Hair Modeling and Rendering Using Ray-Tracing on GPU

Nasim Sedaghat

A thesis

in

the Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

August 2008

Your file Votre référence
ISBN: 978-0-494-45502-9
Our file Notre référence
ISBN: 978-0-494-45502-9

Abstract

Hair Modeling and Rendering Using Ray-Tracing on GPU

Nasim Sedaghat

Realistic human hair simulation, especially in real-time, is one the most challenging problems in Computer Graphics due to the unique nature of hair strands. The major obstacle is the large number of hairs and rather small diameter of individual hair strands. Specifically, we are dealing with four problems in hair simulation: modeling, rendering, collision detection, and animation. In this thesis, we focus our studies on hair modeling and hair rendering.

For modeling hair, we based our algorithm on key strand hair modeling where hairs are considered as wisps of similar hairs with one key strand per wisp. We added various parameters to this method to achieve more realistically-looking hair. Also, we introduced a new technique to produce short human hairstyles easily.

We proposed a new 3D geometric object called *Continual Cylinders* to represent hairs in 3D space based on the fact that the cross section of hairs can be approximated by ovals. Simulating hair strands with this rather simple geometric object helps us generate realistic hairstyles without affecting running time.

Rendering a sufficient number of individual hairs to convey realism in real-time is a very hard task in Computer Graphics. To be able to do this we rendered hair using GPU in this thesis and we successfully rendered about 10,000 hairs in real-time. Moreover, we used the ray tracing method, along with GPU, which is a time-consuming rendering technique that is mostly used to produce very high quality images. Here as well, we introduced and used various methods to achieve realism such as hairs self-shadowing, hairs fineness, slightly different hairs color between strands and within a strand itself.

Acknowledgments

I would like to gratefully acknowledge the enthusiastic and valuable supervision of Dr. Fevens who helped me a lot during this research and whose ideas always inspired me. Also, I would like to thank him for his helping me write this thesis with his great suggestions and with his patience.

I would also like to thank my friends and family for their support and who were always there for me.

Dedicated to my wonderful sister, Yasaman.

Table of Contents

List of Figures

Chapter1. INTRODUCTION

1.1 Human Hair

Hair is one of the most important factors of a human's first impression and ones overall appearance. The same person can appear unexpectedly different with different hairstyles. Strictly speaking, each person has a unique hairstyle and not only one, but a lot of them during the life. People treat their hair in various ways such as: washing, combing, cutting, using hair products, styling, etc. Hair plays an important role in almost everybody's life.

Research [Rob00] estimates that, on average, a human's scalp contains about 100,000 hair strands. On the other hand, the width of each hair strand is less than 200 micrometers. Therefore, we are dealing with a large number of hair strands and a rather small diameter for each individual strand. The nature of hair is complex and unique as you can see in Figure 1-1; the surface of strands of hair is not smooth. The exact physical properties of hair or the true nature of hairs are hard to model. Also, everybody has different hair types.

There are generally four major types of human head hair: *fine*, *medium*, *coarse* and *wiry*. Hair can also be thin, medium or thick and it can be from very straight to extremely curly. The natural color of hair can be blonde, black, brown or red. The diameter of hairs can vary from 17 micrometers to 181 micrometers. The cross section of an individual hair strand can also vary from oval for very curly hairs to circular for straight hairs [Rob00].

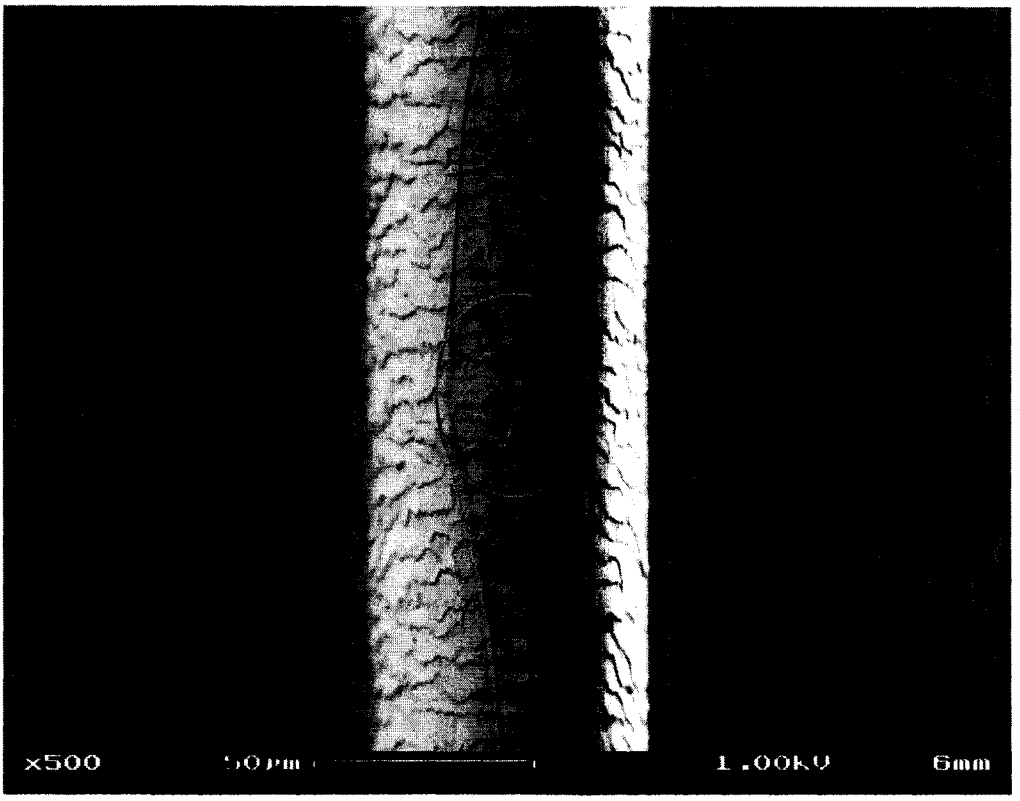


Figure 1-1: Microscopic view of a single hair strand

There is a micrograph of a nanowire curled into a loop in front of a strand of human hair. The nanowire is about 50 nanometers wide [To03].

1.2 Purposes of hair simulation

Hair needs to be represented for many simulation purposes such as, for example, a stunt-double in movies, in virtual reality environments, computer games, or animations. Each simulation has specific requirements in terms of how realistic the hair looks and how quickly it is rendered.

Sometimes in these simulations, we need **realistic** hair simulation that need not necessarily be done in real-time. Instead, we are able to render hairs **offline** using an essentially unlimited amount of computer power. For example, in a movie, a *stunt-double*'s hair has to be as similar as possible to the real actor's hair, yet, we have practically no limit on rendering time.

In some other simulations, it may be necessary to render hairs in **real-time** but the result need **not** necessarily be **realistic**. For example in *computer games*, every scene need to be rendered in less than 1/15 of a second, yet, the hair models only need to be good-looking not necessarily realistic.

In *computer animations* on the other hand, we may need realistic hair or semi-realistic hair or even non-realistic hair. Also, we have no limit on rendering time and animations are usually rendered **offline**. Today high quality and heavily detailed animations take a large amount of rendering time. For example, each frame[1] of the movie *Ratatouille* took an average of six hours on a 2.66 GHz processor to render. Overall, it took 1532 CPU-years to render the whole movie. Of course rendering it with only one CPU was impossible (it would take 1532 years!); therefore, about 3200 processors has been used. In Appendix A you can find rendering time of some famous animations with more details.

Finally, there are situations where we need **realistic** hair that can be rendered in **real-time**. For example, in a virtual barber shop simulation[2], we need virtual hair that can react realistically to the barber's actions (although we may not actually need realistic-looking hair). Such situations require a combination of the most difficult aspects of hair simulation: realism and fast rendering time. Having both at the same time with today's technology is quite challenging because we need to simulate enough amount of fine hair considering all environmental influences on hair which consequently require more rendering time. In the future, we may find it interesting to have a program that shows our barber's customers how exactly they will look like with particular hairstyles while reproducing hairs with exactly the same properties as that of the customers. This of course needs realism and real-time hair simulation at the same time.

[1] A frame is a single image of a film or any motion picture. More than 15 frames per second are required to consider a set of frames as a continuous film.
[2] Virtual barber shop is an environment that barbers can be trained before start on working in real world.

Representing hair virtually plays an important role in the field of human simulation as it is one of the most important factors of a human's look. Before recently, even in animations hair strands were not simulated individually and characters' heads were either covered by hats or consisted of a number of patches of the same color or texture; for instance, *Toy Story 2* (1999) treated hair the same as all the other objects of the scene. Then by 2001 producers of animations focused on hair more; for example, *Monster Inc.* was the first animation that succeeded to render millions of individual hair strands (Sullivan, the main character of the movie, had 4 million hairs in some of the scenes). Also other movies like *Shrek* (2001) or *The Incredibles* (2004) decided not to overlook the role of hairs in designing an animation's characters. And recently, animations like *Beowulf* (2007) and *Ratatouille* (2007) made it hard for future animation producers to ignore the importance of hair in human simulation (see Figure 1-2-left).

Video games are yet another type of situation because of the fact that we need real-time simulation for games. Recently, games quality become remarkably high as you can see in Figure 1-2-right, still, hairs are difficult to simulate in real-time and this is one of the recent challenge in the gaming industry.

Figure 1-2 : Hair simulation; Games vs. Animations
Left) a snapshot of the movie Beowulf with realistic-looking hair [IM08]
Right) a snapshot of the game Assassin's Creed (2008) where hairs are not represented individually [Ubi08].

1.3 Difficulties of hair simulation

The unique nature of hair makes it difficult to model and time-consuming to render. The main reasons are:

1. We are dealing with **large number of hairs**. To claim we can simulate realistic hair, we need to render about 100,000 hair strands one by one. To see the real difficulty of the problem, let us suppose that each individual hair strand consists of 200 points. This means that we should render 20 million points (regardless of all the equations for realism and motion). Using OpenGL and C++, rendering 20 million points with a 2.99 GHz CPU would take about one minute per frame.

2. **Thickness of each hair strand** is relatively **very small**. In a normal viewing condition, the thickness of an individual hair strand is less than the size of a pixel, the basic unit of the composition of an image on a television screen, computer monitor, or similar display, in a frame. When graphically representing a scene, we usually deal with rendering polygons which contain a number of pixels of the same color whereas for hair strands not only is it hard to represent them by polygons but also most of the times their thickness is less than the size of the pixel. Therefore, it is not a straightforward procedure to simulate hairs.

3. **Hairstyles are uncountable**. Hairstyles can vary from person to person and from time to time. Also, people change the overall shape of their hair in many ways such as combing, using hair products, or coloring (partially or completely). A single hairstyle can vary too; it can be wet or dry, clean or dirty, electrolyzed, orderly or disheveled, or a mixture of any of these. All of these parameters affect the hair model.

4. Another problem is **hair texture**. As we mentioned in previous section hair types varies from person to person. Also, the surface of an individual hair strand

is not smooth which make light and shadow computations tough. It is also hard to mimic the texture that a hair really has (see Figure 1-1).

1.4 Issues in hair simulation

Basically we are dealing with four issues for representing hair as a 3D object: Hair Modeling, Hair Rendering, Collision Detection and Animation.

1.4.1 Hair Modeling

3D modeling is the procedure of representing any 3D object as a wireframe object or defining a 3D object mathematically. Here, hair modeling means to define a 3D model for hairs and this involves geometry, distribution, density, orientation of each hair strand, and hairstyle.

Geometry: We need to represent strands as geometric objects. The cross section of hair strands can be represented as an oval. The strand itself can be represented as a curve. Yet, since hair strands are so thin that usually it is hard for viewer to distinguish them individually, simple geometric objects such as **poly-lines**, **splines**, **polygons,** or even **points** are widely used to represent hairs.

Distribution: There are thousands of hair stands that should be attached to the scalp of the head, and the distribution of hairs along scalp is not uniform. Therefore, the hair model should include position of each individual hair on the scalp of the head. Although it may at first seem unimportant due to large number of hair strands, positioning hairs is important for at least one reason -- it is not even straightforward to have a uniform distribution because the surface of the scalp is not a simple geometric object but usually is represented by a mesh of triangles.

Density: Not only the number of hairs can vary from head to head, but also the density of hairs in one head can vary from place to place of the scalp.

Orientation of each hair strands: Hair strands are free to move, therefore, every one of them can have a particular orientation at any time. No matter what geometric object we choose to represent hair, it should be able to handle the orientation problem.

Hairstyle: Finally and most importantly, we should be able to produce different hairstyles (ideally all potential hairstyles).

1.4.2 Hair Rendering

Rendering is the procedure of assigning a color to each pixel of the screen. Due to the large number of hair strands, there is a huge trade-off between time and quality in the hair rendering area. Unlike most of the Computer Graphics problems, triangulation and other mesh-based techniques do not work here (as we mentioned before, the thickness of hair strand is typically less than the size of a pixel); therefore, rendering likewise is not a straightforward procedure. Hair rendering involves color, light, shadow, blending, etc:

Color: Not only does everybody have different hair colors and every individual hair strand has a color of its own, but also the color may vary along a single hair strand itself.

Light: Lighting affects each and every single hair strand. Although without any special illumination calculations i.e. only considering solid color of hair strands, we may still produce recognizable (Figure 1-3) hair, light plays a crucial role in rendering realistic hair.

Figure 1-3: Rendering hair strands without using illumination models
Hairs are rendered as 3D vertices with a solid color

Figure 1-4: Self shadowing comparison up) without down) with [BMC05]

Shadows: Like other Computer Graphics problems, here also environmental shadowing is important. But what is more important than that in hair simulation is self-shadowing. As you can see in Figure 1-4, self-shadowing affects realism greatly and therefore cannot be ignored.

Blending: As mentioned before, the thickness of a human hair under normal viewing conditions is less than the size of a pixel. The best solution proposed so far is to use pixel blending methods. Pixel blending is a simple method that combines the color of the current pixel with the color of the very next object behind current object using which we can produce very thin and fine-looking hair strands.

Different rendering techniques can be employed to assign a color to each pixel of the frame. Although this can be done with a pixel by pixel approach, other

common techniques are usually used, like **rasterisation**, **ray tracing**, and **radiosity**:

Rasterisation: Normally, the graphical scene consists of some number of polygons defined by vertices. The simple task of transforming[3] these vertices in 3D world into the corresponding 2D pixels of the screen is called rasterisation. The color of each pixel is then decided with respect to the corresponding polygon.

Radiosity: Here, every part of the surface of every polygon is considered as a light source. Therefore, all the surfaces can emit energy and affect the color of other surfaces. It is easy to see that with radiosity rendering technique, we will need more than one pass to render all the objects (refer to Background and Literature Review for more information).

Ray Tracing: Here, the light path is treated as a straight line partially passing through objects such that one part of the light ray reflects off the surface of the object and part of it transmits through the object. Light will be tracked from the collision point over and over again as it intersects new objects until desired quality of illumination is reached.

The hair rendering phase very much depends on the hair modeling phase. For example **Physics-based** rendering depends on the physical object used to represent hair; all properties such as light scattering, reflection, color, etc are defined relatively. In **image-based** rendering the recovered hair strand are points that can be considered as simple as connected lines or as complex as Generalized Cylinder[4]. Subsequently, using images from different viewing conditions would let us capture overall lighting and highlights. Unlike image-

[3] Mapping functions such as rotations, reflections, and translations.
[4] This is of course just in theory. We can not treat recovered strands from images as a 3D geometric object yet; they are so thin if distinguishable.

based and physics-based approaches, geometry-based rendering totally depends on the underlying method of geometric-modeling.

1.4.3 Hair Animation

Hair animation is the dynamic motion of hair. Hair animation is the most important reason that we model hair. Therefore, the underlying hair model should be able to be employed for different hair movements. A Physics-based approach is not so common in hair animation area because of the complexity of hair strands. The trade-off between time and quality is provoked here as well due to the complexity of the hair dynamics.

1.4.4 Collision Detection

Collision detection entails hair-hair interactions and collisions between hairs and other objects of the scene. Hair-hair interactions problem is very important in hair dynamics. This is undoubtedly the hardest problem in the area of hair modeling and yet to be an area of significant research progress.

1.5 Real-time rendering with GPU

Graphics **P**rocessing **U**nit or GPU is a special purpose microprocessor that has been designed to do the complex graphical computations. GPUs have a parallel structure that makes them so efficient for graphical purposes especially 3D graphics which is so demandable these days. A GPU can take the load off the CPU; today's most time-consuming operation in Computer Graphics is rendering (Appendix A), thus a GPU is faster than a CPU and can let CPU do other calculations while it is busy with rendering.

The parallel structure of GPUs makes them a robust tool for real-time rendering purposes. The only serious disadvantage of GPUs is its limitation such as size of loops, size of arrays, number of instructions, etc. (for more information, refer to Appendix B).

Many programs have been made to render fully on GPU instead of the CPU or GPU-CPU Hybrid.

1.6 Our Contribution

Among the problems discussed in section 1.4, hair modeling and rendering are the main concerns of this thesis. Potentially, the method that will be introduced has the ability to implement hair dynamics but the focus is on hair modeling and rendering areas.

Our method of hair simulation is suited best for video games and other virtual environments in which real-time hair simulation is a must whereas quality is as high as possible.

1.6.1 Hair Modeling

Although in an ideal world a completely correct physical model for hair would make modeling very easy, most of the methods used today are trying to create outputs that resemble desired hair shapes using not necessarily a correct physical model of hairs but a good estimation of reality. Again, ideally we should be able to represent any kind of hairstyle with our simulation tool. Therefore, two of the most important areas of research in hair modeling are **representing hairs as geometric objects** and **ability to model any kind of hairstyles**

with out too much effort (practically, it is almost impossible to model every strand one by one).

Today, the best geometric object for representing hair is the Generalized Cylinder[5] whose sweeping-area's attributes can vary (cross sections can be estimated by ovals but they are not necessarily the same and should be able to vary along the hair strand curve). As we will discuss in the forth chapter, an estimation of the Generalized Cylinder, the *Continual Cylinder*, is introduced for the first time in our thesis to represent hairs. We discuss fully why Continual Cylinders were defined and chosen over other choices such as Generalized Cylinders, lines, polygons, or points.

The second important factor of a hair model is that a variety of hairstyles should be possible to model. Our hair model has the ability to model most of the normal hairstyles and also some of the abnormal ones (like new fashions or wired innovations!). Users have a very high control over the hairstyle with our hair model while needing to set only a relatively small number of parameters.

1.6.2 Hair Rendering

Recently, many different approaches have been developed in offline hair simulation [NT04, CJ+04, CK05, MI05, WOQ05, BA+06, WBC07]. They are capable of producing near-realistic and also fancy-looking hair for animations. There are some real-time methods that can render a reasonable number of hairs as well [KH00, KHS04, Os07]. Real-time hair rendering is a difficult task because of the huge deal of data; about 10,000 hair strands or reasonable number of hairs, should be rendered. Note that although we normally have 100,000 hairs, 10,000 hairs look fine as well as you can see in Figure 1-3-left. In both offline and real-time approaches, realism is a challenging issue and an open problem.

[5] Volume made by moving a 2D shape, here can be an oval, along an arbitrary curve.

The goal of this thesis is to model and render hairs in real-time as completely as, and as realistically as, possible. For achieving these goals we used ray tracing methods and *GPU* programming. We use **Ray tracing** for the sake of **realism** and **GPU** for the sake of **efficiency**.

Using GPU for rendering makes graphics calculations faster due to the highly parallel structure of it. Ray tracing method has been used for its good estimation of reality. The focus is on rendering an adequate number of hair strands as realistic as possible and for achieving this, ray tracing has been used along with GPU programming which, to our knowledge, has never been used in the area of hair simulation.

In this thesis, a simplified version of the key strand method introduced in chapter3 is used as underlying model for the rendering part. This simplified version contains most of the important aspects of the introduced model i.e. hair strands geometry, having wisps ruling by a key strand, and some of the realism issues. With this version, we succeeded to render about 10,000 hair strands in real-time i.e. about 15 to 25 frames per second.

1.7 Overview of the thesis

In chapter 2, we provided some background in Computer Graphics and we reviewed some of the most significant works that have been done in the area of hair simulation.

Chapter 3 focuses on hair modeling. We introduced our version of key strands-based hair modeling along with various realism issues and provided the results of this hair model.

We defined and introduced our geometric object to represent hairs, Continual Cylinders, in chapter 4. We also provided the reason for this selection especially over Generalized Cylinders.

Hair rendering procedure is described in the chapter 5; starting with real-time rendering with GPU. Then we demonstrate the CPU role in our algorithm and introducing ray tracing on GPU algorithm, and then we applied some realism issues in our algorithm. We completed this chapter by a discussion about running time of our algorithm.

Finally, conclusion and future work are discussed in chapter 6.

Chapter2. Background and Literature Review

2.1 Computer Graphics; Modeling and Rendering

Recent advances in computer technology have made it possible to bring Computer Graphics into play and use it as a powerful tool for many purposes in Art, Science, Engineering, industry, business, advertising, etc. Initially, Computer Graphics was used as a tool for displaying **data graphs and charts** to summarize data such as financial, statistical, scientific, economic, or engineering data. Another important use of Computer Graphics was, and is, **computer-aided design** (CAD). CAD methods are used in the design of various products such as buildings, automobiles, aircraft, computers, etc. Software packages for CAD applications provide a multi-window 3D environment for users to be able to design products as easily and as fast as possible (see Figure 2-1).

Animations and video games have grabbed much attention recently. 3D Computer Graphics is now widely used in the production of animations and video games. Also, in film making, unusual or sometimes impossible tasks can be done using Computer Graphics rather than employing a stunt double especially in Science Fiction and Action movies [HB04].

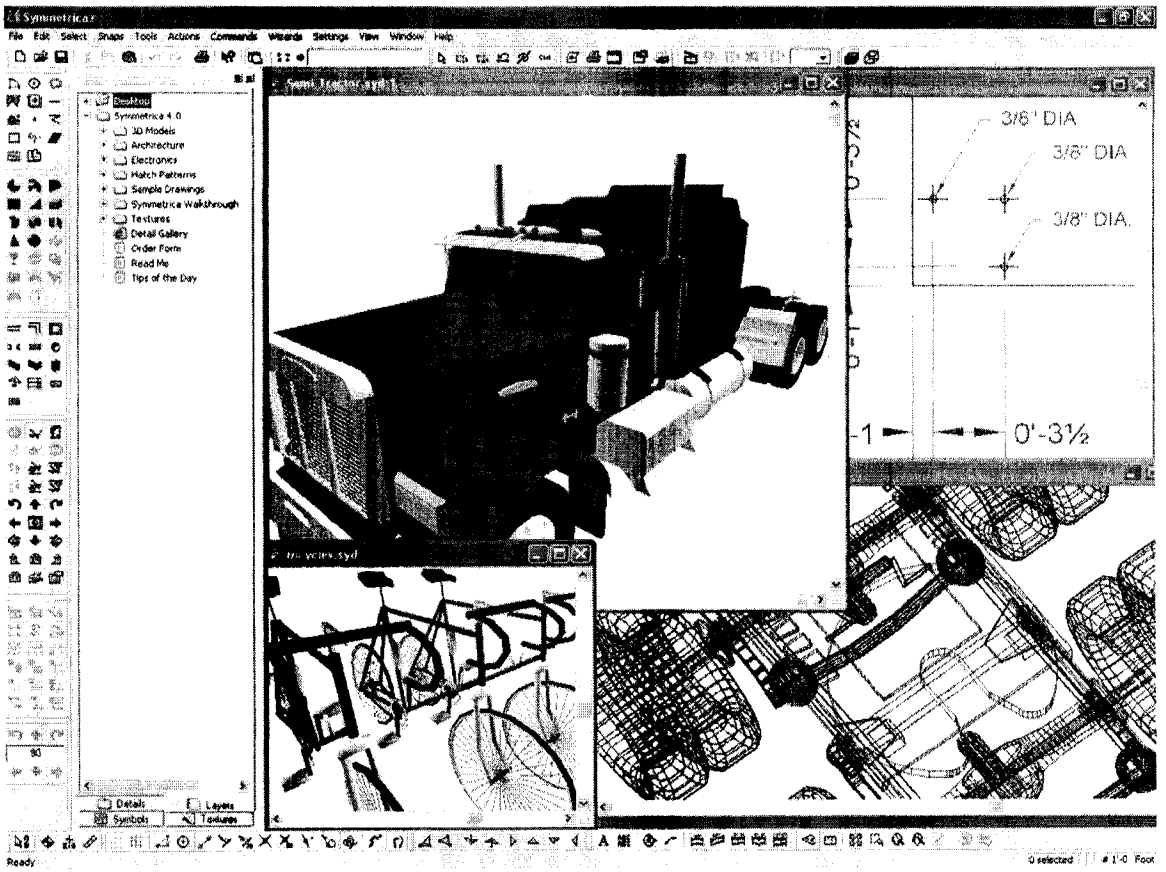


Figure 2-1: CAD
[HB04]

Apart from the previously mentioned major subfields in Computer Graphics (modeling, rendering, collision detection, and animation), many other related subfields interest us such as, 3D representation, viewing, illumination, shadowing, texture mapping, and alpha mapping. We will take a quick look at these issues in the next few sections.

