acknowledgement

I would like to thank my family, specially my sister Cemre for supporting and helping me throughout my thesis, and my cat, Nala, for giving me the idea to model a fur. Also I would like to thank to my advisor for his patience, support and advices throughout my time, both as a student and during my thesis.

This thesis could not be completed without the "help" of my friends, their help and support made me to continue on this path of research and science. Furthermore, some of them indeed tried to put me in the right path and showed me the way. Thanks to them my journey has become enjoyable, and also I would like to thank them for tolerating me during this time.

Finally, I would like to thank my school and my department for providing a nice research environment and giving me the opportunity to do my masters.

Contents

1 Introduction **1**

2 Background **4**

2.1 Fur Models 4

2.2 Rendering Techniques 7

2.3 Animation Techniques 12

3 Implementation **15**

3.1 Modeling the Fur 15

3.1.1 Fur Strand Generation 17

3.1.2 Rendering Texture 19

3.1.3 Collision and Anti-aliasing 22

3.1.4 Patching to an Object 23

3.2 Rendering 24

3.2.1 Bump Map 25

3.2.2 Shadows 27

3.2.3 Lighting 29

3.2.4 Spots 31

3.3 Animation 33

3.3.1 Integration of Rendering 36

3.4 Burning Fur 37

4 Results **42**

4.1 Modelling 42

4.2 Rendering 45

4.3 Wind and Fire 47

5 Conclusion **49**

List of Figures

3.1 A fur rendered with our system 16

3.2 A strand produced by Bézier curve 17

3.3 Encapsulation of a Bézier spline 18

3.4 A fur encapsulated with our spherical cones to produce thick and thin fur. 19

3.5 A normal sloped and a higher sloped fur 19

3.6 A view of the shells from the top and from the sides 20

3.7 Inner and the outer part of a strand 21

3.8 Shells rendered with the tangent values 22

3.9 Difference between an affine (linear) texture mapping and a perspectively corrected mapping. Wikipedia, "Texture mapping," 2012. [Online; accessed 22-July-2012]. 23

3.10 A human face modeled thickened and thinned by our method . . . 24

3.11 A human leg rendered with our fur rendering method 25

3.12 A bull rendered with our fur rendering method 26

3.13 Rendering result of specularity blended with strand color and the actual specularity 30

3.14 A fur rendered with white stripes to demonstrate the fur of a cow 33

3.15 Renderings done with different spot colors 34

3.16 Excessive bending of a strand 36

3.17 Wind, bending the strands on the torus, and change in the ligthing 38

3.18 Burning effect of fur 40

3.19 Propagation of burning 41

4.1 Different fur renderings captured from varying distances 46

List of Tables

4.1 The relationship between the model resolution and frame rates . . 44

4.2 The relationship between the number of textures and frame rates 44

4.3 The relationship between the texture resolution and frame rates . 44

4.4 The specular tangents, number of shadow maps, number of spots and frame rates 46

4.5 The relationship between the distance and frame rates. Distance is the length between the virtual camera and the object 47

4.6 The relationship between the seconds passed in the simulation of fire and frame rates, all with 6 spots 48

Chapter 1

Introduction

Computer games are catching up with the visual reality. The character animations, renderings and the scenery is more realistic, also physical phenomena; such as explosions, fire, water appear more convincing. Most of these were not possible years ago, but with the drastic improvements in GPU technology, these have become feasible. Yet there are some aspects that are hindered in computer games for the sake of processing speed, measured in frames per second (fps). An example is animal fur simulations, since this contains huge amounts of geometry, animating and rendering simultaneously is not possible in real-time. Moreover, in games not only the geometry itself produces realistic objects, they have to be rendered. If a game seems like having an uninterrupted motion, it is called real-time and the game is enjoyable. But if the game lags, then the user cannot enjoy the experience.

One of the significant problems of real-time fur rendering comes from displaying and transfering huge amounts of geometry or data. Since vast majority of the animal kingdom possess fur, the animals modeled in the games should look realistic. Currently this is being done by having one layer of texture patched on to the models. Although it creates an acceptable appearance yet this needs to be improved.

Since 1980's researchers are trying to model animal fur. Many of the early

approaches cannot execute real-time, but since the games need to be further evolved for realism, researchers are trying to improve these methods. The very first method has been introduced by Kajiya and Kay [1]. Their volumetric modeling of the fur and rendering this volume by ray tracing has produced an excellent furry object and constructed a solid base for the ongoing research. During that time real-time fur was not quite possible due to the hardware restrictions. Later with the improvements in texture processing applications, researchers have tried to model the fur by using textures. They put these textures atop of each other so that they look like 3D object, when viewed from a distance. However most of these methods have not emphasized in rendering of the fur. Sheppard [2] rendered fur in real-time by using bump map textures. Furthermore, to increase the reality of the fur textures, implementing a fast shadow texture has been researched. Animation of fur has not been researched at all, but animation of grass by [3] provides information about animating textures in vertex shaders. This demonstrates that textures can be animated without geometry. Nevertheless, the effects of physical phenomena such as burning of real-time fur animation has not been researched at all. The most recent development was NVIDIA's rendering of a geometric fur in GPU [4].

Such animation systems must be adaptive so that they can be easily modified. This is important in order to cope with the current technological achievements and needs of the gamers. In this thesis we have focused on modelling real-time fur animation by rendering furs by textures rather than geometry or volume, since real-time geometric modelling requires expensive hardware. But using textures could limit such expanditures. To test the limits of the method we added a windforce to animate it, and a fire to see if it can be burnt. Furthermore, since fragment shader is more capable of doing pixel operations, the animation of the texture is done in fragment shader, whereas current methods use vertex shader. In addition, in our system, diffuse and specular reflections, shadows, and colorful spots, all rendering tecniques used to increase fur's reality, are moved to shaders to increase frame rates. These methods are currently implemented in CPU.

Chapter 2 presents a comprehensive investigation of the previous work done in generating and rendering fur. Different types of fur models, lighting models and

ways to animate these are explained in detail. Chapter 3 explains the proposed system, the modeling and rendering problems that we have faced and the solutions to these problems are given in detail. Furthermore, animating fur using fragment shader is demonstrated; also burning fur using texture mapping is demonstrated. In addition, generating different types of color spots and patterns on the overall fur is demonstrated. Chapter 4 presents experimental results, the performance of the system with different settings. Chapter 5 concludes the thesis with a summary of the current system and future directions for possible improvements.

Chapter 2

Background

Fur has been an improtant topic in computer graphics, in over thirty years, different fur generation techniques have been developed. Since the begining of the demands for realistic rendering quality; the application areas, and the rendering elements such as lighting, shadow have been improved by researchers.

Almost every animal possesses a fur and every animal has a different kind of fur varying in; color, markings such as spots of Dalmatians, patterns of cat fur, types of textures such as smooth, rough, curly, straight, and broken, length of hair and health of the hair coat such as shiny, dull, wet, burnt [5]. All of these differences make a complete fur method nearly impossible.

This chapter is divided into three sections. In the first section different types of fur models that have been developed are explained. The second chapter presents the rendering techniques that are used to render these different models of fur. In the last section animation of these models are demonstrated.

2.1 Fur Models

Throughout the course of computer graphics, different types of fur models are proposed, each targetting a different type of applications, one being the film

industry and the other being computer games. In this section different types of fur models are presented.

Gary Miller created fur with geometry where hair strands are pyramid like structures, and used these strands to render a furry looking caterpillar [6]. Csuri et al. used triangles to patch surfaces of primitive objects [7].

Perlin used hypertextures to model fur and many other objects [8]. Hypertexture models the object as a function of density, and it consists of two different regions; a hard unmodifiable region and a soft region which can be modifiable by a function. With these two, Perlin produced a flexible volumetric definition. There are two main functions that determine the hypertexture; first an object denisty function which fills the space, the second one is density modulation function which alters the soft region. Perlin defined several density modulation functions such as; bias, gain, noise, etc. To create fur, these are defined in the soft region to make it changable. Then the noise function is used to determine the density of the fur, and with the second function the user can change the type of the fur from curly, to straight.

Volumetric model to create a furry object is demonstrated by Kajiya and Kay in 1989 [1]. It has been the most succesful looking and pleasing fur model up to date. Their main goal was to render high complexity scene with a wide range of detail. The state of the art was rendering with geometry and adding a hierarchy of scale that corresponds to the level of detail of the object. But this method due to being a geometric model leads to an intractable aliasing problem, so with the current model, the authors state that modeling of a fur with a very fine detail almost becomes impossible. To overcome this problem they state an adequate solution, named texel.

A texel is a 3-dimensional texture map in which both a surface frame - normal, tangent, and binormal- and the parameters of a lighting model are distributed freely throughout the volume. Also to make the texel look realistic, the authors fill it with a data which assembles the properties of a furry bear. Their algorithm for filling the texture for the bear is as follows. The hair strands are distributed as a Poisson disk that has a torus topology so that having a single texel does not

produce seams on the bear. Also to avoid brush like appearance they divided the fur into two seperate regions called as undercoat and uppercoat. Where undearcoat is distributed at lower levels of the texel to demonstrate dense short fur and an uppercoat to demonstrate the sparse long furs.

Neyret extended the traditional volumetric approach by encoding the volume by an octree, so the amount of storage is simplified [9]. In his representations, each voxel has a density amount of occupation and a simpler reflectance model. By doing this, Neyret achieved an efficient multiscale model. Later expanding his work, Neyret showed how to convert the usual 3D models into texels [10]. Neyret's work is further developed by Meyer and Neyret [11]. Authors represent a volume by thin slices of 2D textures mapped on each other. They acknowledge that these textures must be transparent in order to make the lower levels visible. Their main goal is to use the current hardware so they can achieve interactive speeds.

Van Gelder and Wilhelms model the fur by creating NURBS curves [12]. Also to give the fur its natural look they add a stiffness factor and add gravity to the space so the fur bends. Authors note that when viewed form afar the furry animal looks like a smooth object due to the fact that the hairs blend together. To solve this problem they automatically decrease the amount of fur displayed with a function defined by distance.

Lengyel used Meyer and Neyret's idea to simulate real-time fur [13]. Meyer and Neyret's work is closer to real-time speeds so Lengyel decreased the amount of textures to be used to represent a volume and achieved a real-time fur. His method generates a geometric fur in 3D space and rather cutting thin slices in the previous work, he cuts bigger slices.

Lengyel named these fur volumes as shells. By using this technique Lengyel achieved real-time speeds, but taking bigger slices has a disadvantage. Since the layers are 2D textures, and the amount of textures that are used to represent is less, it causes view problems at angles over 45°. Lengey et al. addressed this problem in [14]. Their solution is to add a new texture type called fin. Where shell is a series of lateral slices, fin is a one vertical slice representing the view of

fur geometry at perpendicular view. Moreover, to give the fur its smooth look, authors point the importance of choosing a correct alpha value for each texture.

Papaioanou introduced a simple and fast way to produce a fur texture without the need of a geometry [15]. Papaioanou used a noise function to create the fur. He calls the textures, opacity map. Papaioanou created the opacity maps as follows: first a grayscale image is produced from noise, afterwards it is smoothed by a blurring filter and the resulting intensity of each of these pixels in the image defines the hair length. After that, Papaioanou used the height value to decide if the current opacity map at current height will be filled out. To add color to these maps, an additional texture is associated.

NVIDIA [16] tested the current capabilities of their GPUs by creating 27,448 splines by geometry shader, and patching onto a surface and made it look like fur. Angelidis also used geometry to create animal fur [17]. To create a realistic fur rather than creating a line for a hair, Angelidis created a series of connected ellipsodial segments.

McGuire used displacement maps to simulate a furry object [18]. McGuire acknowledges that this method can indeed render a short but fast fur. In 2010, a work done by Pouli et al. has demonstrated the feasibility of rendering fur into images directly [19]. In their work they estimate the local depth and lighting information creating a 2.5D world from that image. Later they render the fur to this 2.5D world to recreate the final image.

2.2 Rendering Techniques

Rendering is the process of creating a final realistically viewable 2D image from a scene with techniques, such as; lighting, reflectance and shadow maps. Depending on the area of use, it is done in real-time or non real-time. The results of these two differ a lot in quality, since in movies people want realistic scenary whereas in computer games gamers demand realism and interactivity [20].

In our fur case, we have different types of models to fill the scene, and now its time to make fur look like fur. In this section different types of rendering techniques are explained.

