151226

FUR AND HAIR MODELING

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of
Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Program.

151226

by
Hulusi BAYSAL

July, 2004
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We certify that we have read this thesis and **"FUR AND HAIR MODELING"** completed by **Hulusi BAYSAL** under supervision of **Prof. Dr. Alp KUT** and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Alp KUT

Supervisor

[illegible]	[illegible]
Doç. Dr. Yalçın ÇEBİ	Doç. Dr. Ender Y. Bulgun
(Committee Member)	(Committee Member)

Approved by the

Graduate School of Natural and Applied Sciences

[illegible]

Prof. Dr. Cahit HELVACI

Director

ACKNOWLEDGMENTS

First, I would like to thank my supervisor, Prof. Dr. Alp KUT, for his guidance and assistance in this thesis.

I also thank my friends Semih UTKU and Gıyasettin ÖZCAN for their support and motivation.

Finally, I would like to thank my father Rıza, my mother Melike and my sister Selcan for their endless support and encouragement.

ABSTRACT

Hair simulation has been always a complex process in the field of human modeling and therefore has been addressed by several researchers. The difficulties of processing hair result from the huge number of hair strands on model, thin structure of hair and complexity of hair to hair interaction. Various techniques have been proposed to overcome these difficulties in hair modeling.

Mass spring systems are commonly used to represent deformable objects such as hair, cloth and facial expressions, for real-time simulation. In this thesis, we designed a simple physics model using mass spring systems. In our model, each hair strand is generated and animated individually. The motion of hair strands is determined according to Newton's Laws of Motion.

Keywords: Hair modeling, fur modeling, physically based modeling, mass-spring systems, simulation.

ÖZET

Saç simulasyonu, bilgisayar ortamında insan modellenmesinde her zaman çok karmaşık bir işlem olarak karşımıza çıkmaktadır ve bu nedenle çok sayıda araştırmacının ilgi odağı olmuştur. Saçın modellenmesindeki zorlukların temel kaynakları model üzerinde çok sayıda saç telinin bulunması, saçın çok ince bir yapıda olması ve saçın diğer saçlarla olan karmaşık etkileşimidir. Bu zorlukların üstesinden gelmek için çeşitli saç modelleme teknikleri sunulmuştur.

Kütle-yay sistemleri, saç, kumaş ve yüz ifadesi gibi deforme olabilen nesnelerin gerçek zamanlı olarak modellenmesinde yaygın olarak kullanılan bir yöntemdir. Bu tezde, kütle-yay sistemleri kullanarak basit bir fiziksel model oluşturduk. Modelimizde, her bir saç teli ayrı olarak oluşturup hareketlendirilmektedir. Saçın tüm hareketleri Newton'un temel hareket kanunlarını esas alarak belirlenmiştir.

Anahtar Sözcükler: Saç modellemesi, kürk modellemesi, fiziksel temelli modelleme, kütle-yay sistemleri, simulasyon.

CONTENTS

Chapter Five
MASS SPRING BASED HAIR MODEL

Page

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER ONE
INTRODUCTION

Computer generated virtual human models are widely used in many areas such as computer games, advertisement clips, and movies. As a result of advances in computer graphics technology, we are able to see virtual human models, which can not be distinguished from real humans. One of the many challenging subjects in simulating these believable virtual human models has been to produce realistic looking hair.

Two decades before, first virtual humans had a polygonal hair structure which has no sense of reality. But advances in computer graphics area forced graphics developers to find new techniques in hair modeling. Realistic visual depiction of virtual human models has improved over the years. Details have become more conspicuous. It is a fact that, hair is a distinguishing characteristic of human and very important to generate realistic human models.

1.1 Hair & Fur Characteristics

Hair is a slender, threadlike fiber growing from the body. It is 97% protein, with the remaining 3% made up of amino acids, minerals and other trace elements. Hair consists mainly of keratin, which is also responsible for the elasticity of fingernails. Hair is a non-living tissue. The living tissue that makes your hair grow is hidden inside the hair follicle. The shaft, the part of a hair that you see, is made of cells that aren't living anymore. We actually have hair of various types all over our bodies. In fact, the only places we do not have some form of hair are the soles of our feet, the palm of our hands, and our lips. (WEB_1, 2003)

The hair that grows on our heads is quite possibly the most decorative feature of the human body. While it provides insulation for our heads, protecting it from the heat and cold, hair also has definite ornamental purposes and is very often used when people describe the way we look. Hair is the fastest growing tissue in the body. 35 meters of hair fiber is produced every day on the average adult scalp.

There are typically 100,000 to 150,000 individual hair strands on a human head. Geometrically, they are long thin curved cylinders (Figure 1.1) having varying thickness. A single hair has a thickness of 0.02 - 0.04 mm, so that 20 - 50 hair strands next to each other make one millimeter. The strands of hair can have any degree of waviness from straight to curly. The hair color can change from white to grey, red to brown, due to the pigmentation. The detailing of these properties is useful when modeling and rendering the hair, as use of parameters derived from real hair will enhance the realism of the model.

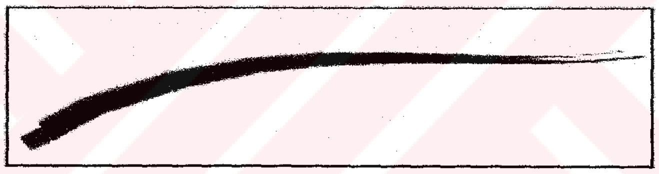


Figure 1.1 Close-Up view of a single hair strand

At first glance, fur appears to be having same properties with hair. Sometimes, the terms fur and hair are used interchangeably. But, there are significant differences between hair and fur.

Relatively Short: Compared to human hair, animal fur is shorter. In general, fur is introduced as short hair.

Covers Most of the Body: Fur covers the most of the animal bodies, however, hair is only located on the scalp of human. Therefore, it is clear that fur is applied over a bigger area and it is more difficult and expensive to model.

Different Types: Animals can have different types of fur such as wavy and smooth on different parts of the body. On the other hand, human has a single type hair model, wavy or smooth.

Vary In Color: Animal fur can vary in color over the body. But, generally, human hair color is identical. For example, tiger or zebra fur has a striped structure with different colors.

Fur is very essential for creating realistic animals and relatively short in relationship to the size of the body, so it is the skin layer that defines what the shape of the animal is. Therefore, fur does not always need dynamics or any type of automatic movement.

1.2 Why Is It Hard To Simulate Hair?

(Watanabe, Y., & Suenaga, Y., 1989) gave a good example of the complexity involved in rendering hair: "If, for example, we suppose a hair is approximated by 32 cylinders and each cylinder is approximated by 32 triangular patches, and if a person has 100,000 hairs, the total number of triangular patches would be 102,400,000. It is impossible to draw such a big number of triangular patches in a reasonably short time." Although this calculation does not include the complexity of hair dynamics or hair to head and hair to hair collision, it is a huge number. The ray tracing of the 102 million faces has not been counted either, and this could potentially be the most expensive part of the process. The difficulties of hair simulation can be listed as follows:

- Huge number of hair strands on a scalp
- Geometric intricacies(complexity) of individual hair
- Complex interaction of light and shadow among the hairs
- Small scale of thickness of one hair compared to the rendered image
- Dealing with hair to hair and hair to head interaction while in motion

1.3 Hair Simulation Phases

It is really hard to divide hair simulation into phases due to number of different approaches. (Magnenat-Thalmann, N. et al. 2002) proposed that, there are three main aspects in hair simulation. These are hair shape modeling, hair dynamics or animation and hair rendering. Hair shape modeling deals with exact or fake creation of thousands of individual hair strands. Four attributes are very considerable in hair shape modeling: hair geometry, density, distribution and orientation. These attributes directly affect the final image. For example, hair geometry differentiates, whether the hair is curly or smooth or to have a semi-bald head, hair distribution is modified. For the reasons, it would be worthwhile to handle them carefully. Dynamics of hair means hair movement, the collision of hair with other objects and self-collision of hair particularly when there is long hair. The movement of hair is governed by physical law, so physically based modeling approach may make good results. Animating single strand is not a difficult problem using physically based approach. However, the number of hair makes it impossible or impractical to implement. The rendering of hair deals with hair color, shadow, lightning and varying degree of transparency and anti-aliasing. Each of these aspects is a different research topic in computer graphics area and many researchers spent effort on them.

1.3.1 Hair Shape Modeling

The purpose of hair modeling is to produce a simple and ordinary hairstyle. Complex hair styles are combination physical properties of individual hair strand and complex hair to hair and hair to body interactions. Hair shape should be modeled in appropriate format to simulate hair movement at interactive speeds. Therefore, it would be considerable to treat hair shape modeling as a separate problem.

First attempts of hair shape models were based on explicit hair models. In explicit hair modeling, each hair strand is processed as individually for shape and dynamics. Each hair strand has parameters such as density, length, orientation. Model is the process of creating hair style from individual hair strands. Explicit hair models are very stylish and close to reality. Unfortunately, they are hard to implement and expensive methods for hair dynamics. The most obvious approach is to model each individual hair as a curved cylinder. The hair width is less than size of a pixel. This causes a serious aliasing problem.

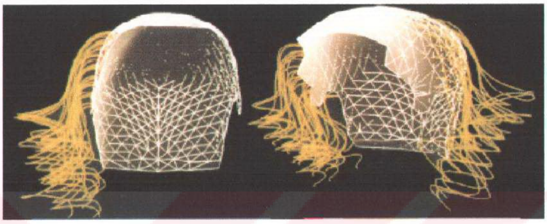


Figure 1.2 Hair styling by defining curves (Thalmann N. et al , 2002)

To overcome these difficulties of explicit hair modeling technique, it is considered as a bunch of hair instead of individual hair strands in cluster models. The major problem of explicit hair modeling is huge number of hair strands. Defining strands in a bucket will decrease the number of elements that must be modeled.

Fur is more complex structure which has explicit geometric definition. Using volumetric textures, fur can be modeled as a volumetric density function. Volumetric Textures are a logical extension of 2D textures. With this method we see much detail on the surface in Figure 1.3.

Figure 1.3 Volumetric Textures (Perlin K. et al. 1989)

This method gives an illusion of the fur like medium without defining geometry of each and every fiber. However this method has a limitation, only fur or very short hair can be modeled as a volumetric density function. Volumetric textures are successful ways of implementing complexity of nature and they are rich in detail.

1.3.2 Hair Dynamics

Generally, dynamic models are basically based on individual hair structure. An individual hair strand is modeled as a combination of rigid segments and these segments are connected to each other. Segments have bending stiffness at each joint. The individual hair segments are solved for the movement of hair due to internal and external forces. A single hair strand is modeled as a series of line segments as seen in Figure 1.5.

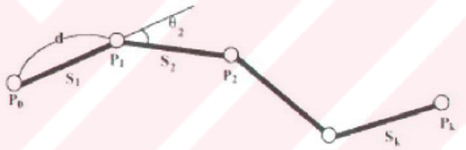


Figure 1.4 Single Hair Strand Model

In the case of fur, which is generally modeled as volumetric texture, the explicit model approach for the animation can not be used effectively. Instead of this, volume density functions should be used to facilitate this task.

1.3.3 Hair Rendering

Hair rendering is a huge problem in human simulation which is examined by researchers many years. Lots of techniques were used to create photorealistic hair models. The hair rendering problem arises due to complex interaction of light, small width of individual hair strand and detailed geometry of hair models. The rendering of hair therefore causes a considerable anti-aliasing problem. Reflections of light and cast of shadows play key role in hair rendering process.

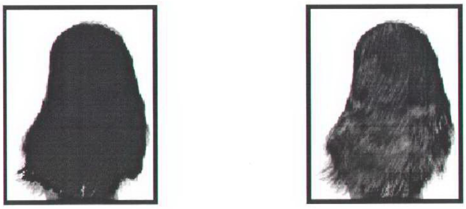


Figure 1.5 Lightning is crucial in hair simulation.