2.1.1 3D Modeling

Modeling is the process of representing a 3D object mathematically so that it can be used later for rendering or CAD. There are various issues that should be addressed in this process; the representation should well fit the object, it should be easy to render, it should be easy to create, etc.

The most commonly used 3D object representation is a **polygon mesh**. Each object can be defined as accurate as desired by using a set of polygons estimating its surface. With polygon meshes, not only can any object such as the human body, furniture, buildings, etc., be estimated, but also any level of detail is possible (see Figure 2-2). They are also easy to use; a set of polygons with their normal vector can define any object. Normal vectors are used for illumination calculations. And finally, polygon meshes are easy to render.

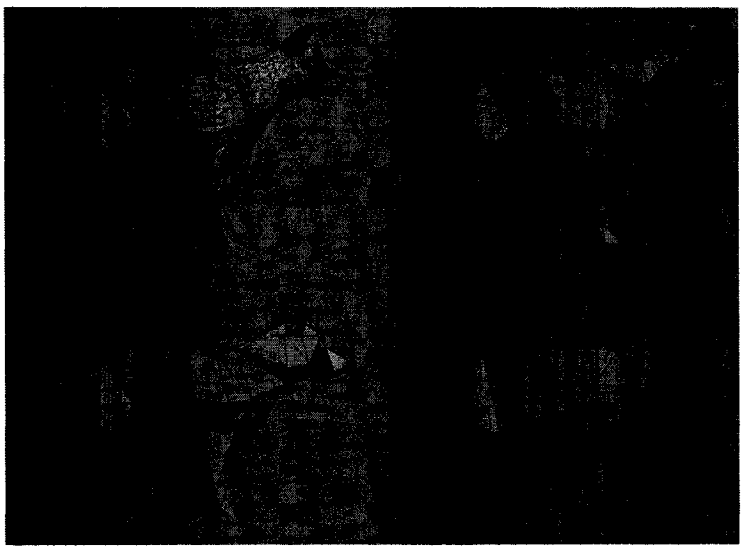


Figure 2-2: 3D modeling with Polygon meshes
[HB04]

Parametric curves and surfaces are another popular way to represent 3D objects. Some objects like spheres are hard to represent precisely by polygons and also we might already have some mathematical formula to represent them. Parametric curves are defined as $p(t) = [x(t), y(t), z(t)]$ and parametric surfaces are defined as $p(u,v) = [x(u,v), y(u,v), z(u,v)]$. They work well with smooth surfaces and are much more compact than polygon meshes.

An **implicit** surface representation is when we define an object by a mathematic formula: F(x,y,z) = 0. Implicit surfaces are the best for ray tracing methods because of the very easy intersection test between a ray and an implicit surface. They defined objects clearly and precisely. The only problem which as well makes this approach the least commonly used one, is that finding F for a given surface is not always easy. To overcome this we can divide object to other objects with known F functions but a whole new problems would come into view.

2.1.2 Illumination

A *lighting model* in Computer Graphics is used to calculate the color of an illuminated point of an object. Projecting all 3D objects of the scene to the 2D screen, lighting model defines the color of each pixel of the screen. Any object in the scene that emits radiant energy is a light source and can affect the final pixel color. The final color depends on interactions between incident radiant energy and material of the object which means light source properties and object properties. An accurate light model is too complicated for today's technology so estimations are used. The most popular light model divides light to **ambient**, **diffuse**, and **specular** light.

Ambient light, that is the same for all the objects, sets a general brightness level for a scene (like light effect of the sun). We can add an intensity parameter, $k_a * I_a$ to light calculation of every objects where k_a is ambient coefficient depending on object material properties.

Diffuse reflection for a surface can be estimated by assuming that light is scattered with equal intensity in all directions independent of the viewing position. Therefore,

$$I_{diff} = k_d I_l (N.L)$$

Where k_d is the diffuse reflection coefficient depending on object material properties, I_l is the light intensity of the light source and N.L is cosine between the normal vector of the point (N) and the vector to the light source (L).

Specular reflection (Figure 2-3) or a highlight can be seen on the shiny surfaces which is the result of total reflection of light in a concentrated region on the surface of an object. There is a famous specular light model called the *Phong* specular-reflection model which is used widely both because of its good estimation of reality and simplicity. In the Phong model specular reflection depends on viewing condition, intensity of light source, and reflection vector:

$$I_{spec} = k_s I_l (V.R)^{n_s}$$

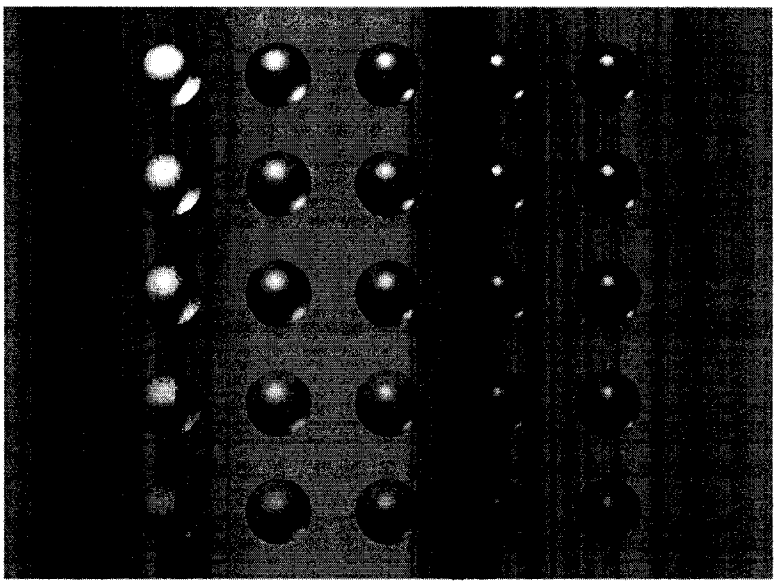


Figure 2-3: Specular light
Left-top has the strongest specularities and bottom-right has the least [So08]

where V is the viewing vector, R is the Reflection vector and n_s is a specular reflection exponent.

Therefore, all together, illumination at a point is determined by:

$$I = k_a * I_a + k_d * I_l(N.L) + k_s * I_l(V.R)^n_s$$

2.1.3 High-Quality Rendering; Ray Tracing vs. Radiosity

One of the popular method of rendering is **Rasterisation** in which each pixel of the 2D screen is normally belongs to let us say a polygon of a 3D object which the color of this pixel can be correspondingly assigned based on the color of the polygon and lighting effects. The process of determining this color is called rasterisation.

But, today's high-quality rendering is done mainly by using two popular algorithms: Ray tracing and Radiosity.

Radiosity is an algorithm that simulates diffuse inter-reflection between patches or small surfaces. It is a method that traces the energy emitted from light sources which reflects between objects. Therefore, light emitting is not limited to light sources and light sources are not estimated by point light sources. Here, there is no need for ambient light because of the fact that every surface emits energy (as in reality) -- so we can produce high quality results after enough number of iterations. The first iteration differentiates all those patches that receive light directly from light sources. In second pass, these patches are considered as new light sources. Therefore, after enough steps a high quality result can be produced.

In comparison, **Ray Tracing** treats light path as a straight line passing through object such that one part of it reflects and one part transmits through the object (again a good estimation of reality). In 1968 Arthur Appel *et al.* [Ap68] introduced the ray casting idea. Ray casting is to shoot rays from the eye, one

per pixel, and find the closest object blocking the path of that ray. Followed by ray casting idea, Turner Whitted [Wh79] made a very big step in producing high quality images in 1979, by introducing an illumination method using ray tracing.

In Turner's algorithm rays are followed even after they hit an object in the scene. Depending on the material of the object, the ray will divide into refraction, reflection, or and shadow rays. Reflection and refraction rays are treated the same but with different origins, directions and intensities. Shadow rays are those who between their intersection point and their origin there exists another obstacle. Many high quality images have been produced since the idea of ray tracing; effects like reflections and shadows are easy to implement and results are adequately realistic. The only serious disadvantage of ray tracing is that even if we follow rays for only 3 or 4 hits it will be terribly time-consuming.

The quality of the result depends on number of times a ray is followed after changing its direction. Ray tracing works very well with specular lights because of the handling of reflected rays (note that they are calculated in the first pass).

Both Radiosity and Ray Tracing algorithms are time consuming and can only be used limitedly. A Few passes for Radiosity is acceptable and rays can be followed few number of times. Both of these models handle shadows easily, whereas Radiosity is good for diffuse lights and Ray Tracing is good for specular lights. Also, Radiosity works well for the scenes consisting of a set of polygons while ray tracing works better with the scenes consisting of other geometric objects.

2.2 Hair simulation

The research that have been done in hair simulating can be divided into two groups: before 1993 and after 1997[6]. Although the importance of early researches cannot be ignored, most of the progress was made after 1997 due to the fast growth of computer hardware. Before 1997 hair rendering could not be done anything near real-time because of the great deal of data and rather poor technology; even offline hair rendering was so primitive that all of the animations lacked good hair models. As mentioned before, perhaps the first time that individual hair strands actually rendered in an animation was 2001 in the *Monsters Inc.* animation. Heroes and heroines in early animations had static hairs and individual hairs were hardly distinguishable. Along with hardware progression and new methods of hair modeling and rendering, hair has become a quality factor of today's animations and increasingly in future video games.

There are 3 different approaches to modeling hair: Physics-based, Image-based, and Geometry-based [WaB+07].

Physics-based hair modeling is trying to represent hair by an object with already known physical characteristics such as spring which we have certain formula for its behavior. In 1992, Anjio *et al.* [AUK92] introduced a physics-based model, using cantilever beams[7] to represent strands.

In physics-based methods some parameters are defined for local and global shape of hair and then the model would determine all the hair strands; *Local Parameters* such as curliness, waviness, wetness, cleanness and *Global*

[6] It might be important to notice this gap in hair simulation area where no research has been done, possibly because of the fact that not until recently technology could let us worry about objects as fine as hairs.
[7] A straight beam with a one-sided and fixed support.

Parameters such as gravity, wind pressure, and other forces that affect hair shape. Physics-based methods are good for dynamic motion of hair, because they are able to generate hair model based on a set of parameters that modifying them with respect to environmental changes would generate a different model; therefore, dynamic motion of hair can be simulated based on physical equations rather than manually. **They do not need much user effort**; the only user task is to define some sets of parameters. The problem of these methods is that the output model may not be the exact expected one. This is mostly because of the fact that the true behavior of hair is hard to be defined by physical equations and approximations cannot explain this behavior well enough and as a result **user control** over the shape of the hair **is not high**.

Image-based hair modeling is trying to reconstruct hair from different photographs of different viewports. In image-based hair modeling the problem is that identifying a single hair strand is very hard and usually impossible, even with high resolution photographs. In practice, a group of strands rather an individual strands can be identified. There are few researches in this area, again both because of the unique nature of hairs and because of the necessity of non-realistic/cartoon hair models in animations and games. **User control and effort are both low.**

Geometry-based hair modeling is widely used in today's technology. Geometry-based models provide some tools for the users to design hairs **interactively**. Compared with physics-based and geometry-based models, **they give users more control** and consequently they need **more user effort**. So far, these methods are best in producing high quality and natural/nice looking hair.

Geometric-based Hair Modeling and Rendering provides some tools for the users to design hair interactively in which hairs are represented using geometric objects. Comparing with physics-based and image-based models, they gave

users more control and therefore more effort. It would be a tedious work if each and every strand had to be designed one by one. Therefore, various methods are used to make this easier which will be discussed later in this chapter while focusing on geometry-based hair modeling because of the variety of hairstyles that can be rendered with geometry-based methods.

As mentioned before, it is very time-consuming to model 100,000 hair stands one by one and place 100,000 hair strands on the scalp of a head; therefore, there should be methods for generalization. There are three major approaches in geometry-based hair modeling and rendering area; considering hairs as **cluster** of hairs, **implicit** hair modeling, and **sketchy** hair modeling.

2.3 Cluster Hair Modeling and Rendering

Hairs do not act independent of each other. Adjacent hairs usually form wisps or clusters [WS89], due to this fact many algorithms consider hairs as cluster of hairs [BA+06, Bek+03, BMC05, CK05, CS+99, KiN02, WS89, XY01, YX+00, ZW06]. Obviously, the more clusters we have the more accurate our design will be, yet this does not mean that modeling each and every hair gives us the best approach; as in [YCJ02] they modeled each and every hair strands individually but then strands were pulled together to form wisps. There are three major approaches in this area: **Key Strand** methods, **Generalized Cylinder** methods and **multi-resolution** methods.

2.3.1 Key Strand Methods

The idea of these methods is to define some number of strands as key strands (typically 10) and all the other strands would follow their master strand. By

following the master strands we mean all properties of an ordinary strand such as length, orientation, texture, color, curliness, etc., are the same as the master strand except for its position. Therefore, we will have some number (typically 10) of wisps containing one master strand and some ordinary strands, such that all of the strands in a wisp are the same. This method has first introduced by Watanabe and Suenaga in 1989 [WS89]. Number of wisps, number of hairs in a wisp, hair length, hair thickness, and hair color were the inputs of their system. Various hairstyles could be created by changing these parameters, but parameters for defining wisps and hair strands have to be entered manually.

For hair modeling using key-strand methods four important questions should be addressed:

1- How do we design our key strands?
2- How do we select our wisps on the scalp of the head?
3- What are the positions (or distributions) of normal strands?
4- How should we avoid having unnatural hairs caused by wisps of exactly same strands?

In 1999, Chen *et al.* implemented a system for producing various hairstyles [CS+99]. They used **trigonal prism wisps** where trigonal prisms in one wisp are linked by three 3D B-splines (Figure 2-4-a). They used 2D arrays to define the distribution of hair strands in a wisp (Figure 2-4-b). Each wisp can have at most 256 hair strands. Therefore for having 100,000 hairs users should define at least 390 wisps (which is 3*390 lines).

This would take hours to do; so to overcome this problem, they divide the skull into eight layers each consists of four "quarter rings" and assume that the shape of the wisps within the same "quarter rings" is the same, then their system can generate hairstyles by putting few curves on the skull.

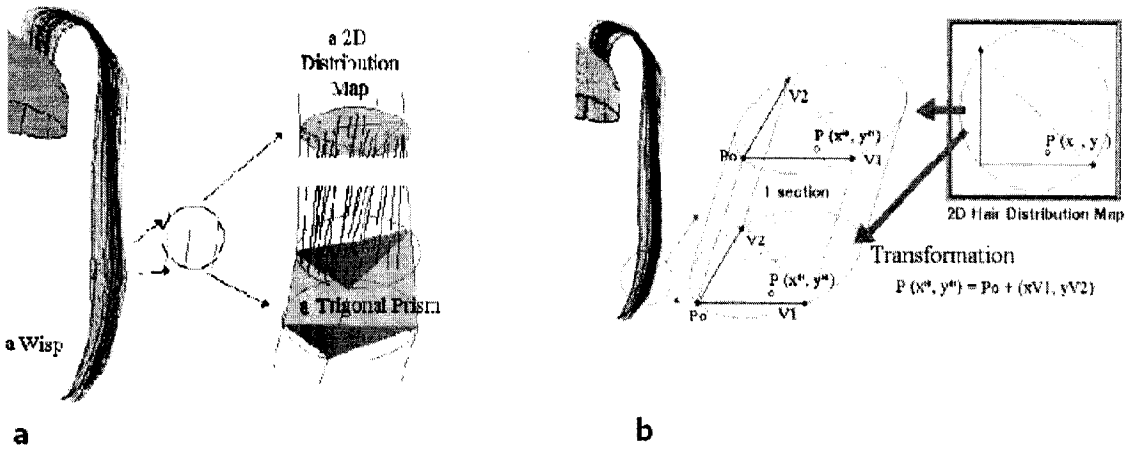


Figure 2-4: A system of 3D hairstyle synthesis based on the wisp model
a) Trigonal prism wisp model b) The relationship between a wisp and its 2D hair distribution map [CS+99]

Perhaps hair with the best quality has been produced by Choe and Ko in 2005 [CK05]. They used a statistical wisp model in which each wisp consists of one master strand and some number of "member strands" and the shape of a wisp is a *Generalized Cylinder* (refer to section 2.3.2).

The geometry of a single strand in their model is based on the fact that hair strands of a single person resemble each other. For each hairstyle their program produces a prototype consisting of some number of splines, from which all different master strands will be produced (see Figure 2-5).

Users need to define the number of wisps (n) and the number of hair strands (m). The root positions of master strands are distributed evenly over the scalp. Wisps are formed by Voronoi diagrams and within each wisps m/n number of root positions for member strands will be defined randomly. A Voronoi diagram is a partitioning of the space with respect to a set of points such that each cell

contains one and only one point and all other positions in the cell are closer to the cell's main point than any other points [BK+00].

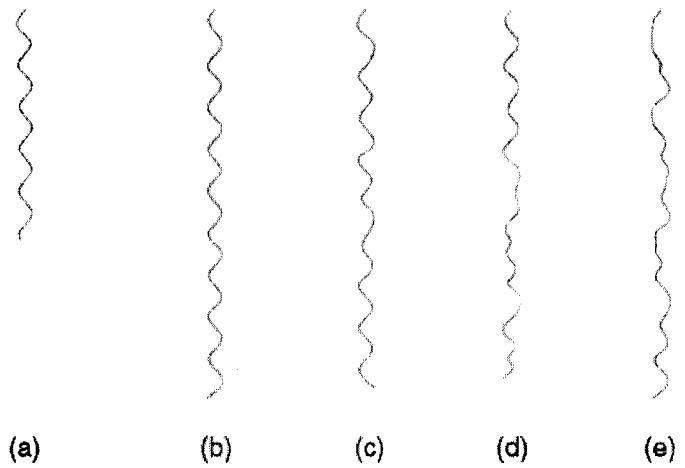


Figure 2-5: Different master strands produced by one single prototype
The prototype variations (b, c, d, e) of the prototype (a) [CK05]

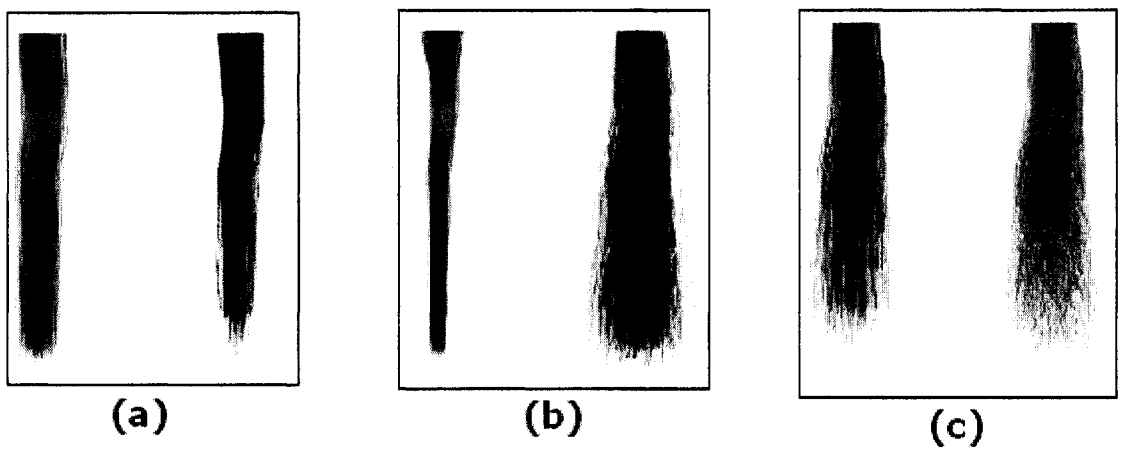


Figure 2-6: Parameters of a wisp
Wisps generated by two different (a) length (b) radius (c) fuzziness parameter [CK05]

The degree of similarity among member strands in a wisp is based on 3 parameters: length distribution, radius, and fuzziness value (see Figure 2-6). Therefore, member strands are formed from master strands and by applying these three functions.

Figure 2-7: Hairstyles produced by statistical wisp model
Final row is hairstyles inspired by real one [CK05].

This method is able to produce fairly realistic hair (Figure 2-7), however, not in real-time. Also, another drawback is that their model lacks the ability to handle all sorts of hairstyles especially those not having a uniform model for overall shape of the hair partly because of the fact that their wisp size cannot vary and partly because of the trade-off between user-effort and user-control.

Another similar method is the method introduced by Zhang *et al.* in 2006 [ZW06]. They also generate hairs based on key-strand method. They used same triangles that form the scalp of the head, as locations of wisps, in the sense that users can choose from 1 to as many triangles as they want from the scalp of the head model to verify the boundaries of wisps. Length and distance Gaussian-based distributions are two parameters that differentiate member strands of one wisp so that hairs could appear more realistic. In contrast to Choe and Ko's method, their method can handle a variety of hairstyles but it lacks realistic hair rendering.

2.3.2 Generalized Cylinder Methods

Another way of modeling a large number of hairs is to define more than one level of abstraction; we can define hairs first in cluster level and then add some details to each cluster to model hair strands. *Generalized Cylinder* seems to be reasonable for this matter because each individual hair can be presented by a Generalized Cylinder.

Generalized Cylinder:

A Generalized Cylinder (GC) is defined by sweeping a *2D shape* along a *curve* with respect to a *scaling function* [CA+04]. Scaling function change the size of 2D shape or the GC's cross section along the curve or GC's axis (see Figure 2-8).

As you can see in Figure 2-8, GCs seem to be a very close approximation for hair strands; cross section of hairs can be estimated by ovals (GC's cross section) and strands themselves can be represented by curve (GC's axis) and thickness of a strand decreases along the curve (GC's scaling function). Yet, it is not easy to

define each and every hair strand by GCs which bring Yang *et al.* [YX+00] to the idea of "cluster hair modeling using Generalized Cylinder".

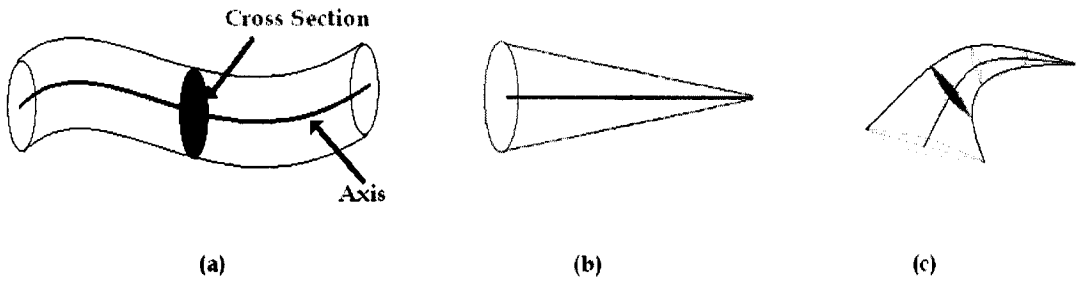


Figure 2-8: Generalized Cylinder
(a) GC whose cross section is an oval, axis is a curve, and scaling function is 1.
(b) GC whose cross section is an oval, axis is a line, and scaling function varies.
(c) ©GC whose cross section is arbitrary, axis is a curve, and scaling function varies.