Kajiya and Kay's rendering of fur has revolutionized this area and has constructed a solid background for fur research [1]. To render the volume, which they named as a texel, they use ray tracing. The novelity of their work comes from their calculation of diffuse and the specular values. The diffuse component of the hair reflection model is obtained by integrating a Lambert surface model along the circumference of half cylinder facing the light source. To achieve this t, l', b are calculated, t is the vector which is perpendicular to the texel basis, l' is the projection of the light vector onto the texel plane, and b is the vector which is orthogonal to these both vectors, t and l'.

$$l' = \frac{l - (t \cdot l)t}{\|l - (t \cdot l)t\|} \tag{2.1}$$

$$b = l \times t \tag{2.2}$$

To paramatrize the normal, n, along the cylinder at degree θ they use the following function

$$n = b\cos(\theta) + l'\sin(\theta) \tag{2.3}$$

Lambertian diffuse equality is $\psi(\theta) = k_d l \cdot n$ thus to find the the total amount of light, the diffuse equation has to be integrated along the circumference of the cone.

$$\psi_{diffuse} = k_d \int_0^{\pi} l \cdot n r d\theta \tag{2.4}$$

$$= k_d \sin(t, l) \tag{2.5}$$

The resulting equation was indeed simple and proven to be useful, so they continue with the computation of the specular component. To calculate the specular

reflection, Kajiya and Kay stated that they could have delievered a specular term starting from ad hoc Phong specular model but the resulting model was way too complex thus they chose to invent one of their own. Their own specular calculation comes form the idea that light striking the hair is reflected along the tangent thus the reflected light produces a cone where angle at the apex is equal to the angle of incidince.

They used the following formula $\psi_{specular} = k_s \cos^p(e, e')$ where e is the vector pointing toward the eye and e' is the specular reflecton vector in the cone closest to the eye. And the value p is the Phong exponent specifying the sharpness of the light. This can also be computed using the angle of incidence and the angle of reflection.

$$\psi_{specular} = k_s cos^p(\theta - \theta') \tag{2.6}$$

$$= k_s \Big(t \cdot lt \cdot e + \sin(t, l) \sin(t, e) \Big)^p \tag{2.7}$$

Shadowing is done by casting another ray from a point in texel to the light and depending on a collision they darken the point. Although they have produced very good looking furry figures, their rendering took about two hours by using twelve 3090 processors and four 3081 processors, but they state that the average CPU usage was between 30% - 50%.

Banks further investigated this method as a problem of codimensionality [21], where codimension is the dimension difference between the object to be rendered and dimension of the world space. In Kajiya and Kay's problem, codimensionality is two, since a texel point is a point and the world has three dimensions. Banks also explained the physics behind Kajiya and Kay's ad hoc specular reflection calculation. Banks tried to solve the excessive brightness problem as it is acknowledged that the problem occurs due to codimensionality. If codimensionality is higher, then the probability of the normal of an object being perpendicular to the world normal is higher. It is solved by exponentiating the diffuse calculation.

Banks also attempted to solve the self-shadowing problem of fur, using the

surfaces negative normal, then calculating the dot product of the negative normal and the light. Using this value to clamp the total illumination is done by diffuse and specular lighting. Moreover, Banks added an attenuation function. The attenuation is calculated by the length of the light that travels through the fur and using this distance he attenuated the fur. Total simulation time of Banks fur took 38 seconds, the scene consisted of 128×128 torus mesh and fur consisting of 2,408,448 line segments.

Lengyel [13] stated that current hardware mode can be used to calculate Kajiya and Kay's diffuse and specular components, by choosing a half vector for specularity and the current normal vector for diffuse. On his contuing paper he also shaded the model by Kajiya and Kay's equations obtaining a very good looking real-time fur [14].

Papaioanou [15] has introduced a self shadowing for the fur, if a volumetric shell represents the fur. Since shadows darken the inner parts of the fur, the fur looks more realistic for a real-time rendering. He devised this method by stating Z-buffer maps fail to cast shadow on lower fur levels, ray traced shadows are good but slow, and there is no need for accurate shadows fur furry regions. His method is, for a given fur layer i, all layers above it will cast shadow on the current layer. So he shifted each layer above by calculating the projection of the layer above to current layer with respect to the light. Then masked the layer to black to give a shadowy appearance.

Goldman [22] took into account that in most scenes the furry animals are viewed afar, and a bundle of hairs took a space less than a pixel. Thus he represented an algorithm that he refers as fake fur rendering. Goldman also proposed a directional lighting system whereas in Kajiya and Kay's diffuse it is not addressed. To handle shadowing, he used a gradient functon to deal with shadow quickly, also this method decides the illumination factor.

Goldman's calculations are done as follows: First within a region mean hair geometry is calculated, then diffuse, specularity, shadows, and reflected luminance are calculated for this mean hair and then it is scattered for the region. His assumption is that hair parameters do not change abruptly over a surface. His

method is used in the movie called *101 Dalmatians*, Dalmatians far away are rendered with this algorithm.

Stalling et al. [23] solved Banks equations for L, T, light and tangent, respectively, instead of relying on a specially choosen and calculated normal vector due to the problem of codimensionality, Stalling wanted to express in terms of L, T. Furthermore, Stalling optimized their method by storing a texture map that stored precomputed values. So they formulized a matrix to change the texture coordinates. They calculated two texture coordinates using diffuse and specular formulas, and used these two coordinates as lookup values. They note that using a texture lookup causes a minor deviation in color since texture coordinates are interpolated linearly. In the tests concurred that use 14,200 line primitives, they can achieve 25 fps.

For real-time purposes textures are used in games. Textures immitate a fake reality of the real geometry, since textures do not have any kind of geometrical information of the geometry inside the texture. Actual diffuse and specular rendering formulas are not capable of rendering the geometry inside the texture. In order to overcome this problem the idea of bump mapping is introduced by Blinn [24], so even a fur which is only a texture can be rendered.

Bump mapping technique uses another map called normal map to store the geometric information needed to render the texture. As implied by its name the normal map stores the surface normals of a geometry. However using the normal map is not that straightforward since normal map stores the value in texture space, and the value used in diffuse and specular are from world space so a transformation of these are required. To transform a value between these to different spaces, a tangent space is used. Tangent space is a plane that has three vectors T, B, N, tangent, binormal, and normal, respectively. While tangent and binormal are on this plane normal is perpendicular to this plane and orthogonal of these two vectors. Multiplying the matrix composed of T, B, N with the vector in world space, we can transform the vector into the texture space. After all these transformation, calculating diffuse and specular values are straightforward.

In his bachelor of science thesis, Sheppard [2] used bump maps to render the

fur textures. An improvement to bump map, a technique called displacement map was introduced by Cook [25]. Displacement map changes the surface coordinate based on the value stored in the texture, but it does not have any kind of rendering information. Later this work is further expanded by McGuire [18] called Parallax Steep Mapping, which can produce displacement mapping, self-shadowing and-self occlusion. McGuire used this method to produce fur on a creature.

Marschner [26] et al. addressed the scattering of light from hair, and Zinke and Weber [27] extended the scattering problem by a better physical solution based on bidirectional fiber scattering distribution functions. Later Angelidis [17] based his lighting model on Marschner's model and applied this model to render the fur, whereas these two methods are used to render human hair, Angelidis achieved a very good looking fur as a result.

Habel [28] et al. used GPU ray tracing to render grass textures in real-time. Later Berger [29] used this technique to render fur in real-time. Furthermore, Berger added Goldman's scattering function to increase the realism.

2.3 Animation Techniques

After Kajiya and Kay [1] published their rendering of fur, Neyret [9] investigated how these texels can be animated. The animation is done by mainly controlling the vertical edges of the volumes. Moving these edges cause free form deformations (FFD) on the content. Neyret also included different types of texel deformations such as a texel representig a cloth or as a texel representing a rigid surface.

Meyer and Neyret [11] extended the animation method to animate volumetric textures which is also introduced in the paper. Volumetric texture consists of slices as textures sliced from a 3D geometry. Later, rather than animating fur with volumetric textures, animating grass has become popular since having a few amounts of slice was adequate for real-time grass, and computer games scene was demanding.

Perbet and Cani [30] represented the grass in three different level of detail (LOD). They used a 3D representation of grass when near, a volumetric texture in the middle and a 2D texture when afar. These 3 different LOD animations are done as follows; first each object has a receiver that receives the information then this information is used to animate the geometry. In the first LOD, the 3D model, each blade of the grass has a receiver. In the second LOD the vertical edges near the volumetric texture have receivers. In the last level of detail, the 2D texture, there are no receivers as authors state that they did not want to animate these. Perbet and Cani also defined several wind types and demonstrated the shift between LODs. In their tests they have reached 25, 12.5, and 8 fps on an ONYX 2 Infinite Reality [31] machine with setting low, medium, and high, respectively. They used 160, 320, 500 blades per patch, the 2.5D range is; 3-8, 2-12, 3-20, and the number of segments per blade of grass is 3, 4, 8 in low, medium, and high settings, respectively.

Bakay et al. presented a simple method for grass animation, where the grass model is based on shell approach [3]. Animation is done by moving each vector with respect to the wind vector which is stored at each vertex. Wind effect is achieved by moving every vertex along its normal and wind vector. Also authors presented a formula for calculating how much to move a vertex, based on current shell value and wind intesity. Their results show that shell numbers is inversely propotional with the fps. They achieved 6,5 fps rendering speed with 32 shells, the test is conducted on a 1.7 GHz Pentium Processor and 32 Mb NVIDIA GeForce 3.

Guerrez et al. further improved Perbet and Cani's work for on the fly grass generation and with a new function to tread on the grass [32]. They animate the grass based on generating a line segment from the previous to the current position and producing a scalar field with a finite radius of influence around the object.

Later Banisch and Wütrich introduced physical laws to animate fur and grass in real-time [33]. Their main idea is to combine shell based approach with mass spring system. The animation is done by laterally displacing the shells. Also

with the mass-spring system they show that the grass can be parted.

Balyaev et al. further improved the geometric animation model, by applying an inertial animation model [34]. Habel et al. [28] animated the grass by changing the u, v coordinates of the texture rather than moving the vertices of the texture.

Angelidis [17] used spring mass system to animate the fur. Also they handle the collision with rigid objects. When collision happens the furs are seperated to the sides, even if physically it would curl along itself.

Chapter 3

Implementation

In this section we present our proposed system, this chapter consists of 4 sections, first our model of the fur is presented, next the rendering system we propose is explained, after that the animation model that we use in our system is described, lastly our fire system is demonstrated. Figure 3.1 demonstrates the a fur rendered with our system.

3.1 Modeling the Fur

Modelling of the fur is the main step in this system, since every other technique; rendering, animation, and burning will depend on this model. So the model has to be flexible. From the previous works we have seen that the geometric methods are the ones that are easiest to implement and the most flexible of all. This is due to animation of the geometry is simpler than an animation of a volume. But due to the fact that the geometric models are not quite possible to render and animate in real-time in most of the computer configurations, we have choosen one from volumetric models. The Kajiya and Kay's texel approach is good for representing a fur volume but the model takes up so much space and also its rendering is costly. So we propose to use the shell method stated by Lengyel, since achieving interactive speeds are easier, as it is the main goal of our system.

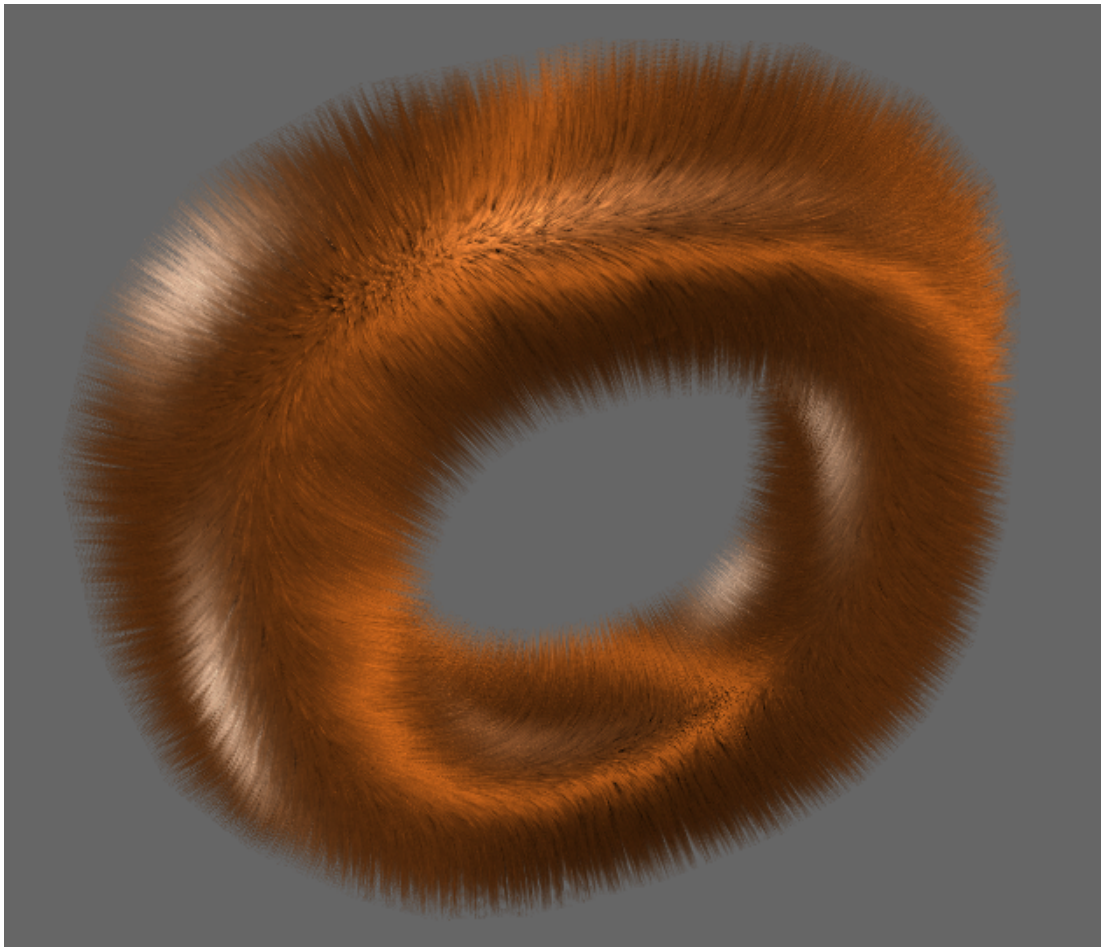

Figure 3.1: A fur rendered with our system

On the other hand, since we have lost all the geometric information, this method is not that much flexible at all (Animation and burning is not easy to integrate).