Lightning is an important phase in hair modeling. Hair has a semi transparent structure. So lightning must be handled carefully. In Figure 1.5, there are two hair models. Right model has a realistic lightning. The other one looks like a black area not hair.

1.4 Goal of This Thesis

This thesis focuses on three main parts which are related to each other. First part of the thesis can be thought as a survey about current and previous hair simulation models. In this section, models are categorized according to the techniques which they used and the way how they were implemented. Not only categorizing the techniques, also they are briefly described.

In chapter three, topic is physically based modeling, which has brought an important new approach to computer animation. The strong relationship between physics, mathematics and computer graphics is exhaustively examined and it is emphasized that physically based modeling plays important role in the realization of the real world to the screen. In addition, types of physically based models are studied by giving examples. Furthermore, Newtonian Rules of Motion is explained which is very crucial for physically based modeling.

The fourth chapter covers the details of mass spring systems which we used in our hair model. Before explaining our model we gave some other models which also use mass spring systems such as facial and cloth animation. The final chapter focuses on our hair model and future work about hair simulation.

CHAPTER TWO
PREVIOUS APPROACHES

Hair rendering is one of the most challenging subjects in computer graphics community. To challenge the hair modeling problem due to large number of individual hair strands in a model and to obtain photorealistic results, many years researchers published different solution approaches in hair modeling, hair dynamics and hair rendering fields. Some of them gave acceptable results, on the other hand, some techniques were quite well in theory, but not applicable.

2.1 Ancient Approaches

Research in hair modeling area has speed up last a few years. Because of inefficient hardware and algorithm complexity, people used very simple hair models in first human characters. However, recently realistic hair models have become necessary due to movie and computer game industry. This compulsion pushed hair modeling society to find new techniques. Before examining these techniques, the ancient approaches should be studied carefully to understand the importance of the situation.

The first technique was "coloring" method which was very simple and used frequently in oldest graphics applications such as first computer games. Actually, coloring is not a hair modeling technique. Only specified part of the head is colored to give the sense of the hair. There was no complexity; hence, it was very cheap. From this point, we will use the terms cheap and expensive for techniques. When it is considered that a technique is cheap, this means the technique is easily applicable and does not need too much effort during modeling and computation time during rendering. On the other hand, expensive means more computation time and effort.

Computer graphics developers try to extend coloring method by using gradient colors. In gradient colors, there is a smooth change from one color to another. As mentioned before in introduction section, hair color changes from root of the hair strand to the top and different parts of the scalp may have different color. Therefore, it will be a suitable choice to use gradient colors instead of one definite color. In Figure 2.1, a simple gradient color can be seen.

Figure 2.1 Gradient Colors

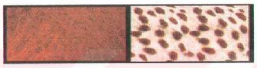


Figure 2.2 Texture Mapping

Concentrated on, applying gradient colors, developers ignored the hair for many years. By significant changes in computer graphics fields, they met three dimensional world and new concepts arose. One of them is texture mapping technique and it is quite popular in computer graphics. Texture mapping (Figure 2.2) is a graphic design process which a two-dimensional surface, called a texture map, is wrapped around a three-dimensional object. It is mostly used in computer games and real-time graphics applications. Textures, which look like hair or fur, are wrapped on models. Coloring method is not valid today. On the contrary, texture mapping is frequently used in most computer games to give hair to human characters today.

2.2 Hair Shape Modeling Approaches

2.2.1 Explicit Hair Models

The techniques that explained above have no sense of three-dimension. Therefore it is pointless to talk about hair dynamics or long hair models in these techniques. (Csuri, C et al., 1979) were the first to render fur-like volumes in 1979. In this approach, each hair strand was modeled as a single triangle laid on a surface. Constructed hair strands are rendered using a Z-buffer algorithm for hidden surface removal.

(Miller, G.S.P., 1988) obtained better results by rendering furry animals. The rendered images originated from explicit hairs. In this method, each hair strand was modeled with triangles to give the shape of a pyramid. The number of hairs was small and their thickness was large. Therefore, it would become impractical when applied to human scalp due to large number of hairs and thin structure of a hair strand.

(Daldegan, A. et al., 1993) presented a system that the user can define characteristics of individual hair strands and construct a hair style based on them. To define the characteristics of hair strands, user should set parameters such as density, spread and location.

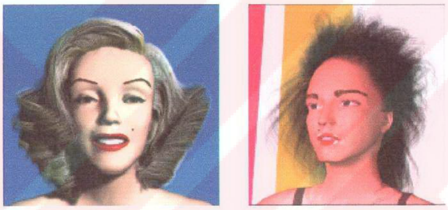


Figure 2.3 Hair styling with curves (Thalmann N. et al , 2002)

2.2.2 Cluster Model Approaches

The explicit models are really nice looking and realistic. However, they are computationally expensive. For example, it will take hours to model hair styles like in Figure 2.3. To overcome these difficulties hair is defined as bunch of hair strands in cluster model. Clustering is also known as wisps modeling.

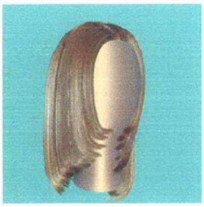


Figure 2.4 Cluster Hair Model, (Yan et al. ,1999)

(Yan X. D. et al., 1999) modeled the wisps as generalized cylinders and rendered the hair using the blend of ray-tracing generalized cylinders and volumetric textures as seen in Figure 2.4.

2.2.3 Volumetric Model Approaches

(Perlin K. et al., 1989) introduced hypertextures, which can model fur without defining geometry for each single hair strand and this method gives the sense of fur like view (Figure 2.5). They used simple analytical functions to model a furry ball and furry donut.

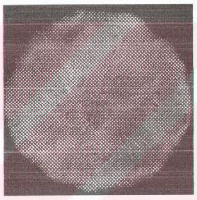


Figure 2.5 Fur as a Volumetric Texture (Perlin K. et al. 1989)

2.3 Hair Rendering Approaches

Hair rendering can be examined according to modeling structure. Hair rendering approaches can be divided into two categories: with or without modeling.

2.3.1 Hair Rendering with Modeling

(Watanabe Y. & Suenaga Y., 1989) introduced a wisp model which uses prisms to represent hair segments. Model gave a poor result when flat shading was used, however when smooth shading was applied the result was spectacular. When styling hair, it tends to be in the form of bundles or wisps. With introducing wisps, calculation costs can be reduced. To control the generation of hair, parameters such as number of wisps, number of hairs in a wisp, length of hair can be set. Generation based on wisps is very useful. After forming a single strand which follows the wisp parameters exactly, all other strands can be generated from this initial hair by adding some random factor to the points generated.

(LeBlanc A. et al., 1991) proposed a method which focused on hair rendering. The method is pixel blending which is similar to that used in particle systems. Each pixel of the image is treated as a square, and the portion of each hair that passes over the pixel contributes to the intensity of the pixel. Pixel blending requires that the pixels are blended in order from furthest to nearest the viewer, and naturally, hairs that are behind solid objects should not be blended. Shadowing is another important phase in this method. The hair casts shadows not only on itself, but also the surface underneath. Hence, it is necessary to calculate shadows. They also introduced shadow buffer algorithms that are ideal for hair rendering.

(Rosenblum R. et al., 1991) worked on not only static hair, also worked on hair dynamics. Their model needed some requirements:

- No restrictive upper limit on the number of hairs allowed in an image
- The images must be spatially anti-aliased
- The images must be shadowed
- The storage structure for a strand of hair must be amenable to efficient rendering
- Total execution time must be kept low so animation sequences can be rendered in a reasonable amount of time.

In the view of hair dynamics, in their model each hair strand was modeled as two masses put in a fixed distance by a spring. In this method, the new mass velocity, and therefore position could be calculated for all points in the hair.

(Anjyo K. et al., 1992) used cantilever beams to simulate hair. It was a straight beam with a fixed support at one end only. The bending occurs due to gravity. In cantilever beam approach, there is a problem. The hairs which have vertical normals grow quite long before they start to bend down, because, they are almost parallel to gravity. Anjyo introduced the concept of joint stiffness between consecutive hair segments.

(Daldegan A. & Magnenat-Thalmann N., 1993) introduced a model which wisps can be created and points in 3D world, which the wisps that will follow, can be defined. In this model collision detection is avoided by leaving the style to the user. But this brings a big problem; the amount of work done by user is huge. The rendering process is same with the model of (LeBlanc A. et al., 1991).

(Ando M. & Morishima S., 1995) showed in their approach that it is important to define the shape of the curve instead of local structure of each hair. The shape of hair is determined by shape control points. These shape control points were connected with segments.

(Rankin J. and Hall R., 1996) used a method called "linked chain of rigid micro-edges" to simulate hair. The attractive part of this method is avoidance of collision. This is done by changing the colors when a collision occurs. In previous approaches the change was done in the shape of hair instead of color.

2.3.2 Hair Rendering Without Modeling

As told before in this text, computational time to render a full head of hair is too big. Some researchers avoided this problem by applying textures to the scalp of human head. (Kajiya J. T. & Kay T. L., 1989) used 3d volumetric structures (texels) to make the teddy bear furry, in Figure 2.6. Texel is essentially an array that represents the density of the micro surfaces at each point. Texels are 3 dimensional texture maps. Texels are volumetric structures that hold complex collection of surfaces.

In the field of hair rendering, texels are used to store hair. They can be simplified by removing the lighting model components, as the bidirectional reflectance function is constant for each hair and common to all hairs, so long as the hair doesn't change color.

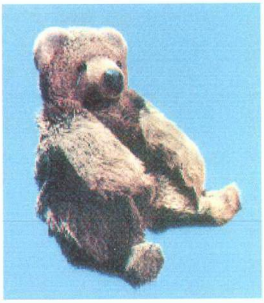


Figure 2.6 Furry Teddy Bear, (Kajiya J. T. & Kay T. L. ,1989)

(Sourin A. I. et al., 1996) proposed a new approach in hair modeling. In this approach, all detailed complex solids are produced from a simple formula. Solid objects are represented as a set of 3D points and they were defined with function f.

$$G : f(x, y, z) \geq 0$$

G represents the solid object and f is a continuous function. To deform the solid f function is modified. There are three situations in this model:

- $f > 0$: Points inside the solid.
- $f = 0$: Points on the solid.
- $f < 0$: Points outside the solid.

They used a noise function to define hairs. Each single hair strand is modeled as piece of solid noise. This method was very impressive for fur rendering

2.4 Comparison of Hair Models

(Magnenat-Thalmann N. et al. 2002) presented a survey about technological advances in hair simulation for computer graphics. They examined hair simulation process in the view of three aspects: hair shape modeling, hair rendering and hair dynamics. In table 2.1, they summarized their role with their effectiveness and limitations for each aspect of the hair simulation.

Table 2.1 Comparison Of Hair Models (Magnenat-Thalmann N. et al. 2002)

	Hair Modeling	**Hair Animation**	**Hair Rendering**
Explicit Models	Effective -tedious to model -not suitable for knots and braids	Adequate - expensive due to size - inappropriate for hair to hair interaction	Fast - inadequate for self-shadowing
Particle Systems	inappropriate	Adhoc - lacks physical basis - no hair-hair interaction	Effective - lacks shadowing and self-shadowing
Volumetric Textures	Effective -Not suitable for long hair	Limited - via Animated Shape Perturbation	Effective - expensive
Cluster Model	Effective -not suitable for knots and braids	Not done - via Animated Shape Perturbation	Effective

Research in hair simulation despite the difficulties has been encouraging and shown evident improvements over years. People do not accept human characters without hair or animals without fur. This forced developers to find new techniques. Hair simulation is still an open issue and it is certain that further improvements will occur in future.