Their main consideration was to have a "compact representation" for having high-quality hairs without defining them explicitly. For this reason, they used volume density model. Volume density model has first used by Perlin [PH89] to produce a furry object defined as a soft object. Yang *et al.* modified their model and combined it with Generalized Cylinder [YX+00]. The density values on the *base cross section* (Figure 2-9-c) are produced with a pseudo-random function. The pseudo-random function generates a set of points as centers of hairs (Figure 2-9-a), then the density value of each point or center of hair will be defined by a Gaussian distribution (Figure 2-9-c). Now, the density value at any point can be calculated by projecting the point onto the base cross section plane. In other words, we have a special GC whose sweeping area (cross section) is the base cross section (with all the dots as centers of hairs). In summary Yang *et al.* represent a cluster of hair by these data sets:

- Axis curve

- Cross-section-shape curve
- Base-density map
- Hair color and light properties

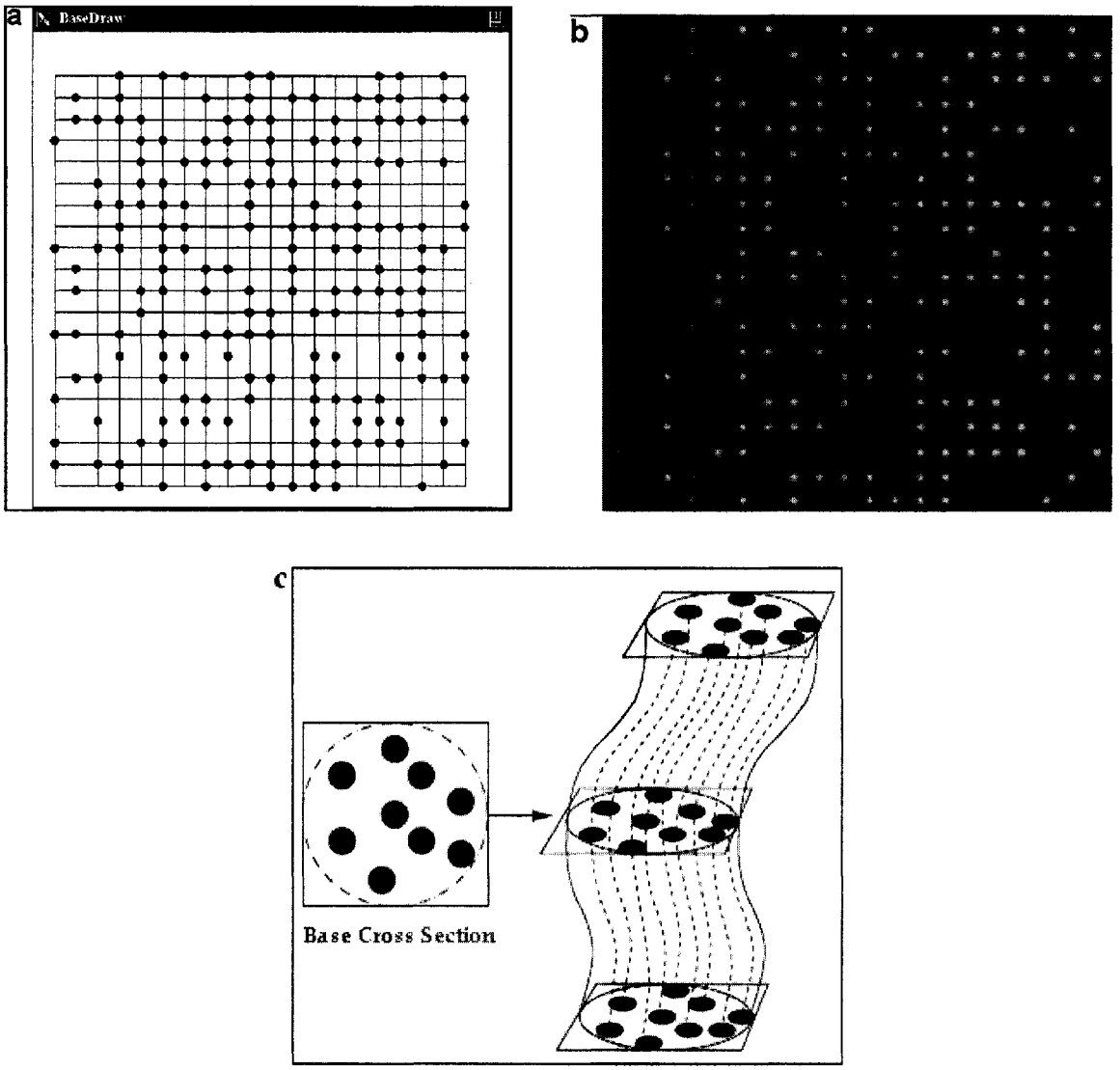


Figure 2-9: Hair modeling with GCs
a) Distribution points on a gird map [YX^{+}00]
b) Base density map [YX^{+}00]
c) Base Cross Section and hairs within a cluster (GC) of hair [YX^{+}00]

They provided a modeling tool based on their cluster hair model which needs reasonable amount of user effort to design a hairstyle. Their model has seven modules: cluster drawing and editing, cross-section-shape editing, base-density map editing, hair color and properties, hair arrangement, and shape preview (see Figure 2-10).

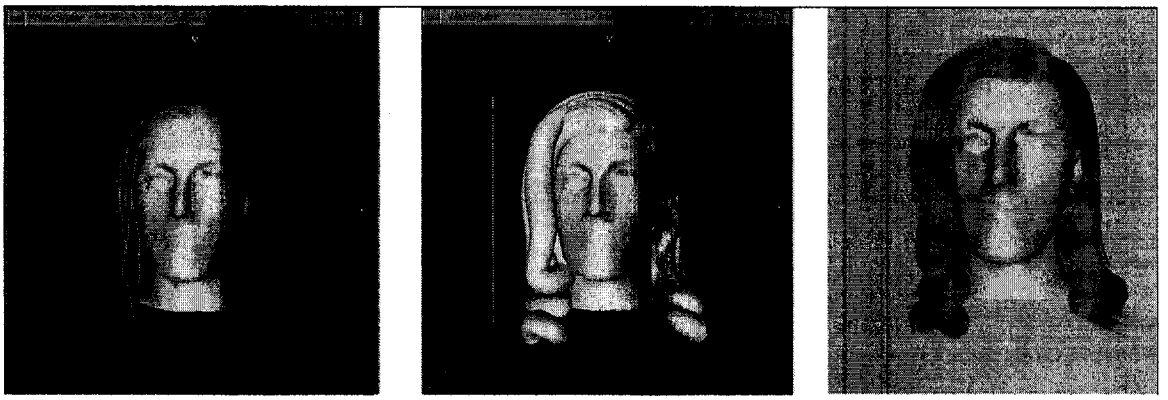
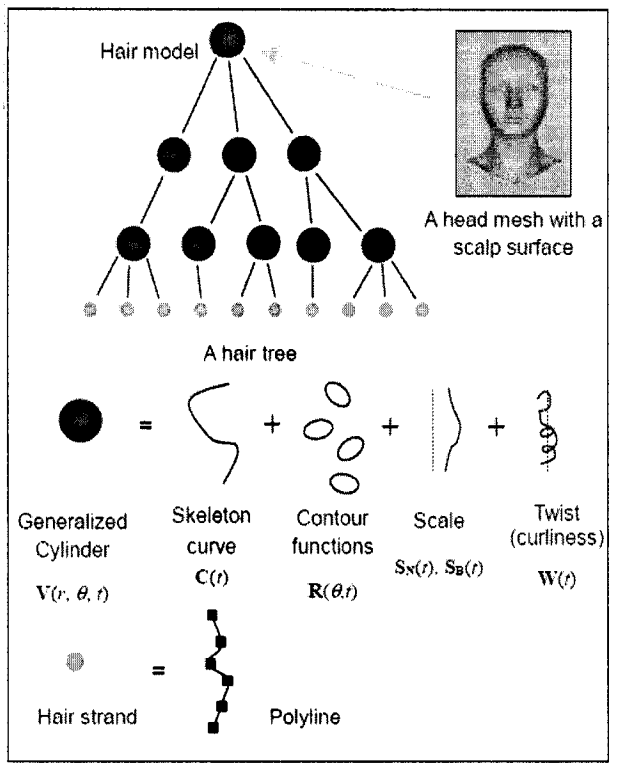


Figure 2-10: Cluster hair model steps of designing
[YX+00]

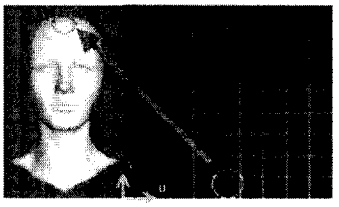
2.3.3 Multi-resolution Methods

Multi-resolution hair modeling was introduced by Kim and Neumann in 2002 [KiN02]. It is another way of cluster hair modeling but perhaps with more details. A hair model is constructed hierarchically in this method. At the first level of modeling, we have small number of clusters, some of which will be divided later to some number of clusters themselves in level two, and we can continue like this (see Figure 2-11-a).

Clusters are represented by Generalized Cylinders. To place clusters on top of the scalp users need to select a contour (Figure 2-11-b right), then choose its location in 2D *scalp space* (Figure 2-11-b middle), its 3D position will be defined consequently (Figure 2-11-b left).



(a)



(b)

Figure 2-11: Multi-Resolution method
(a) Graphical view of multi-resolution hair modeling[KiN02]
(b) Scalp space and contours[KiN02]

Each parent cluster can be subdivided as many times as users specify to child clusters. One hair strand should be assigned to the clusters of the highest level for each root contour. Then strands of hair would be explicitly rendered. Users can control *alpha values* of strands, the number of strands in each cluster and the number of segments for each strand because they are represented by poly-lines. There is a tradeoff between performance and quality. The more strands and/or segments per strands we have, the more rendering time we need.

Alpha Blending, used by [LTT91] for the first time in hair simulation, is a method to add transparency information to the representation of an object. After alpha blending, color of points of a surface, points of a polygon, or simply a set of points will be:

$$\text{New Color} = \alpha * \text{Old\_Color} + (1 - \alpha) * \text{Background\_Color}$$

Where α, a normalized value between 0 and 1, is the transparency coefficient. $\alpha=1$ means the object is opaque and little amount of α correspond to near full transparent objects.

By presenting a user-friendly tool and a based-on-reality method, Kim and Neumann provided a model with reasonable amount of user control and effort which can also generate nice-looking and semi-natural hairstyles. Also, their model can be used to design almost every hairstyle but it is offline.

2.4 Hair Modeling and Rendering from a Sketch

Modeling hair based on a hand-drawn sketch, is for generating *non-photorealistic*[8] hairs for games and cartoons. This method is based on the fact that we might not need realistic looking hair and we might just look for nice cartoon shaded hairs.

2.4.1 Sketchy Hairstyles

Sketchy hair modeling is a top-bottom approach to hair modeling. It asks users to draw the silhouette of the hairstyle and then they can modify the automatically generated hairstyle to design the desired hairstyle. Unlike cluster hair modeling, sketchy hairstyles are based on the artists' hand-drown sketches for a cartoon character; therefore, they are mostly suitable for non-photo realistic hairstyle images.

Mao *et al.*, has first introduced and implemented sketchy hairstyle model in 2002 [MK+02]. In their system which is completed in 2005[MI05], hairstyles can be designed with very low user effort. First, users need to specify the area to grow

[8] Non-photorealistic images are rendered with NPR (non-photorealistic rendering) techniques which contains all sort of rendering techniques that generate images not realistically.

hairs on the scalp of a 3D head model. Then, they have to specify the partition line; partition line is the line that partition head to front and side/back. After that, users must specify the silhouette of hairstyle for both front part of the head and side part. When the system automatically generates *cluster lines*[9], user can modify the total hair shape (see Figure 2-12).

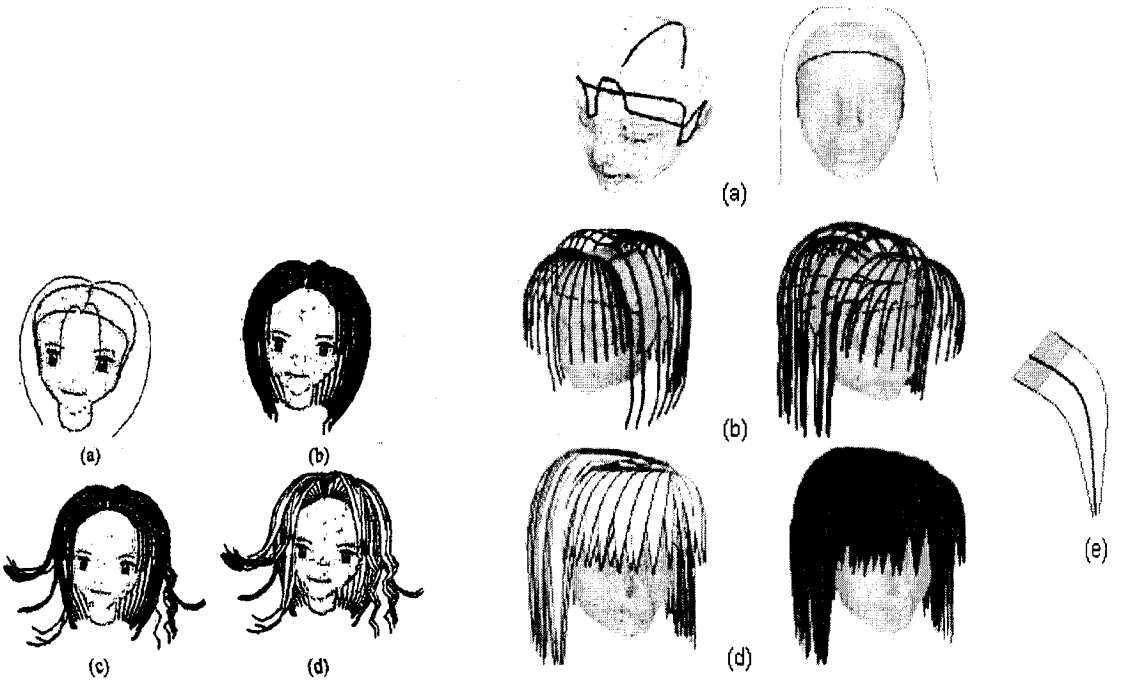


Figure 2-12: Sketchy hairstyle modeling procedure
(a) user specified hair growth area, partition line, and front and side silhouette lines (b) automatically generated cluster lines (c) local modification (d) total result (e) polygon patch assigned to a cluster line (red) [MK+04]

Cluster lines are actually representative curves based on which polygon patches will then be created (see Figure 2-12-e). Cluster polygons are approximately tangential to the scalp and a set of polygons will be assigned to each cluster line of the hairstyle. Cluster lines can then be modified to have more complex hairstyles (see Figure 2-12-c).

[9] Cluster lines (Figure 2-12-b) are defined for rendering purposes. The details of the algorithm can be found in [MK05].

Non-photo realistic hair modeling has also been worked by Cote *et al.* [CJ+04]. Their model assigned some polygon patches to a hand-drown sketch of the desired hairstyle. Polygon patches are defined based on silhouette of hairs.

2.4.2 Physically-based hair styling from a sketch

In 2005, Sugisaki *et al.* represented hair form hand-drawn sketches by an animator [SY05]. They used springs as a physical model for hair strands. With their method one can interactively generate cartoon hairs animation like those that are obtained by hand.

Later, In June 2007, Wither *et al.* implemented a very user friendly hair model. They also used a combination of sketchy and physically-based modeling hair. Users can design hairstyles in 3 steps [WBC07]:

1-Define the scalp area: users need to draw a stroke to define the scalp of the head; scalp is the location in which hairs grow.

2-Draw example strands and volume stroke: After specifying scalp area users have twenty to thirty choices of roots or *guide strand* locations to draw example strands within this area. Users can have only one example strand or as many as they want. If more than one strand has been specified, Voronoi diagram would divide head respectfully. Therefore, some number of guide strand (or master strand) would cover the scalp area. Volume stroke determines the global volume (silhouette) of hair (see Figure 2-13).

Figure 2-13: Physically based hair modeling from a sketch
Left) example strands and volume stroke
Middle) Guide strands right) Result [WBC07]

3-Generate full head of hair: They used the same method as [CS+99] (described in section 2.3.1) for rendering full head.

Note that their method is actually an interface to physically-based hair styling introduced by [BA+05].

2.5 Implicit Hair Modeling and Rendering

Instead of modeling each and every hair individually, we can consider bunch of them as one geometric object, such as layers or Generalized Cylinder. As long as individual hairs aren't represented as an object in the scene, we are dealing with implicit hair modeling. Implicit hair modeling can be done in real-time for it avoids both obstacles of hair simulating; large number of hair strands and rather small diameter of each one of them.

Koh and Huang introduced the first real-time hair model and renderer in 2000 [KH00]. They modeled hairs in strips using NURBS for its accuracy and compact representation, supporting of local and global shape editing, and inherent continuity. Hair strips can be manipulated by users by manipulating parameters

such as location, orientation, and weights for knots. To generate a certain hairstyle, some number of hair strips should be assigned to the scalp of the head (see Figure 2-14).

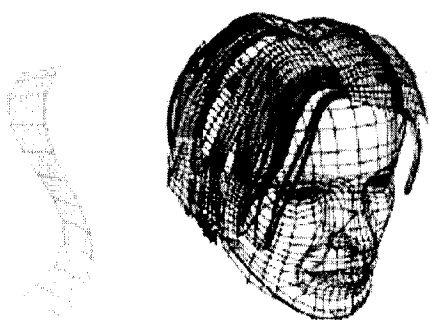


Figure 2-14: Hair Modeling with Strips
Left) a hair strip
Right) all the hair strips [KH00]

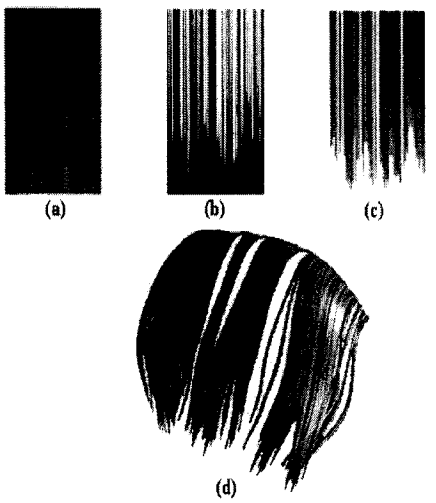


Figure 2-15: Hair Strips Rendering
(a) texture map (b) alpha map
(c) combination of (a) and (b)
(d) result [KH00]

Rendering has been done using *texture mapping* with the Alpha channel; a combination of a texture map (Figure 2-15-a) and an alpha map (Figure 2-15-b) is the final texture for rendering NURBS.

Texture mapping is the mapping of a 1D, 2D, or 3D function onto a surface in space or simply a method to attach images to objects. Now every points are defined as (x,y,z,u,v) where u and v are texture coordinates corresponding to (x,y,z). Texture mapping has many uses such as, surface color, specular reflection, transparency, shadows, surface displacement, etc.

They succeeded rendering hair in real-time. First of all, with just few number of hair strips we can have a fair hairstyle and secondly, different levels of resolution can be selected based on the distance to the viewpoint which is a fast process too. The major drawback of their method is that it is only good for straight hairs. Curly hair would have the problems of collision detection and complicated hairstyles may need too much effort to design NURBS surfaces for.

2.6 GPU

Graphics Processing Unit or GPU is a micro processor used for rendering purposes which has been introduced in 1999 by NVIDIA Corporation. The technical definition of GPU provided by NVIDIA is "a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second."

A GPU can be placed on the video card or directly into the motherboard. It is dedicated to graphics purposes and able to offload all time-consuming and complex graphical calculations from CPU. Using GPU helps us render very complex scenes without affecting performance (see Figure 2-16).

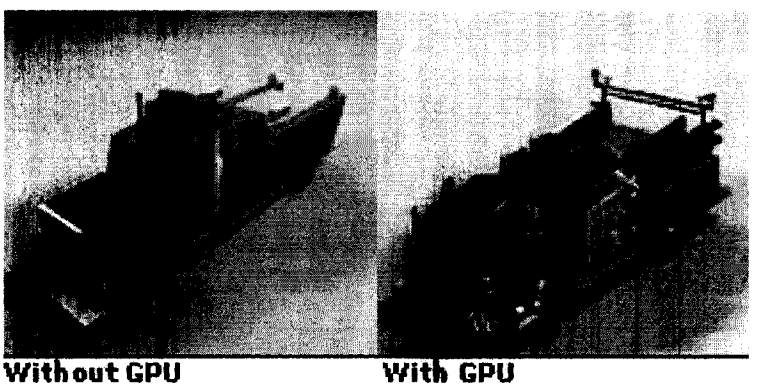


Figure 2-16: Rendering Comparison with/without GPU
[Nv08]

Although GPU is designed for mathematics intensive tasks and can be used for complex calculations about vectors, matrixes, and transformations, it is mostly used for 3D rendering nowadays.

What interests us in GPUs is their ability to create real-time high quality images without overworking the CPU. This is because of the highly parallel structure of GPUs; a GPU is a stream processor which means that a set of data flows through a sequence of steps (Fig 4-3) and a set of output will be produced. GPU performance grows a lot faster than CPU[10] (see Figure 2-17); this is both because of the specialized nature of GPU which makes it easy to use additional transistors for computation (not cache) and also because of the multi-billion dollar video game market [Ng07].

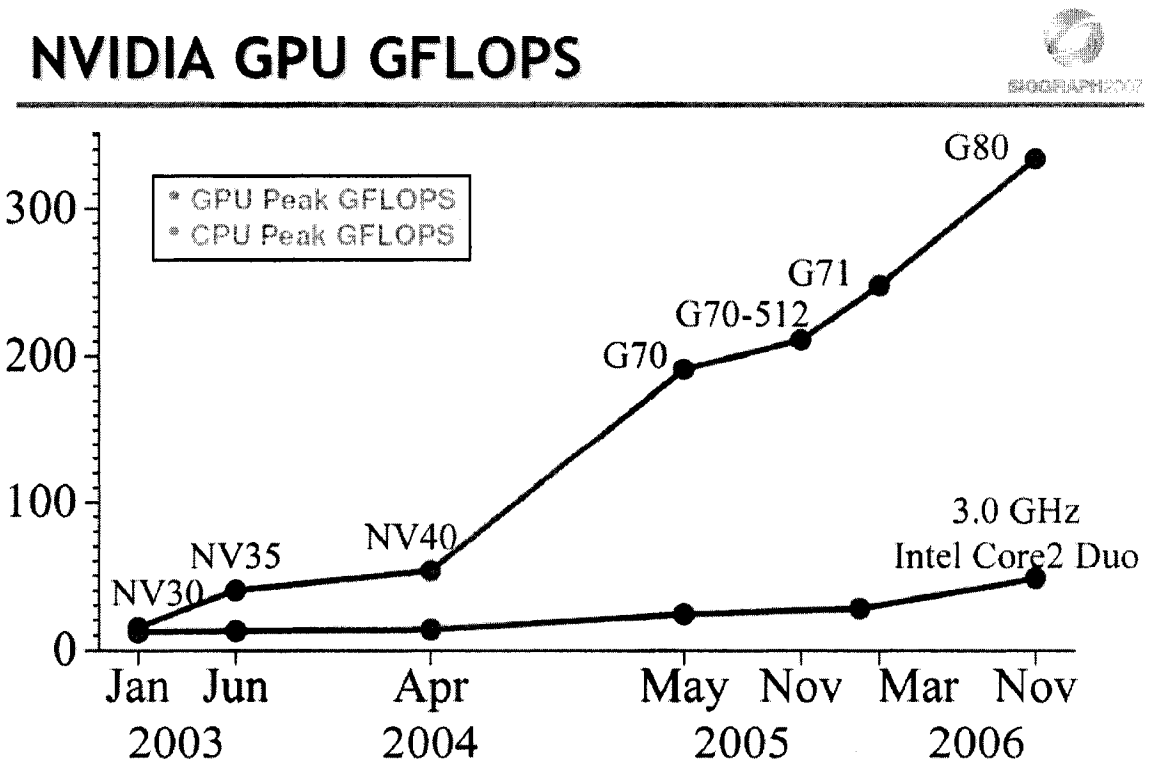


Figure 2-17: GPU performance over time vs. CPU [Ng07]

[10] Performance growth is measured using GFLOPS which is Giga FLoating OPerations per second.

2.6.1 Graphic Pipeline

Graphic pipeline is the model of stages where graphic data go through. As you can see in Figure 2-18 the pipeline input is 3D models and the output is a frame buffer or simply pixels that will be shown on the screen.

The pipeline can be divided into two general stages: one is geometry stage or Lighting/Transformation. Light calculations provide lighting effect on the scene and transformation calculations transfer 3D objects (commonly represented by triangles; therefore, we are dealing with triangles and the corresponding coordinates of their vertices) to 2D screen coordinates. The second stage is rendering in which the area between previous-stage's coordinates will be filled by pixels; each having a specified color (typically RGBA).