We have divided the modelling of the fur into four parts. In the first section the actual geometry is created, then the geometry is rendered to the textures. After that collision of the strands and the anti-aliasing of the textures are explained, and in the last section we explain the patching process on an object.

3.1.1 Fur Strand Generation

Before we render the fur to the shell textures it has to be created, and from our experiences most of the furs look like curved lines thus we have to create a geometry that can produce curved lines. There are lots of different spline models proposed in mathematics, but we choose to use Bézier curves to model the geometric furs that will be rendered onto textures. The Bézier splines are easier to control than most. Once a strand is generated, many strands can be generated from similar equations.

Each Bézier curve is created similarly, with two differences; height, and the angle of the curve. Actually the angle is not calculated but an angle is decided then the end and the tips of the Bézier points are created. We choose the first Bézier point close to the tip, and the second Bézier around the middle of the strand. Figure 3.5 demonstrates a higher sloped fur. The variety of the strands are done by adding randomness to the initial values of tip, end and Bézier control points. Thus each fur has become slightly different than the other. Although the system does not try to create two different types of fur layers, the height difference of the system can be changed to simulate this. Figure 3.2 demonstrates a strand produced by this method.

Figure 3.2: A strand produced by Bézier curve

After achieving a relative difference in strands, we cover a horizontal surface with these strands. Now we use a random generator to decide where the starting

points of the splines will be planted on the surface. The problem with this system is that the strands can collide with each other. Even if the geometry does not collide when the furs are rendered to the texture since they are rasterized, there can be collision problems. Also using the default random function increases the number of collisions occured, so we use Matsumoto's random generator to have a better distribution of fur strands [35]. The other problems that the collision causes will be explained later.

By using the Bézier spline we have achieved a nice looking thin fur model. Due to the fact that these are modeled by lines, the rendering equations will not work correctly. Lines are made of points thus the normals are the same everywhere. In our system we must have different normal values around the strands. To achieve this the line is encapsulated within a circular cone. Figure 3.3 shows the encapsulation of the Bézier Spline.

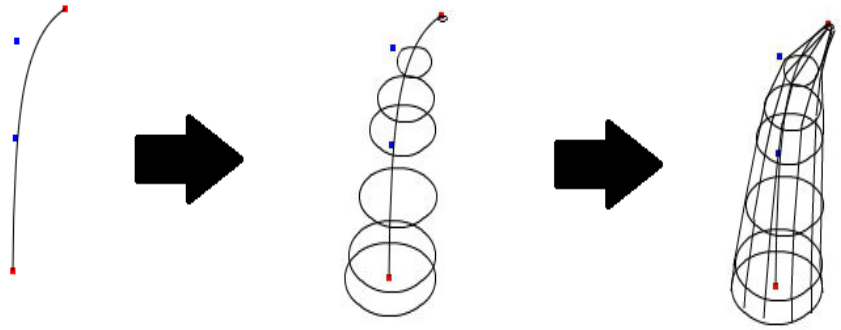

Figure 3.3: Encapsulation of a Bézier spline

The circle represents the thickness of the fur, and the tip and the bottom thicknesses are decided by the user for all fur strands. The randomization of the thickness is varied by a randomizer. Figure 3.4 demonstrates a thick and a thin fur. The circle resoultion of the strand also represents lighting resolution. When the resolution is higher there will be more vertices representing the circle, and causes more accurate tangent values for that segment. Using textures to represent the fur is advantegous in here since the resolution of the circle does not change rendering speed.

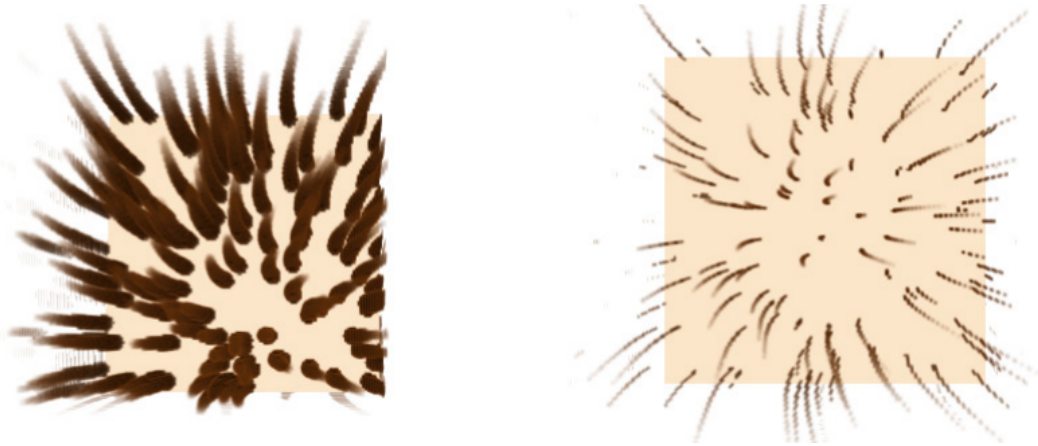

Figure 3.4: A fur encapsulated with our spherical cones to produce thick and thin fur.

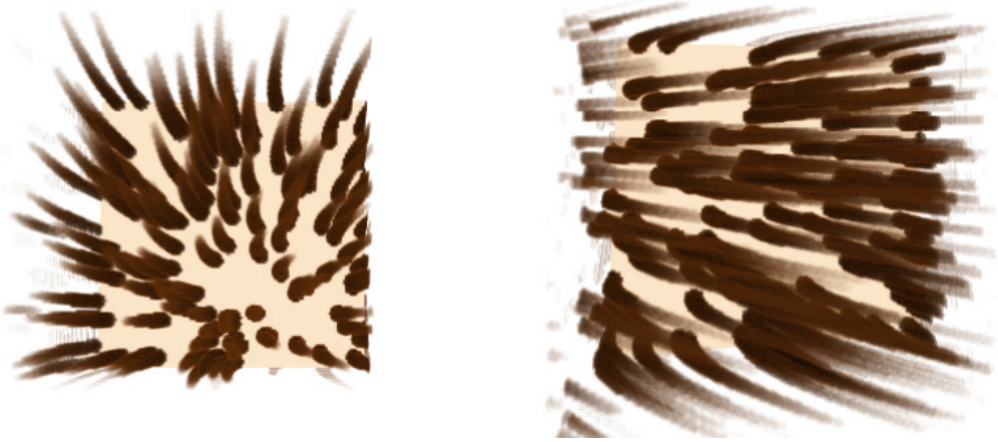

Figure 3.5: A normal sloped and a higher sloped fur

3.1.2 Rendering Texture

3.1.2.1 Shells

We now have an accurate and flexible representation of fur strand, and the next step in our method is to render these strands to the texture. Each layer of a texture should contain the representation of a strand masked between certain heights. This is done by rendering only one slice of the fur strand. The strand is already segmented, with the number of shells specified. The maximum height of the strands are divided by the number of the shells, then this number is used to

segment the strand. Then for a texture which will display the certain height, the strand is rendered for only that height.

The textures are initially created with full transparency, so the empty points in the textures will enable us to see the lower textures. Also the current rendered strand will have a value defined by the Gaussian filter, the input is the height of the current texture, this Gaussian value is also used by Lengyel et al. [14].

Figure 3.6: A view of the shells from the top and from the sides

Coloring of the textures is also important; a general color map is used for fur color variation, and a normal map texture for rendering. For each shell level, having a color and a normal map is quite expensive, this doubles the speed and the memmory requirements. Normal map is a must for the rendering to be calculated correctly. If we use the same normal map for each shell, rendering of each shell will be the same, which is incorrect, as the furs slightly bend towards top. But color map only provides a slight variation in color, which is not that much important. Moreover, when the textures are rendered a variance in color is achieved by diffuse and specular lighting.

In Kajiya and Kay's [1] and Stalling's [23] approaches, calculations of the tangent value is used to calculate the lighting, rather than the normal value, Figure 3.8 demonstrates shells rendered with the tangent values. Therefore in our system we calculate the tangent values of the strands, but when it is segmented, the inner part of the strand also becomes visible, whereas in reality we do not see this. But in our textures this is visible and has to be dealt with. Figure 3.7

shows the outer and the inner part of the fur. We propose to color them with the tangent values of the outer part of the fur. If the fur is stable this inner part will not be seen most of the time. Then rendering this are would be adding additional complexity, but due to wind factor this part can become visible. So we colored the inner parts with the corresponding outer parts color.

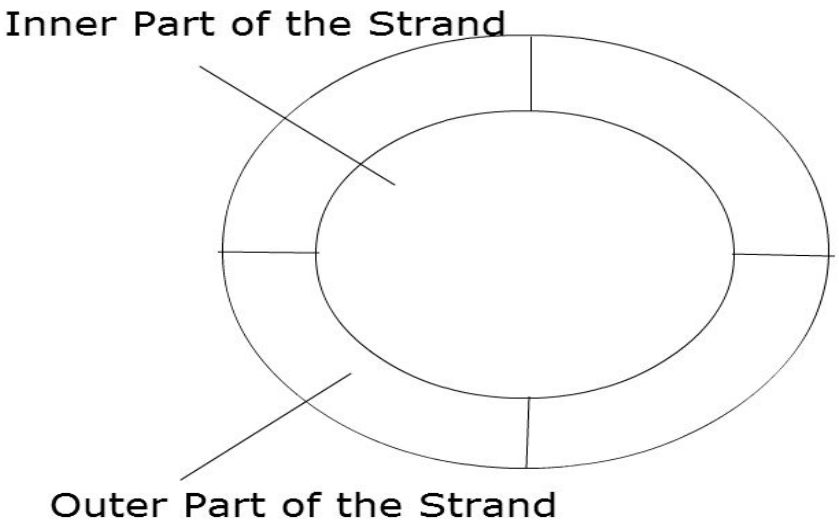

Figure 3.7: Inner and the outer part of a strand

3.1.2.2 Fins

Fin textures are devised by Lengyel to overcome the visual deffects that happen at angles greater than 45°. Figure 3.6 demonstrates the visual deffect. Fin textures are placed at the sides of the shells. To create fin textures again a geometry is needed and it is provided by our strand generator. From our observations the fin textures must be at least the height of the shell textures. Longer the strands the more furry the object looks.