CHAPTER THREE
PHYSICALLY BASED MODELING

In recent years, physically based modeling has emerged as an important new approach to computer animation and computer graphics modeling. It was first introduced in the late 80's to apply common physics and engineering disciplines to the subject of computer graphics modeling and animation. These disciplines are not fairly new in physics and engineering, but what is new about physically-based modeling is the way in which they are applied.

3.1 Physically Based vs. Physics Simulation

The first computer animations were drawn on individual sheets of paper or celluloid. When a series of sheets is transferred to film and run at a speed of 24 frames per second, an illusion of motion is perceived. From a simulation perspective, the sheets reflect a declarative type of model that is similar to a finite state automaton, one state per sheet. Therefore, early animation models were finite state automata. The next generation of animation programs allowed users to automatically create in between states by specifying certain key states (key frames) and then using geometric smoothing methods such as a Bezier or cubic spline. Now, given this knowledge, we can see how progress has been made in computer animation. Specifically, we can create more cohesive models by basing the animation on more complex model types, such as event modeling, functional modeling or constraint modeling. The term "physically based" normally refers to constraint based, equational, models derived from physical laws; however, the movement called physically based modeling is a step in the direction of using simulation models to replace the older automata type models, inherent in frame based simulation with more comprehensive system models. (WEB_2, 1994)

Physically-based modeling in computer graphics is the simulation of physical systems for the purposes of creating computer graphics images. Generally, these graphics images are animated. Anyway, when physics is subject, it is impossible to think system which has no motion. As a part of the realization of the real world to the screen, physical based modeling has an important role. It helps us to visualize the objects when they are interacting with the environment or any controls are applying. In physically based modeling, physics is just a tool. It is only used to create nice and interesting computer graphics. It does not matter if the model is correct for the given situation, as long as it gives appropriate results. The most important issue is the appearance and applicability of the model. Therefore, the accuracy of the model is ignored easily. Physically based modeling is a mixture of three fields: physics, numerical programming and computer graphics. By correct combination of these there fields, realistic looking computer graphics images can be achieved.

Physically based modeling is not a physics simulation. The main difference between physically based modeling and physics simulation is accuracy. In physical simulation, researchers try to gain more insight into the physical world. The stress is on accuracy, prediction of the real world and the discovery of new phenomena, for example, simulating the collision of two galaxies, behavior of a tornado, fluid flow, the corrosion or the fracture of a metal plate. Another type of physical simulation is used for engineering purposes to simulate the behavior of machines such as airplanes or cars, and is closer to physically based modeling because it often uses computer graphics and requires real-time solution. As we mentioned before, physically based modeling is a mixture of three components. In the next part of this chapter these components will be explained briefly. (Turner, R., 1993)

3.2 Physics

The term physics in physically based modeling is usually the Newtonian classical mechanic rules. Objects are the collection of masses. These masses are the either free moving, static or confined in rigid bodies. Although the masses themselves are simple, the forces acting between them can be complex, depending on the type of physical model. Actually these forces do not represent the forces of the nature. They are imaginary forces that is, approximations to the actual forces resulting from very complex physical processes. For example, in a simulation gravity force must not be same as the real world. If the simulation gives the well result, it does not matter how much it effected. The important matter is how it effected such as in which direction, how long etc.

Physics is obviously a field of study in its own right and not a sub-category of mathematics. Nevertheless, physics and mathematics are closely tied to one another in several areas within computer graphics. Examples of graphics problems that involve physics include how light interacts with the surfaces of objects, how light bounces around in a complex environment, the way people and animals move, and the motion of water and wind. Knowledge of physics is important for simulating all of these phenomena.

We can see already that we are beginning to deal not with exact mathematical representations of physical reality but rather with approximations. This is of course true for most physical and engineering calculations, but it is especially true in physically based modeling because the only real arbiter of physical accuracy in this case is our eyes. Physical simulation is little more than just another algorithm for generating interesting computer graphics and animation behavior. Because the algorithm is rooted in physical law, there is a better chance that we will get visually appealing results.

The main disadvantage of physically based modeling is the control of the system. The system tends to behave what the physics rules wants not we want. Friction, parameters and force fields are elements of this modeling world. The developer should adjust this friction and tune parameters until a desired behavior is achieved. Although these control mechanism is processed, there is still a big problem, kinematics control. This control is accomplished through use of constraints, which force the physically based modeling to conform to certain geometries or paths or to perform a certain task, while still behaving in a physical way.

3.3 Numerical Programming

Physical models construct the basis of physically based modeling, but how to simulate them is another topic. This topic leads us to numerical programming. This is one of the oldest fields in computer science which is concerned with finding numerical solutions to solve mathematical problems. Although numerical programming covers many topics, such as solving linear systems, finding roots of equations, and evaluating transcendental functions, the most important one for physically based modeling is the solutions of differential equations. Since most physical laws, and our models, can be expressed in the form of a differential equation, implementing a physically based modeling requires solving this differential equation numerically.

The main emphasis in numerical differential equation solving is on highly specific problems in science and engineering. In early days of computers, CPUs were slow and expensive; therefore, it was hard to interact with a numerical solution as it happens. But, with evolving technology, faster CPUs make us to achieve observing results and deciding what looks nice and realistic. Furthermore, a user can actually construct his model interactively, using the physical behavior as a modeling tool. Computer graphics developers do not want to develop complex numerical programming algorithms. The techniques need to be straightforward and easy to comprehend.

3.4 Computer Graphics

Computer graphics is the third and most important component of physically based modeling. Previous components are used to achieve realistic and nice looking computer graphics images. Physically based modeling can, in fact, be thought of as a natural evolution of computer graphics modeling techniques. Computers graphics also provides data structures and software techniques which allow complex physically-based models.

Physically-based modeling adds new levels of representation to graphics objects. In addition to geometry, forces, torques, velocities, accelerations, kinetic and potential energies, heat, and other physical quantities are used to control the creation and evolution of models. Simulated physical laws govern model behavior, and animators can guide their models using physically-based control systems. Physically-based models are responsive to one another and to the simulated physical worlds that they inhabit.

3.5 Classical Mechanics

Classical mechanics is the study of the motion of bodies, in accordance with the general principles first pronounced by Sir Isaac Newton in his "Philosophiae Naturalis Principia Mathematica (1687)", commonly known as the Principia. Classical mechanics was the first branch of Physics to be discovered, and is the foundation upon which all other branches of Physics are built. Moreover, classical mechanics has many important applications in other areas of science, such as Astronomy (e.g., celestial mechanics), Chemistry (e.g., the dynamics of molecular collisions), Geology (e.g., the propagation of sound waves, generated by earthquakes, through the Earth's crust), and Engineering (e.g., the equilibrium and stability of structures).

Most measurements in physics are made with respect to fundamental quantities or units. On the other hand, computer programs and mathematics deal only in pure numerical quantities. Physical quantities may be constructs of the human mind, but they are based on something real that exists "out there". We choose as our fundamental quantities a unit of time, a unit of space or distance, and a unit of matter or mass.

In classical mechanics, the main concept is the motion of objects. To find a way of describing the motion of objects, a mathematical formulation that allows us to represent the position, velocity, and acceleration of moving objects must be defined. This formulation must express how these quantities are related to each other in time. In other words Kinematics is concerned with describing the way in which objects move. There are 3 basic properties (functions) of a particle in motion:

Position: A position function can be either scalar-valued (for motion in one dimension) or vector-valued (for motion in two or three dimensions). At each point in time its value represents the position of an object at that time.

Velocity: It is the time-derivative of the position function, and gives the velocity of an object at each point in time.

Acceleration: This is the time-derivative of the velocity function, and the second time-derivative of the position function. It gives the value of the acceleration of an object at each point in time.

3.6 Dynamics

The study of dynamics begins with an introduction of the concepts of force and mass, then goes on to introduce the basic laws of dynamics, Newton's Three Laws. Dynamics describes why physical systems move in certain ways, not just how, and allows us to predict the behavior of a physical system.

3.6.1 Definition of Force

A force is an interaction between two objects involving a push or a pull. To have a force, there must be two objects, one object pushes and the other gets pushed. A force is something that happens between two objects. The things that make objects move are forces. A force is a vector quantity, as it has both a magnitude and a direction. The magnitude of force can be changed by pushing it harder or softer and direction can be changed by pushing it in one way or another. Consequently, it is obvious that force causes motion. Isaac Newton put forth a variety of laws which explain why objects move (or don't move) as they do. These three laws have become known as Newton's three laws of motion. (WEB_3, 2001)

3.6.2 Newtonian Rules of Motion

Newton's first law of motion is often stated as, an object at rest tends to stay at rest and an object in motion tends to stay in motion with the same speed and in the same direction unless acted upon by an unbalanced force. In other words, an object moves with constant velocity unless acted upon by a net external force. When velocity is zero, it means no there is no motion.

Mass is another important concept, that plays a role in the relation between force and motion. A mass is the amount of matter in a given body. Mass is a scalar quantity, it has no direction, and is the object itself, not its location. Mass is measured in kilograms (kg). A certain given objects mass will be the same on earth, on the moon, or in empty space. But the weight of object in these different locations will change.

Newton's Second Law gives us a quantitative relation between force and motion. Newton's Second Law says that the net force (F) acting upon an object causes acceleration (a), with the magnitude of the acceleration directly proportional to the net force and inversely proportional to the mass (m).

$$\sum F = ma \qquad (3.1)$$

The Second Law relates two vector quantities, force and acceleration. Because both force and acceleration are vector quantities, it is important to understand that the acceleration of an object will always be in the same direction as the sum of forces applied to the object. The magnitude of acceleration depends on the mass of the object, but is always proportional to the force. Newton's Second Law gives an exact relation between the vectors force and motion. Actually, the law shows, the force is simply the product of mass times acceleration.

All forces result from the interaction of two bodies. One body exerts a force on another. When we push a box across the floor, our hands and arms certainly feel a force in the opposite direction. Newton described his forces as acting on and between point masses, and in his third law of motion he states that for every action there is an equal and opposite reaction. If body A exerts a force on body B, let us denote this force by F_{AB}. Newton's Third Law, then, states that:

$$F_{AB} = -F_{AB} \tag{3.2}$$

Newton's third law also gives us a more complete definition of a force. Instead of merely a push or a pull, we can now understand a force as the mutual interaction between two bodies. Whenever, two bodies interact in the physical world, force results. Whether it be two balls bouncing off each other or the electrical attraction between a proton and an electron, the interaction of two bodies results in two equal and opposite forces, one acting on each body involved in the interaction. Our model is constructed with masses and is based on these Newtonian forces.

3.7 Types of Physically Based Models

There are lots of physically based modeling techniques that have been proposed for computer animation. According to (Hégron G. & Arnaldi B. ,1992), it is hard to classify models, because of crowded topics in physics and engineering. However, they have divided methods into four categories: free-particle systems, mass-spring systems, continuous systems, and rigid body systems.

The particle system is not defined well in computer graphics. It has been used to describe modeling techniques, rendering techniques, and even types of animation. The particles may be completely independent of each other, or there may be some forces between them. A particle system is composed of one or more individual particles (Reeves W. T. , 1983).Particles are graphical primitives such as points or lines. Free particle systems are a good choice for models of fluid, smoke, fire sand etc. Figure 3.1 is an example of fire animation. I used Particle Illusion 3.0 software to construct the animations. (www.wondertouch.com)

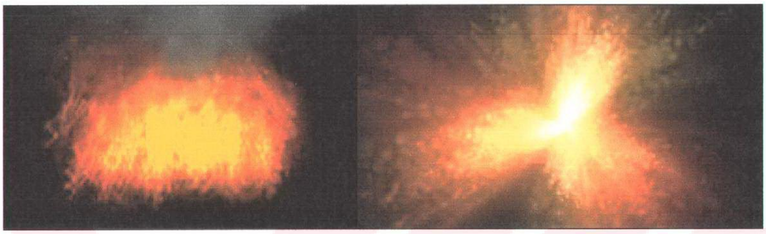


Figure 3.1 Fire animation with a particle systems using software P.Illusion 3.0

Mass-spring systems are commonly used to represent deformable models for real-time simulations. An object is modeled as a collection of point masses connected by springs in a lattice structure (Figure 4.2). Mass-spring systems are simple and easy to implement. Our hair simulation is a mass-spring system model. In next chapters, mass-spring systems will be explained in details.