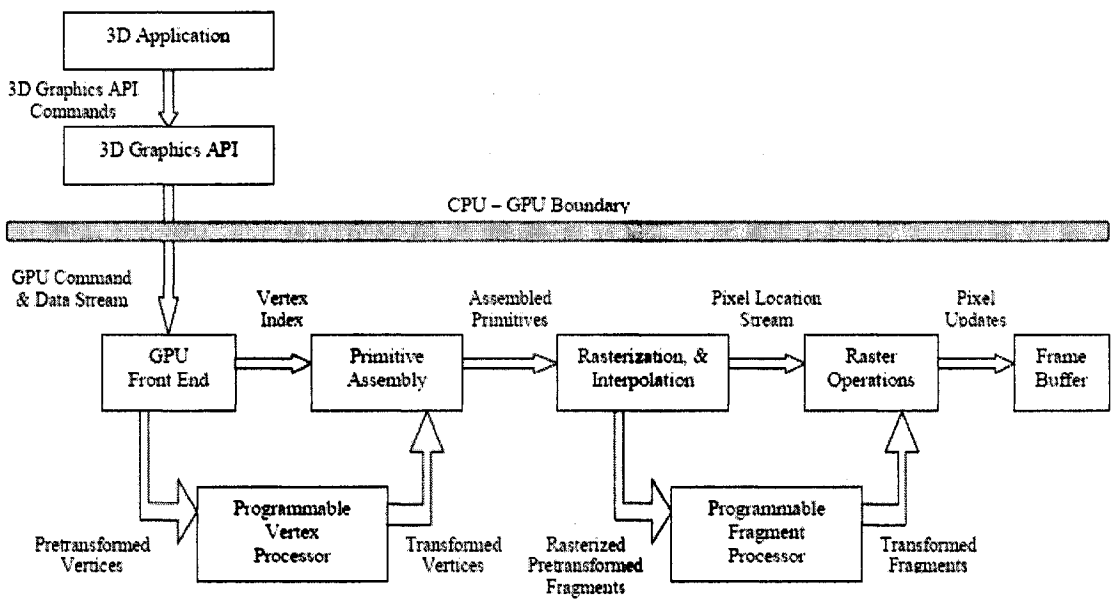


Figure 2-18: Cg's GPU Model [Kir+04]

GPUs have two fully programmable processors; **Vertex** processor and **Pixel** processor (also called vertex shader and pixel shader). Vertex processor involves the operations that occur at each vertex which are vertex and normal transformations; most notably, texture coordinate generation, and lighting calculations. Pixel or Fragment processor is responsible for the final color of every pixel (RGBA). A fragment contains all of the data necessary to update a single location in the frame buffer. Object specifications and rendering attributes are passed to the GPU from the CPU and get modified by the two processors to determine the final color of every pixel of the scene [Ng07].

2.6.2 Hair modeling and rendering with GPU so far

To our knowledge, hair simulation using GPU has done once before [Os07] (there are some methods like [CK05] which used GPU along with CPU but [Os07] rendered hair completely with GPU). Oshita used GPU to render 10,000 hair strands in real-time with a particle-based[11] hair simulation method (Figure 2-19). Although they have modeled hair strand individually, their hair model lacks realism because of the fact that strands are made of simple particles, but they probably presented the best real-time model for dynamics of hair.

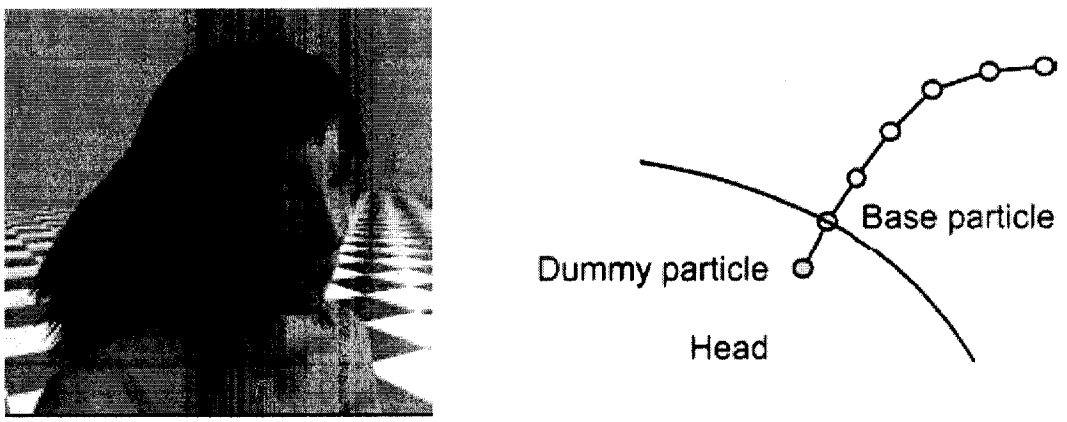


Figure 2-19: Particle-based hair simulation using GPU

[11] Particle systems are the applications which describe one or more objects using a collection of disjoint pieces. (Very useful for objects like fire, clouds, water sprays, etc.) [HB04]

Chapter3. HAIR MODELING

We have seen in the literature review chapter that hairs can be simulated using geometry-based, image-based, or physics-based methods. Image-based hair simulation is hard and inaccurate due to the low quality of images with respect to the thickness of hairs. A physics-based approach is also inaccurate because of the complex nature of hairs. Among geometry-based approaches, we chose cluster hair simulating for the quality of hairs simulated by this method and among cluster hair simulating approaches key-strand-hair modeling was chosen. Generalized Cylinder methods do not simulate hairs individually and multi-resolution methods are quite similar to the key-strands methods with the ability of having multi-resolution for the hair model. Yet, the hair modeling approach that will be introduced here to generate the whole hairstyle (based on cluster hair modeling using key strands) also have the ability of having multiple resolutions. The key fact about key strands hair simulation is that hair strands are defined based on only a little number of key strands which need to be defined directly and all the other will be generated automatically.

Beside the quality of a hairstyle, another important factor of hair modeling is the ability of the model to generate different sort of hairstyles. Key strand hair modeling seems to be reasonable for this because of its high user-control.

In this chapter, first we will explain our version of the key strand hair modeling method, then hair realism issues will be discussed, and finally some results of the model will be demonstrated.

3.1 Modeling hair using Key strand methods

Figure 3-1: Hair modeling steps
Left) Designed key strands on the scalp
Right) Generated hairstyle with all the ordinary strands

As we discussed in the literature review, the key strands hair modeling method is to define a set of key strands and generate ordinary strands based on them; ordinary strands does not have any property of their own and they inherit all the properties from the key strands assigned to them. This way, we only have to design key strands (typically 10) and the hair model will be generated automatically[12] (see Figure 3-1).

[12] Although the hair model consists of both ordinary and key strands, number of key strands is negligible.

Note that all of the images of this chapter are merely to demonstrate our version of key strand **hair modeling** algorithm, i.e., the strands of hairs are all rendered as 3D points in 3D space with a solid color and no special rendering methods or illumination calculations have been used. Also, it is important to note that these images are generated offline. For instant, Figure 3-1-right consists of 100,000 hairs and it took about 70 seconds to render it using OpenGL and a 2.99 GHz CPU.

As it mentioned in chapter 2, for hair modeling using the key-strand method four important questions should be addressed:

1- How de we design our key strands? **(Section 3.1.2)**
2- How do we select our wisps on the scalp of the head? **(section 3.1.3)**
3- What are the positions (or distributions) of ordinary strands? **(section 3.1.4)**
4- How should we avoid having unnatural hairs caused by wisps of exactly same strands? **(section 3.2)**

3.1.1 Definitions

Before introducing our model, we should define some simple attributes of the model shown in Figure 3-2 which is a top view of the head shown by a circle:

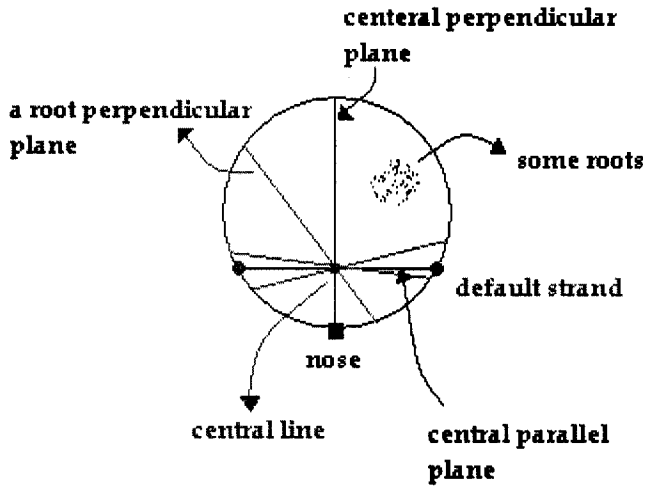


Figure 3-2: Top view of head model

Root: both key strands root and ordinary strands root are considered as root in a 1D array; $\mathbf{R}_i$.

Default strand: an arbitrary strand in the right side of the scalp which is in front of the right ear. This strand is actually used to define a local axis for the head model to help us deal with key strands orientation later. Another property of this strand is that user can think of it as the exact same copy (even the same orientation) of the spline that s/he designed using a Bézier curve.

Central perpendicular plane: the plane that divides head into left side and right side (again it is used to define a local axis and can be any approximate plane).

Central parallel plane: the plane perpendicular to central perpendicular plane contained default strand's root.

Central line: the intersection between central parallel plane and central perpendicular plane.

Root normal vector: every hair root has a normal vector which is either equal to the average of its adjacent triangles' normals (in case of being a triangle vertex) or equal to its container triangle normal vector; **$N(R_i)$**.

Root tangent plane: the plane which contains R_i and perpendicular to $N(R_i)$ is called root tangent plane; **$T(R_i)$**.

Root perpendicular plane: the plane contains the root and central line.

Root main tangent vector: intersection between root tangent and perpendicular plane starting from the root.

Root second tangent vector: vector starting from the root which is perpendicular to root normal vector and root main tangent vector.

Root local XYZ axis: root normal vector, root main tangent vector, and root second tangent vector.

3.1.2 Designing Key Strands

Designing a key strand consists of the following actions:

1-Locate the key strand's control points (Figure 3-3-left)

2-Set the key strand's length and color.

3-Locate the key strand on the scalp (Figure 3-3-right)

4-Set the radius of the strand (will be discussed in chapter 4)

5-Define the key strand's orientation (Figure 3-4)

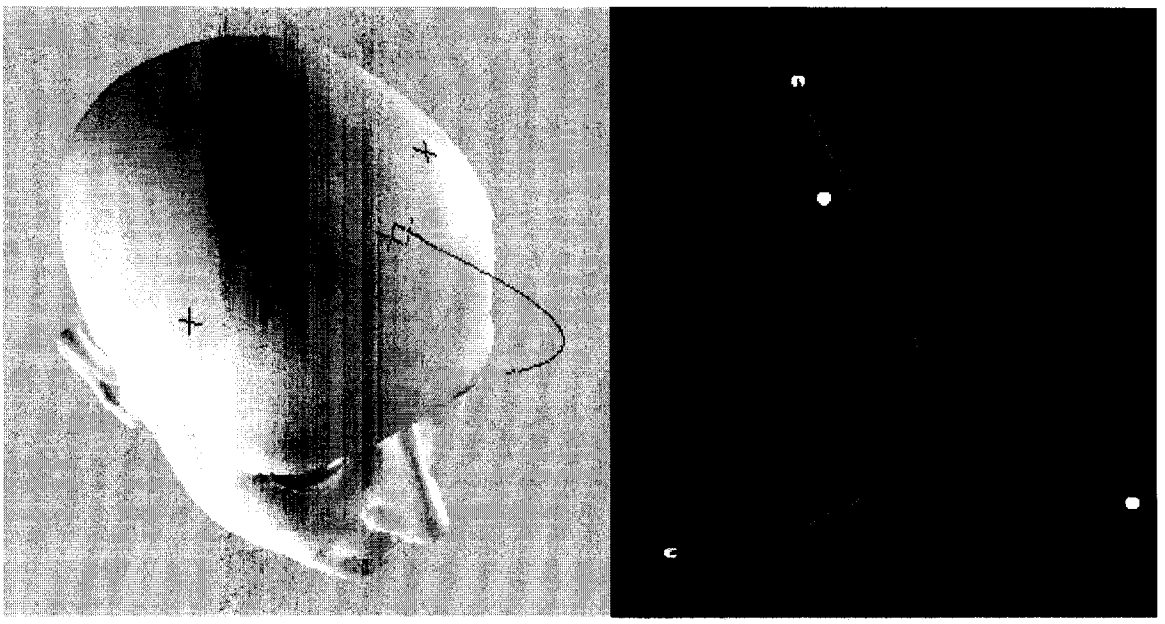


Figure 3-3: Defining key strand locations and splines
Left) key root locations with the first key strand assigned to the first root location
Right) Bézier spline defining the first key strand curve

A very common way to define objects with shape of a curve in Computer Graphics is to use *parametric curves*[13], one of which is *Bézier curve* that we used in this project.

Bézier curve: a parametrical curve that can be controlled by a set of control points based on location of which the infinitely continuous curve can be manipulated (refer to appendix C for more information).

Resolution: the number of points that form a Bézier curve or the number of cylinders that form a single strand.

Number and location of the control points of a typical key strand's Bézier curve determine the number of twists in the strand. Therefore, depend on how curly is the strand, the Bézier curve should be designed (for example, spline of Figure 3-

[13] When x,y, and z components of a curve are defined in terms of a fourth variable, t:
Curve = (X(t), Y(t), Z(t)).

4-right is a curly hair formed by 12 control points). The first control point of the Bézier curve is the root of the strand.

Initially, the 2D curve representing the key strand is put on the scalp in a way that its plane matches the **root perpendicular plane** (Figure 3-4-Left-1) and its first control point matches the root. This may not be the exact desired orientation for the strand; therefore, we are able to rotate the key strand around root local XYZ axis; root normal vector, root main tangent vector, and root second tangent vector, until achieving the desired orientation for the key strand (see Figure 3-4-Left).

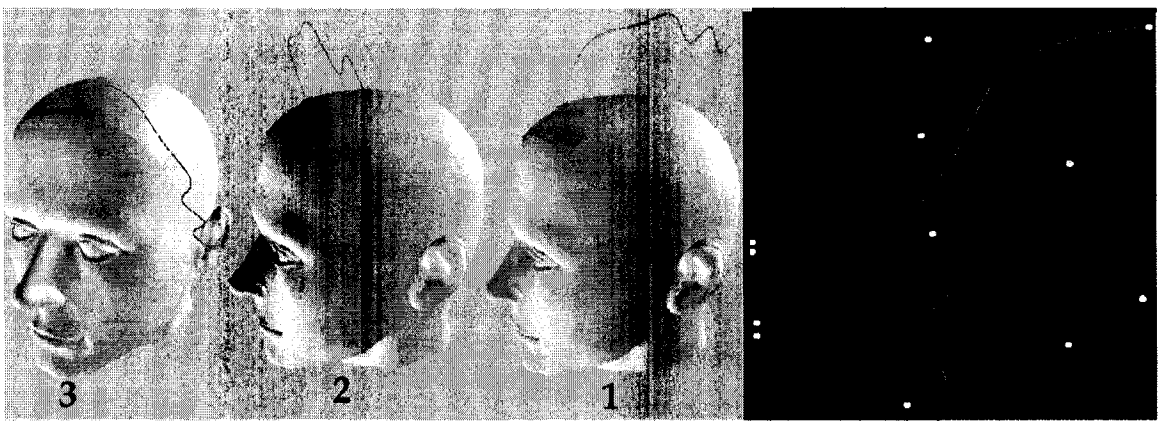


Figure 3-4: Specifying orientation of key strands
Right) Key strand curve Left) 1-Initial location of the key strand 2-rotating along root local XYZ axis 3-Final location of the key strand

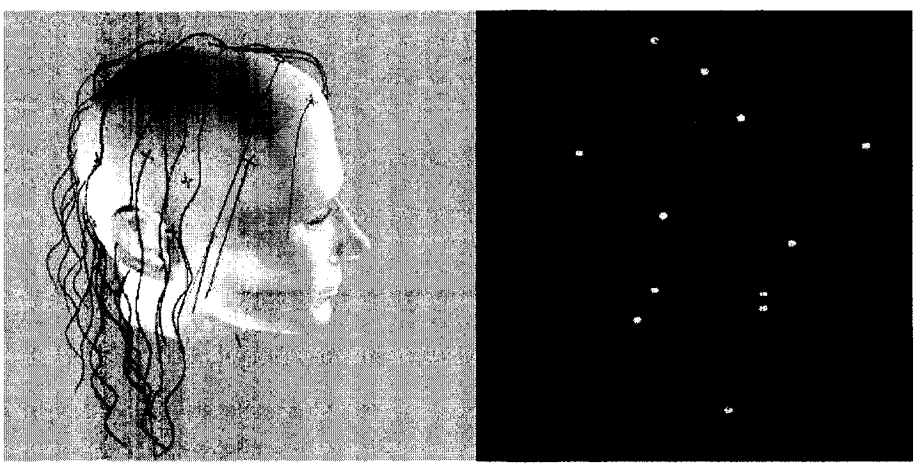


Figure 3-5: Designing key strands

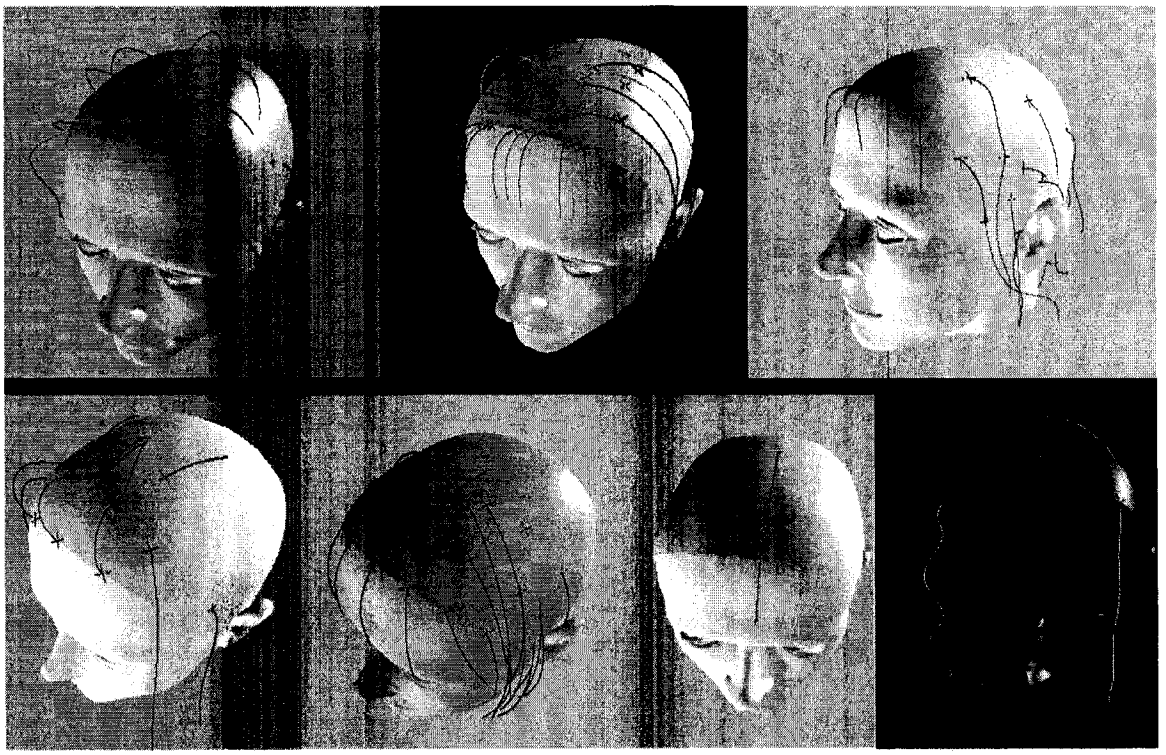


Figure 3-6: Key strands of sample hairstyles

Doing exactly the same, any number of key strands can be defined all over the scalp (see Figure 3-5 and 3-6). Typically users need to define 10 key strands which are quite a few compared to the great number of human hairs on the scalp.

With this way of defining and orienting key strands all sort of hairs can be designed except for space-3D hairs (those that curves of which cannot be drawn on a plane).

3.1.3 Positioning wisps on the scalp of the head

One possible solution is what Choe and Ko have used which is to position wisps uniformly (every wisp has the same size) on the scalp [CK05]. Drawbacks of this approach have been demonstrated in chapter 2. In contrary to their method, if

we let **different wisp sizes** as in [ZW06], user-effort will be sometimes more but instead user-control over the hairstyle will be increased and as a result more complicated hairstyles can be designed.

The human head is commonly represented by triangle meshes. Therefore, we have some number of points which form the scalp after triangulation. In this thesis, the same set of points has been used as potential locations for the root of **key strands**. There are about 1000 points defining the scalp distributed almost uniformly around the head which makes it possible to choose the root of key strands almost everywhere on the scalp (see Figure 3-7).

The root location of the key strands which is initially located on the middle-top part of the scalp, as denoted by a cross in Figure 3-7, can be relocated to choose the **root location** of key strands. For each of these roots, a spline should then be defined to represent key strands. After defining all key root positions, boundaries of wisps can be calculated; ordinary hair strands are members of the wisp of the nearest key strand to them. In other word:

> Ordinary strand O_i belongs to a wisp W_j defined by a key strand K_j for i and j where distance between roots of O_i and K_j are minimum.

This can be called a **Semi Voronoi decomposition** because we do not consider length of the path from root of O_i to root of K_j on the scalp instead we define wisps based on actual distance between O_i and K_j.

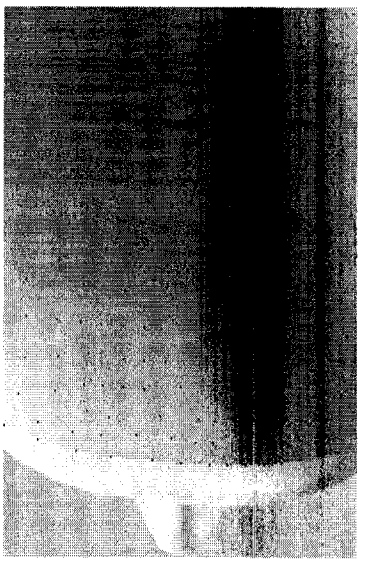


Figure 3-7: Selecting root locations of the key hair strands among about 1000 possible choices

With this method we can model most of the hairstyles, including those with bizarre deformations, spending reasonable effort.

3.1.4 Ordinary hair roots distribution

The scalp of the head is defined by some number of triangles (about 1000 in our model). Now that we defined key strand hair roots we can use the same set of triangles defining the scalp and locate ordinary hair roots based on area of each triangle. Suppose A_i is the area of **T_i (i^{th} triangle)** then, area of the scalp is $A = A_1+A_2+...+A_n$. Now suppose we want to have m ordinary strands (typically 100,000) therefore, T_i has $(A_i/A)*m$ number of ordinary hair roots. Ordinary roots are positioned randomly inside each triangle based on the fact that triangles are small in size; therefore gaps are not probable. We used bilinear interpolation:

$$R_{ij} = \alpha\,(\beta T_{i\_x} + (1-\beta)T_{i\_y}) + (1-\alpha)T_{i\_z} \qquad \text{for j from 1 to (Ai/A)*m}$$

where α and β are random numbers from 0 to 1. And R_{ij} is the j^{th} root of i^{th} triangle of the scalp.

This way we can position m roots on the scalp in a semi-uniform way; this approach is better than uniform distribution because naturally, hair roots are not following any particular distribution and seem to be random. Plus, it is better than choosing completely random locations if we want to avoid gaps. Here, the probability of having some areas with no hairs is so low and even so, these areas will be so small (at most the size of a triangle in the mesh).

Here, any number of wisps can share a triangle's ordinary roots. This distribution is just a fair distribution of roots over the scalp without affecting by wisps definition. Whereas most of the cluster hair modeling methods [CS+99, CK05, YX+00, KiN02] defined wisps and then defined some number of roots in those wisps but in this thesis, hair root locations and wisp locations are independent.

3.1.5 Assigning general parameters of the hairstyle

Up to this point, we defined all the key strands and positions of ordinary roots. To generate the whole hair model, some general parameters need to be assigned as well. General parameters can affect hair shape widely and can be modified after viewing the results to reach the desired hairstyle. General parameters are:

1-Total number of hairs (Figure 3-8).
2-Hairs maximum and minimum length (will be discussed in the next section).
3-Tidiness value (will be discussed in the next section).
4-N-Influence value (will be discussed in the next section).
5-Blending flag (will be discussed in chapter 5)
6-Parameters involve rendering (will be discussed in chapter 5)
7-Intelligence flag(will be discussed in the next section).