The fins are only drawn at the silhouettes, since at these angles the shell method causes view problems. Thus creating the fins from the same geometry does not require to use the same geometry as in shells. In fin textures there

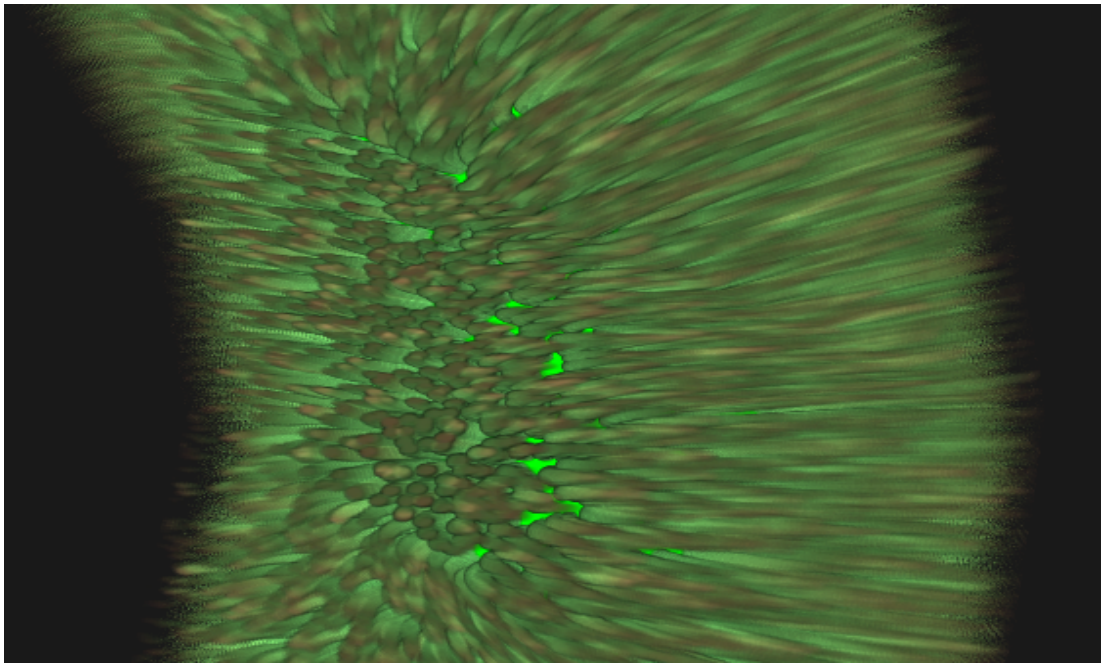

Figure 3.8: Shells rendered with the tangent values

should not be that much geometry since we are looking from sides. In our system we create fin textures using the same strand generation method but we limit the area and increase the height. We can understand the silhouttes by having the dot product of the $(eyePoint - triangleVertice), verticeNormal$. For our system if the angle of these two is between 70° and 110° or 160° and 200° we draw the fin texture. The angle values can change depending on the user.

3.1.3 Collision and Anti-aliasing

Collision causes a problem when writing to the texture. If two strands are colliding, the strand rendered later will overwrite the tangent value. But when rendered, it looks like a thicker fur, and most of the time it is not recognizable. So in our system we do not check collisions. One might want to check for collisions, but if one wants to simulate a denser fur without collision, it is not possible.

To achieve a better look in textures we use anti-aliasing, first the textures resolution is increased then the approximated values are passed to the texture

with the smaller resolution. Also anti-aliasing algorithm can cause collisions.

Using a collision check and anti-aliasing increases the amount of precomputation, and they do not change the view of the fur that much thus these can be negligible.

3.1.4 Patching to an Object

Upto now we have only created a fur that represents a cube, to achieve the furry feeling we have to patch these shells onto an object. If the object is properly mapped for texturing, mapping of the shell textures are easy, but if they are not, visual deffects occur. An example to this can be seen in Figure 3.9 [36]. In our system we did not add an algorithm to handle this. Although it is a must to patch arbitrary objects, it is not in our thesis' scope.

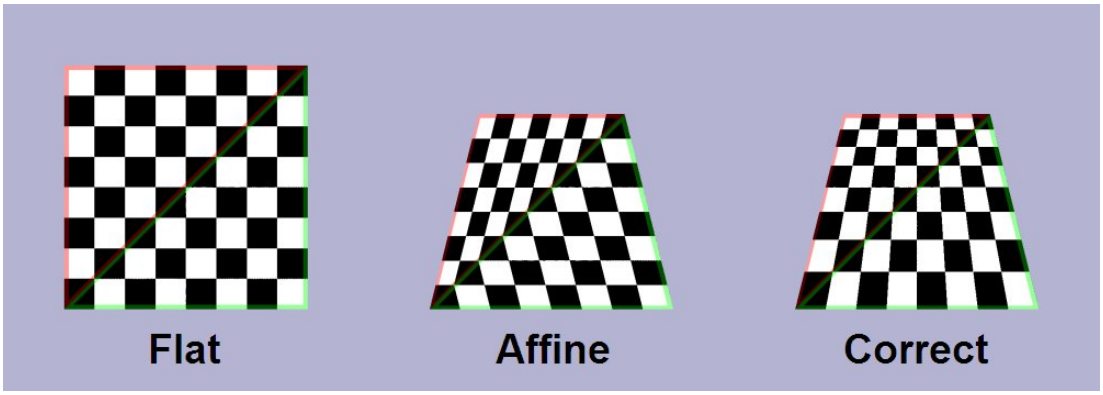

Figure 3.9: Difference between an affine (linear) texture mapping and a perspectively corrected mapping. Wikipedia, "Texture mapping," 2012. [Online; accessed 22-July-2012].

For patching, we have developed a method that extrudes the vertice normals of a given object. So for every given object we created an object loader. This calculates the surface normals, then uses these to calculate the vertex normals. By using this vertex normals we create a surfaces above the current surface. We are doing this to provide a surface to patch each texture. The reason that we are using vertex normals rather than surface normals is to prevent some of the surface collisions and not to leave spaces between these newly created surfaces.

In other words, this method creates an object looking thicker or thinner than the original, and encapsulates the original object. Figure 3.10 thickening and thinning is demonstrated. The object is thickened by a user defined value. It signifies the difference between the original and the thickest creation. By other means, it thickens or thins the object. This algorithm might create collisions, and to avoid this we need to calculate the vertice normal for a greater region. The patching of the textures have to be done layer by layer in order to avoid Z-buffer alpha problems.

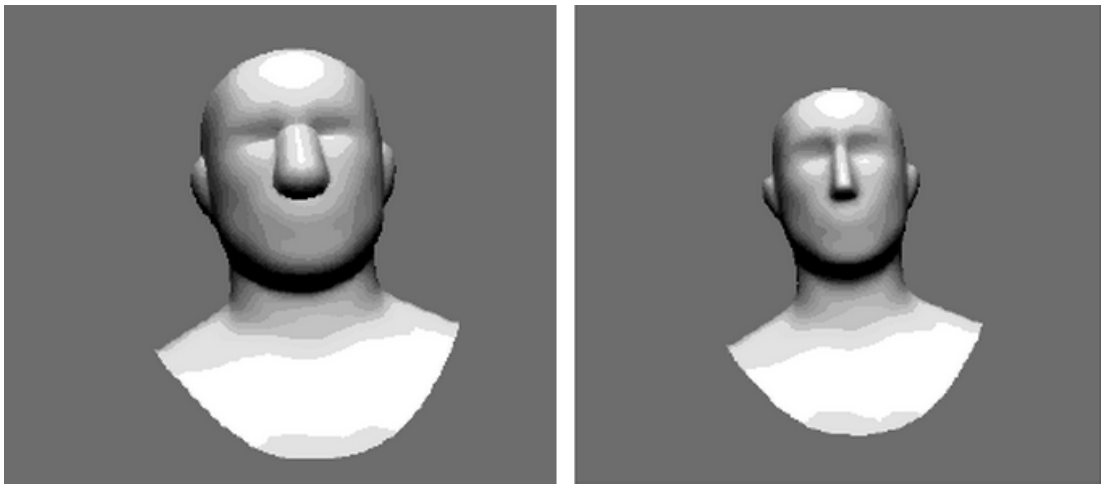

Figure 3.10: A human face modeled thickened and thinned by our method

If we are using less amount of textures, we intensify the placing on the top, and increase the spacing in the bottom, the reason for that is, we do not have any fin textures. The main objective figure of our thesis has become the torus since extruding this object does not create collisions. Figures 3.11 and 3.12 demonstrate the results that we obtained on a human leg and a bull.

3.2 Rendering

In this section we present the algorithm for rendering the fur model that we have defined in the previous section. This section is divided into 4 subsections; in the first section the tangent space calculations for the bump map is given, next

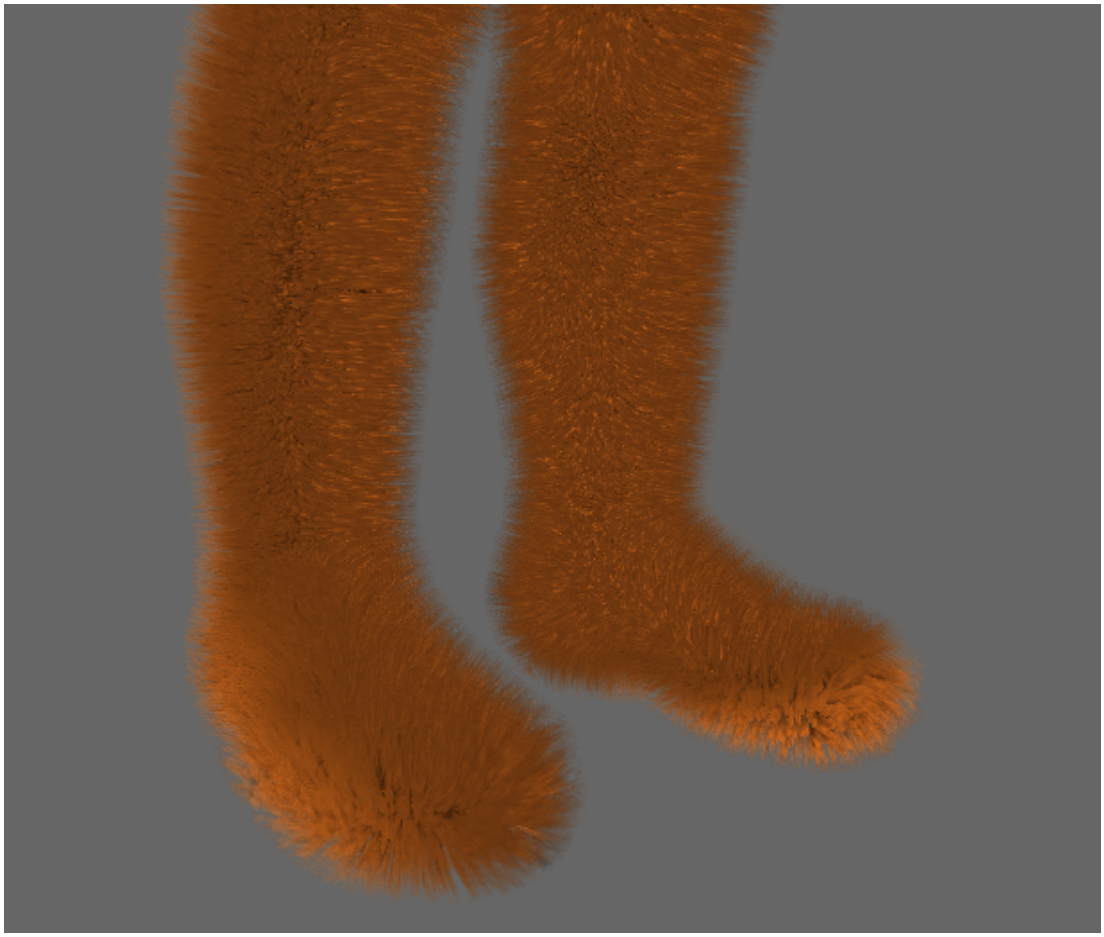

Figure 3.11: A human leg rendered with our fur rendering method

the shadow approximation is presented, following with a lighting calculation to achieve a realistic fur, and lastly the addition of spots to the fur is shown.