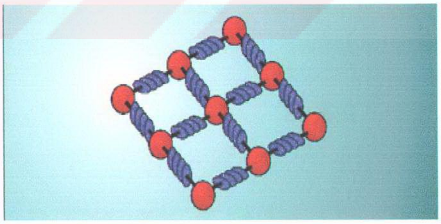


Figure 3.2 Mass-Spring System Layout

Continuous system models are based on continuum mechanics, which studies elastic, deformable and fluid materials. Quantities which were represented as single values (for a particle) or as an array of values (for a particle system); now become scalar, vector fields. Mass, m, for example, becomes mass density.

On account of flexibility problem of mass-spring and continuous systems, a mathematical abstraction approach, rigid body method appeared. Rigid body method is developed in different fields of science such as physics, engineering, robotics and automotive. System of rigid bodies is quite different from a system of particles in the sense that the elements of the system are rigid bodies that have not only the mass but also the volume. A rigid body is defined as a finite or infinite collection of mass points which are constrained to maintain a fixed distance from each other. In true life, all objects are somewhat flexible, but the rigid body approximation is a very good one for modeling a variety of everyday objects.

3.8 Differential Equations

In many ways, differential equations are the heart of analysis and calculus, two of the most important branches of mathematics for over the past 300 years. Differential equations are an integral part or a goal of many undergraduate calculus courses. As an important mathematical tool for the physical sciences, the differential equation has no equal. So it is widely accepted that differential equations are important in both applied and pure mathematics.

3.8.1 Modeling with Differential Equations

In physically based modeling, differential equations are a standard mathematical technique which is used to simulate a physical model. In other words, the behavior of a physical system can be defined by a differential equation. Firstly, the system examined carefully, then, it is put forward an idea about behavior of the system. This method is based on Newtonian laws of motion. The result is normally a differential equation of forces expressed in terms of differentials of the state variables with respect to time or space.

To animate the physically based model, a proper solution must be found. Exactly which solution we find depends on the initial conditions of the state variable, which can be chosen arbitrarily, and can greatly affect the outcome of the solution.

3.8.2 Newtonian physics-based differential equation

As mentioned before, Newton's Second Law gives us a quantitative relation between force and motion. It provides the tool for finding the net force on the object.

$$F_{net} = m\,a$$

Newton's second law of motion defines the net force on an object as its mass multiplied by its acceleration. Acceleration is a property of motion. Acceleration is defined as an object's change of velocity through time. A more mathematically precise definition is that an object's acceleration is the derivative of its velocity with respect to time, t.

$$a = \frac{\partial}{\partial t} v$$

Newton's second law, with the appropriate substitution, now looks like:

$$F_{net} = m\left(\frac{\partial}{\partial t} v\right)$$

An object's velocity is defined as its change of position through time, or more precisely, as the derivative of its position with respect to time, *t*. Define *r* as the position *vector* of an object in some coordinate system, and then the following is obtained:

$$v = \frac{\partial}{\partial t} r$$

From just above it is known that net force is mass multiplied by the first derivative of velocity with respect to time, *t*. Substituting this yields another useful form of Newton's second Law of Motion:

$$F_{net} = m\left(\frac{\partial^2}{\partial t^2} r\right)$$

This form of Newton's second law enables you to find the net force if you know an object's mass and its position through time. (WEB_4, 2004)

CHAPTER FOUR
MASS SPRING SYSTEMS

As desktop computing power and graphics capabilities increased during the 1980's, the graphics community began exploring physically based methods for animation and modeling. These methods use physical principles and computational power for realistic simulation of complex physical processes that would be difficult or impossible to model with purely geometric techniques.

Mass-spring systems are one physically based technique that has been used widely and effectively for modeling deformable objects. An object is modeled as a collection of point masses connected by springs in a lattice structure.

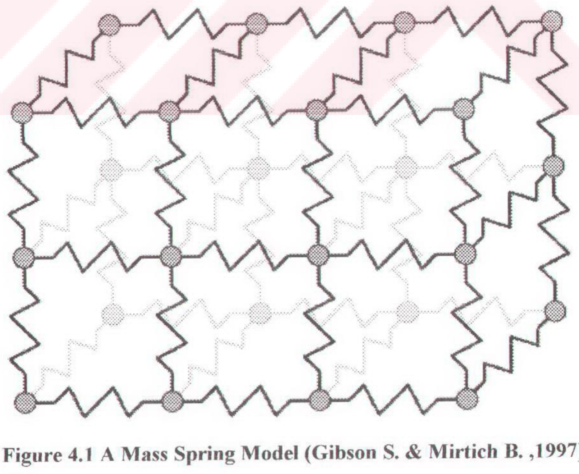


Figure 4.1 A Mass Spring Model (Gibson S. & Mirtich B. ,1997)

4.1 History

Mass spring systems are widely used to model many physical systems. In computer graphics, facial animation, cloth simulation and other elastic models are popular examples of mass spring systems. In face simulation, face is modeled as a two dimensional mesh of points. And these points are connected to each other with springs. Muscle actions of face are handled by a force that is applied to different parts of the face. (Terzopoulos D. and Waters K., 1990) were first to apply mass spring systems to facial animation. They modeled the face as layers like the anatomical structure in real life. In their model, each layer is modeled with different spring constants to maintain realistic behavior. The facial model has 6500 springs, and is animated at interactive rates.

Mass spring systems have been used extensively for animation. (Chadwick J. et al., 1989) combined mass-spring models with free form deformations to animated muscles in human character animation. The dynamic deformation of the muscle is calculated by modifying the motion of the lattice points.

(Tu, X.,& Terzopoulos, D., 1994) used mass spring system approach to model an artificial fish with full dynamics. They used 23 mass points and 91 springs. Muscles that form the body are moved to propel the fish.

The most popular method for cloth modeling is to represent the cloth as a system of masses connected with springs. Mass-spring techniques were introduced for cloth simulation by (Breen D. et al, 1994). (Baraff D. & Witkin A., 1998), introduced implicit techniques for cloth, used a formulation that has elements from both continuum and mass-spring systems.

4.2 Mass-Spring Basics

Springs usually occur physically as a coil of metal, and their idealizations have pretty simple behavior: compressing the spring will result in the spring pushing back, and stretching the spring will have it trying to pull back towards the start position, so any displacement along the axis of the spring will be countered by an opposite force that will tend to move the spring back to it's original position.

If a pull force on a spring is applied and stretched, and then a work is done on the spring. That is because a force applied over a displacement. Since work is the transfer of energy, it must be accounted for to what the energy was transferred. It is considered that the energy was transferred into the spring. The work becomes stored energy in the spring. The work becomes potential energy in the spring. A spring can be stretched or compressed. The same mathematics holds for stretching as for compressing springs.

When a spring (Figure 4.2) is compressed or extended, a force is produced in the opposite direction by the spring. The equations describing spring motion can be derived from Newton's Laws of Motion.

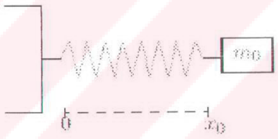


Figure 4.2 Mass Spring 1

For a simple spring, the following points can be observed:

- When a spring is compressed or extended, it will produce a resistance. The more compression or extension gives greater resistance.
- When an ideal spring is compressed or extended and then released, it tends to oscillate.
- When a simple spring is compressed or extended and then released, it tends to return to its rest length (x_0 position).
- Spring constant determines the compression and extension force.

From these points, a spring formula can be produced. (4.1)

$$\mathbf{F = -k * \Delta x} \qquad \mathbf{(4.1)}$$
$$\mathbf{\Delta x = x_c - x_i}$$

- **F** is the resultant force.
- **k** is the spring coefficient.
- **Δx** is the distance from it's rest position ($\mathbf{x_i = x_0}$)
- $\mathbf{x_c}$ is the current distance.
- $\mathbf{x_i}$ is the distance at the inertial position.

In the view of mathematics, a spring can be defined as being two points separated from a distance x. If these points are not changed, no force will be generated, because Δx is zero. Compressing the spring will change the value of Δx to negative. Hence, a positive force will be generated by the spring. On the contrary, stretching the spring will generate a negative force (Figure 4.3).

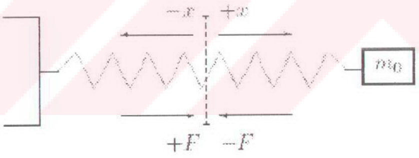


Figure 4.3 Mass Spring 2

The spring coefficient (k) is also called elasticity coefficient and affects the spring's final force. A spring with bigger coefficient creates a larger force. On the other hand, a spring with a smaller k is more flexible because it creates a smaller force from its inertial state.

4.3 Mass Spring Systems in CG

4.3.1 Cloth Simulation

In most graphics applications it is only necessary to provide the illusion of physically accurate behavior. In previous chapter, we have seen that, in physical based modeling, accurate results are not expected; the main purpose is to create nice graphics. One of the most popular approaches for simulating cloth is to model the cloth as mesh of mass-springs arranged in triangles.

In general mass spring model to simulate cloth, there are nodes, edges and triangles. A node is the mass point in space and edge is the pair of connected nodes by a simple spring. As usual, a triangle is three nodes connected with three edges. Simulation is done by applying forces on these nodes. Nodes have three crucial parameters; position, velocity and acceleration. For each time step, these parameters should be updated to generate the motion. In order to update parameters, some forces such as inner force of springs, bending forces, external forces must be applied to nodes. The net force on a node is the sum of all forces acting on node.

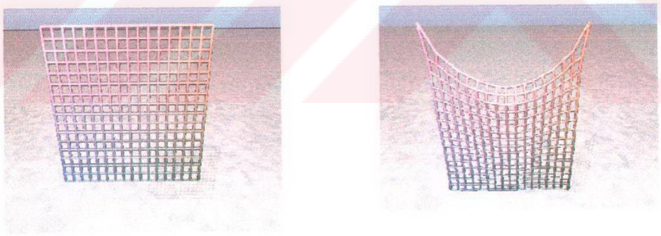


Figure 4.4 Deformation Of Elastic Model Of A Sheet by Provot X et al.

(Provot X., 1995) proposed a physical based model for animating cloth objects that is derived from elastically deformable models (Figure 4.4). Cloth is modeled as deformable surface composed of masses and springs by using fundamental law of dynamics. Provot used the Euler method to integrate energy equation through time to be able to get the position of positions at each time step. He also observed that when a hanging cloth is simulated using the classical model, unrealistic deformations occur at the constraint points. He was able to correct this problem by detecting the deformation rate of the springs around the constraint points. (WEB_5, 1999). One benefit of the model was that using this model it was possible to model a piece of cloth flowing through any time of liquid, including water.

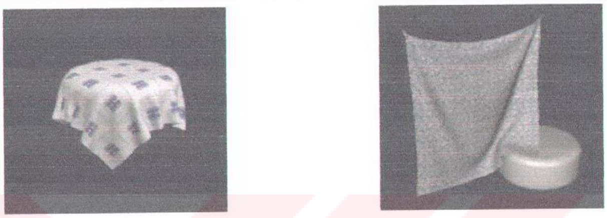


Figure 4.5 Cloth draping on objects by Baraf D. & Witkin A., 1998)

(Baraff D. & Witkin A., 1998) proposed a system that can stably take large steps in cloth simulation. The biggest bottleneck in cloth simulation systems is that time steps must be small to avoid numerical instability. The simulator models cloth as a triangular mesh, with internal cloth forces derived using a simple continuum formulation that supports modeling operations such as local anisotropic stretch or compression. The resulting simulation system is significantly faster than previous accounts of cloth simulation systems in the literature.