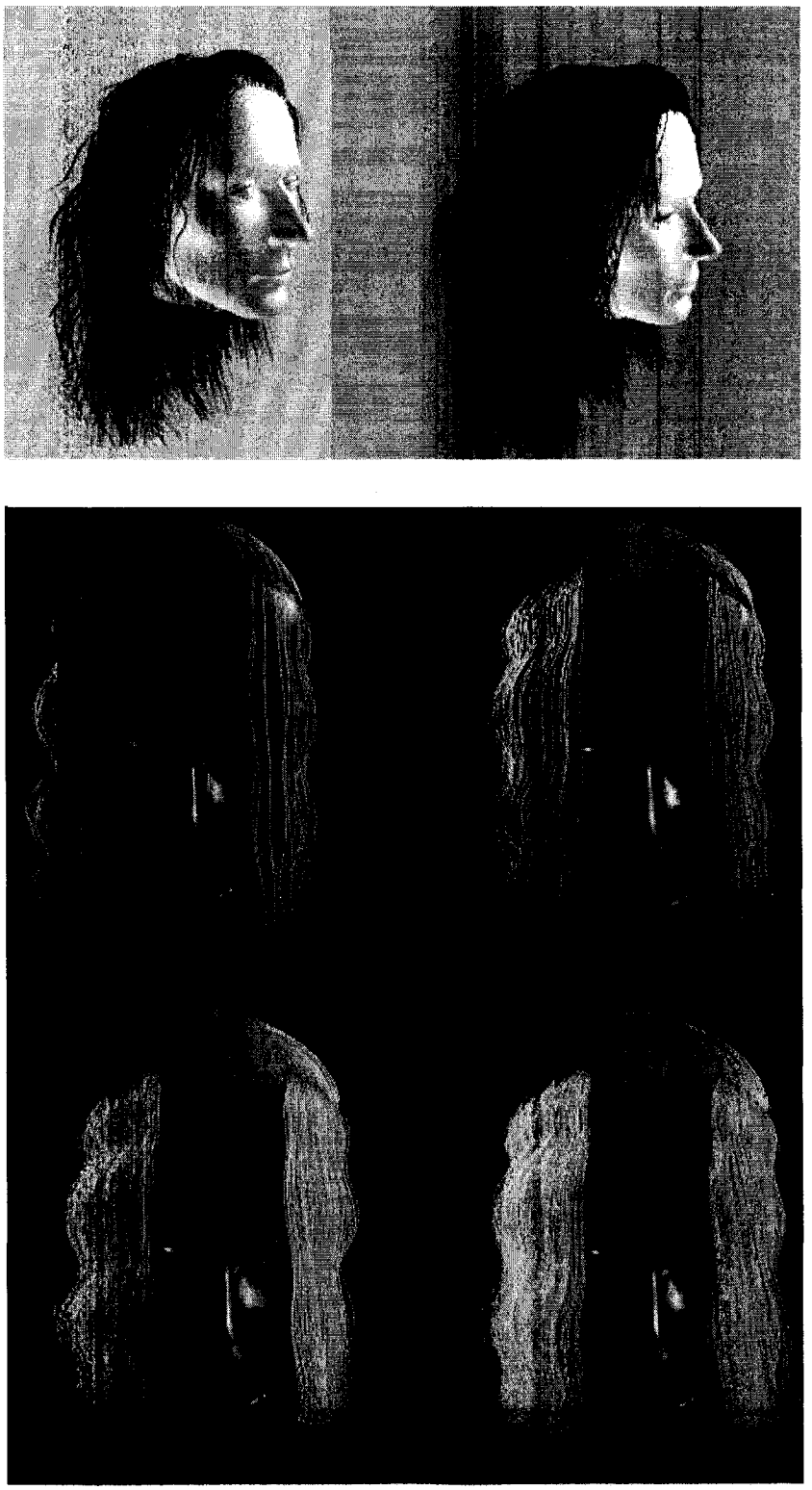


Figure 3-8: Total number of hair comparison
Top-Left) 10,000 Top-Right) 100,000 Middle-Left) 1000
Middle-Right) 5000 Bottom-Left) 20,000 Bottom-Right) 100,000

3.2 Realism

Up to this point we discussed how to define curves for key strands and root locations for ordinary strands and also determined wisp boundaries. For generating ordinary strands curves based on key strands, we need to be aware of realism; if all ordinary strands are defined exactly the same as their master strands, the hairstyle would look very unnatural. So, we modify some attributes of ordinary strands to make them slightly different from each other and therefore more realistic. Color, length, orientation, and shape are the parameters that can be modified for each ordinary strand. We will discuss about color variation along a strand and between all strands and also other rendering issues in chapter 5, here, **length, orientation, interactions between wisps,** and **shape of hairs** are addressed.

3.2.1 Length of hairs

Naturally hair strands have different **lengths** unless newly cut. Here, we assigned a maximum and a minimum number for all hairs based on which, hair length is randomly assigned between min and max (see Figure 3-9).

Figure 3-9: hairstyles with different max and min length values
Left) Min=Max **Middle)** Min = 0.9 Max **Right)** Min = 0.6 Max

3.2.2 Tidiness of hairs

Although **orientation** of hair would seem unnecessary to play with, it can affect realism a lot even with small changes. Therefore, we assigned slightly different random numbers to strands orientation. *Tidiness* value (between 0 and 1) determines the maximum difference between orientation of an ordinary strand and its master strand. If we assign one to this value, all strands would have the exact same orientation as their master strand; whereas if zero is assigned to *tidiness,* hairstyle would become quite messy (see Figure 3-10).

Root of hair strands has three local axes; root first and second tangent vector and root normal vector. As it was mentioned before, orientation of key strands are adjustable and the orientation of ordinary strands is the same as the key assigned to them. If we rotate ordinary strands slightly around these three local axes, a feel of untidiness can be achieved as you can see in Figure 3-10.

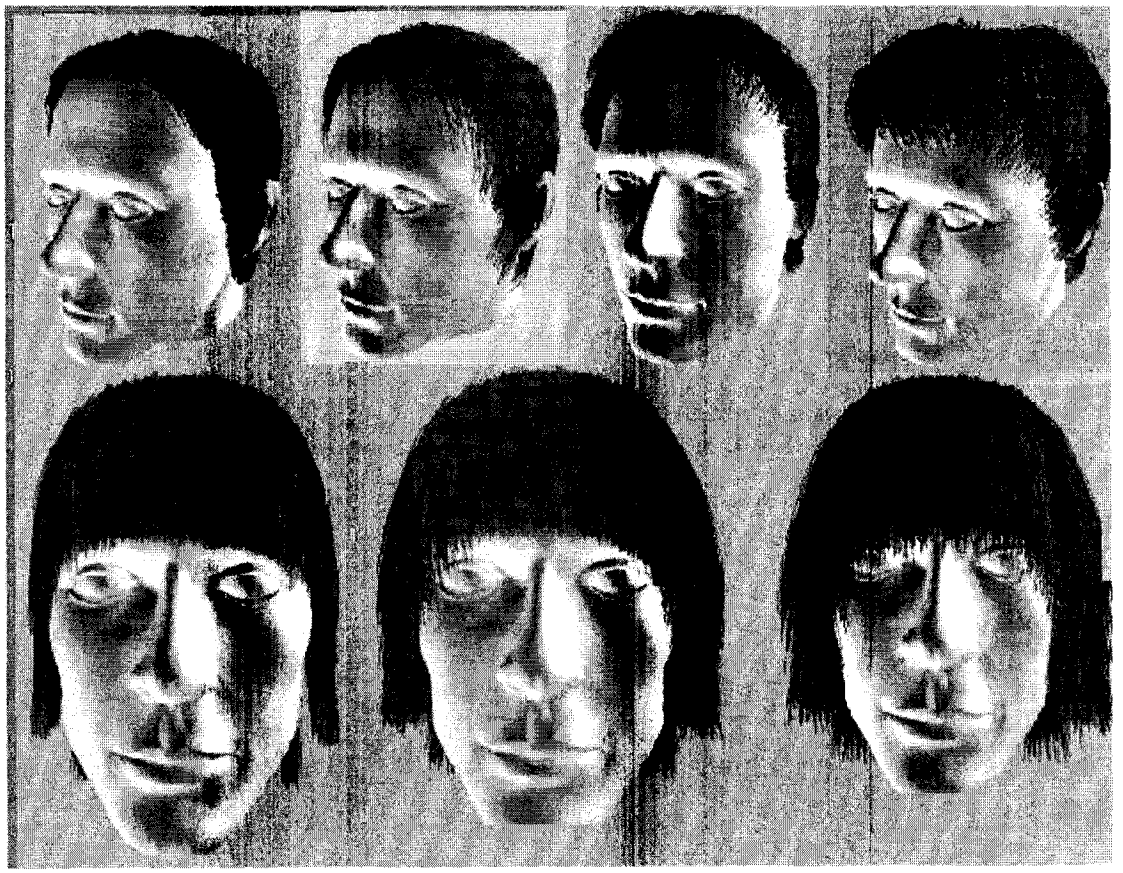


Figure 3-10: Hair model with different tidiness
Tidiness value is decreasing from left to right

3.2.3 Interaction between wisps

Although having slightly different orientation would help us prevent the main drawback of key-strand hair modeling which is to have unnatural clusters of hair rather than individual hair strands, it still cannot conceal the fact that we actually design quite a few key strands and all the others are just copies of them. Especially when key strands do not resemble each other and each one has structure of its own. Therefore, in this work, we used an interpolation factor between wisps, *Neighbours influence or shortly N-influence,* which helps

interaction between wisps of hairs. *N-influence* relates a wisp to a number of nearest neighbours. Neighbour wisps can affect strands of a wisp depending on the amount of N-influence value (between 0 and 100).

For k^{th} point of the j^{th} ordinary strand of the i^{th} wisp, P_{ijk}[14], instead of only using $P_{ijk} = K_{ik}$ (copying all the points of the key strand to the entire group of ordinary strands following it) we can calculate location of $P_{(main)jk}$ as (here i = main is actually i^{th} wisp which we call main for the sake of readability and neighbour1 and neighbour2 are nearest wisps to the main wisp):

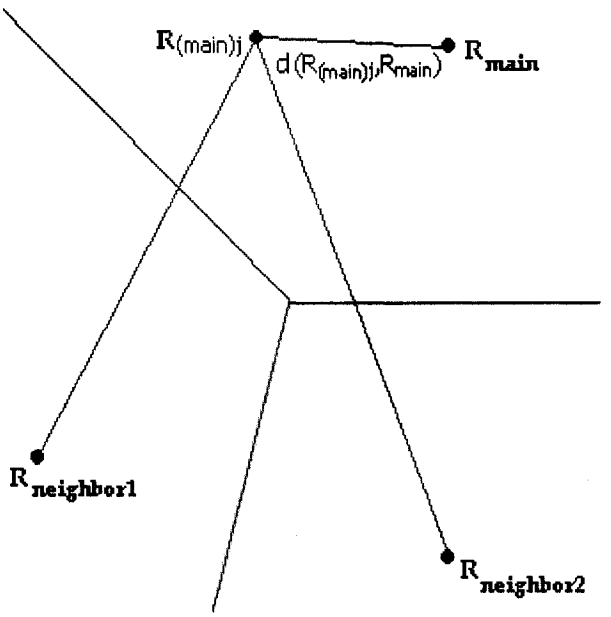


Figure 3-11 : Relating neighboring wisps to each other

$\alpha = 1/d(R_{(main)j}, R_{main})$

[14] Note that in the actual program all the arrays are 1D for the sake of performance as you will see in some of the following sections. Here, 3 indices are used for simplicity. In the reality, R_i is the i^{th} root either key root or ordinary root and P_i is an array of all cylinders top point in a certain order.

$\beta = 1/d(R_{(main)j}, R_{neighbour1})$

$\gamma = 1/d(R_{(main)j}, R_{neighbour2})$

$sum = \alpha*(101\text{-N-influence}) + \beta + \gamma$

$P_{(main)jk} = (\alpha*(101\text{-N-influence})/sum)*K_{(main)k} + (\beta/sum)*K_{(neighbor1)k} + (\gamma/sum)*K_{(neighbor2)k}$

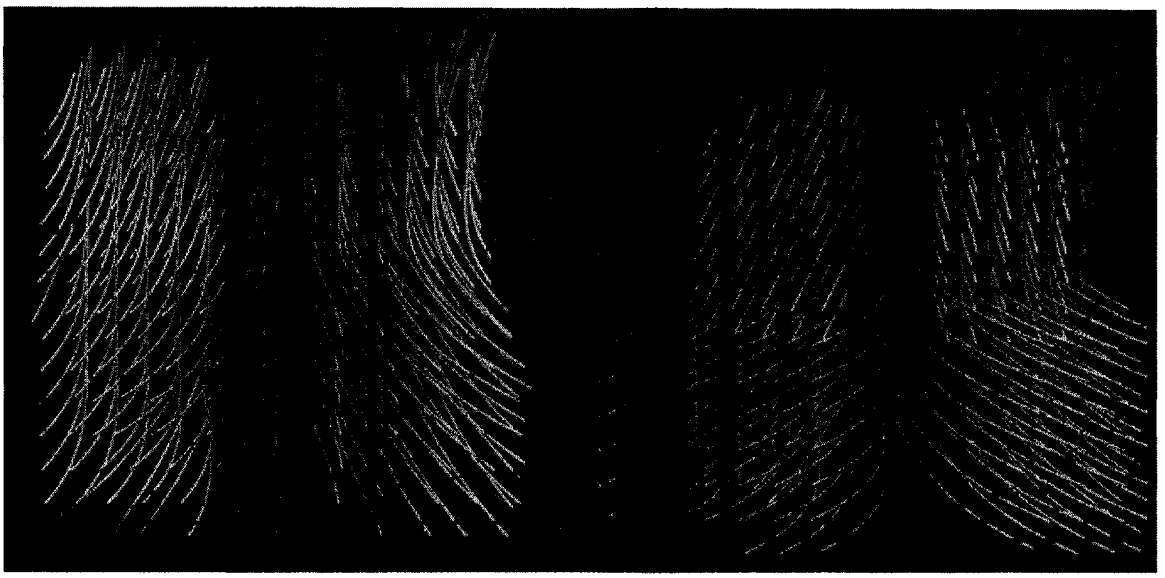


Figure 3-12: Interpolation between wisps of hair
left) N-influence=100 right) N-influence = 1

Where $R_{(main)j}$ is the root location of j^{th} ordinary strand of main wisp (the one which P_{ijk} belongs to), R_x is the root location of key strand x, d(a,b) shows the distance between point a and b in space, $K_{(x)k}$ is the key strand x's k^{th} point. As you can see in Figure 3-12, if, for example, N-influence is 100, then neighbor wisps would affect an ordinary strand with respect to their distance from root of the examining ordinary strand. Obviously since main key strand is closer to the current ordinary strand, its effect is always the most. If N-influence equals 1, then ordinary strand curve would not noticeably be different from the main key

strand (β/sum and γ/sum will be about zero and α*(101-N-influence)/sum will be about 1) (see Figure 3-12).

3.2.4 Intelligence Flag

We used interpolation between wisps and random orientations in order to avoid unnatural looking hair wisps, yet we have two problems; if we use interpolation method, user-control over hair would be less when N-influence value is high. If we use random orientation method, we need to define too many key strands to avoid undesired hairstyles (at least 10). Using both methods together would help but may limit the variety of hairstyles. Therefore, another parameter has been proposed in this thesis to cover these drawbacks. It is called the *intelligence flag*.

If we turn the intelligence flag off, hair would be created with respect to the **root's key strand normal vector**, if not, the orientation of each strand would be defined with respect to the **root normal vector**.

Initially we suppose that strands root plane is equal to the default root plane. If we turn intelligence flag off this plane should rotate with respect to the key strand normal vector for each wisp. Otherwise, it should rotate with respect to the root normal vector itself.

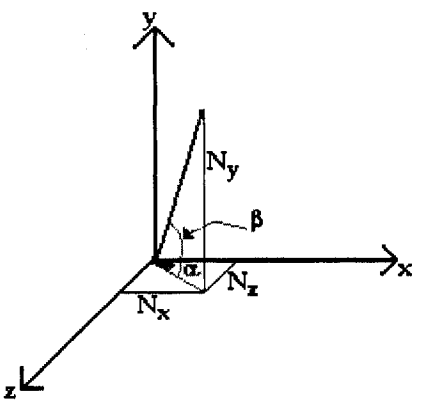


Figure 3-13: Root Normal Vector

Default root Normal vector, (1,0,0), can be mapped to the root normal vector, (N_x, N_y, N_z), using α and β angles. Where:

$$\alpha = \cos^{-1}(N_x / \sqrt{(N_x^2+N_z^2)})$$

$$\beta = \tan^{-1}(N_y/ \sqrt{(N_x^2+N_z^2)})$$

Therefore, rotating plane of the default strand α degree around (0,1,0) and β degree around (1,0,1) will give us the correct rotation. In other words, angle between the strand curve and the scalp at initial position (default strand) will be equal to the angle between the strand curve and the scalp at the new position (R_x, R_y, R_z).

Using intelligence flag is of great importance for the mere reason that we can design a full hairstyle with only one single key strand (see Figure 3-14).

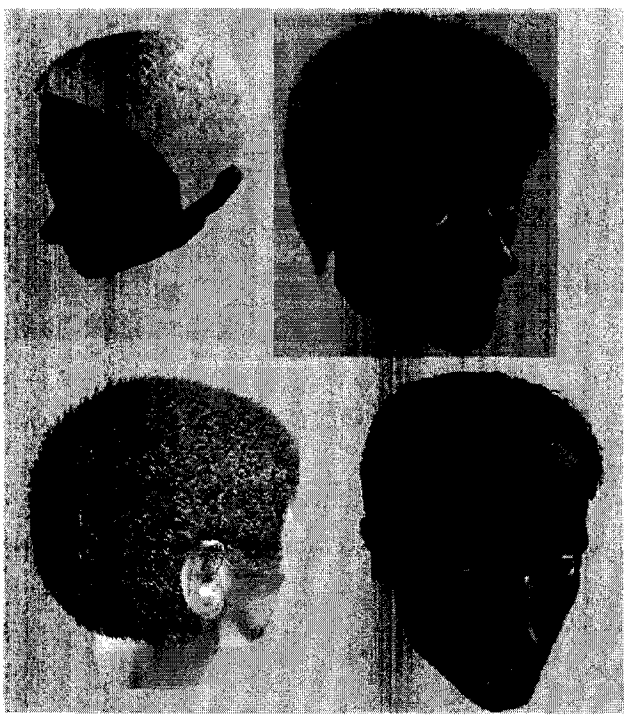


Figure 3-14: Hair design with one key strand

Although using this method is very suitable for short hairs, some long hair models can be designed more easily with it as well. Depending on the hairstyle, we can use the benefits of intelligence flag. For hairstyles that we exactly know the wisps and what we are looking for, we can turn off intelligence flag and for those that we just have an idea and we do not want to put too much effort, we can turn on intelligence flag (see Figure 3-15).

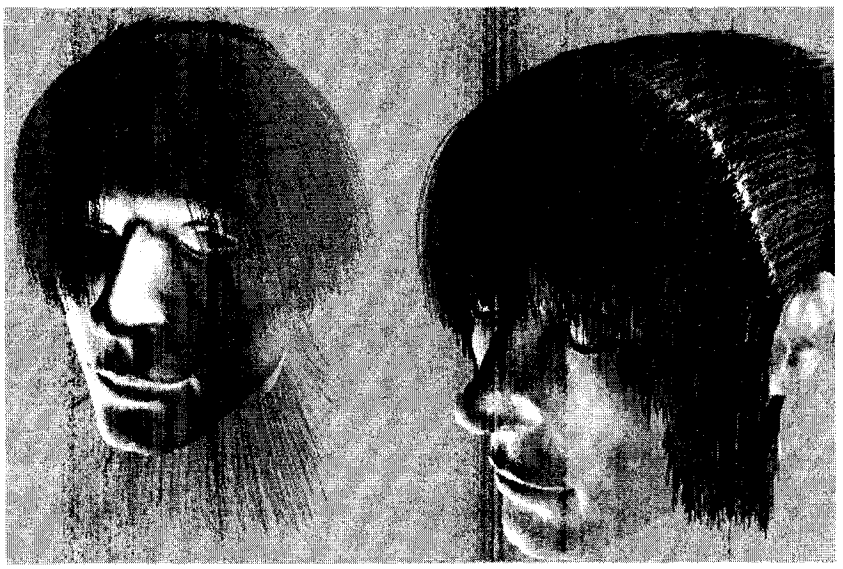


Figure 3-15: Intelligence flag effects
Left) intelligence flag is on Right) intelligence flag is off

3.3 Result Images

We generated hairstyles based on the key strand method introducing some additional parameters for the sake of realism such as assigning slightly random strand lengths and orientations. Our version of key strands method supports different wisp sizes, distributes hair roots semi-uniformly, includes random hair orientation, defines neighbor wisps interactions, and introduces an automatic way of assigning hair orientation (intelligence flag). Using our method, various hairstyles can be modeled (see Figure 3-16 and Figure 3-17).

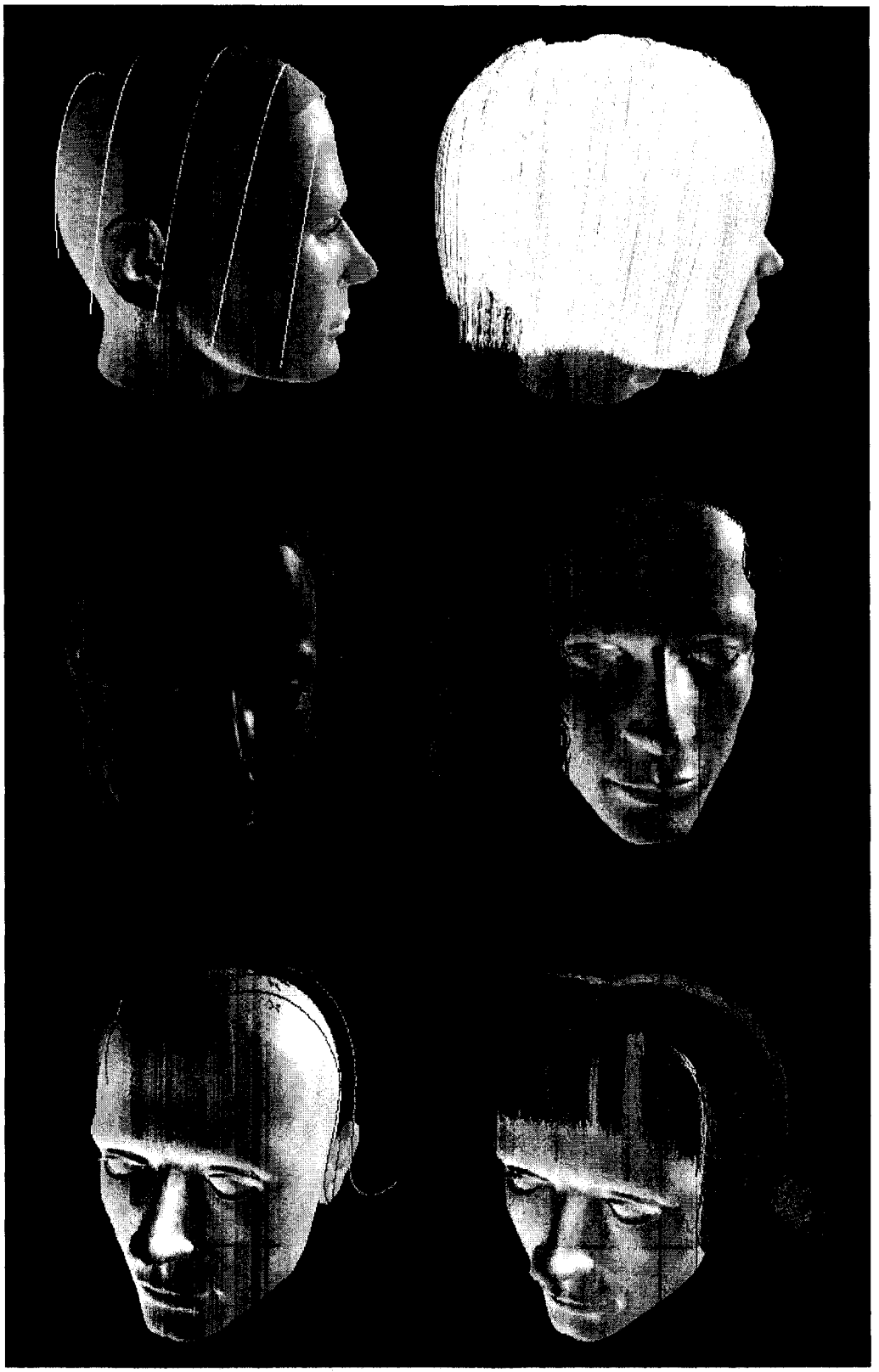


Figure 3-16: Key strand hair modeling

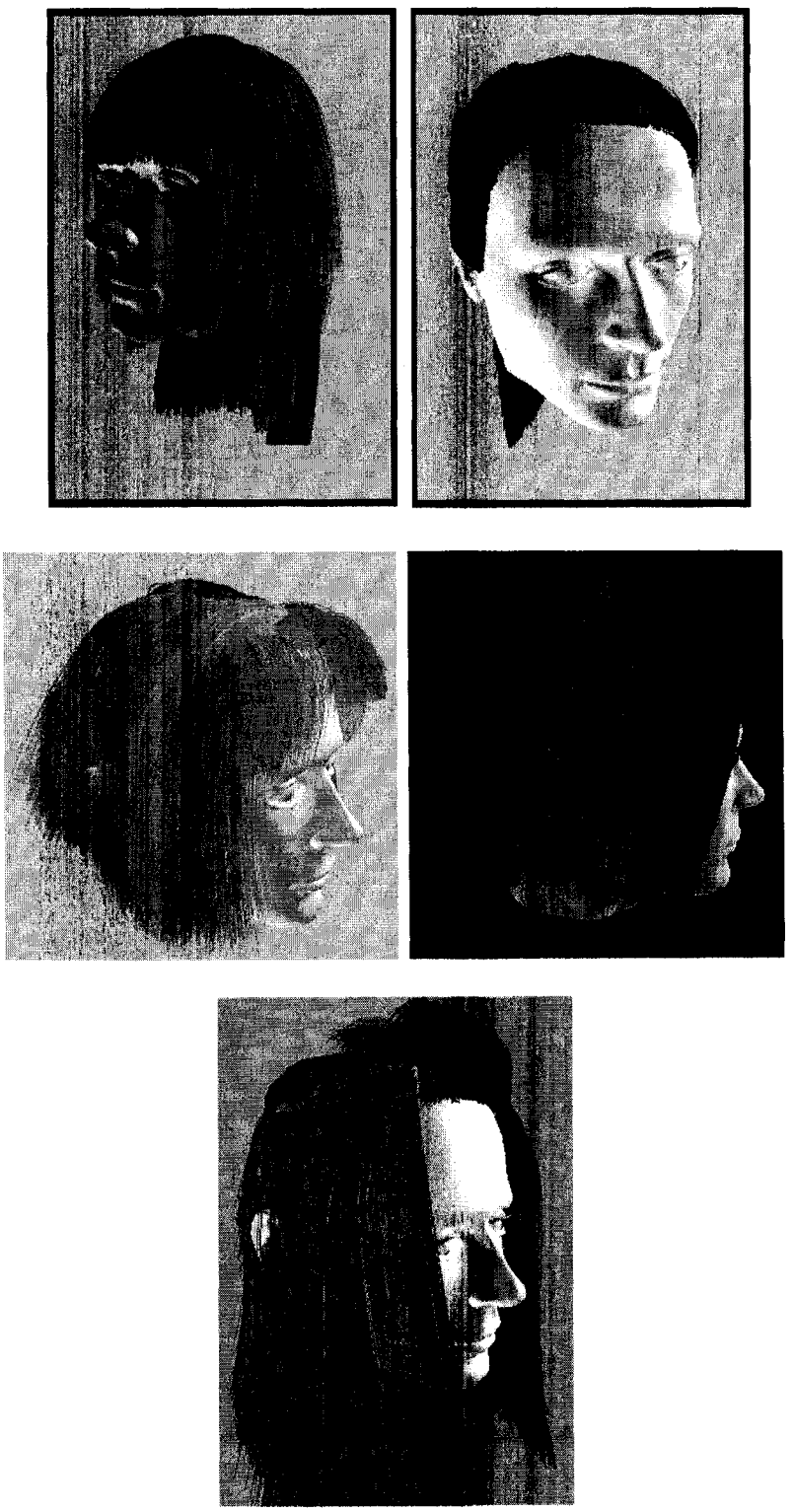


Figure 3-17: Hairstyles
Top-Left) Modeled with 7 key strands **Top-Right)** A Tied hairstyle
Middle) Straight hairstyles
Bottom) Modeled with 50,000 hairs

Chapter4. STRANDS AS CONTINUAL CYLINDERS

As was mentioned before, the goal of this thesis is to simulate hair realistically in real-time with a hair model potentially suitable for animation and collision detection. Also the hair model should be capable of providing high **control** over the shape of the hairs with the ability to model a variety of **hairstyles**, yet not requiring too much user-**effort**. We also mentioned that for the sake of quality we will use ray tracing to render this model (ray tracing makes it easy to implement self shadowing, reflection as in specular light, and alpha blending -- in the fifth chapter the reason for this selection over other possible choices will be discussed fully). For acquiring ray tracing's benefits, the hair strands should be represented as a 3D objects and cannot be represented as poly-lines or points; therefore, a new geometric object called *Continual Cylinder* is introduced for representing individual hairs in this thesis.