3.2.1 Bump Map

To calculate the lighting values correctly, the tangent space must be calculated. Our object loader does this for us, after finding the surface normal, it calculates the tangent and the binormal values too. These tangent and binormal are calculated from a surface with given texture points as follows [37];

$$\overrightarrow{T} = \frac{(v_3 - v_1)(p_2 - p_1) - (v_2 - v_1)(p_3 - p_1)}{(u_2 - u_1)(v_3 - v_1) - (v_2 - v_1)(u_3 - u_1)} \tag{3.1}$$

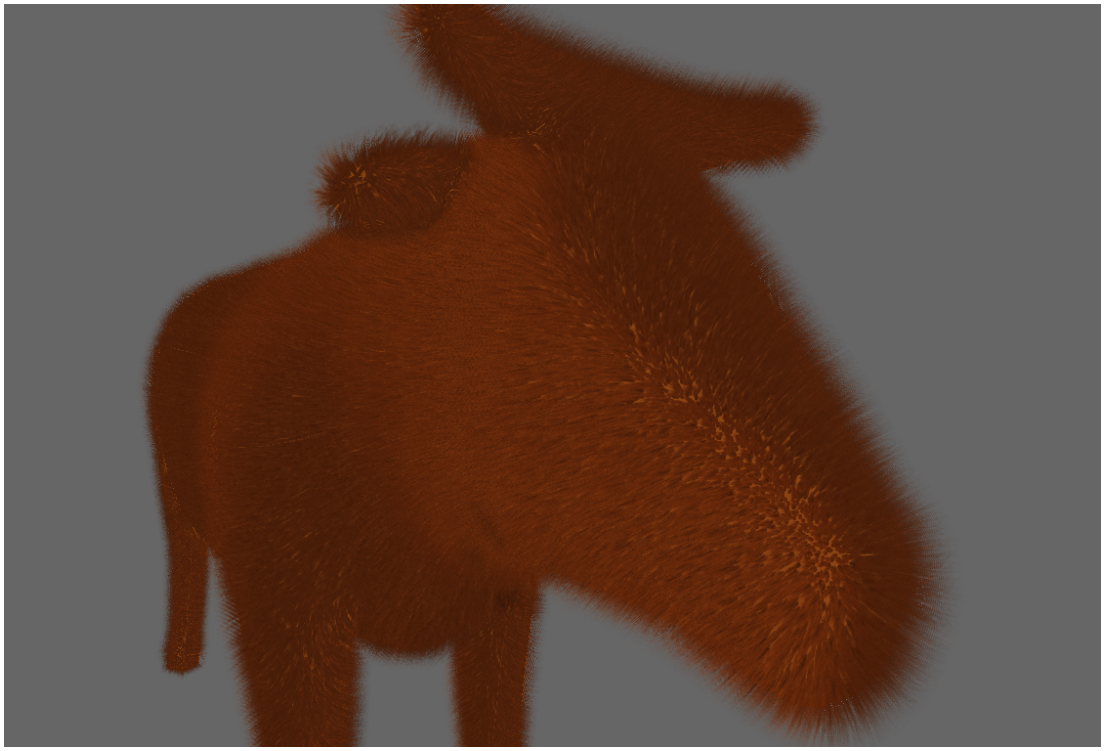

Figure 3.12: A bull rendered with our fur rendering method

$$\overrightarrow{B} = \frac{(u_3 - u_1)(p_2 - p_1) - (u_2 - u_1)(p_3 - p_1)}{(v_2 - v_1)(v_3 - v_1) - (u_2 - u_1)(v_3 - v_1)} \tag{3.2}$$

$$\overrightarrow{N} = \overrightarrow{T} \times \overrightarrow{B} \tag{3.3}$$

Also the normal values are recalculated just to make sure the original is correct. In the Equations 3.1 and 3.2, p represents the vertices of a surface, and the subscript represent the vertex number. The u, v represents the texture coordinates of a vertex, and the subscript again represents the vertex number.

We transform the eye and the light vector by multiplying them with the T, B, N matrix. After these operations, eye and the light vector are now in the tangent space, same as the tangent values of the strands that we stored in the textures.

3.2.2 Shadows

The fur closer to the base mesh should have a darker color due to the fact that the light cannot penetrate that much deep and we cannot achieve this with the current diffuse and specular calculations. Also strands cast shadow in reality. So we have to model a shadow that can achieve both darkening of the lower parts and to cast shadows.

For fur rendering with textures, two different shadows have been proposed. One is by Papaioannou [15] and the other is by Banks [21]. The algorithm devised by Papaioannou is a more realistic model but the latter model done by Banks' is faster but an ad hoc model. If we use Papaioannou's model we have to check the textures above for occlusion. However, in the Banks' method, it will just darken the area closer to the base, which is fast but not suitable for animation, since every strand in a shell texture will be darkened by the same amount. Thus we chose to use a model based on Papaioannou's method.

One main difference between Papaioannou's and ours is that we do not have the knowledge of all the shell textures. This can be done, but as our goal is to mimic reality thus so we do not need this. We use the current shell texture to cast shadow on itself. To readers note, this will not compute correctly for curly strands. However in our case the furs are slightly bent, and the bending occurs after the half height of the shells. Since we are doing pixel operations in fragment shaders, an algorithm is devised accordingly.

A vector is calculated from the current point to the light vector, then looking at the differences in vectoral changes in x, y, z, we understand how much the surface has to be translated, we named this vector as *shadowChange* vector. However, the texture map we are using has two dimensions and thus we have to transform these values. Transformation is simply done by looking into the tangent, and binormal values. Tangent values give us the knowledge about the alignment of the texture's u coordinate and binormal gives us the aligment about the v coordinate. Our analogy of these two are as follows;

- If we move along the u direction in texture space, in world space we will be moving along the tangent vector.
- If we move along the v direction in texture space, in world space we will be moving along the binormal vector.
- To move along only in the x axis we use $au + bv$, where a, b are carefully choosen to make the resulting vectors y component zero. So if we move in the texture space $au + bv$ amount we will only move in x-z plane in world space.
- To move along only in the y axis we use $cu + dv$, where c, d are carefully choosen to make the resulting vectors x component zero. So if we move in the texture space $cu + dv$ amount we will only move in y-z plane in world space.

So the transformation is done by removing the z from the equation, and the changes in x, y are normalized using tangent, and binormal values. However, the x and y values of *shadowChange* vector can be 0 and the z can be a value other than 0. This will make the method above to calculate no difference at all. Thus we modified this by removing the lowest amount of change in the dimension x, y, z. Then use the u and v to calculate the look up value by using the other two dimensions.

After calculating the amount in u, v, let this vector be *shadowLookUp*, we add it to the current texture location that we are at and check if the according pixel is transparent or not.

- If it is transparent;
  - We do not shade the current pixel.
- Else;
  - shadowValue is increased.

In our system we redo the above algorithm depending on the closeness to the bottom. We use a user defined value n, then use this to recalculate the shadow value. We divide the *(numberOfTotalShell - currentShell)* with n, let this number be m. This is the amount that we will use to recalcuate the shadow. We recalculate the shadow as, we multiply the *shadowLookUp* by a fraction of m, and repeat this process m times. So by using this method, the bottom of the fur is colored darker and a better approximation is achieved. Also due to the wind force we animate the shadows, but this is presented in the animation section.

3.2.3 Lighting

The very well know formula for lighting in real-time rendering is,

$$\psi_{finalColor} = \psi_{ambient} + strandColor \times \psi_{diffuse} + \psi_{specularity} \tag{3.4}$$

We will explain how we modified this equation in our proposed model. We use Kajiya and Kay's [1] equations to render the shells we have defined. Although this equations have been further modified by many researchers, the original equations fit our system better. The equations can be found in the Chapter 2 (Equations 2.5 and 2.7). As stated by Banks [21], the diffuse causes an excess lighting thus has to be dampened. In our implementation, the dampening is done by exponentiating the diffuse value by 4.8 as stated by Banks, but when we have added specularity to this dampened model, the specularity lits the fur a little too much. When we increase the exponent of the specular, now the fur looks dull, and the specularity hits a very small region.

To fix this problem we change our system towards a hair rendering method devised by Scheuermann [38]. So we change our calculation to;

$$\psi_{finalColor} = \psi_{ambient} + strandColor(\psi_{diffuse} + \psi_{specularitynew}) + (\psi_{specularity1})^n \tag{3.5}$$

Figure 3.13: Rendering result of specularity blended with strand color and the actual specularity

The difference is that; the specular value is also multiplied by strandColor, and another specular term is introduced. The new specular term is the sum of the three specular values, having three different tangent values. One of these is the actual tangent value from the strand, and the other two are created from changing the tangent values. The new two tangent values are the altered values of the actual tangent along the normal, back and forth.

By using these two tangent values the specular distribution becomes more uniform. Figure 3.13 shows the result of this technique. On the other hand since the specular calculation is costly and the rendering is closer to the hair than the fur, so we have modified this equation into;

$$\psi_{finalColor} = \psi_{ambient} + strandColor(\psi_{diffuse} + \psi_{specularity}) + \psi_{dampenedspecularity} \tag{3.6}$$

This equation is not accurate but when rendered it looks perfect. The main problem is that we cannot achieve specular highlights when the color is darker. The reason is that the specular highlights are not looking natural. If we have made another pass in the fragment shader, we should blur the texture so it would scatter better. Therefore there will be no reason to dampen the value. The dampening is due to the fact that only some pixels in the textures are being lit. This looks like adding sparkles to the fur, also when the animation is added the situation becomes even less manageable.

Multiplying the specularity with strandColor adds a better color definition to

the fur, and the dampened specularity calculated with low exponent. Specularity is dampened to achieve a little bit of scattering. For our system; the dampening is calculated by fixing the values over 0.6 to 0.3, and the rest provides a gradient form 0 to 0.3, these values are found by trying, different values on our torus. Since the value is not that high, it can blend in with the strandColor.

We also have to add the shadow calculation to this function, we simply subtract the shadow value we have calculated in the section above, but now we have to look at the maximal and the minimal values.

$$\begin{aligned}\psi_{finalColor} =& \psi_{ambient} + strandColor(\psi_{diffuse} + \psi_{specularity}) \\ &+ \psi_{dampenedspecularity} - \psi_{shadow}\end{aligned} \tag{3.7}$$

Ambient, diffuse, specularity all ranges from [0,1], the dampened specularity ranges from [0, 0.3], and the shadow ranges from [0, -5]. If the fur color is black since the specularity is dampened we cannot achieve a white color, and if the fur color is close to white an excess specularity can be seen. Also for a dense fur, shadow can color the surface region full black. As these might create undesirable results, we multiply each of these by a user defined value.

3.2.4 Spots

Spots add a variety to the fur, for example; Dalmatians are known from their black spots over their white fur. The main way to produce a spot is to texture the object with a predefined texture file. This is the easiest way to add a spot to the fur, designed by a modeller in a modelling program. There is nothing interesting in this so we propose two different algorithms to cover the object with spots.

We can achieve this effect by two different methods; we can define a geometric function where the fur in that area will be colored, or we can calculate the spots as a lighting component where the area that the light hits will be colored. The

first method is easier to implement since we only add a geometric function such as an ellipsoid function. After setting the parameters of the ellipsoid, we check if the point lays within the ellipsoid. The problem of this method is that we have to give different ellipsodial functions.

The latter method is to use a light to produce spots on the object. Although this method is not very deterministic, we do not know what kind of spot it will form on the object, and this indeterminacy can be the type of spot we want to achieve. We can use the light to perform specular lighting on the object, but rather than coloring them like in specular lighting, we can color them as a different color. The light can be fixed to the object so if the object moves the spot will still be there. In our system, we use the specular value to add spots and stripes to the objects.

Moreover, with the lighting method we can precompute the specular values and save it to the texture and perform simple lookups, whereas it is not that easy to do this in geometric method. The coloring of these two methods are similar. In our system, we have defined the colors of the spots accordingly to the color of the fur. For our brown fur we have added light brown, white and a yellowish brown to achieve a difference. We used white stripes over the brown fur to make the fur on the torus look like a fur on a cow, Figure 3.14 shows the result. Each different light can cause different spots, but adding a new spot decreases the speed. The coloring is done by setting an inner circle and an outer circle. In the inner circle the full color of the predefined spot is given, and in the outer region we add a gradient function to slowly change the color from the spot and to the color of the fur. Then this calculated color is inserted to the Function 3.7 as $strandColor$. Figure 3.15 shows the results of renderings done with different spot colors on torus.