4.3.2 Facial Animation

Modeling and animation of human faces is one of the most difficult tasks in computer graphics like other physical models. Face has a very complex, flexible 3-D surface. It has color and texture variations and usually contains creases and wrinkles. There are more than 200 muscles in face connected to bones and tissues. The muscles lie underneath the skin and have a great influence on the appearance of the face. In muscle based facial animation systems, mass-spring model is widely used to represent the elastic properties of the face.

(Kähler K. & Haber J.,2001) presented a muscle model and methods for muscle construction that allow to easily create animatable facial models from given face geometry. The influence of muscle contraction onto the skin is simulated using a mass-spring system that connects the skull, muscle, and skin layers of this model.

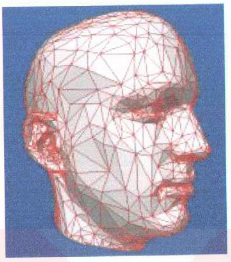


Figure 4.6 Head Model. (Kähler K. & Haber J. , 2001)

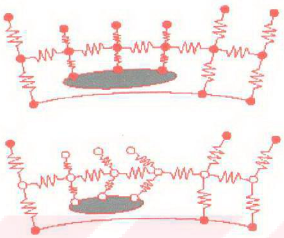


Figure 4.7 Mass-Spring Model. (Kähler K. & Haber J. , 2001)

As seen in Figure 4.7, elastic property of skin is simulated using masses and springs. The springs connect the skin nodes to muscles and skull. The skin is modeled as a layered structure to avoid skull penetration and preserve volume by additional springs.

(Lee Y. et al. , 1995) presented a mass-system contains 3 layers: cutaneous layer, subcutaneous fatty tissue layer and muscle layer which is the first 3D model of a human face. Jaw rotation is specified through a menu-driven interface in the model.

4.3.3 Jello Cube Simulation

Jello cube is another popular implementation of mass spring systems. The jello cube is a piece of 3D elastic material which can be bent, stretched, squeezed. It behaves like an elastic material; when stretched, it tends to contract. When squeezed together, it tends to inflate back to the original shape. Without external forces, it eventually restores to the original shape.

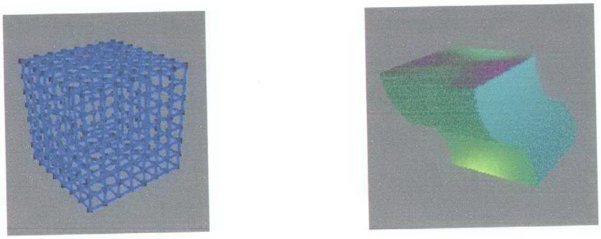


Figure 4.8 Jello Cube

Jello cube can be physically modeled as a mesh in three dimensions. The mesh consists of a set of points, links connecting points, and various properties embedded. Each point is initially defined with a position in space and a mass. Every mass point connected to other points by springs (Figure 4.8).

4.4 Advantages & Limitations

(Gibson S. & Mirtich B. ,1997) presented a survey of the work done in modeling deformable objects within the computer graphics community. They presented some purely geometric approaches for modeling deformable objects, but focused on physically based approaches including mass spring models. Mass spring systems are simple physical models with well understood dynamics. There are other physical models like continuum models which can not be animated at possible rates. On the other hand, mass spring systems are easy to construct and can be animated at acceptable rates even on a desktop computers. Parallel computation is easily used in mass spring models due to nature of nodded structure.

But mass spring models have some disadvantages. The model is an approximation of the physics that occurs in a continuous body. The structure can be only modified through spring constants and these constants are not properly determined easily to derive from measured material properties. In addition, some certain constraints may not be naturally expressed in the model. Very thin surfaces and some volumetric objects are difficult to model using mass spring approach. In these cases, some additional springs must be used and this increases computational cost.

Stiffness is another problem when spring constants are large. Large spring constants are needed to model objects that are nearly rigid, or to model hard constraints due to physical interactions. Problems in stiff systems arise due to poor stability, requiring the numerical integrator to take small time steps.

CHAPTER FIVE

MASS-SPRING BASED HAIR MODEL

Deformable objects which are required to improve the realism of the virtual reality applications have flexible structure. They are very useful in modeling clothes, facial expressions and human characters. However, they are known to be computationally expensive to model and render.

In human hair simulation, to give the motion, the shapes of hair strands are changed. If there is a change in the shape of an object during simulation, it is said to be a deformable object. Therefore, it is a meaningful approach to model human hair as deformable object. Modeling and simulation of deformable objects has a long history in computer graphics society. They have been studied for nearly two decades with different objectives. As we mentioned in previous chapter, mass spring systems are well appropriate for deformable objects. In this chapter, we will present a simple physics model to simulate human hair using mass spring systems.

5.1 Physical World Design

In real life, all objects act according to physics rules. For example, free falling of a ball is under the control of physics. When we set a ball free, the gravitational force makes it fall. There are forces in nature and these forces generate the motion. Motion is such an ingrained part of our daily lives that we rarely pause to contemplate it. We rarely notice the more subtle details of the motions of objects around us.

According to Wikipedia Encyclopedia (WEB_6, 2004), computer graphics is the field of visual computing, where one utilizes computers both to generate visual images synthetically and to integrate or alter visual and spatial information sampled from real world. Consequently, to get computer generated images, a system should be constituted that simulates the nature. In fact, real world is a complete physical simulation without missing parts.

There are lots of advantages of being able to simulate nature. It expands the creativity of animators and effects supervisors. Animation refers to the process in which each frame is produced individually generated as a computer graphic repeatedly making small changes to a model. Simulating the nature automatizes this process. Instead of generating each frame one by one, animation generation process is done by deforming the model. The deformation of objects and environment are handled by some functions that imitate the real world.

5.1.1 The Simulation Engine

To simulate the natural behavior of objects in universe, a physical simulation engine must be constructed. However, we have explained the differences between physically based simulation and physical simulation; we will use the term "physical simulation engine" for our physically based model. Physical Simulation engine supplies the ability to objects and environment to behave in a physically realistic fashion. This means objects in the simulated world can be configured to behave just as real objects do, according to the laws of physics.

5.1.2 Factors for a Stable Simulation Engine

The physical simulation engine can be so sensitive due to configuration of objects. This may cause some performance problems and unexpected results. There are lots of factors involved and sometimes it can be difficult to find the exact reason. Below are some topics to watch out for when dealing with physical simulation to build a stable simulation engine:

- Working with too large or too small objects may cause instabilities. In physically based modeling, the main properties of objects such as weight, length or size may differ from the real world.

- ➢ The characteristics of objects which are related with motion like velocity or acceleration can lead to unrealistic behaviors. To avoid from these behaviors the forces acting on objects should be designed attentively.
- ➢ Collisions between objects may result unexpected problems. Of course, it is not possible to handle all kinds of collisions just like in nature. But enough attention must be given to this topic.
- ➢ Unnecessary parts must be extracted from simulation for best performance. For example, when simulating the fall of a ball, the air friction should be ignored due to small effect on the system. If not, it will increase the cost of simulation and maybe the system will become non-applicable.

5.1.3 The Design

Generally, physical simulations are known to be complex models and many different approaches were developed by researchers in previous decades. A most common and plain model is shown in figure 5.1. We have constructed our model according to this process diagram.

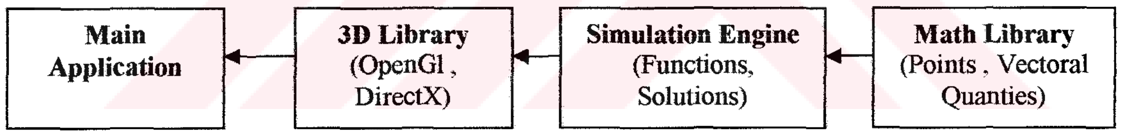

Main Application
3D Library (OpenGl , DirectX)
Simulation Engine (Functions, Solutions)
Math Library (Points , Vectoral Quanties)

Figure 5.1 Physical Simulation Process

To build our physical environment, firstly basic elements such as points, vectoral quantities must be declared. The math library includes these elements. For example, to simulate free fall of a ball; the position of ball, direction of gravity and velocity of ball must be described in this library. Simulation engine gives the ability of movement to the objects that are created with the basic elements in math library. Physical simulation is considered to be a three dimensional application due to nature of real world. In nature, all objects exist in three dimensional world. Consequently, using a 3D Library would be a good choice to perform the simulation. In this project, we used OpenGL to construct a 3 dimensional environment. Nehe Productions OpenGl tutorial (WEB_7, 2003) helped us to build this simulation.

5.1.4 Math Library

In fact, a math library does not need to include too many elements to build a three dimensional model. There are three coordinates in the universe: x, y and z. By using these coordinates all objects's positions can be defined. In addition to this, velocity, acceleration and force components must be used. To define all these properties vectors can be used.

5.1.4.1 Vectors

A study of motion will involve the introduction of a variety of quantities which are used to describe the physical world. Distance, position, velocity, acceleration, force, mass, energy, momentum, power etc. are examples of such quantities. All one of these quantities are either scalar or vector. Displacement, velocity, force and acceleration have a size or magnitude, but also they have associated with the idea of a direction. A vector is a quantity that has two aspects. It has a size, or magnitude, and a direction. In contrast, the quantities are called scalars that have only size. Vector quantities are not fully described unless both magnitude and direction are listed. (WEB_8, 2004)

For example, distance and displacement are two quantities which may seem to mean the same thing, yet have distinctly different definitions and meanings. Distance is a scalar quantity and it refers to "how much ground an object has covered" during its motion. Displacement is a vector quantity which refers to "how far out of place an object is"; it is the object's change in position.

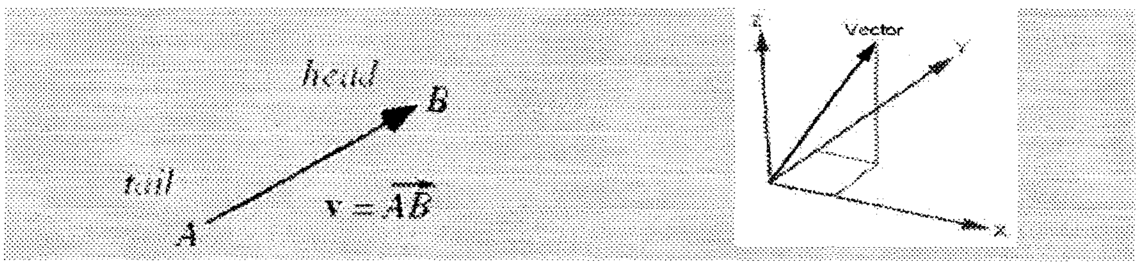


Figure 5.2 Vectors have a direction (left), A vector in coordinate system (right)

Graphically, a vector is represented by an arrow, defining the direction, and the length of the arrow defines the vector's magnitude (Figure 5.2). Mathematical way of representing a point in the space is named as vector. A vector is composed of three numbers, generally: x, y and z. These numbers are used to show how far it must be gone in three different directions. Vectors are written in the format of V(x, y, z). V(4,1,3) means there is 4-step movement in x axis, 1-step movement in y-axis and 3-step movement in z-axis. (WEB_9, 2004)

5.1.4.2 Vector Design

To build the basics of physical model, a vector class should be declared. This class will be used for points, vectors, position, velocity and acceleration. As explained before, this vector should have 3 elements.

```
Class Vector
{
  float x;   float y;   float z;
}
```

The math library only includes this class. In addition to these x, y and z values, the class contains the vectoral operations such as addition, subtraction etc. As mentioned before, force is defined as a vector and it has a direction and magnitude. Therefore, different forces acting on an object should be added to get the total force.