In the following sections, first we define Continual Cylinders, then we demonstrate the reasons for this definition and finally we use them to model our hair strands.

4.1 Continual Cylinders

In this thesis, unlike other hair models to our knowledge, we modeled hair strands with a new 3D object that we called *Continual Cylinders*. Each hair strand is modeled using a chain of cylinders where cross sections of each cylinder are tangent to the previous and the next cylinder (of course, except for the first and the last cylinder) as you can see in Figure 4-1. Therefore, a number of adjacent cylinders (typically 25) are the objects that form all the hair strands.

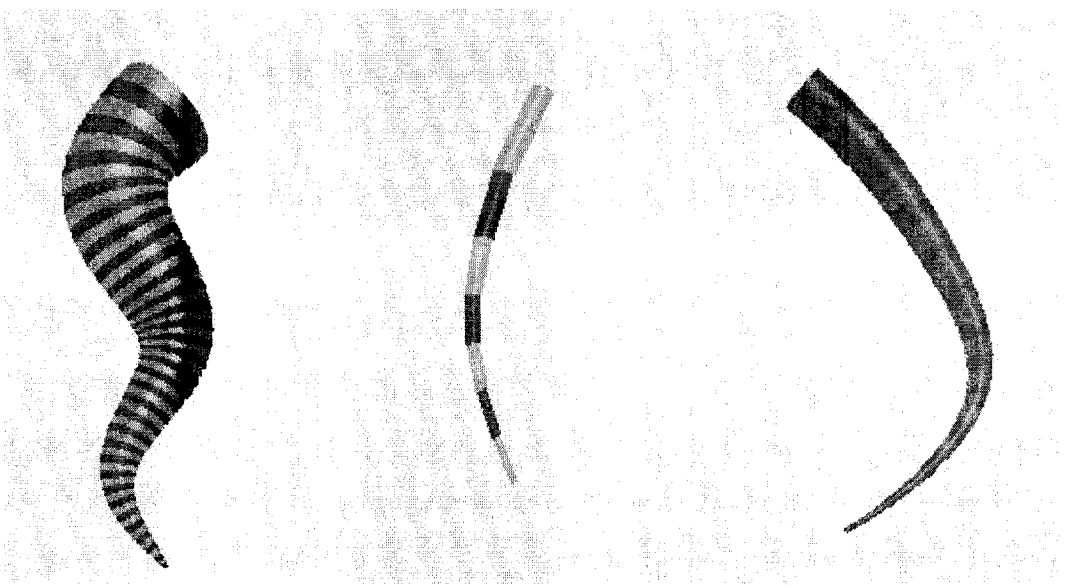


Figure 4-1: Continual Cylinders
Left) 47 Cylinders **Middle)** 7 cylinders **Right)** 23 cylinders

A **cylinder** is the surface formed by all the points at a fixed distance, cylinder radius, from a certain straight line, cylinder axis. Suppose that we cut this cylinder by two arbitrary planes; cross sections will be two ovals. Let us call centers of these two ovals, **cylinder's top point** and **cylinder's tip point**.

Motivated by Generalized Cylinders and splines[15], we propose a new geometric object called a Continual Cylinder. A Continual Cylinder is defined as a set of n cylinders, $\{C_1, C_2, ..., C_n\}$, where C_i's tip point is equal to C_{i+1}'s top point (P_{i+1}) and C_i is cut by these two planes:

$$Plane_{i1} : N_{i1\_x}*(x - P_{i\_x}) + N_{i1\_y}*(y - P_{i\_y}) + N_{i1\_z}*(z - P_{i\_z}) = 0$$

$$Plane_{i2} : N_{i2\_x}*(x - P_{(i+1)\_x}) + N_{i2\_y}*(y - P_{(i+1)\_y}) + N_{i2\_z}*(z - P_{(i+1)\_z}) = 0$$

$$N_{i1} = ((P_{(i+1)\_x} - P_{i\_x}) , (P_{(i+1)\_y} - P_{i\_y}) , (P_{(i+1)\_z} - P_{i\_z}))$$

$$N_{i2} = ((P_{(i+2)\_x} - P_{(i+1)\_x}) , (P_{(i+2)\_y} - P_{(i+1)\_y}) , (P_{(i+2)\_z} - P_{(i+1)\_z}))$$

where $P_i = (P_{i\_x}, P_{i\_y}, P_{i\_z})$ is C_i's top point.

Every strand in this thesis is formed by a Continual Cylinder and therefore can be defined with this vector:

$$S = \{(r_0, P_{0\_x}, P_{0\_y}, P_{0\_z}) , (r_1, P_{1\_x}, P_{1\_y}, P_{1\_z}) , ... , (r_{n-1}, P_{(n-1)\_x}, P_{(n-1)\_y}, P_{(n-1)\_z})\}$$

where r_i is the radius of i^{th} cylinder.

This definition does not necessarily generate smooth curves which we expect for a strand, because we did not assign any value to the positions of the cylinders top points (P_is) yet. We will discuss shortly how to arrange the top points in order to satisfy this necessity.

4.2 Reasons of defining Continual Cylinders

There are various reasons for which Continual Cylinders have been defined to model hair strands:

1. Hairs can be approximated by cylinders almost realistically.

[15] A piecewise polynomial function.

2. The model is 3D; therefore, it can be rendered using ray tracing and Phong illumination model.
3. Different quality levels can be applied by using less or more number of cylinders in the chain.
4. Properties of an individual hair can vary along the hair this way (as we go from one cylinder to the next). Properties such as, color, thickness, alpha value, etc. (Figure 4-2).
5. The scene can be easily divided into smaller partitions without worrying about long hairs falling apart.

Generalized Cylinders seem to be a better representation at first. Firstly, because of their neat formula which is a very good approximation to hair strands and secondly, for the sake of rendering using ray tracing; the less objects we have in the scene, the less rendering time we should spend (in ray tracing methods most of the time is spent for ray-object intersection calculations). But **Generalized Cylinders are not appropriate** in our context. Here, we are dealing with at least 10,000 hair strands or let us assume for now Generalized Cylinders. Checking each and every one of them to see if it hits each and every ray could take too much time. Therefore, even if we represent hairs by Generalized Cylinders, we need to **divide them into pieces** so that we can handle the great number of objects in the scene.

Another important reason for using Continual Cylinders rather than Generalized Cylinders is because of those properties that can vary along a single strand such as color, thickness, etc. As you can see in Figure 4-2 this can largely affect realism. Also, the significant role of **alpha value** in hair simulation cannot be ignored. Hairs are very fine and cannot be approximated by volumes such as Generalized Cylinder unless having an alpha value assigned to them to make them look finer and therefore realistic (as you may recall that in a normal viewing condition thickness of a strand is less than the size of a pixel which is

why alpha blending is of great importance here). Since alpha value may vary along the curve we again need to deconstruct the Generalized Cylinders into some smaller parts with different alpha values.

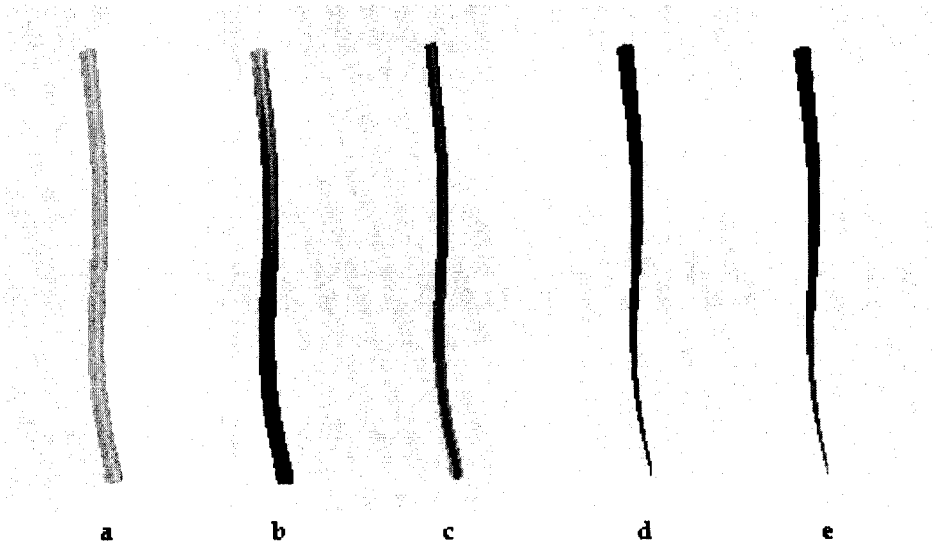


Figure 4-2: Modifying properties of hairs along a strand
a) The primitive hair strand b) Color variation along the strand c) alpha value variation d) radius variation e) Alpha, color, and radius are changed along the strand

4.3 Strands as a smooth curve

Up to this point, we defined a geometric object to represent strands of hair. Now the problem is to determine the locations of the cylinders' top point (having the cylinder's top point and its radius is enough to define a cylinder because this cylinder's tip point is the next cylinder's top point) to be able to form a smooth strand. Hair strands can be considered (actually estimated but a very good estimation) as curves. As we mentioned, a very common way to define objects with shape of a curve in Computer Graphics is to use parametric curves, one of which is Bézier *curve* that we used in this project.

Control points are a fixed number of points who control the behavior of the curve. A Bézier curve is actually a set of points close enough to be seen as a

curve. By changing location of control points we can have different shapes for strands (see Figure 4-3). Each Bézier Curve consists of n points (larger n means higher resolution) and consequently n-1 cylinders; for example the axis of the Continual Cylinder in Figure 4-1-left is actually a Bézier Curve defined by 10 control points. Here, each point of a Bézier curve corresponds to a cylinder top point (Figure 4-1-left is created based on a 48 points Bézier curve created by 10 control points).

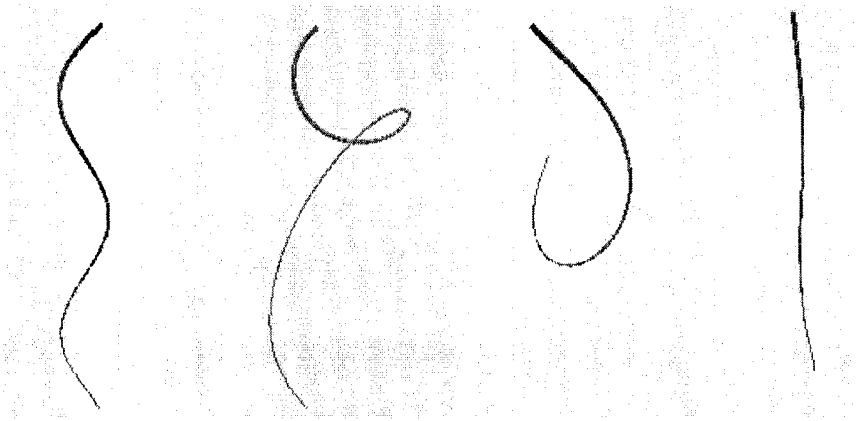


Figure 4-3: Some hair strands generated by Continual Cylinder

The reason that Bézier curves are chosen in this thesis is for their being smooth and continuous which makes them a good representation for the shape of the hair. This means that we can estimate strands of hair no matter how much curly they are, and we can have as less as 3 control points for straight hairs which is so easy to define.

A strand's resolution is adjustable and refers to the number of cylinders that form the hair strand. Although it might seem that higher resolutions generate strands with higher quality, it is important to note that after certain amount for resolution, it will be impossible to distinguish the difference between the strands with higher resolution and lower one. This is because of the fact that cylinders are tangent to each other and if they form a smooth curve with a large enough

number of cylinders (typically 25), it will not matter if we increase the resolution; again we will have the exact same smooth curve. This does not mean that the quality of a single strand is limited, but it means we reach the highest possible quality (which of course is limited to the size of a pixel in screen!) with relatively small number of objects.

Normally, the roots of human hairs are stronger and therefore thicker than the tips of hairs[Si67]; therefore, we let the cylinder thickness of strands decrease whenever it is appropriate (see Figure 4-3):

$$\textbf{Radius}(C_i) = r*(1-i/\text{resolution})$$

where **r** is the radius of the root cylinder, and C_i refers to cylinder number i (C_0 is the root); therefore with a typical resolution, 25:

Radius(root) = r

Raduis(C_1) = r*0.96

Raduis(C_2) = r*0.92

...

Raduis(C_{24}) = r*0.04 ~ 0

Note that it is not necessary to decrease the radius of a hair strand. We can either use the same radius for each cylinder or decrease it at a certain point to a certain radius. Decreasing the radius to zero is appropriate for two situations: when we have very long hairs and when we have unnatural or cartoon hairs.

Chapter5. HAIR RENDERING

Geometry-based hair rendering largely depends on the underlying hair model. As we have seen in previous chapters hairs are modeled using Continual Cylinders; therefore, we are dealing with a number of cylinders (a very large number i.e. about 250,000) which need to be rendered. Rendering is done using GPU in this thesis. GPUs are dedicated to graphical purposes only and can be used to render graphical scenes much faster. For the sake of realism ray tracing methods are used to render cylinders.

As we have seen in literature review chapter, real-time rendering of high quality hair is very hard. As we have seen, real-time hair rendering has been done in some researches, for example, [XL+06] has used thick Generalized Cylinders to achieve real-time performance, [KH00] simulated hair strands as small number of strips, and [Os07] introduced a particle-based method using GPU for hair simulation. Yet, most of the even recent methods are capable of rendering only a few strands in real-time and rendering time of the whole head is nothing near real-time: [KiN02] could render 10,000 hairs with their multi-resolution method in about 6 seconds, [BA+05] method "is fast enough to handle several hair strands in real-time, and a hundred strands within a few seconds", and [CK05] simulated very realistic hair models using both GPU and CPU to render 20,000 hair in about 10 seconds.

We succeeded to render enough number of 3D hair strands (10,000) using Ray Tracing along with several realism issues.

Also, performance of most of the previous algorithms largely depends on the length of hairs; the longer the hair, the more segments are required to model

them and therefore, the more rendering time is needed. In this thesis, length of hair does not effect performance (will be discussed in section 5.5) and we are only limited by GPU functionality (section 5.1.2). This means that our algorithm can be used to render more number of hair strands by using a better GPU in future.

In this Chapter, first we discuss about real-time rendering with GPU and its limitations, then we speak about CPU part in rendering, after that we describe our *ray tracing on GPU* algorithm, then we color our pixels realistically, and finally we talk about the running time of our algorithm.

5.1 Real-time Rendering with GPU

The term *real-time rendering* in Computer Graphics commonly refers to computer-animation. In real-time computer animation, each stage of the sequence is viewed as it is created [HB04]. Here, by real-time rendering we mean that our images are rendered in less than $1/15^{th}$ of a second, or in other words, with the frame rate of more than 15 per second. Therefore, we can move the camera around the hair model and see different views of it in real-time.

As we mentioned in Background, since 2002, GPU has been used for real-time rendering because it is a special purpose microprocessor that has been designed to do complex graphical computations. Having millions of transistors to do 3D graphics calculations, it can generate 3D scenes with better quality comparing with the CPU [NV08] (also see Figure 2-16), but it has rather fewer resources (see section 5.1.2).

For programming a GPU, several high-level languages have been developed recently which will be discussed shortly.

5.1.1 Shading Language

A shader is an independent compilation unit consisting of a set of software instructions used to program the GPU pipeline. GPU can be programmed by the complete set of shaders (All pixel and vertex shaders) that are compiled and linked together. Shading languages used to be low-level but since 2002 high-level GPU programming has emerged. Three popular shading languages are as follows:

1-Cg (developed by nVidia)

2-GLSL (part of the core OpenGL 2.1)

3-HLSL (developed by Microsoft for use with DirectX)

All three have been created to give more control over GPU pipeline and are hardware-specific languages. Cg and HLSL are very similar and actually co-developed. The major difference is that HLSL is only suitable for DirectX code while Cg can compile to both DirectX and OpenGL. HLSL can only be used in windows while Cg and GLSL are multi-platform. Since GLSL is only good for graphics cards supporting OpenGL and real-time hair modeling is mostly used for game industry where DirectX is widely used (esp. XBox consoles), we chose Cg for programming GPU in this work.

5.1.2 GPU limitations

Unlike the CPU, the GPU has very limited resources; based on the work done by [PS+06] GPUs only provide a subset of the functionality commonly found on CPUs and the following are the most important limitations of GPU which makes the implementation of complex single-pass programs a difficult task:

- The instruction count is effectively limited to 4096 instructions.
- There is only a small number of registers available, e. g. current graphics boards from NVIDIA have 32 registers and there is no further writable random access memory.
- Because of the parallel nature of the fragment processing units, it is not easy to use inter-pixels information.

A more complete list of GPU limitations is provided in Appendix B.

Whenever we pass a limit of a GPU, the results will deteriorate (see Figure 5-1). Therefore, although we can make rendering faster using the GPU, still we are limited. The good thing is that GPU grows very fast due to the rich industry of gaming and therefore real-time high quality hair simulating may not be a very hard-to-reach goal to achieve before long.

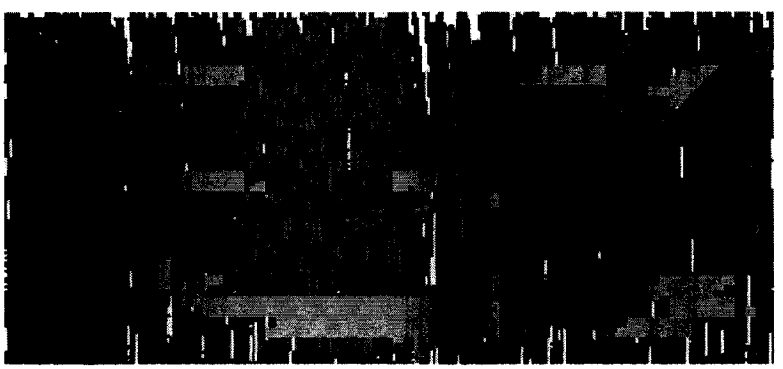


Figure 5-1: GPU limitations problem
GPU acts incorrectly when we exceed some of its limitations

5.2 CPU Part; transferring required data to GPU

Before explaining our algorithm, let us clarify CPU and GPU roles. All the rendering has been done using GPU. The CPU is responsible for modeling, pre-calculations and transferring rendering parameters to GPU. Subsequently, CPU transfers the following data to GPU:

1-Lighting information: eye location, light sources locations, and shadow flag.

2-Cylinders information: Top, tip, and next point of each cylinder (P_{top}, P_{tip}, P_{nxt}), cylinders color, and cylinders radius.

3-Scene information: scene's partitioning information, number of total points defining a strand (or resolution), and number of total hairs.

While the CPU is responsible for dividing scene into a number of partitions and calculating the location and length of the central axis of cylinders and lighting information, the GPU is responsible for rendering all those cylinders based on provided information.

Although data can be transferred directly to the GPU using limited size arrays, the common way of passing data from CPU to GPU is using textures. The reason is that textures can be treated as random access memories[16] in GPU. Textures here resemble arrays of CPU and they can contain large amount of data. We used them to pass P_{top}, P_{tip}, P_{nxt}, cylinders color, and scene's partitioning information (see Figure 5-2). Having all of the information, pixel processor is

[16] Or RAM is a type of computer memory that can store data in any order and therefore return them in constant time.

ready to render all of the Continual Cylinders to simulate hair with the algorithm that will be discussed in next section.

In this section, we first discuss how to divide the scene into some number of components, called voxel[17], and then how to assign cylinders to each voxel.

5.2.1 Dividing the scene into Voxels

To achieve higher performance while using ray tracing methods, we need to partition our scene into some smaller components each contains some number of the objects we plan to render; otherwise, we have to verify intersection test between all the rays and all the objects of the scene which could be quite time-consuming. Here, the scene is divided into some number of same size voxels (Figure 5-2-left) within each, some of our previously designed cylinders will be placed. The size of voxels are the same because cylinders distribution is almost uniform i.e. for most of the voxels either we have several cylinders in a voxel or we have none. This is because of the fact that hair strands are adjacent to each other.

Rays are emitted from a single point of view in every direction. Whenever a ray hits the first non-empty voxel in its way from the light source towards the end of the scene, cylinders within that voxel are checked to see whether they have any intersection with the ray or not. In case of a collision, the intersection point (between the ray and the hitting cylinder) will be rendered with the proper color. If the ray is able to pass through the first non-empty voxel without hitting any cylinder, the next non-empty voxel will be checked for a collision until finding a hit or until the ray falls out of the end of the grid without hitting by any object.

[17] Pixel to screen is like voxel to 3D space.