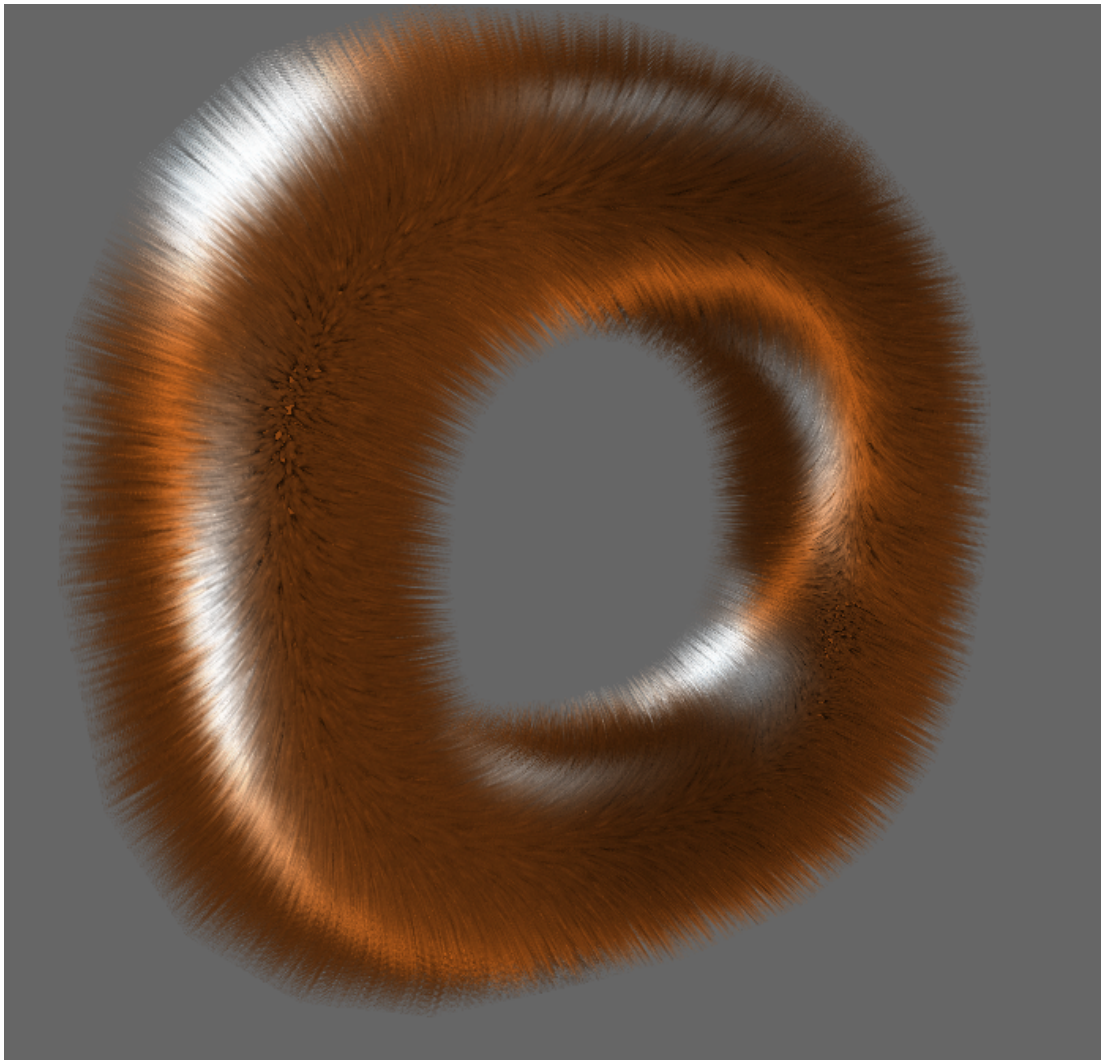

Figure 3.14: A fur rendered with white stripes to demonstrate the fur of a cow

3.3 Animation

In our system, we propose an animaton based on a wind model. Our system does not handle animaton due to collision from other objects, but if the objects have a geometric model it can be integrated.

To simulate the wind, we should be able to bend the fur strands. If we have had geometry simulating the bending behaviour this would have been easy, but since we have designed the model on shell method it will be ad hoc. The main goal is to simulate a bending of these textures.

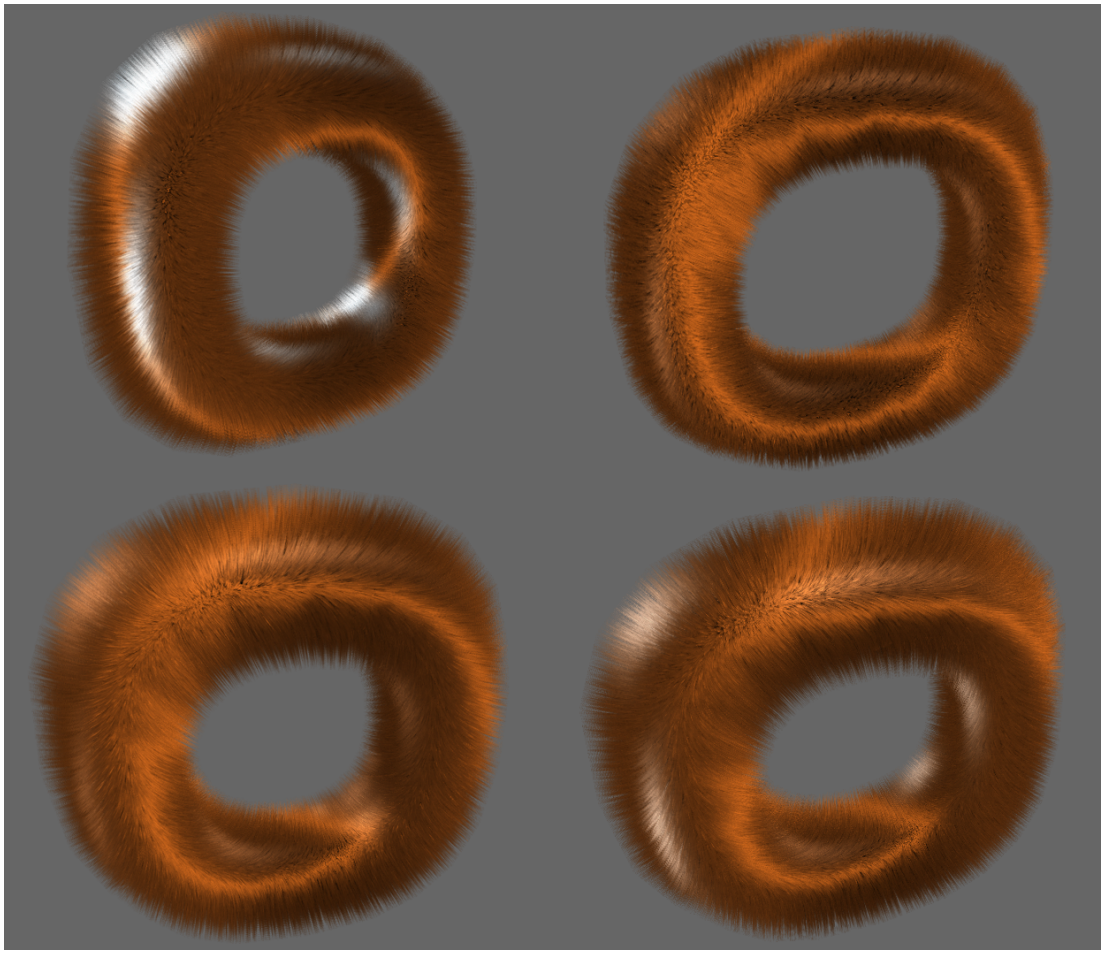

Figure 3.15: Renderings done with different spot colors

In an actual wind model, wind will apply force to the strand, and will bend the fur. Then due to elasticity of the fur, it will try to return to its original state. This will cause a vibration in the fur since when pushed by the wind, the strand will try to go to the opposite direction. So in our model, we should have two different forces acting upon a single fur; one pushing it and the other producing the vibration. Furthermore, the force that elasticity applies increases as the fur bends.

Since our system does most of the animation in the fragment shader, too much bending of the fur creates a messy effect, therefore we have put limits on the bending amount. However each texture will react differently to a wind force. So we need to know how much does each texture will move and store this value

or we need to give different wind directions at each time allowing the fur to go back and forth.

If we do the animation solely in the fragment shader, the problem in Figure 3.16 will occur. In order to solve this, we should also move the textures in the direction of the negative normal of the surface, so the length of the fur will seem same.

We choose to animate in texture space since doing the animation by moving the vertices of the textures will make the animation look like a FFD. Animation in the texture is done in a reverse manner. As we are looking at the current pixel we have to know which pixel will be there, rather than where the current pixel will go. Finding the right direction in the texture space can be tricky, but by using the same method we have used in calculating for shadows allows us to approximate how to move the pixels of the texture in world space.

- As we need to remove an axis, we remove the axis that we moved the most in the vertex shader, where we moved the texture along the negative normal of the surface. The example below assumes that we removed the z direction.
- If we move along the u direction in texture space, in world space we will be moving along the tangent vector.
- If we move along the v direction in texture space, in world space we will be moving along the binormal vector.
- To move along in the x axis we use $au + bv$, where a, b are carefully choosen to make the resulting vectors y component zero. So, if we move in the texture space $au + bv$ amount we will only move in x-z in world space.
- To move along in the y axis we use $cu + dv$, where c, d are carefully choosen to make the resulting vectors x component zero. So, if we move in the texture space $cu + dv$ amount we will only move in y-z plane in world space.

We use a directional wind model in our system and the wind direction is passed to vertex shader. In the vertex shader the displacement amount is calculated

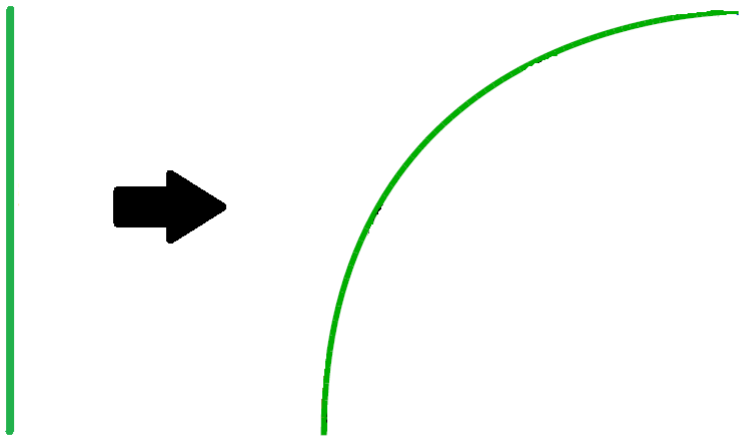

Figure 3.16: Excessive bending of a strand

by taking the dot product of the wind vector and the normal vector. However this gives us the cosine result while we are interested in the sine result, but the transformation is trivial. Using the dot products result enables us to animate the textures that are parallel to the wind. However if the texture is diagonal to the wind, the wind will sweep the furs to sides. When the texture is divided, some strands are split into two, which is unnatural. Then the sine value is multiplied with the current shell texture and this value is passed to the fragment shader.

In the fragment shader, by using this value we calculate how much should we move in the direction $x - y$ plane. Then we transform the current point with this amount to find out which pixel will come here.

3.3.1 Integration of Rendering

Since we animate the strands, the rendering methods we have mentioned before should be integrated also. In our system, we tried to achieve this, but we do not have that much data to give a physically correct integration. All we know is the tangent values of the strands, how much we will move it along the dimensions and the current geometrical location in the world space.

The rendering is done by using the tangent values so the tangent values should change with animation. Shadows also use the texture which is now animated so we will have to recalculate the shadows. The spots and stripes method does not use textures, but to animate it with the current system causes problems so it is not animated. Also burning of the fur is not affected by the animation, meaning that the fire does not spread along the wind direction.

The change in the tangent values cannot be calculated accurately since we do not have any geometry information. We update the tangent value by adding a fraction of the wind vector and normalizing the value. To update the shadow lookup value, we approximate the wind force at the shell and then we add this displacement value to the shadow's displacement, then check if the shadow will occur at the pixel. Figure 3.17 demonstrates the lighting change in the torus.

Integration of the spot is not done since when we move the light, an unexpected spot might appear; or if we change the normals, again an unexpected spot might occur on somewhere in the object. Even if we use the geometric method, unwanted spots can occur since we are not moving any geometry at all. The key to solve this problem is to use another pass in the fragment shader, where we will color the color map by the spot color and then in the next pass we will animate the texture already colored.

Although we have defined the animation, the tangent values that come to fragment shader are interpolated values so we could not animate the fur as good as we desire.

3.4 Burning Fur

In our proposed system, we also integrate the burning of the fur. To the authors knowledge, there has not been any burning of the fur. Unlike burning of wood, since fur exists on animals, thankfully there was no footage that demonstrates this. So we had to estimate this behaviour.

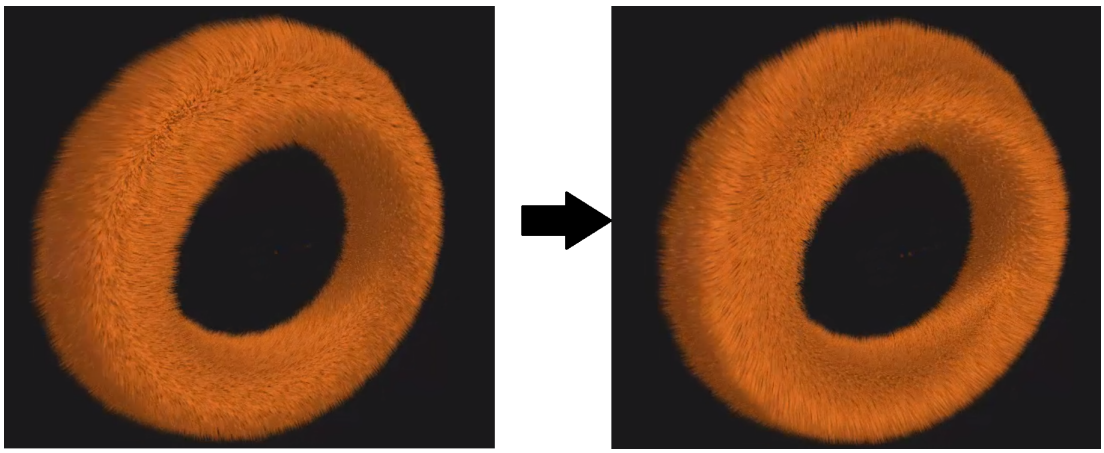

Figure 3.17: Wind, bending the strands on the torus, and change in the ligthing

Since we are not sure whether fur is a fuel source, we burned the fur with matches that are stable at certain points around the fur. Our goal is to produce a real-time fur, not a fire, thus we did not model the fire. So we had to give the appearance of the burning. Matches will instantly burn the target area, and the burnt area will start to expand in size. This effect can be achieved by using the spot method we have used, since it changes the color of an area. But fire is not just about chaning the color of the texture, we have to deform the texture as well.