5.1.4.3 Vectoral Operations

Vectors can be added together, subtracted and multiplied by scalars. Vector addition is the operation of adding two or more vectors together into a vector sum. Force is a good example to show the properties of vectors. According to Newton’s laws of motion, the net force acting on an object is determined by computing the vector sum of all the individual forces.

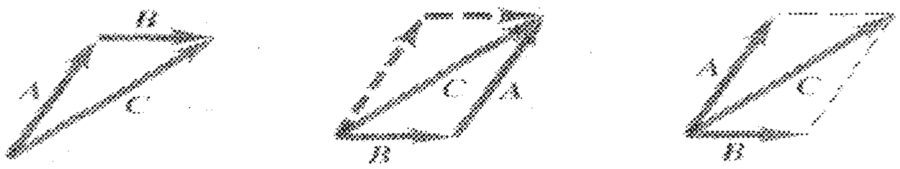


Figure 5.3 Addition Of Vectors (C = A + B)

The result vector is the addition of each coordinate of one vector with the corresponding coordinate value of the other vector. It is not important if there are two or more vectors; all vectors can be added in this manner.

```
Class Vector
{
  float x;   float y;     float z;
  Vector Add(Vector Other)
  {
    Return Vector(x + Other.x, y + Other.y, z + Other.z);
  }
}
```

Vector subtraction, division and multiplication of a vector with a scalar are other operations included in this class. Vector multiplication is not uniquely defined, but a number of different types of products, such as the dot product, cross product, and tensor direct product can be defined for pairs of vectors. The class also includes the operation to get the length of the vector. The length of the vector is calculated by the following formula for Vector (x , y , z):

$$\text{Vector Length} = \sqrt{x^2 + y^2 + z^2} \qquad (5.1)$$

5.1.5 Definition of Mass

According to Newton's Second Law of Motion, a force applied to an object makes it accelerate. But the same force applied to a bowling ball and balloon makes each accelerate differently. The bowling ball accelerates less because it has a higher mass. Mass is the amount of material in an object.

Sometimes, mass appears to be same with weight. In fact, mass turns to weight when gravitation exists. Weight differs on different locations due to gravitation diversity. On the other hand, there is only one value for mass of an object in all conditions. Weight is a measure of force and depends on both the amount of mass and the amount of gravity. Mass is important for calculating how quickly things accelerate when we apply a force to them. It is certain that, a car accelerates slower if more passengers in it.

5.1.6 Mass Design

Mass is an object with position and velocity. The "mass" class in our simulation should contain velocity of the mass, position of the mass, force acting on mass and the mass value of the mass.

```
Class Mass
{
  Vector position;
  Vector velocity;
  Vector force;
  float massvalue;
}
```

Position, velocity and force are vectoral quantities and they are declared as an instance of Vector class. Mass value is scalar; therefore it is declared as float type. The object tends to move in the direction of velocity. If there is a change in the position of an object, velocity is responsible for this. The change in the position depends on also how much time passed. In chapter3, we discussed Newtonian Laws. According to Newton's First Law of Motion, the velocity of a mass changes if there is a force acting on it.

Acceleration is the change in velocity per unit time. Second Motion Law of Newton shows that the net force acting upon an object causes acceleration, with the magnitude of the acceleration directly proportional to the net force and inversely proportional to the mass. There are several forces acting on the mass. The total force is the vectoral sum of all the individual forces. "Mass" class handles the calculation of the net force by a function.

```
Class Mass
{
   ...
   Void AddToNetForce (Vector F1)
   {
      this.force = Add (F1)
   }
}
```

The physical simulation has "simulate" function that performs the calculation of new values of position and velocity. The function takes "time" as a parameter. Consequently, for each time segment, position and velocity values are changed according to force and mass values. The change in the velocity is calculated with formula:

Change In Velocity = (Force / mass) * Time (5.2)

And it is added to current velocity to get the new value. The change in the position depends on velocity and time passed; it is calculated with following formula:

Change In Position = Velocity * Time (5.3)

```
Class Mass
{
   ...
   Void simulate (float time)
   {
      velocity = (force / massvalue) * time;
      position = velocity * time
   }
}
```

As seen in code segment above, the velocity and position depends on the time passed. So this "simulate" function should be iterated at each time segment. Iterating time may be implemented by different methods. One of the most famous and simple methods is "Euler Method" which is widely used in computer games. It calculates the next value of velocity and position for mass object.

5.2 Simulation Design

5.2.1 Simulation Process

When a physical simulation works, at each time segment same process is executed. This means some functions and operations are defined and the change is ensured by the time parameter. In this process, firstly all forces are applied and then new values of position and velocity are calculated. According to these calculated values, the image is drawn with a 3D library. To create the masses and perform operations on them a "Simulation" class is needed. In this class the masses are created and initialized. A simulation generally includes lots of masses, so it will be congruous to define a masses array.

```
Class Simulation
{
   Mass[] Array Masses; Float Number;
   Void create (int Number)
   {
        For (i=1; i< Number+1 ; i++)
        Masses[i] = new Mass();
   }
}
```

The "Mass" class has a simulate function which calculates the velocity and position of the mass. To call this function, a "simulateall" function is needed in "Simulation" class, for all of the masses that have created.

```
Void simulateall (float time)
{
  For (i=1; i< Number+1 ; i++)
    Masses[i].simulate(time);
}
```

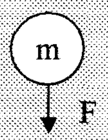
To get a complete simulation class, another function must be defined which tells to mass what to do. Without declaring it, simulation class will only create the masses and call the simulate functions of them. Still the position and velocity of masses are not declared and no force is applied. Thus, there will be no change in the mass position and this will cause no motion. In next topic, the importance of this function will be shown with an example.

5.2.2 Performing Simulation

By building "Mass" and "Simulation" classes, a basic simulation engine has come into existence. As an example, free fall of a ball will be simulated in this topic. In this simulation, ball is considered as a single mass. To perform the simulation a new class will be used which was inherited from "Simulation" class as a base class. This new class has a new function which was not included in the base class. This function holds the operations such as initial value assignment and applying force.

```
Class FreeFall : Simulation
{  Vector gravitation = (0.0, -9.81, 0.0);
   Void Initialize(){
    Masses[1].position = Vector(0.0,0.0,0.0);
    Masses[1].velocity = Vector(0.0,0.0,0.0);
    Masses[1].mass = 1;
  }
}
```

The mass has a velocity of zero in all directions due to free fall. It's position is set to (0,0,0) in the coordinate system and massvalue is set to 1. Gravitation force is the only force which should be applied to object to get a free fall effect. Actually, there are lots of other forces acting on object such as air friction. However it is impossible to calculate all forces that exist in nature. That would be a cost to computational time and some forces may be ignored to reduce this.



Force of gravitation is equal to the product of mass and gravitational acceleration.
F = m * g

Figure 5.4 Gravitational Force

When an object is dropped on earth surface, its velocity increases 9.81 meters per second in each second. Therefore, gravitational acceleration is 9.81 m/sec^2 on earth. The function that performs the gravitational force is shown below.

```
Perform(){
  Masses[1].AddToNetForce(                 Masses[1].massvalue
*gravitation)
  Masses[1].simulate(time);
}
```

Gravitation is a vectoral quantity so it is set to *(0.0, -9.81, 0.0)*. The fall of the object will be in the y axis in negative direction. This is why the gravitational acceleration is set to negative.

5.2.3 Application with OpenGl Library

The application generates the images with the values that are taken from the objects which were instantiated from simulation classes. This project focuses on an application which uses OpenGL as 3D Library. There are common procedures in OpenGL applications. The application has three steps:

- Creation of Simulation Object (Class FreeFall)
- Performing Calculations (FrameUpdate)
- Drawing Objects On Screen (GLRender)

The first step is executed once when the application starts. The creation of the simulation object is handled with this step. The created object is used in other two steps.

```
   FreeFall* FreeFallObj = new FreeFall();
```

Applications that use OpenGL are based on frames. To generate the animation each frame must be calculated and drawn. Second and third step performs these operations. These functions work simultaneously. Thus, according to calculated values in step 2 the image is drawn to screen in step3.

```
Function FrameUpdate(){
   FreeFallObj.Perform();
}
```

GlRender function is executed after each frame update. It contains the appropriate OpenGL commands to build the simulation visually. The code below simulates the free fall of a 4 sized point.

```
Function GLRender(){
   ...
   Mass* MassFoo = FreeFallObj.Get();
   Vector* position = MassFoo.position;
      glPointSize(4);
      glBegin(GL_POINTS);
         glVertex3f(position.x, position.y, position.z);
      glEnd();
   ...
}
```

5.3 Spring Motion

5.3.1 Spring Design

A spring is a device that has an equilibrium position, and when moved from that position, it pulls itself back toward it. A mass is connected to a spring and spring should pull the mass towards the connection point. This system can be modeled by claiming that there is a force on this object proportional to its displacement from the equilibrium position. In equation (4.1), if this position is denoted as origin, x is the displacement from origin. Here k is the proportionality constant which is proportional to the strength of the spring.

```
Class HarmonicMotion(){
  Vector ConnectionPosition = (0.0,0.0,0.0);
  Float k; // Spring Constant
  Mass M;
  Void initialize {
  M.velocity = (0.0,0.0,0.0);
  M.position = ConnectionPosition + (20.0,0.0,0.0);
  }
}
```

To model a harmonic motion, a mass should be declared first and its position should be set enough far away from connection position. The force applied to the mass is the product of spring stiffness (spring constant) and the distance vector. The mass may be in two sides of the spring during harmonic motion; one the left hand side or right hand side. The mass is pulled to the opposite direction with the force generated by the spring. This is why there is negative sign at formula in Equation (4.1).

```
Perform(){
Vector DeltaDistance=Mass.position - ConnectionPosition;
Vector SpringForce = -( DeltaDistance * k)
  Mass.AddToNetForce( SpringForce )
```

```
    Mass.simulate(time);
}
```

DeltaDistance is the vector from the position of the mass to the connection position. It is calculated by using the vectoral difference of mass position vector and connection position vector. The generated force is the only force that is applied to mass. The system oscillates forever until other forces are applied.

5.3.2 Friction Force

In physics, all systems have energy which is either kinetic or potential. All forms of energy are associated with motion. For example, any given body has kinetic energy if it is in motion. In real life, a system does not conserve all of its energy forever. Friction is the main cause of energy leaks in the system and the reason why energy is not fully conserved. Friction is a non-conservative force. Non-conservative forces are forces that cause a change in total energy.

Friction Force = - (Friction Constant * Current Velocity) (5.4)

In previous topic, the first energy is generated by the spring as a pull force. When the mass reaches connection point all energy is transferred to mass as velocity. This energy transformation lasts forever until no other force is applied. To get a realistic animation, some friction force should be added to simulation. Friction force opposes motion by working in the opposite direction. For the example which was explained in previous topic Equation 5.4 should be added to simulation to handle friction process.

5.4 Mass-Spring Design Model For Hair Dynamics

Up to now, we have built a simulation class which does nothing and examined how springs work with an example. From now on, we will construct hair model by using previously built models: Vector, Mass and Simulation. One individual hair strand includes particles connected to each other. In next topic, a single hair strand generation will be shown.

5.4.1 Single Hair Strand Generation

In our hair model which uses mass spring systems, there are masses and forces acting on them. From now on, we will use term "particle" instead of mass because of not confuse it with mass object. The hair strand is a curved line which has a deformable structure. This means its shape changes by the time passes. To generate a hair strand lines between particles are used.