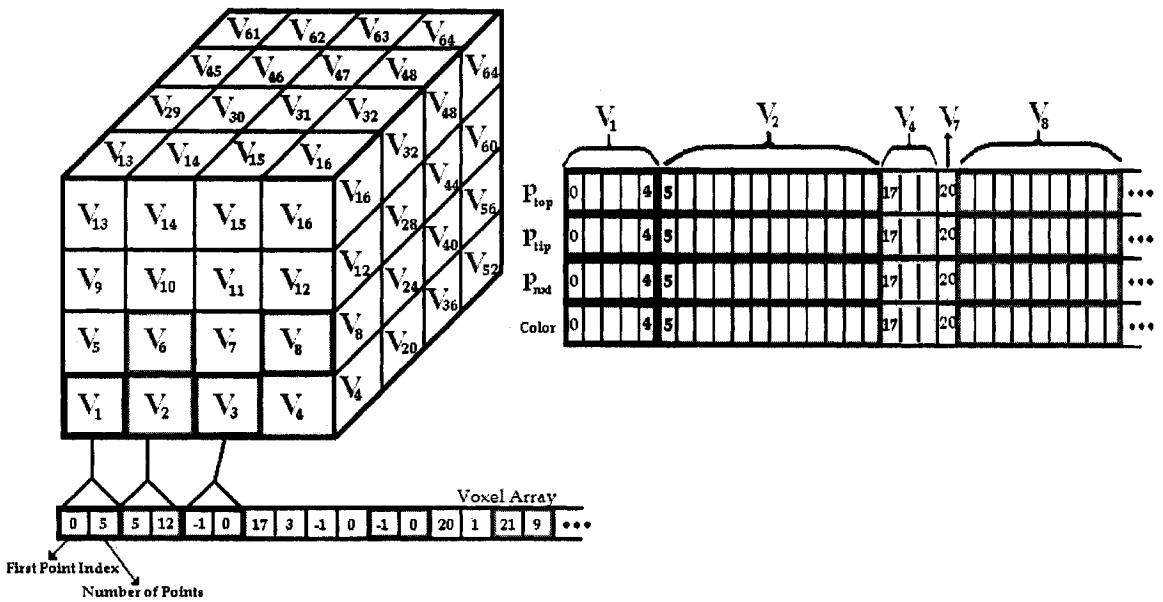


Figure 5-2: Keeping scene data
Left) scene is divided into some number of voxels within which some **number of points** is placed. **Right)** cylinders are defined by four parameters and placed in order.

5.2.2 Assigning voxels to cylinders

As we mentioned in chapter 4, our hair model is a set of cylinders cut by two planes and therefore each cylinder can be defined by three points: the cylinder top point, the cylinder tip point (= next cylinder top point), and next cylinder tip point (= the after-next-cylinder's top point); P_{top}, P_{tip}, and P_{nxt}. Also we mentioned that P_{top} can uniquely represent each cylinder.

To assign voxels to cylinders, the line formed by P_{top} and P_{tip} or the axis of the cylinder is considered. If the axis of the cylinder is completely inside one voxel, the cylinder will add to the end of the list of that voxel. If not, the cylinder will add to the end of the list of all those voxels which contain the axis. A 1D **voxel array** (Figure 5-2-left) contains *first point index* of each voxel and *number of*

points in each voxel. **First point index** shows the index of the first cylinder in the cylinder's list of a voxel and **number of points** shows the number of cylinders in the list.

As an example let's consider V_4 of Figure 5-2. First point index of this voxel is 17 and it contains 3 cylinders. Center of the top circle of the first cylinder (first cylinder's top point) is saved in the 17^{th} slot of P_{top} array, center of the top circle of the second cylinder (second cylinder's top point) is saved in the 18^{th} slot of P_{top} and consequently third one in the 19^{th} slot. Tip point of the first cylinder of the list is saved in P_{tip_17} and so on.

The algorithm to find whether a point P is in a voxel or not, is a simple lg(n)[18] recursive algorithm which n is the number of voxels in one dimension as follow:

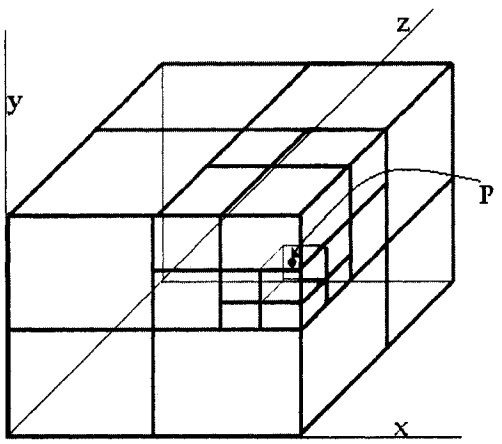


Figure 5-3: assignment of voxels to points (representation of cylinders)

At first we consider the whole scene as one big voxel which contains P. We check to see which half of the scene with respect to x, y, and z axis contains P,

[18] lg(n) is used for logarithm in base 2.

continuing like this and discarding 7/8 of voxels each time and after few steps (typically 6) we can find the proper voxel (see Figure 5-3).

Our experiments show that partitioning the scene to more than 128*128*128 voxels will make the specific GPU that we used i.e. nVidia Quadro 4500, to deteriorate (see Figure 5-1). Therefore, in the worst case scenario the running time of the algorithm to find the proper voxel for every P_{top} (or every cylinder) will be n*lg(128) = 7*n. Where n is the number of cylinders which is:

n = resolution*NOS. (Normally 250,000)

where NOS is the number of total hair strands in a hair model.

Voxels are assigned to cylinders using the CPU and the final voxel array will send to the GPU for applying ray tracing method. In addition to voxels, P_{top}, P_{tip}, and P_{nxt} arrays, cylinders color, and radius are as well sent to the GPU which will be discussed in section 5.4.

5.3 Ray tracing on GPU

Ray tracing on GPU is first introduced by [PB+02] in 2002. They succeeded to render scenes consisting of triangles using ray tracing in real-time. We used roughly the same algorithm for rendering hairs, but instead of triangles we are dealing with cylinders represented by points.

In our algorithm for ray tracing on GPU, we first assign rays emitted from eye, then, we find the intersection point of the ray and hair strands, and finally we decide about the color of this intersection point (or pixel of the screen).

5.3.1 Why Ray Tracing?

In Ray tracing methods most of the computation time is spending on ray-object intersection calculations. Either we have to calculate intersection between every ray and every object or we should divide the scene to reduce the number of intersection tests.

In this thesis, ray tracing has been selected rather than Radiosity for the reason that hairs are modeled by cylinders and not polygons and therefore, we cannot treated them like surfaces that emit energy as easily as polygons, whereas intersection calculations between rays and cylinders are very easy using ray tracing because cylinders are implicitly represented. Also, specular light plays a very important role in hair simulation which as we discussed in chapter 2, ray tracing can successfully simulate specular lights (see Figure 5-4).

Although rasterisation is very fast, it denies the fact that the color of pixels is not independent of each other. For example consider a complete mirror; using basic rasterisation ignores the fundamental property of a mirror. Also, for shadows we have somewhat the same problem. Comparing rasterisation and ray tracing is of little interest here; let us just focus on the fact that ray tracing produce results with better quality so it is reasonable to render our scene using ray tracing if it doesn't affect performance.

Figure 5-4: Hairs with/without specular effect
Left) No specular light – Middle) Little specular light
Right) More specular light

5.3.2 Assigning rays

As we mentioned in chapter 2, simple ray tracing is to follow all rays emitted from the eye position through the screen and render the corresponding pixel with respect to the intersection point (see Figure 5-5). Therefore, we have a one to one correspondence between pixels and rays.

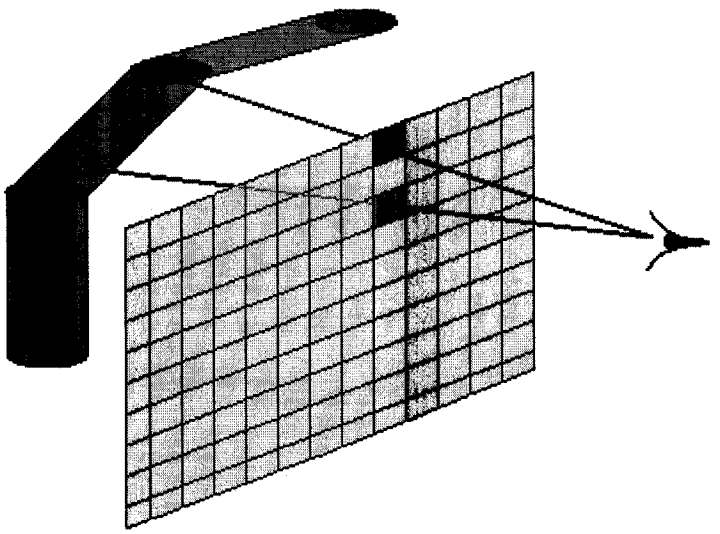


Figure 5-5: Ray Tracing
Red lines are rays and the plane is the screen. Here we have a scene consisting of some objects; the color of each pixel is equal to the color of the intersection point of the ray emitting from eyes through that pixel and an object.

The GPU pixel shader is responsible for the color of every pixel of the screen. Inputs of the pixel shader are:

1-the pixel location,

2-attributes passed from the CPU, and

3-attributes passed from the vertex shader.

Having the pixel location (p) and passing eye location (eye) from the CPU side, the corresponding ray can be calculated as:

$$r = eye + t*(p-eye)$$

Here, the ray origin equals the eye location (rO = eye) and the ray direction equals <p – eye> (rD = <p-eye>). Having r, we are ready to run the intersection algorithm. Note that, rays as well as pixels are considered independently in the GPU.

5.3.3 Finding the first non-empty voxel for a single Ray

The algorithm that is used to find the first non-empty voxel that a ray hits, is the same as the one introduced by Amanatides and Woo in 1987 [AW87]. Details of this algorithm are provided in appendix D.

A voxel is defined by x,y, and z coordinates considering $V_0 = (0,0,0)$ in Figure 5-2 (for example $V_{31} = (2,3,-1)$ in Figure 5-2-left). A ray, r is defined by two parameters; its origin, rO, and its direction, rD:

$$r = rO + t*rD,$$

Using rO and rD, in O(1) we can find the very first voxel that r hits. This voxel is the one that contains the collision point of r and the plane **z=0**. Therefore:

$$V_I = (\mathbf{floor}(NOP*(rO_x + (-rO_z/rD_z)*rD_x)) , \mathbf{floor}(NOP*(rO_y + (-rO_z/rD_z)*rD_y)), 0)$$

Where NOP is the number of partitions in one dimension (typically 64).

Having V_I, V_{II} -- or the next voxel that r goes through, can be found depending on the ray direction (rD):

$V_{II} = (V_{I\_x} + 1, V_{I\_y}, V_{I\_z})$ or

$V_{II} = (V_{I\_x}, V_{I\_y} + 1, V_{I\_z})$ or

$V_{II} = (V_{I\_x}, V_{I\_y}, V_{I\_z} + 1)$ or

Continuing like this (for more information refer to appendix D), we can easily find all those voxels that a ray meets during its way towards the end of the scene. The first one whose number-of-points (see Figure 5-2-left) is not 0, is the one that we are looking for.

Running time of this algorithm is O(n) which n is the number of voxels in one direction (typically 64). Because a ray can meet at most n voxels and to check whether a voxel is empty or not is O(1) as we just need to check the second index of it in the corresponding index of the voxel array (Figure 5-2).

5.3.4 Finding the first collision

There are a number of cylinders in the first non-empty voxel that we have found in the previous step. All of the cylinders must be checked to see whether they have collision with the ray or not. Note that the first cylinder with a positive answer is the actual first cylinder that the ray hits because cylinders are already put in the array with respect to their z value.

If the ray does not hit any of the cylinders in the list of the first non-empty voxel, it should be examined again to find the next non-empty voxel in its way until either a cylinder is found or the ray reaches the end of the scene.

In case of a collision, the corresponding pixel will be colored. The color of the pixel depends on several factors that will be described in the next section.

5.4 Coloring the pixel realistically

A simple math (cylinder-line intersection point) gives us the location of the intersection pixel. To render this pixel we need these values:

1-Normal of the pixel for illumination concerns

2-Color of the cylinder (including alpha value)

3-Whether the pixel is in shadow or not

Within few following few sections we will see how these information help us render pixels more realistically.

5.4.1 Illumination

We used Phong model (as it was discussed in background) to illuminate our pixels in this thesis. For rendering using the Phong model, in addition to lighting and viewing information we need to define a **normal** vector for each and every pixel. Here, normal of the intersection pixel p is:

$$N = [(P_{top}-P_{tip})x(p-P_{tip})]x(P_{top}-P_{tip})$$

where, P_{top} and P_{tip} are of the cylinder that the ray hits, and x shows the cross product.

Therefore, N will be the vector perpendicular to the cylinder (which was found in the previous phase) at point p. We used N to render this point using Phong illumination model.

5.4.2 Pixels color including alpha

Color of the cylinders which passed by the CPU to the GPU is used for rendering and it is important to note that in this thesis, we are able to define a different color for all the cylinders. We can use this for two purposes: 1-slight changes in color along a hair strand (see Figure 5-6-left) and 2-different colors for different hair strands to achieve more realistic hairs (see Figure 5-7 and 5-6-right). Note that hair images of this chapter are generated using a 2.66 GHz CPU and a nVidia Quadro FX 4500 GPU.

The simple algorithm that is used here for slight random color variation is:

$$P_{i\_R} = R + randR*a$$
$$P_{i\_G} = G + randG*b$$
$$P_{i\_B} = B + randB;*c$$

Where, R, G, and B are the specified color of the hair between 0 and 1, randR, randG, and randB are random real numbers between -1 and 1, and a, b, and c, are the percentage of color variation[19].

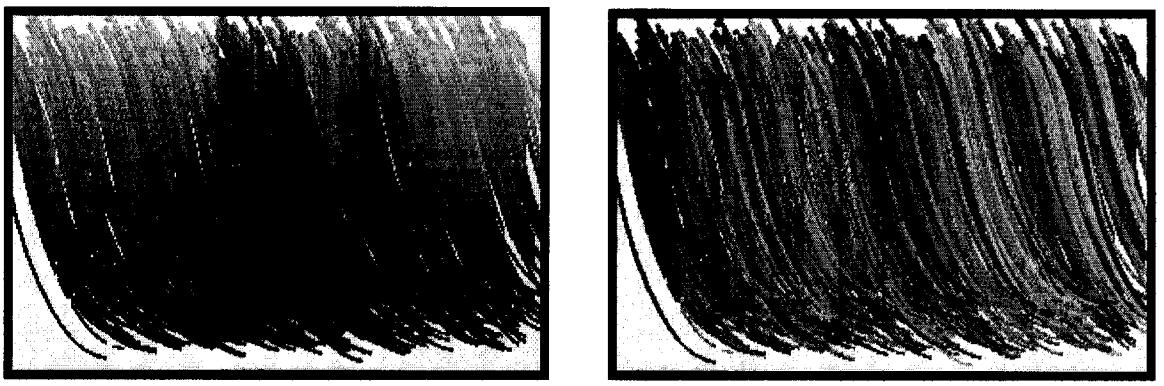


Figure 5-6: Color Variation
Left) Color of hairs vary along the hair
Right) An example of importance of color variation

[19] A very common color model is the RGB color model where R,G, and B are the percentage of red, green, and blue color in a pixel.

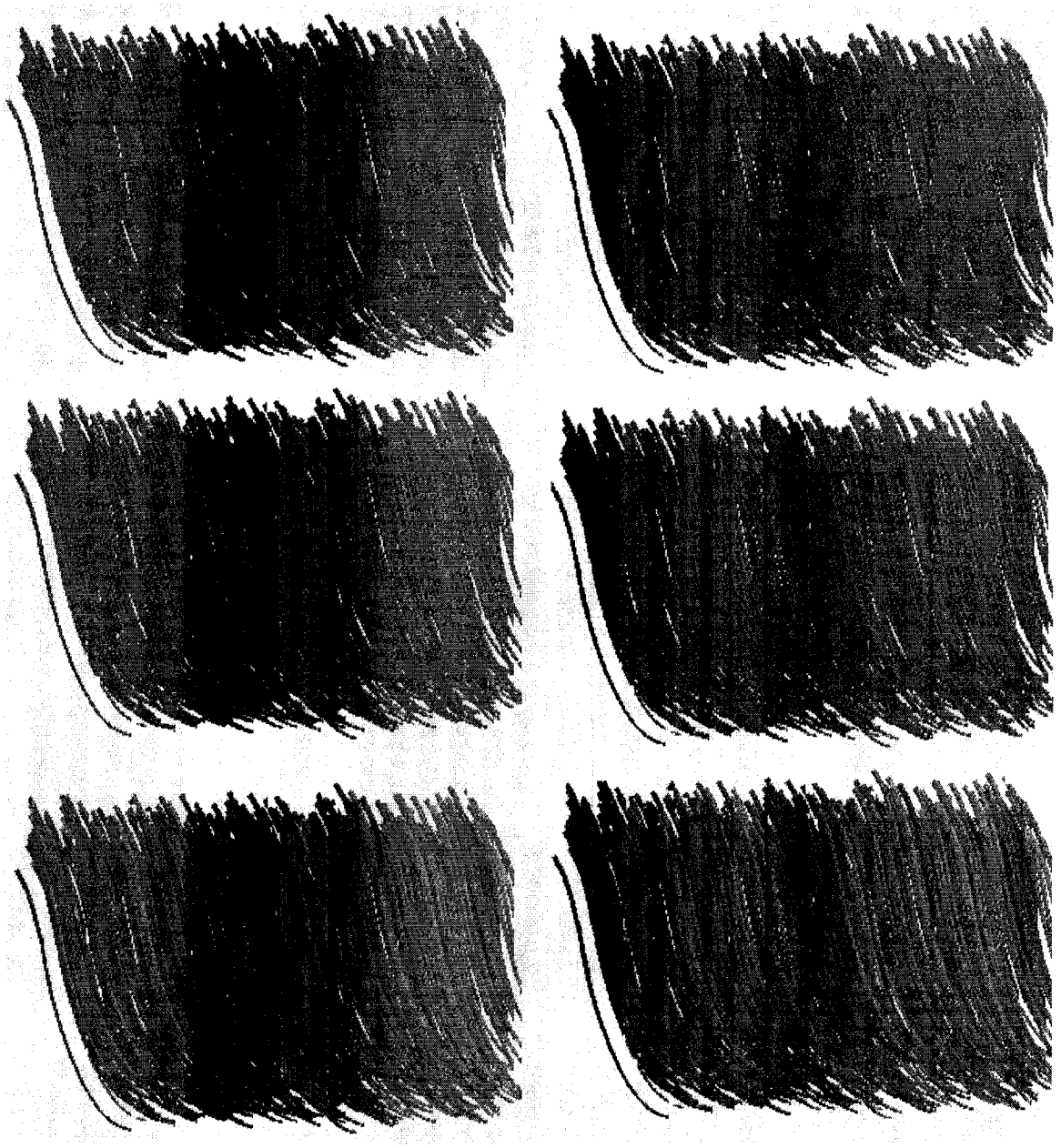


Figure 5-7: Color Variation
Top Row) All cylinders of all Hair strands have the same color
Middle Row) 7% color deference
Bottom Row) 20% color deference

Alpha value can be assigned to pixels as well. Suppose that e is the eye location. We can assign p to rO and (p-e) to rD, and then run the intersection algorithm once more with the new origin and direction for the examining ray. Using the same algorithm discussed in 5.3.4, the new intersection point (NewP) will be the pixel right behind P with respect to the viewing location (e). Then, We can assign P_{color} to the pixel color:

$$P_{color} = \alpha * color + (1-\alpha) * NewP_{color}.$$

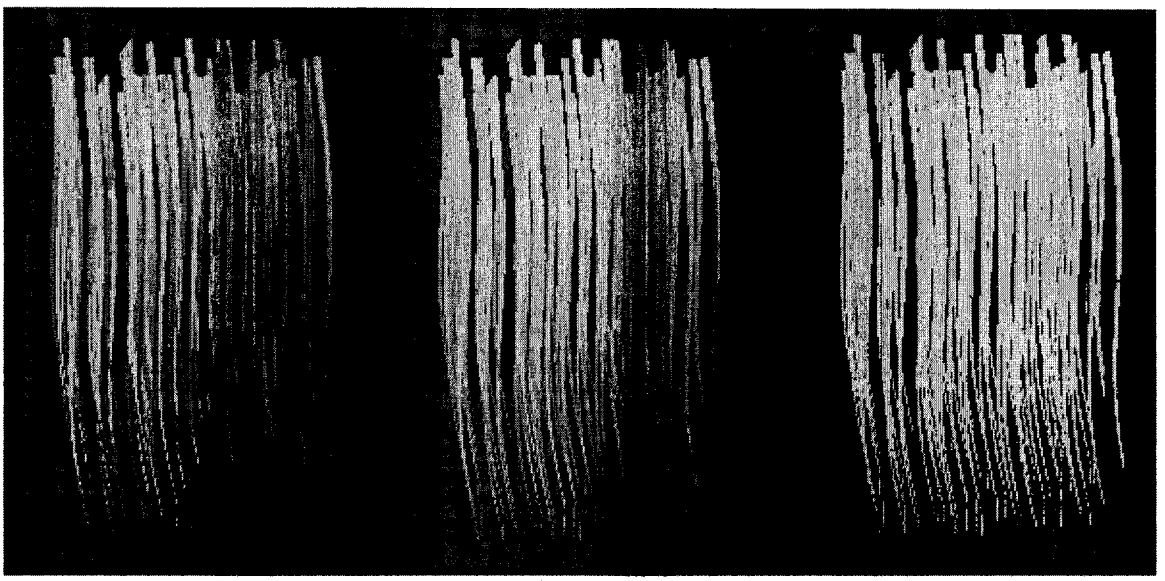


Figure 5-8 : Alpha value variation
Left) Alpha is decreasing fast along hair strands
Right) Alpha start decreasing from 6th (out of 25) cylinder
Right) Alpha=1 in all the cylinders

As you can see in Figure 5-8 and 5-9 assigning alpha value helps hair look more realistic. Although, we also decrease the radius of the hair as we discussed in section 4.3 for the sake of realism, effect of using alpha value cannot be ignored.

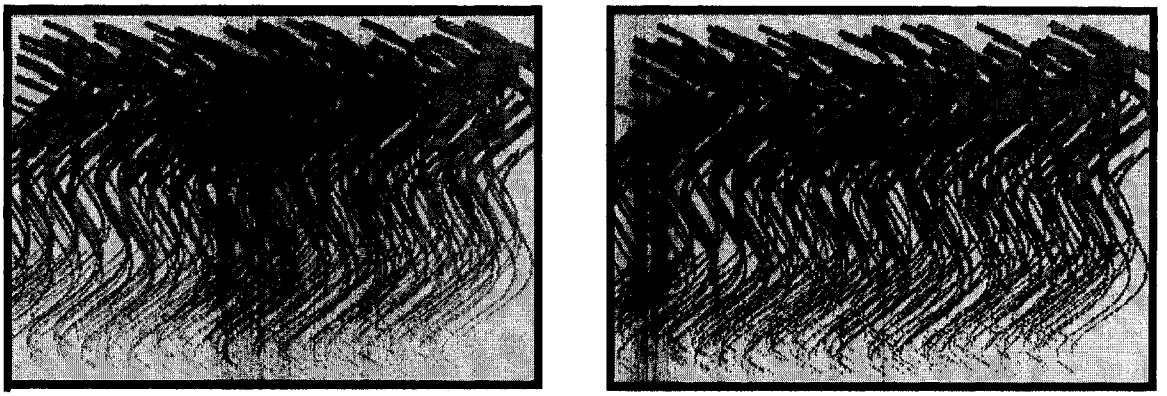


Figure 5-9: Having alpha value vs. Not having alpha value
Left) Alpha is assigned to hair strands
Right) Alpha = 1

The problem is that the rendering time will become about twice as much, because the intersection algorithm should run another time to find the back pixel color and this is because of the fact that the running time of the whole algorithm (or we can also say rendering time) is almost equal to the running time of the intersection tests.

5.4.3 Shadow

To check whether p is in shadow or not, we can assign p to rO and (L-p) to rD, where L is the light source location. If the new ray hits any cylinder in its way to the light source then we can conclude that p is in shadow. Like alpha value, shadows can also affect realism (see Figure 5-10).

Having N (p's normal), light sources locations and types, eye location, and the pixel color (considering both alpha and shadow values), Phong illumination model can be used to color p.