Using the textures to our advantage, we solved the defomation problem. If we simply change the alpha value to make the pixel transparent, it will be seen as the fur strand is destroyed in the process of burning. But this has to be done from the tip to the bottom. Also when you burn a hair the bottom of the hair remains, thus the bottom layer should be kept, but a color difference should state the fur has been burnt. Taking all these to account we have devised our model and rendered of the burning.

We determine the regions to be burnt by setting the lights which will cause spots. We might have used the geometric representation of the spot but this is more convenient since we only have to change the specular exponent, a single data versus, three vectors. Also we can use the same exponent in the spot but we cannot use the same vectors for each spot, causing three vectors per spot. On the other hand, controlling the area of influence in the geometric method is easier,

however we choose the lighting method due the ease of control.

Next we decide on the rendering process, we dissolve the area which is closest very quickly then we would have to expand the region. This is done by changing the exponent. In the begining of the simulation it is inactive, then we change it to a higher value to only affect the region closest to the fire, later we decrease the exponent to increase the area of influence of fire. The problem of this method is due to the fact that we are using the light to expand the area which affects only the area with direct contact to the light, in other words the light will not be able to expand to the other side of the object which is opposite to the light.

After setting how to expand the burnt area, we explain how we destroy the textures and coloring of the remaining textures. We are destroying the textures closest to the initial area and expand this by checking the spot amount, and the shell texture number. The shell texture number determines how fast it will burn, and if the spot intensity is positive than it means the area is burning.

- If the intensity is greater than 0.1, we subtract the spotIntesity from 1 then multiply with the maximum shell.
  - If it is smaller than the current shell value then we set the alpha to 0, as this means that the texture is already burnt.
  - Else if it is bigger we multiply the strand color by *(1 - spotIntensity)*, as this means that the texture is burning.
- Else if the intensity is less than 0.1 we calculate the rendering using Equation 3.7.

So we have a smooth transition between the original color and the burnt color. Figure 3.18 demonstrates the burning effect that we have achieved with our system, and Figure 3.19 shows the propagation of burning.

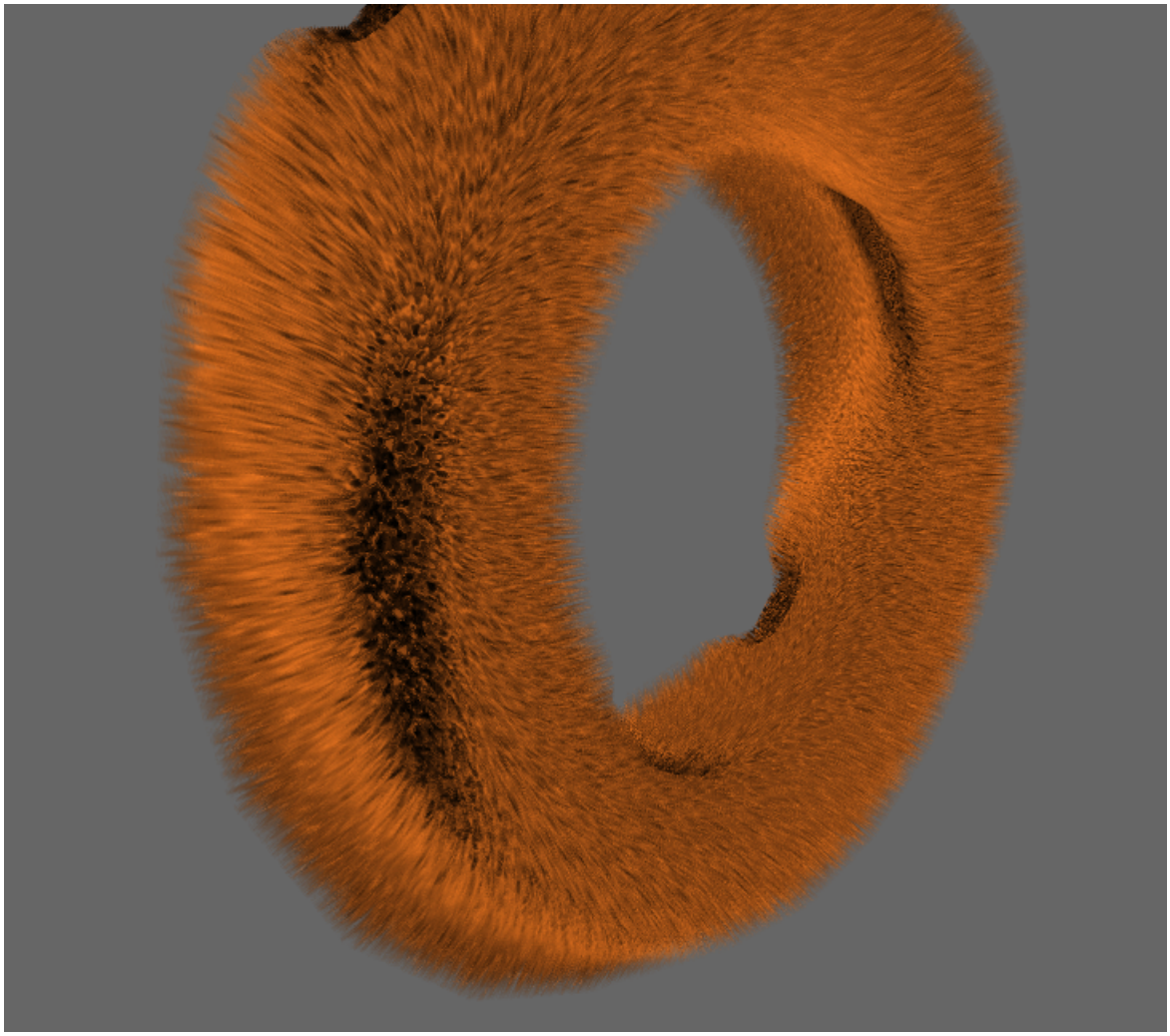

Figure 3.18: Burning effect of fur

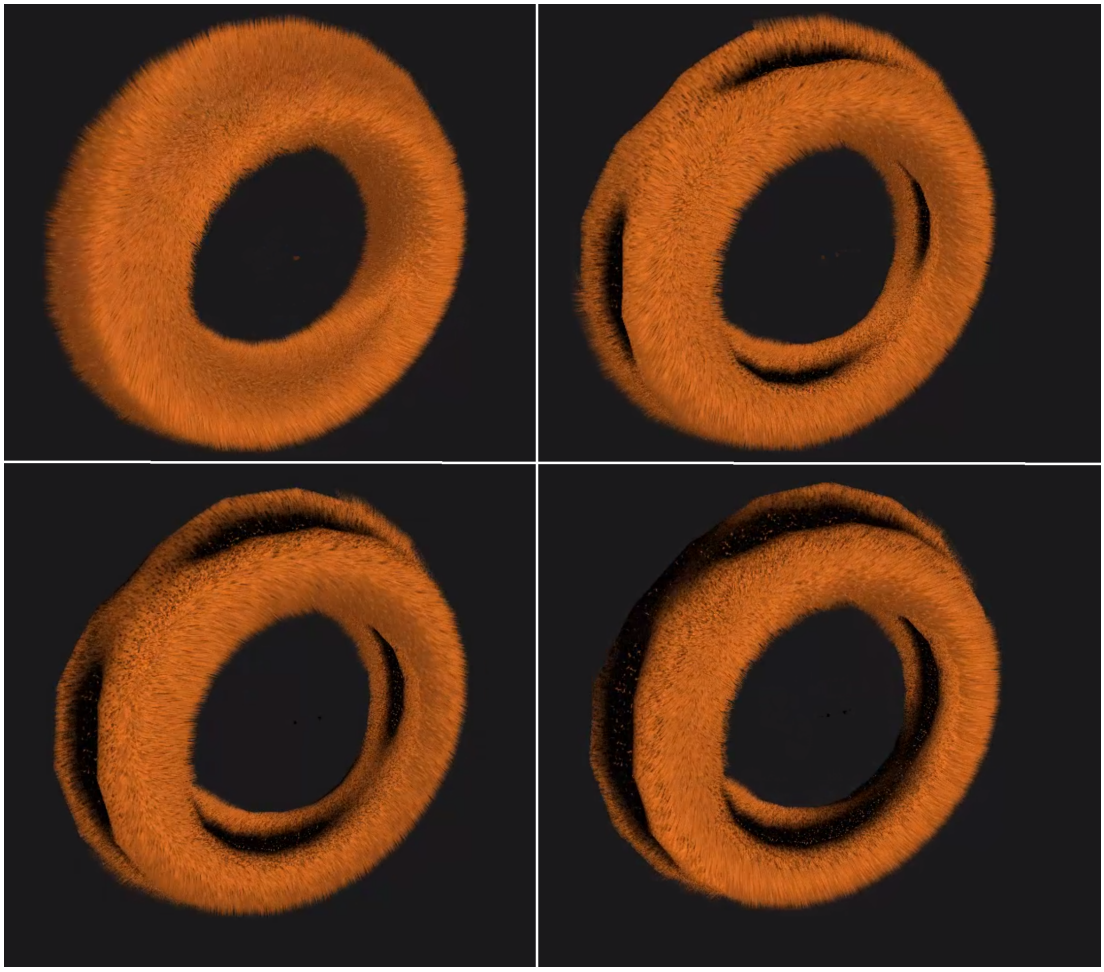

Figure 3.19: Propagation of burning

Chapter 4

Results

We implemented our system using C++, OpenGL 2.1, and NVIDIA CG 4.0. The simulation runs on a laptop that has 2.3 GHz i5-2410M, and a NVIDIA 540GTM graphics card. The resolution of the application is 1366 $\times$ 768. The simulation results are divided into three main sections, changing the modeling parameters with rendering and without rendering, changing the rendering parameters, and lastly changing the wind and the fire parameters.

4.1 Modelling

In this section, we test the weight of the modelling parameters on the system. The parameters that we can change are as follows;

- *modelResolution*, is the number of triangles on the object.
- *modelDifference*, is the total length of shell textures.
- *numberOfStrands*, is the number of strands map to a texture.
- *strandResolution*, is the number of segments we use to sample the Bezier line.

- *circleResolution*, is the number of circle segments while encapsulating the Bezier line with circles.
- *numberOfShells*, is the number of textures representing the geometry.
- *textureResolution*, is the resolution of a texture.

The experiments are done on the torus model. The frame rates are given as frames per second.

- *modelResolution* = 500.
- *modelDifference* = 1,0.
- *numberOfStrands* = 1000.
- *strandResolution* = 20.
- *circleResolution* = 10.
- *numberOfShells* = 10.
- *textureResolution* = 128×128.

These are the base values of the paramaters, since we cannot show each of them in a single table we will show the ones that we have modified. If we modeled the fur using these parameters we will have 150,000,000 triangles representing the fur. We map half of the texture to a triangle, the half of the texture has 500 strands, and we have 500 triangles representing the model. 600 triangles are used to model a strand. ($500 \times 500 \times 600 = 150{,}000{,}000$).

Table 4.1 shows that the number of triangles used to represent the model decreases the frame rate, however for a good gaming experience the furry object should not be that much detailed. As we cover the object with a texture the detail of the model also becomes unnoticable.

Increasing the number of textures allows us to create a longer fur, and decrease the inter shell distance. Since we do not have a fin texture, 10 shells do not give

Model resolution	Frame rates without rendering	Frame rates with rendering
500	>60	>60
1000	>60	60
2000	>60	50
5000	>60	45
25000	>60	29
50000	>60	18
100000	40	13

Table 4.1: The relationship between the model resolution and frame rates

Number of textures	Frame rates without rendering	Frame rates with rendering
10	>60	>60
20	60	52
30	46	40
40	36	32
50	30	27
100	19	15

Table 4.2: The relationship between the number of textures and frame rates

Texture resolution	Frame rates without rendering	Frame rates with Rendering
32×32	>60	>60
64×64	>60	60
128×128	>60	50
256×256	>60	42
512×512	53	32

Table 4.3: The relationship between the texture resolution and frame rates

us the furry look, so for our renderings we use at least 30 shells. However, as we increase the number of the textures, the fps decreases greatly we can see this effect in Table 4.2.

Texture resolution changes the space that the textures take up in the ram also the amount of data is sent to the GPU and the number of pixel operations need to be done in the fragment shader. The results can be seen in Table 4.3. With 128×128 texture the detail is actually good enough for real-time rendering, but with 256×256 texture the look of the fur becomes quite realistic.