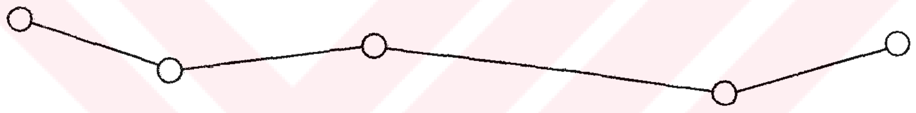


Figure 5.5 Lines between particles generate the hair strand model

In Figure 5.4 above, particles are denoted with a circle. The line which holds the two particles together are denoted as springs. In this figure, there are five particles connected with four springs. These springs will be source of force between two particles.

In topic 5.3, we have built a spring model where we defined a single mass and a spring. In this model, there was a connection point and the mass was oscillating according to this point. Now, we will extend this model to create a tension between two particles. But, if we use the same spring force formula, system will never reach a steady state. In equation 4.1, if Δx is not zero, a force will be generated. Therefore, the particles will continue to move and will never stop. Consequently, the formula should be changed as follows:

$$\text{Spring Force} = -\,(\text{Spring Constant}) * (\Delta x - d) \qquad (5.5)$$

In this equation d denotes the desirable distance between two particles. When the distance between particles is d, no force will be applied and there will be a specific distance between particles. The following spring class holds the two masses and calculates the forces acting on them.

```
Class Spring(){
  Float SpringConstant;
  Float DesiredDistance;
  Mass M1;
  Mass M2;
}
```

In this class, two masses and two constant values are defined. The drawn line between these two masses will be the one segment of our hair strand. By defining more springs in our simulation, we will get a segmented hair strand. “Spring constant” value is used to define spring tension. “Desired Distance” value determines the distance between masses and this is the length of one segment.

5.4.2 Spring Force

Equation (5.5) will be our base formulation to generate the most significant force in our simulation. Firstly, spring vector should be found to get the direction and magnitude. Spring vector is the vector which is defined from one mass position to another.

```
void Perform(){
  Vector SpringVector = M1.position - M2.positon;
  Float SpringVectorLength;
  d = DesiredDistance - SpringVectorLength; // (Δx-d)
  Vector Force = - (SpringConstant)* d; // F= -k * (Δx-d)
}
```

The generated force is applied to both masses. But one force in positive direction, the other is in negative direction. Because, while one mass pulls the other, this will be a push force on the other mass. This force is only the spring tension between masses. There will be other forces acting on masses such as friction, gravitation and wind in our hair model. From now on, all forces that we construct will be applied to both in different directions. The code segment below shows application of forces.

```
void Perform(){
   ...
   M1.AddToNetForce(   Force )
   M2.AddToNetForce( - Force )
}
```

5.4.3 Single Hair Strand Simulation

To animate a single hair strand by using physical laws, we need to generate springs. The number of springs depends on the length of the hair strand and speed of animation. When a proper distance is set between masses, adding new springs increases the hair length. Therefore, the distance between masses is another factor that affects hair length. When there are fewer springs, there would be less calculation proportionally. Consequently, our simulation will work faster.

```
Class Spring(){
   Constructor Spring(Mass Mass1, Mass Mass2){
      this.M1 = Mass1;
      this.M2 = Mass2;
   }
}
```

Before constructing hair simulation class, a part must be added to spring class. There will be lots of masses in our class and these will be connected to each other. Each two pair of the masses will belong to a spring. Therefore, the masses of a spring should be determined by a constructor in spring class as shown in code segment above. Class "Hair simulation" includes the spring generations and performs the simulation for one hair strand.

```
Class HairSimulation
{
  Spring[] Array Springs; Mass[] Array Masses;
  Float SpringNumber;   Float MassNumber;
  Float SpringLength;
  Void initialize()
  {
      For (i=1; i< MassNumber+1 ; i++){
      Masses[i] = new Mass();
      Masses[i].position.x = SpringLength * i;
      }
      For (i=1; i< SpringNumber +1 ; i++){
        Springs[i] = new Spring(Masses[i],Masses[i+1]);
      }
  }
}
```

Now, all the masses are created and these masses will be the milestones of the hair strand. Masses are the joint points of the strand. The two consecutive masses are bound to a spring with a simple for loop. As we remember from previous topic, the "Perform" function generates the spring forces and applies them to each one of the masses. When "Perform" function is executed for each spring, the forces will be calculated for each mass.

```
Void Perform() // Of Class HairSimulation
{
  For (i=1; i< SpringNumber +1 ; i++){
    Springs[i].Perform();  //  Perform  method  of  spring
class
  }
}
```

Until now, simulation of one hair strand is described briefly. But there are missing parts. There will be other forces acting on masses to make the system stable and more realistic. Without gravitation force, particles will be in same row without sense of reality. Also, air friction is a considerable factor to hold the system steady. Unless air friction is applied, the system continues to swing forever.

5.4.4 Gravitational Force

Gravitational force is widely used in most physical simulations. It is the most important force on earth. Every object tends to fall. In our simulation, masses must be under the effect of this force for a realistic view. In "FreeFall" example, we have defined a gravitational force for a single mass. For hair model, it would be enough to generate this force for all masses. "g" is the gravitational acceleration vector.

```
For (i=1; i< MassNumber+1 ; i++){
  Masses[i].AddToNetForce( g * Masses[i].Massvalue )
}
```

5.4.5 Air Friction

As we mentioned in topic 5.4, all systems in real life do not conserve their energy forever. Some part of energy is converted to heat due to friction. In our hair model, if air friction is not defined, hair strand never stops to swing. Therefore this friction force should be applied to all masses in this model.

```
For (i=1; i< MassNumber+1 ; i++)
{
  Masses[i].AddToNetForce(-
(airfriction*Masses[i].velocity))
}
```

Air friction constant, spring constant and mass values are chosen arbitrarily in this model. In physically based models, accurate results are not expected; the most important issue is the appearance and applicability of the model. Therefore, these values do not represent the real world values.

5.4.6 Output: Single Strand

In figure 5.5, the animation of a single hair strand is visualized. 18 masses are used in this example. They are shown as dots to show the joint points of the strand. When we move the first mass which is located at the top, all other masses follow the nearest mass due to spring effect and this gives a smooth movement.

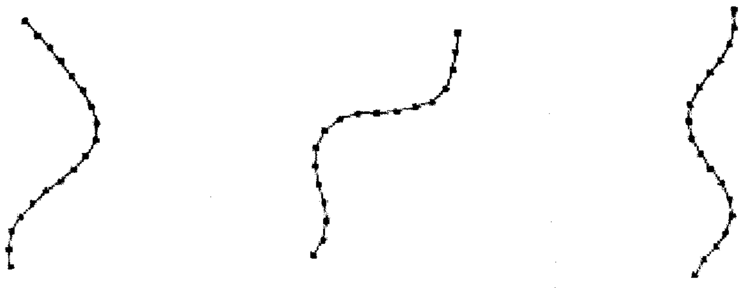


Figure 5.6 Motion Of Hair

Until now, we have built a fully functional model for the animation of a single strand. In next topic, to have a bunch of hair, we will clone this hair model.

5.5 Wisp Of Hair

5.5.1 Extending HairSimulation Class

To increase the number of hair strands in our simulation, we extended "HairSimulation" class. The number of hair strands is passed as a parameter to this new class.

```
Class AllHairSimulation
{
   Int number;
   Constructor AllHairSimulation (int number)
   {
      This.number = number;
   }
   HairSimulation[] All = new HairSimulation(number);
}
```

If same values are set for all hair strands, we will be able to see only one strand on screen, because same values mean same results. Therefore, there must be a difference in parameters of strands. This should be done by defining a difference factor or changing some values such as friction or mass weight.

5.5.2 Wind Effect

To see the animation details, we have added a wind effect to our simulation. The wind can be applied in both directions; from left or right. Wind is set as vectoral quantity in simulation.

```
Class HairSimulation
{
   Vector wind;
   Void Setwind(Vector Wind)
   { this.wind = Wind;}

   For (i=1; i< MassNumber+1 ; i++)
   {
   Masses[i].AddToNetForce(wind);
   }
}
```

5.5.3 Output: Wisp Of Hair

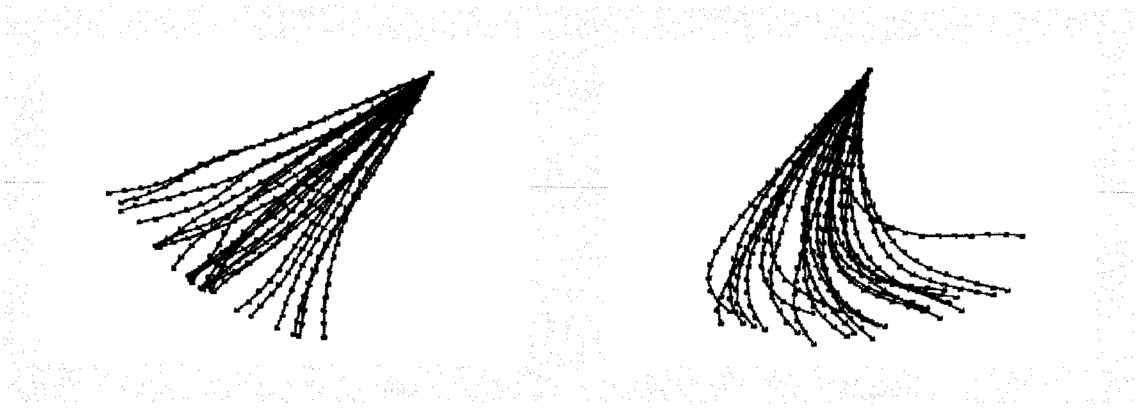


Figure 5.7 Output 1:Wisp Of Hair With Nodes

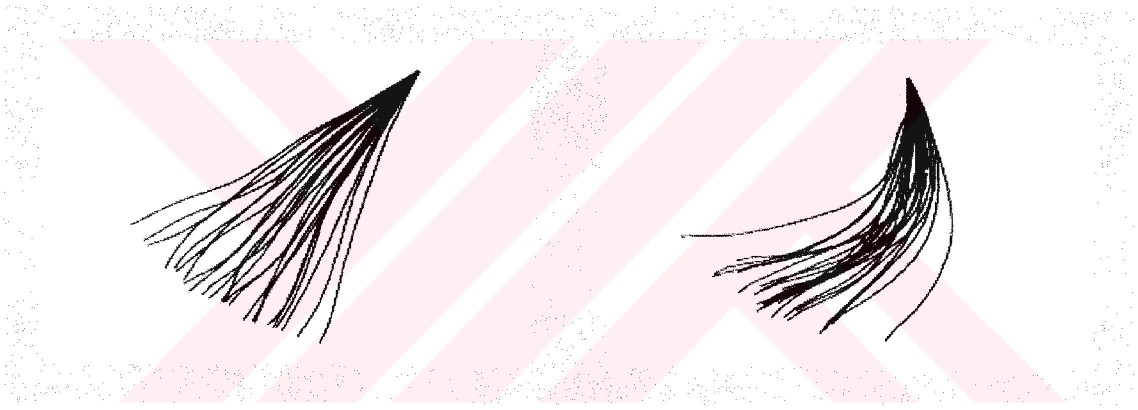


Figure 5.8 Output 2

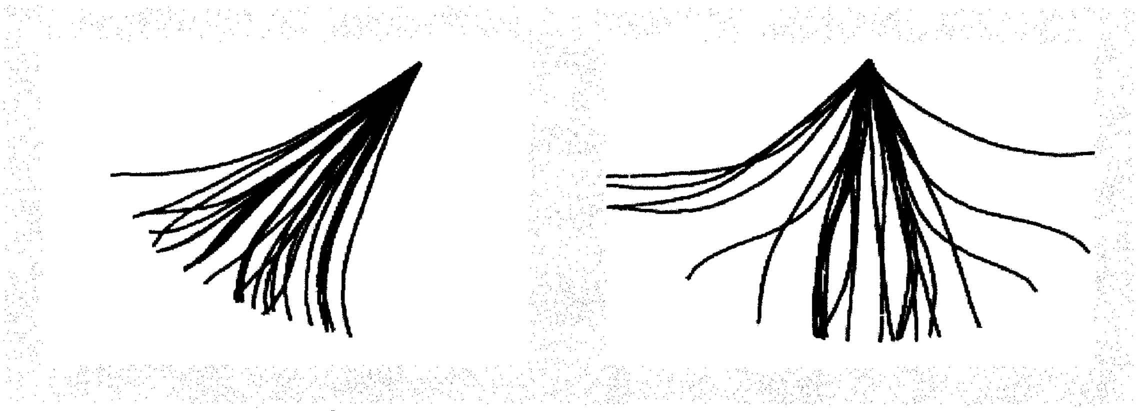


Figure 5.9 Output 3 : Thicker Strands

CHAPTER SIX
CONCLUSION

There are different techniques which are used in hair modeling. In this thesis, we have examined these techniques briefly and introduced a physically based model which uses mass spring basics. Our hair model can be considered as an explicit hair model. Explicit hair models are the most obvious approach to rendering hair is to model each individual hair as a curved line. As seen on table 2.1, explicit models are very expensive due to number of elements.

When each hair strand is generated individually, it costs too much and sometimes the model becomes inapplicable. Our hair model has a segmented structure. Each hair strand is composed of lots of segments which were implemented with masses and springs. Reducing number of segments is a major factor to speed up the animation. But, when the hair strand is modeled with fewer segments, we get lower quality image.

For better performance, the combination of this model and cluster hair models can be a solution. The hair on some parts of scalp shows the same behavior. Therefore, it will be suitable to model a wisp of hair and distribute this wisp over different parts of scalp; instead of modeling all strands on the scalp.

As a future work, the model can be implemented on a parallel environment. The position and velocity values for masses can be calculated on parallel machines for performing these large and complex tasks faster.

REFERENCES

Newman, W.M.,& Sproul, R.F.(1979). Principles Of Interactive Computer Graphics. McGraw-Hill Inc., Newyork

Watanabe, Y., Suenaga Y. (1989) Drawing human hair using wisp model. Proceedings of Computer Graphics International'89 , Springer Verlag

Magnenat-Thalmann, N., Hadap, S.,& Kalra, P.(2002). State of the Art in Hair Simulation. International Workshop on Human Modeling and Animation, Seoul, Korea, pp. 3-9.

Perlin, K.,& Hoffert, E. M.(1989). Hypertexture. Computer Graphics, 23, 253–262.

Csuri, C., Hakathorn, R., Parent. R., Carlson, W., & Howard, M.(1979). Towards an interactive high visual complexity animation system. Computer Graphics, 13

Miller, G.S.P. (1998). From Wire-Frame to Furry Animals, Graphics Interface 1988, pp. 138-146.

Daldegan, A., Thalman, N.M., Kurihara, T.,& Thalmann, D. (1993). An integrated system for modeling, animating and rendering hair, Computer Graphics Forum, pp. 211–221.

Yan, X. D., Xu, Z., Yang, J.,& Wang, T.(1999).The cluster hair model. Journal of Graphics Models and Image Processing. Academic Press.

Leblanc, A., Turner, R., AND Thalmann, D.(1991).Rendering hair using pixel blending and shadow buffer. Journal of Visualization and Computer Animation, 2 , 92–97.

Rosenblum, R., Carlson, W.,& Tripp, E.(1991).Simulating the structure and dynamics of human hair:Modeling, rendering and animation. Journal of Visualzation and Computer Animation. pp.141–148.

Anjyo, K., Usami, Y.,& Kurihara, T.(1992).A simple method for extracting the natural beauty of hair. Computer Graphics (Proceedings of SIGGRAPH 92) , 26, 111–120.

Daldegan, A.,& Magnenat-Thalmann, N.(1993).Creating virtual fur and hair styles for synthetic actors.Springer-Verlag.

Ando, M.,& Morishima, S.(1995). Expression and motion control of hair using fast collision detection methods.Springer.

Rankin, J.,& Hall, R.(1996).A simple naturalistic hair model. Computer Graphics,30,5-9

Kajiya, J. T.,& Kay, T. L.(1989).Rendering fur with three dimensional textures. Computer Graphics, 23, 271-280

Sourin, A.I., Pasko, A.A.,& Savchenko, V.V.(1996).Using real functions with application to hair modelling", Computers and Graphics , 20 , 11-19.

Hégron, G.,& Arnaldi, B.(1992) Computer Animation : Motion and Deformation Control, Eurographics'92 , 120

Turner, R.(1993).Interactive Construction and Animation of Layered Elastic Characters. Doctoral Thesis.

Reeves, W.T.(1983). Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. Computer Graphics. 17, 359-376.

Gibson, S.,& Mirtich, B.(1997)A survey of deformable modeling in computer graphics. Technical Report,MERL

Terzopoulos, D.,& Waters, K.(1990). Physically-based facial modeling,analysis, and animation. Journal of Visualization and Computer Animation, 73 ,80

Tu, X.,& Terzopoulos, D.(1994) Articial Fishes: physics, locomotion,perception, behavior. In Computer Graphics Proceedings,ACM SIGGRAPH.

Chadwick, J., Haumann, D.,& Parent, R.(1989). Layered construction for deformable animated characters.ACM SIGGRAPH, pp 243-252.

Baraf, D.,& Witkin, A.(1998) .Large steps in cloth simulation, ACM Computer Graphics (SIGGRAPH '98 Proceedings), pp. 43-54.

Provot, X.(1995).Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. Graphics Interface '95, pp 147-154.

Terzopoulos, D.,& Waters, K.(1990). Physically-based facial modeling,analysis, and animation. Journal of Visualization and Computer Animation, 73, 80

Lee, Y., Terzopoulos, D.,& Waters, K.(1995).Realistic Modeling for Facial Animation.ACM SIGGRAPH.

WEB_1. (2003). What is Hair. http://www.apollohairsystems.com/hair.html, 03/02/2004

WEB_2. (1994). Role Of Simulation. http://www.cise.ufl.edu/~fishwick/book/section2_2_5.html, 05/04/2004

WEB_3. (2001). Newton's 1st Law Notes
http://www.batesville.k12.in.us/physics/PhyNet/Mechanics/Newton1/, 17/04/2004

WEB_4. (2004). Newton's First Law of Motion.
http://www.physicsclassroom.com/Class/newtlaws/U2L1a.html , 17/04/2004

WEB_5. (2004). Cloth Modeling. http://davis.wpi.edu/~matt/courses/cloth/,
22/05/2004

WEB_6. (2004). Wikipedia Encyclopedia. http://www.wikipedia.org, 10/06/2004

WEB_7. (2003). NeHe OpenGL Tutorials. http://nehe.gamedev.net, 10/08/2003

WEB_8. (2004).Vectors - Fundamentals and Operations.
http://www.glenbrook.k12.il.us/gbssci/phys/Class/vectors/u3l1a.html, 07/06/2004

WEB_7. (2004). The use of mathematics in computer games.
http://www.nrich.maths.org.uk/mathsf/journalf/may00/art3/ , 07/06/2004

APPENDICES

APPENDIX A: 3rd PARTY HAIR & FUR PLUGINS

A1. Maya Fur

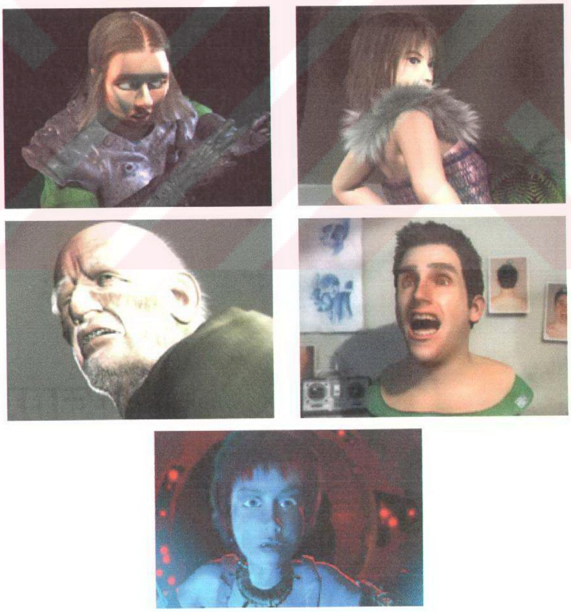
This plugin is released from Alias|Wavefront in first-quarter of 1999. This plugin allows to select a surface in Maya and than cover this surface with hair using some attributes such as curl, transparency, length, color. Maya Fur is integrated within Maya renderer, and results can be output as composite images or as separate render pass for post-processing. Maya Fur is self-shadowing and supports both true 3D cast shadows and render efficient root and back shading. Motion blur and depth of field fully supported.

Sample Images created with MAYA Fur.

A2. Shag:Hair

Shag:Hair is a revolutionary new plug-in system used to add hair and fur to any 3D object. Using control hairs to define the overall look of hair, this software plug-in is implemented as an atmospheric effect inside 3D Studio MAX although it can create true geometry. Standard MAX splines can shape hair strands, which can then become control hairs. Shag:Hair has built-in dynamic properties and controls for such attributes as thickness, clumping, curliness, and collision. Shag: Hair has the added advantage of its own built-in dynamics engine, so that as your character or creature moves, the hair will flow realistically, even to the point of accurately colliding with surrounding geometry.

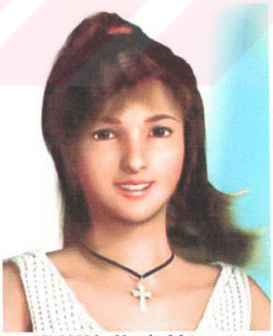
Sample Images created with SHAG:HAIR.



A3. Shave and a Haircut

Shave and a Haircut is a 3D system for generating and rendering complex dynamic particle systems such as grass and hair. "Shave and a Haircut" is shipped as a plug-in that is available for Maya, 3D Studio Max and Cinema4d. Its effects are dynamic and react automatically to physical forces, such as momentum, wind and gravity.

Sample Images created with Shave and a Haircut.

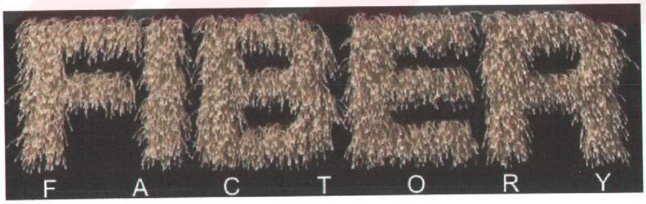


(C) 2003 by Yusuke Matsumoto

A4. Fiber Factory

This Lightwave plug-in makes hair from fibers with any number of sides and segments. The fibers can have various lengths, widths, colors, contours, curls, and other effects; can react dynamically; and when used with bones, can be styled in layout and animated without dynamics. It provides specular, diffuse, and ambient lighting effects to strands of two-point polygons.

Sample Images created with Fiber Factory.



A5. Fur Designer

Created by a long-time film production company, this plug-in for Softimage 3D uses a particle animation system to create hair and fur. It offers styling control, provides texture mapping of all parameters with interactive updates, and supports animation and deformation.

Sample Image created with Fur Designer.

A6. Houdini

Houdini provides fur control through its particle operators (POPs); hair position is based on particle streams. By putting fur into POPs, dynamics happen automatically, hair growth can be animated, and skin can be deformed when hair is tugged. Within Houdini, complex procedural parametric spline tools generate complex procedural fur behavior, thereby affecting the position, orientation, and behavior of individual hairs or groups of hairs to, for example, cause the hair on a cat to rise when the cat arches its back. Sample Image created with Houdini.