The other variables do not change the rendering speed of our system as expected, however they increase the loading speed. With numberOfStrands = 10,000, strandResolution = 20, and circleResolution = 50 it takes 16.3 seconds to start up. In normal conditions the whole process takes 5.6 seconds.

4.2 Rendering

In the rendering section the tests we have been conducted are based on:

- *modelResolution* = 500.
- *modelDifference* =0.6.
- *numberOfStrands* = 3000.
- *strandResolution* = 20.
- *circleResolution* = 20.
- *numberOfShells* = 30.
- *textureResolution* = 128×128.

With these settings our system works at 38fps where the wind force is also active. If modeled with geometry it will have about half a billion triangles. To

Specular tangents	Number of shadow maps	Number of spots	Frame rates
1	height/5	2	38
3	height/5	2	34
1	height/3	2	29
1	height/2	2	26
1	height/1	2	14
1	height/5	4	37
1	height/5	6	36
1	height/5	12	32

Table 4.4: The specular tangents, number of shadow maps, number of spots and frame rates

readers note, the torus in Figures 3.1 and 3.14 rendered in 26 fps, with 12 spots and 40 shells.

Table 4.4 demonstrates the rendering elements' weight on the system. We can see that increasing the specular quality decreases the fps a little but not too much. Furthermore the number of shadow maps cause shadows to be better approximated, but height/5 is enough. Height/n means that if we have a total of m shells the shadow calculation for the bottom layer will use m/n lookups on the texture. As we go up to the tip m decreases down to zero. However, number of shadow maps decrease the frame rates. On the other hand, the number of spots increases the variety and the number can be increased depending on the model, and does not decrease the fps as much as shadow.

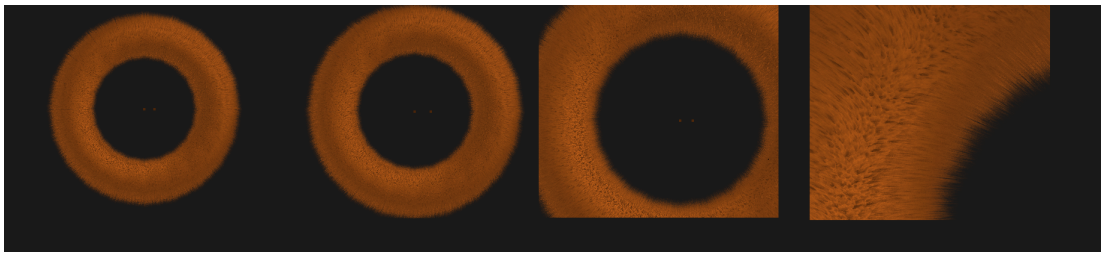

Figure 4.1: Different fur renderings captured from varying distances

Table 4.5 demonstrates the relationship between how far we are away from the fur and the fps of the application. Figure 4.1 shows how the fur looks from

Distance	128×128 texture frame rates	256×256 texture frame rates
24	>60	60
22	>60	50
17	60	34
15	52	28
12	45	22
10	36	14
7	28	10
5	18	10
2	15	10
0	>60	>60

Table 4.5: The relationship between the distance and frame rates. Distance is the length between the virtual camera and the object

these distances. From this table we can see that the closer we are to the fur the execution speed decreases abruptly. However, the frame rates does not drop below 10 fps, our guess is that the amount of textures that we see decrease but the amount of detail we see increase, so this holds the frame rates at 10 fps. The reason that the frame rates is again 60 fps at distance zero is we do not see anything at all.

4.3 Wind and Fire

In this section we demonstrate the stress caused by wind and fire on our system. Since fire is done by the spots technique the base speeds are the same, but when the textures start to destroy the rendering speed increases, since there is no texture to render. The relationship can be seen in Table 4.6. Increasing the number of spots will decrease the fps as the technique is same, but the gradual increase in the fps will be faster, since the fur will be burnt faster.

Our wind model uses only few amounts of instructions in the shader. We

Simulation duration	Frame rates
0	38
1	36
2	42
3	46
4	48
5	49
6	50
7	50
8	51
9	51
10	52
20	56
40	60

Table 4.6: The relationship between the seconds passed in the simulation of fire and frame rates, all with 6 spots

do not put a table about the rendering performance because there are not any parameters about wind. Setting the windforce on and off changes the fps by 2-4 at the most.

Chapter 5

Conclusion

Modelling a real-time fur has been an important research subject, and due rendering high amount of geometry, it has been hard to do this in real-time. Also the rendering equations are being extended and renewed since when rendering hair and fur we are close to the reality but still trying to catch up to it. Furthermore, there is more research to be done, in the animation methods of the texture based models. Moreover, the fur can be interacted with a natural phenomena, eg snow on furs.

In our thesis, we have represented a real-time fur with Kajiya and Kay's rendering functions [1], based on the model that Lengyel [13] proposed with shadow method stated by Papaioannou [15]. After all these to increase the realism we have added spots to the object, animated by the wind model that we have intutively modeled. Later we burned the fur to demonstrate the flexibility of this method.

During the implementation phase we can say everything went just as we have foreseen, but the lack of wind models and force equations mainly forced us to create our own equations. Even if there were equations tuned for animating the fur texture we would have to devise our own since the tangent values had anomalies when passed to the fragment shader. So we had to try which values can create a realistic animation in the fragment shader.

The most enjoyable part was modelling the burning. Since there was no resource to guide us through the implementation we had to devise our own methods. After that we had to fine tune our method, for our torus mesh. After spending a year doing a project, the feeling that you feel after destroying the object you have built was very enjoying.

We have done some implementation on a human face, a bull, and a human leg. However, as we could not fine tune the model and the rendering the results was not that good as our torus. After seeing that many of the projects emphasize on texturing arbitrary objects, we tried to do something new, so we tried to burn the fur. Fur on arbitrary meshes also require some work, as proper texture mapping. Since our technique depends on textures, the better they are mapped, the better they look. There are two ways to fix this problem; the first is to let the modeler to generate texture mapping in a modelling software, or implementing an algorithm that will make the texturing better.

After we have finished the rendering, animating and burning, we wanted to add a modeler tool, where the user can create a strand by modifying the Bezier control points. Then the textures will be created from this self modified fur. However, as we were not sure, whether the current rendering methods would be adequate and the current animation model would animate correctly, we had to leave this as a future work.

When integrating the animation we had lots of problems, since the rendering should differ as the strands bend. So we tried different values to calculate the change amount in the tangent values due bending, and we chose the value by checking the rendering result. Animation of shadows was straightforward as we just add the amount of displacement due to wind. On the other hand, the integration of the spots was difficult, as when we moved the light the spots were moving, but also caused the spots to appear somewhere else on the object.

Bibliography

[1] J. T. Kajiya and T. L. Kay, "Rendering fur with three dimensional textures," *ACM SIGGRAPH'89*, vol. 23, pp. 271–280, ACM, July 1989.

[2] G. Sheppard, "Real-time rendering of fur," 2004. [Online; accessed 10-July-2012].

[3] B. Bakay and W. Heidrich, "Real-time animated grass," in *Proceedings of Eurographics (short paper)*, 2002.

[4] NVIDIA, "Nvidia GeForce GTX 680 Tech Demos - Pillars and Fur 1080p," 2012. [Online; accessed 30-July-2012].

[5] Wikipedia, "Coat (animal)," 2012. [Online; accessed 22-July-2012].

[6] G. S. P. Miller, "From wire-frames to furry animals," in *Proceedings on Graphics interface '88*, (Toronto, Ontario, Canada), pp. 138–145, Canadian Information Processing Society, 1988.

[7] C. Csuri, R. Hackathorn, R. Parent, W. Carlson, and M. Howard, "Towards an interactive high visual complexity animation system," *ACM SIGGRAPH'79*, vol. 13, pp. 289–299, Aug. 1979.

[8] K. Perlin and E. M. Hoffert, "Hypertexture," *ACM SIGGRAPH'89*, vol. 23, pp. 253–262, July 1989.

[9] F. Neyret, "Animated texels," in *Eurographics Workshop on Animation and Simulation'95*, pp. 97–103, 1995.

[10] F. Neyret, "Synthesizing verdant landscapes using volumetric textures," in *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, (London, UK), pp. 215–ff., Springer-Verlag, 1996.

[11] A. Meyer and F. Neyret, "Interactive volumetric textures," in *Eurographics Rendering Workshop* (G. Drettakis and N. Max, eds.), (New York City, NY), pp. 157–168, Eurographics, Springer, July 1998.

[12] A. Van Gelder and J. Wilhelms, "An interactive fur modeling technique," in *Proceedings of the Conference on Graphics Interface'97*, (Toronto, Ontario, Canada), pp. 181–188, Canadian Information Processing Society, 1997.

[13] J. Lengyel, "Real-time fur," *Eurographics Rendering Workshop*, pp. 243–256, 2000.

[14] J. Lengyel, E. Praun, A. Finkelstein, and H. Hoppe, "Real-time fur over arbitrary surfaces," in *Proceedings of the Symposium on Interactive 3D Graphics*, I3D '01, (New York, NY, USA), pp. 227–232, ACM, 2001.

[15] Papaioannou, "A simple and fast technique for fur rendering," Technical Report, Department of Informatics, University of Athens, 2002.

[16] S. Marshall, "Converting production renderman shaders to real-time," in *GPU Gems* (R. Fernando, ed.), pp. 551–565, Addison-Wesley, 2004.

[17] A. Angelidis and B. McCane, "Fur simulation with spring continuum," *The Visual Computer: International Journal of Computer Graphics*, vol. 25, pp. 255–265, Feb. 2009.

[18] M. McGuire and M. McGuire, "Steep parallax mapping," *I3D 2005 Poster*.

[19] T. Pouli, M. Praák, P. Zemčík, D. Gutierrez, and E. Reinhard, "Technical section: Rendering fur directly into images," *Computers and Graphics*, vol. 34, pp. 612–620, Oct. 2010.

[20] Wikipedia, "3d rendering," 2012. [Online; accessed 22-July-2012].

[21] D. C. Banks, "Illumination in diverse codimensions," vol. 28 of *ACM SIGGRAPH'94*, pp. 327–334, ACM, 1994.

[22] D. B. Goldman, "Fake fur rendering," ACM SIGGRAPH'97, (New York, NY, USA), pp. 127–134, 1997.

[23] D. Stalling, M. Zöckler, and H.-C. Hege, "Fast display of illuminated field lines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, pp. 118–128, Apr. 1997.

[24] J. F. Blinn, "Simulation of wrinkled surfaces," *ACM SIGGRAPH'78*, vol. 12, pp. 286–292, Aug. 1978.

[25] R. L. Cook, "Shade trees," *ACM SIGGRAPH'84*, vol. 18, pp. 223–231, Jan. 1984.

[26] S. R. Marschner, H. W. Jensen, M. Cammarano, S. Worley, and P. Hanrahan, "Light scattering from human hair fibers," *ACM Trans. Graph.*, vol. 22, pp. 780–791, July 2003.

[27] A. Zinke and A. Weber, "Light scattering from filaments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 342–356, Mar. 2007.

[28] R. Habel, M. Wimmer, and S. Jeschke, "Instant animated grass," *Journal of WSCG*, vol. 15, pp. 123–128, Jan. 2007.

[29] M. Berger and S. P. Kmoch, "Real-time fur using gpu-based raycasting," 2010. [Online; accessed 22-July-2012].

[30] F. Perbet and M.-P. Cani, "Animating prairies in real-time," in *Proceedings of the Symposium on Interactive 3D Graphics*, I3D '01, (New York, NY, USA), pp. 103–110, ACM, 2001.

[31] Wikipedia, "Infinitereality," 2012. [Online; accessed 26-July-2012].

[32] S. Guerraz, F. Perbet, D. Raulo, F. Faure, and M.-P. Cani, "A procedural approach to animate interactive natural sceneries," in *Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)*, (Washington, DC, USA), pp. 73–79, IEEE Computer Society, 2003.

[33] S. Banisch and C. A. Wüthrich, "Making grass and fur move," 2006. [Online; accessed 25-July-2012].

[34] V. C. S. Belyaev, I. Laevskiy, "Real-time animation, collision and rendering of grassland," *Scientific Visualization*, vol. 3, no. 4, pp. 20–27, 2011.

[35] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, pp. 3–30, Jan. 1998.

[36] Wikipedia, "Texture mapping," 2012. [Online; accessed 22-July-2012].

[37] J. Jouvie, "Lesson 8 : Tangent space." [Online; accessed 22-July-2012].

[38] T. Scheuermann, "Practical real-time hair rendering and shading," in *ACM SIGGRAPH Sketches*, (New York, NY, USA), ACM Press, 2004.