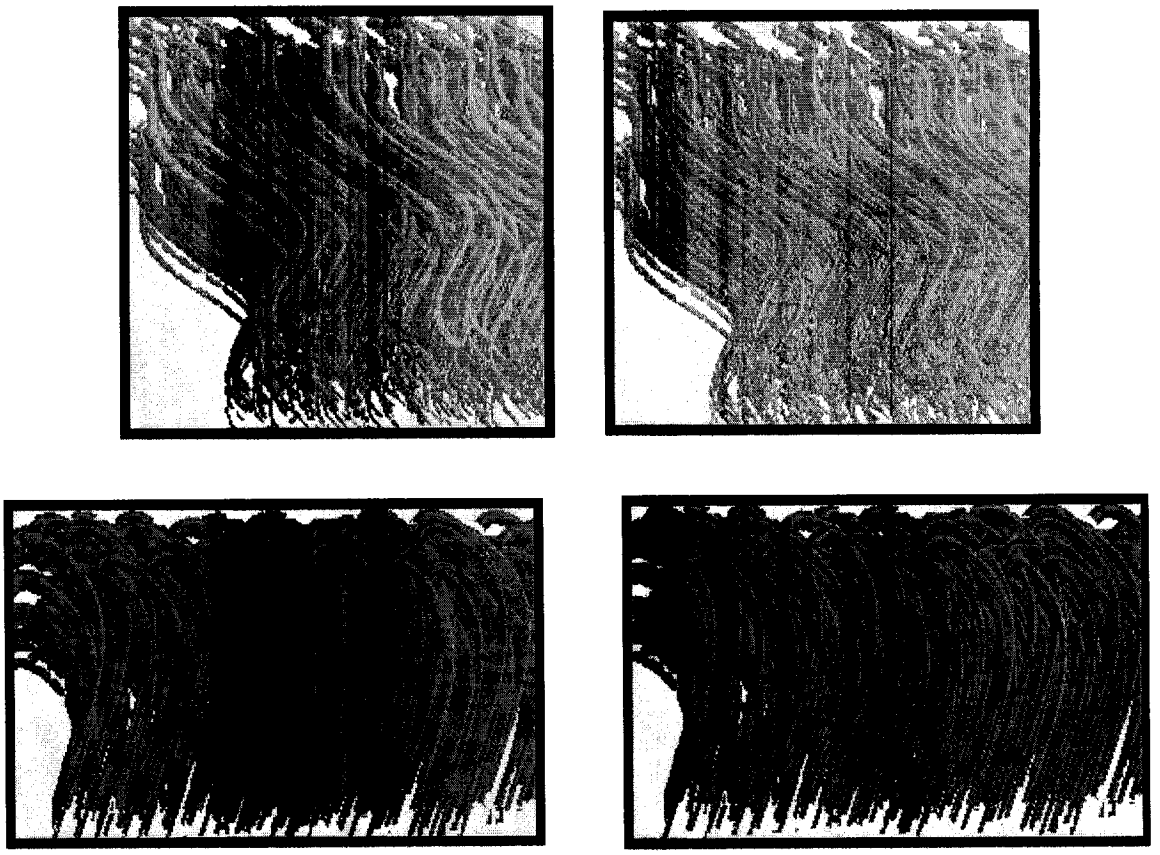


Figure 5-10: Hair Shadow
Left) Rendering hairs with shadow **Right)** Rendering hair without shadow

5.5 Running time

The expected Running time of the algorithm to find the **first collision** described in section 5.3 is n*m where n is the number of voxels in one dimension and m is expected number of cylinders in a voxel. This is because each ray is verified independently to realize whether it hits a cylinder along its way towards the end of the scene or not (GPU treats ray independently as we explained before). Checking whether the ray hits a cylinder is O(1) as it is to find if a line and a cylinder has an intersection or not.

The value of n is at most 128 but normally 64 (as we mentioned, because of the GPU limitations, we cannot divide our scene to more than 128*128*128 pixels) and the value of m depends on the resolution of the hair strands (normally 25). Based on our experiments, as long as m is about 10 i.e. we have at most 10 cylinders in each voxel, GPU works properly and having more than about 10 cylinders in any voxel causes a crash (see Figure 5-11).

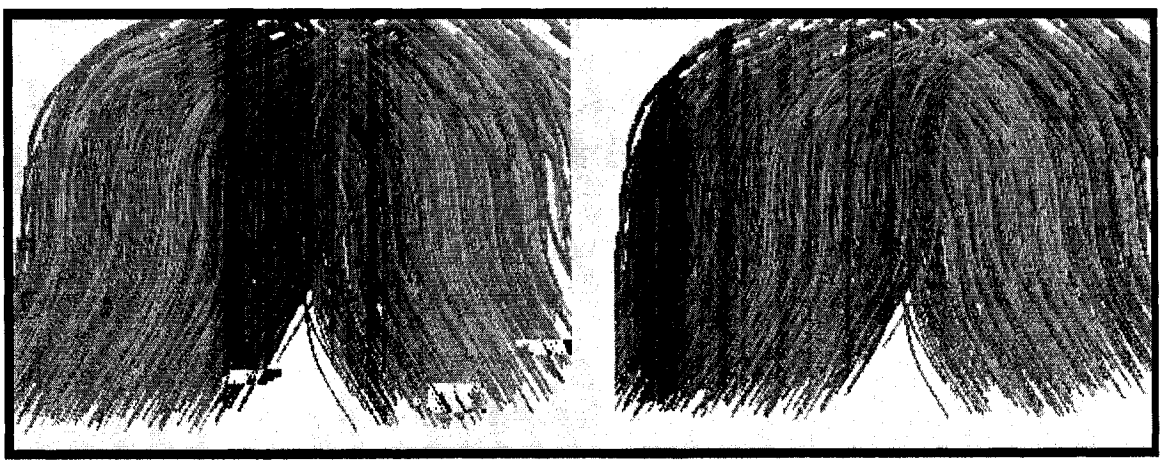


Figure 5-11: GPU fails when some voxels contains too many cylinders
Left) Resolution = 64 which means each hair strands formed using 64 cylinders
Right) Resolution = 23. The difference is not noticeable (refer to section 4.3)

Considering hairs occupy 1/8 of voxels in a head model:

$$k*Res = m*(n*n*n/8)$$

where k is total number of hairs, Res is resolution and therefore k*Res is total number of cylinders, m is expected number of cylinders in each voxel, and n is the number of voxels in one dimension.

Having n = 64, k=10,000, and m=10:

Res = 32

Having 32 cylinders for each strand is more than enough for most of hair models. Our experiments show normally having 23 cylinders to form a strand is enough[20], but since 10 is an expected number it is better to have less resolution or the GPU may still fail in some cases.

5.6 Result images

We rendered about 250,000 cylinders, which could represent about 10,000 hairs depending on resolution (number of segments (or cylinders) a strands have). The curlier a strand is the more cylinders are needed to model it, whereas straight hair can be modeled with even 3 or 4 cylinders.

Figure 5-12: Short Black and White Hairs

[20] As most of the images of this chapter rendered with 23 for resolution.

As we mentioned, we used a simplified version of the complete model introduced in previous chapters; yet, most of the key properties of the model has been implemented (see Figure 5-12 and 5-13 and 5-14)

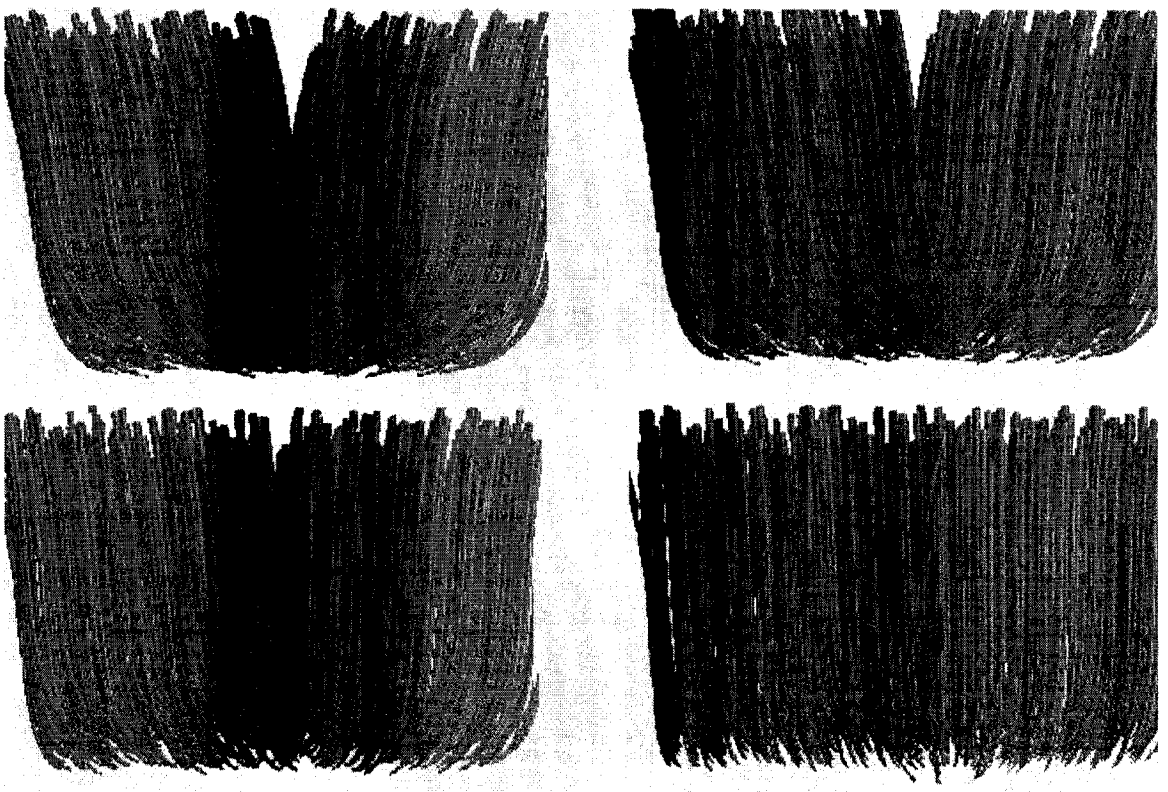


Figure 5-13: Neighbor wisps influence on each other
Top-Left) N-influence = 0 Top-Right) N-influence = 90
Bottom-Left) N-influence = 95 Bottom-right) N-influence = 100[21]

[21] Note that we usually do not want too much influence (+90) and only care about strands which are near wisp boundary.

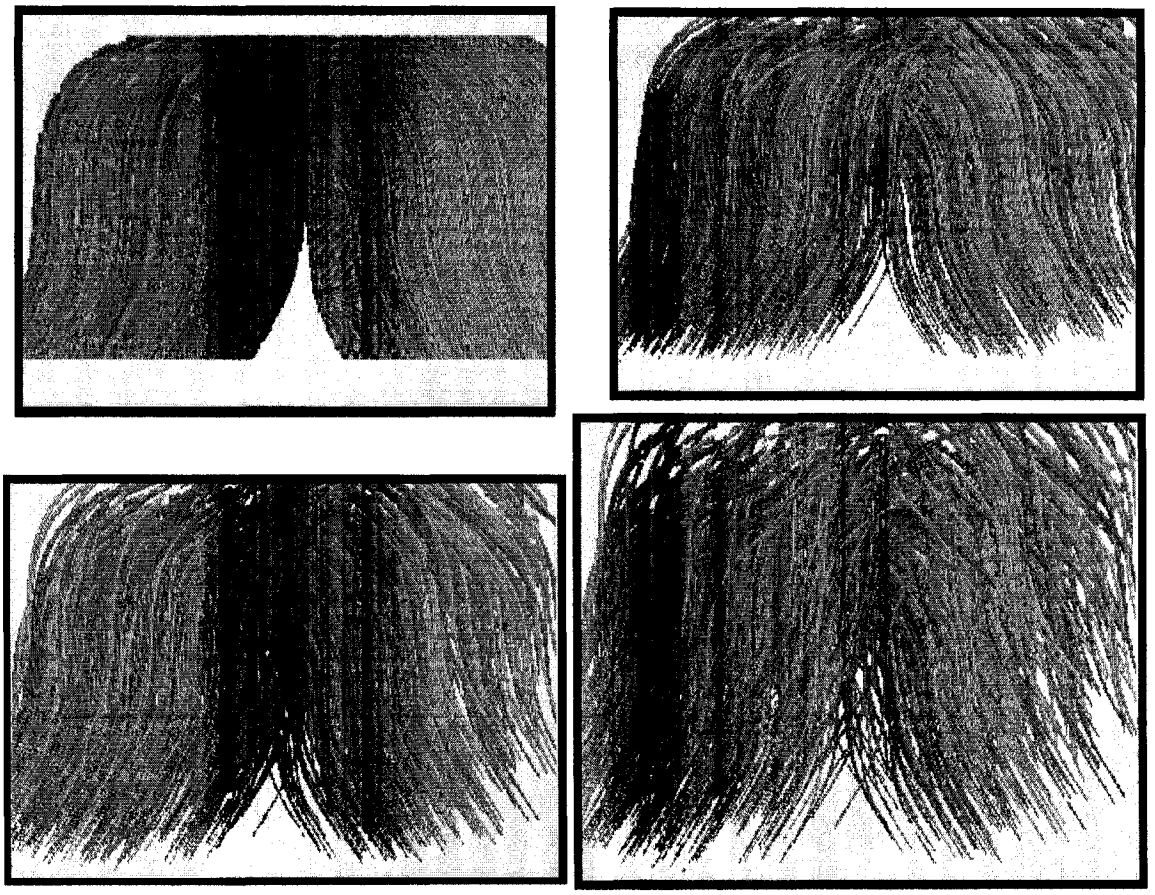


Figure 5-14: Tidiness Comparison
Left-Top) Ordinary strands are exactly the same as key strands except for their position
Right-Top) 10% untidiness
Bottom-Left) 20% untidiness Bottom-Right) 40% untidiness

Chapter6. Conclusion and Future Work

Recently, considerable progress has been made both in high-quality hair simulation and in the fast rendering and animation of hair. The **quality** of hair depends on the hair model and rendering techniques. As we have seen in this thesis, various modeling and rendering parameters have been considered to have more realistically-looking hairs.

In modeling phase, we simulated hairs introducing a simple 3D object, Continual Cylinders, to be able to represent hair strands as close as possible to reality. We considered hairs as wisps (or clusters) of hairs based on natural characteristics of hair and we added some sort of neighbor interactions to avoid unnatural looking hairs. We assigned slightly random length and orientation to strands for demonstrating various hair properties such as tidiness. And, overall we provided a model with the capability of generating different hairstyles.

In the rendering phase, we used alpha blending to be able to simulate such a thin object more realistically[22]. We used ray tracing method which is able to generate high-quality images based on reality. We rendered hair considering self-shadowing which can affect realism greatly. Also, as each hair has the color of its own and possibly some variation between the root to the tip, we assigned different color to each different segment (or cylinder) of all the hairs.

The **performance** of 3D object simulation depends mostly on the underlying rendering technique and the hardware resources. We used ray tracing which, although is one the best methods for producing high quality images, is very time-

[22] The thickness of a single hair is less than the size of a pixel in normal viewing conditions.

consuming. But not long ago, GPUs have been introduced with parallel structure suitable for fast rendering. Therefore, it is not beyond expectation to have individually simulated hairs for games in which rendering and hair dynamics need to be done in real-time.

As we rendered three dimensional hairs using ray tracing on GPU, our algorithm has many benefits over other similar ones. Fast growth of GPU technology will make it possible to render more and more hair strands without changing the algorithm itself. As we mentioned, GPUs have many limitations and therefore may fail to render properly (see Figure 5-1). As a result, every year, the introduced algorithm is able to simulate the hair model with more hair strands using newer and therefore less limited GPUs (see Figure 2-16). Also as we mentioned, in contrast to many algorithms, length of hair strands does not affect rendering time largely[23]. This is again because of the parallel structure of GPUs.

The model that has been introduced in Chapter 3 is a complete model for generating different hairstyles, whereas the rendering procedure that has been introduced in chapter 5 is based on a simplified version of this model. The head has not been rendered along with hair strands. Rendering head along with hairs requires rendering both cylinders (for hairs) and triangles (for head) at the same time and may need further research to keep it real-time.

Also, the location of the roots of hair is of great importance both for quality and for performance. We introduced a semi-uniform way in our model to distribute roots over the head to avoid unnatural hair distribution. Also, it affects performance because we should avoid having too closely placed root locations and render unnecessary hairs. Implementing this algorithm using the head model also needs future research and works.

[23] As we have seen in chapter 5, rendering time mostly depends on number of cylinders in each voxel and having long hairs would not change this number.

In **hair modeling**, being able to generate all possible hairstyles without too much effort is the goal yet to be achieved. Although our model is able to produce many different hairstyles, there still are some certain styles that our algorithm could fail to generate. A hair strand does not necessarily fit in a plane and can occupy 3D space (see Figure 6-1). Providing an easy way to design and then increase this kind of strands can be quite challenging. Also, it is impossible to define wisps following a certain key strand for some hairstyles, for example hairstyles using hair ties can be very tedious to generate with our model. Some tricks like using fake key strands (see Figure 3-17-Top-Left) can be used, but the procedure is both tedious and inaccurate.

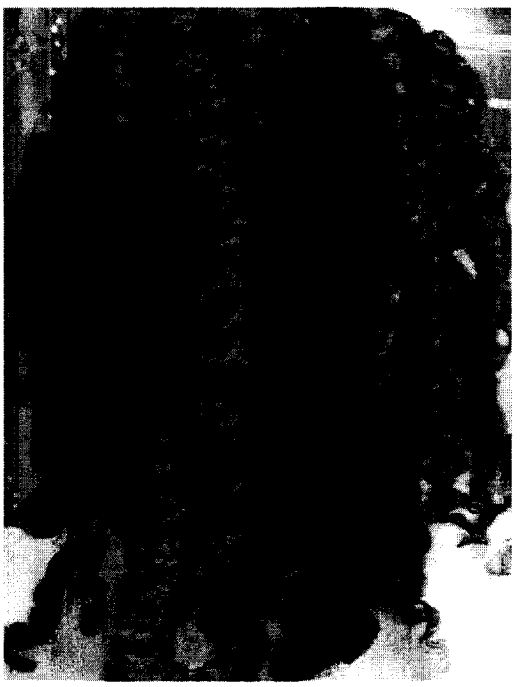


Figure 6-1: Sample hair model with strands occupying 3D space

In **hair rendering**, being able to generate a full head model (with 100,000 hairs) realistically and in real-time is the goal yet to be achieved. Future GPUs

will definitely increase the number of hairs that can be rendered in real-time, yet realism issues need further research. Self-shadowing, alpha blending, and assigning segments colors based on reality is very important, but being able to render hairs with different properties such as wetness, softness, etc. is equally important. For example, wet hairs can be rendered darker but shinier and some certain formula can be assigned to alpha value for various softness (the more alpha value is, the softer hair looks).

Yet, a very normal hairstyle generated by computer even off-line can be easily identified from a natural one. So, rendering hair realistically need further research whereas hair modeling may not be able to generate all sort of hairstyles but those that it can generate can be considered natural.

Hair animation is perhaps the most important reason that we model and render hairs. Like many other hair models, our hairs are modeled using segments. Therefore, considering these segments as springs which we have certain formula for their behavior in Physics, dynamics of hair can be simulated[24]. Also, methods used in Robotics like forward kinematics can be used considering strands as chain of rigid segments. Furthermore, since our model is based on key strand methods, we can simulate dynamics of the key strand of each wisp (typically just 10 key strands) and then, generate the hair model based on new positions and shapes of key strands to produce the whole head animation.

Collision detection and handling between strands and other objects and among strands themselves is very hard and time-consuming. Again hair physical properties could be considered (or estimated by for example springs) to detect and then handle collisions. Using cylinders makes detecting collision easier, yet, we have too many segments even in a single wisp (typically 25000). Certain data

[24] Mass-Spring Systems is the oldest attempt to simulate dynamics of hairs introduced by Rosenblum *et al.* [RCT91].

structures could simplify the process for the reason that some segments cannot collide with each other and need not to be checked.

In summary, we focused our studies on hair modeling and rendering with potential abilities to simulate hair dynamics and run collision tests. Hair simulating is still a very challenging research topic and has progressed a lot recently. An accurate physical model taking into account all properties of hair could solve most of the problems in this area however the difficulties in such an approach necessitate the development of approximation-based simulations.

REFERENCES

[Ap68] A. Appel, "Some techniques for shading machine renderings of solids," in *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[AUK92] K. Anjyo, Y. Usami, and T. Kurihara, "A simple method for extracting the natural beauty of hair," in *Proceedings of ACM SIGGRAPH 1992*, ser. Computer Graphics Proceedings, Annual Conference Series, pp. 111–120, August 1992.

[AW87] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *G. Marechal, editor, Proc. EUROGRAPHICS '87,* pp. 3-10, 1987.

[BA+05] F. Bertails, B. Audoly, B. Querleux, F. Leroy, J.-L. Leveque, and M.-P. Cani, "Predicting natural hair shapes by solving the statics of flexible rods," in *Eurographics (short papers)*, August 2005.

[BA+06] F. Bertails, B. Audoly, M.-P. Cani, B. Querleux, F. Leroy, and J.-L. L´ev^eque, "Super-helices for predicting the dynamics of natural hair," in *ACM Transactions on Graphics (Proceedings of the SIGGRAPH conference)*, August 2006.

[BeK+03] F. Bertails, T.-Y. Kim, M.-P. Cani, and U. Neumann, "Adaptive wisp tree - a multiresolution control structure for simulating dynamic clustering in hair motion," in *ACM SIGGRAPH Symposium on Computer Animation*, pp. 207–213, July 2003.

[BK+00] M. Berg, M. Kreveld, M. Overmars, O. Schwarzkopf, "Computational Geometry: Algorithms and applications," 3rd ed., Springer 2000.

[BMC05] F. Bertails, C. Ménier, and M.-P. Cani, "A practical self-shadowing algorithm for interactive hair animation," in *Graphics Interface*, graphics Interface'05, May 2005.

[CA+04] J. Chaung, N. Ahuja, C. Lin, C. Tsai, and C. Chen, "A potential-based Generalized Cylinder representation," in *Computer & Graphics,* vol. 28, issue 6, pp. 907-918, December 2004.

[CJ+04] M. Côté, P. M. Jodoin, C. Donohue, and V. Ostromoukhov, "Non-photorealistic rendering of hair for animated cartoons," in *Proceeding of GRAPHICON'04,* 2004.

[CK05] B. Choe and H.-S. Ko, "A statiscal wisp model and pseudophysical approcahes for interactive hairstyle generation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 2, March 2005.

[CS+99] L. Chen, S. Saeyor, H. Dohi, and M. Ishizuka, "A system of 3d hairstyle synthesis based on the wisp model," *The Visual Computer*, vol. 15, no. 4, pp. 159–170, 1999.

[FD+90] D. Foley, A. van Dam, S. K. Feiner, J.F.Hughes, "Computer Graphics: Principles and Practice," in *C (2nd Edition)*, Addison-Wesley, Wokingham, 1990.

[Ha05] I. Hanak, "The GPU and Graphics algorithm," Technical Report No. DCSE/TR-*2005-05,* April, 2005.

[Hai08] Healthy Hair Salon and Corrective Skin Clinic, Hair & Skin-ology, http://www.hairandskinology.com/hairologymenu.htm, August 2008.

[HB04] D. Hearn, M. –P. Baker, "Computer Graphics with OpenGL," 3rd ed., Pearson Prentice Hall, 2004.

[IM08] Photoes from Beowulf, IMDB, http://www.imdb.com/media/rm2257426688/tt0442933, July 2008.

[KBR06] J. Kessenich, D. Baldwin, R. Rost, "The OpenGL shading language specification," 3Dlabs, Inc. Ltd, 2006.

[KH00] C. Koh and Z. Huang, "Real-time animation of human hair modeled in strips," in *Computer Animation and Simulation'00*, pp. 101–112, Sept. 2000.

[KH01] C. Koh and Z. Huang, "A simple physics model to animate human hair modeled in 2D strips in real-time," in *Computer Animation and Simulation '01*, pp. 127–138, Sept. 2001.

[KHS04] M. Koster, J. Haber, and H.-P. Seidel, "Real-time rendering of human hair using programmable graphics hardware," in *Computer Graphics International (CGI)*, pp. 248–256, June 2004.

[Kir+04] D. Kirk, "Cg Toolkit User's Manual. A Developer's Guide to Programmable Graphics (Release 1.2)," *NVIDIA Corporation*, January 2004.

[KiN02] T.-Y. Kim and U. Neumann, "Interactive multiresolution hair modeling and editing," *ACM Transactions on Graphics*, proceedings of ACM SIGGRAPH 2002,vol. 21, no. 3, pp. 620–629, July 2002.

[KK90] J. Kajiya and T. Kay, "Rendering fur with three dimensional textures," in *Proceedings of ACM SIGGRAPH 89*, ser. Computer Graphics Proceedings, Annual Conference Series, pp. 271–280, 1989.

[LTT91] A. M. LeBlanc, R. Turner, and D. Thalmann, "Rendering hair using pixel blending and shadow buffers," *The Journal of Visualization and Computer Animation*, vol. 2, no. 3, pp. 92–97, 1991.

[MI05] X. Mao, S. Isobe, K. Anjyo, and A. Imamiya, "Sketchy hairstyles," in *Proceedings of Computer Graphics International*, 2005.

[MK+02] X. Mao, K. Kashio, H. Kato, A. Imamiya, "Intractive hairstyle modeling using a sketching interface," Lecture Notes in Computre Science Vol. 2330, Proceedings of the International Conference on Computational Science-Part II, pp. 131-140, 2002.

[MK+04] X. Mao, H. Kato, A. Imamiya, K. Anjyo, "Sketch interface based expressive hairstyle modeling and rendering," in *Computer Graphics international 04 proceedings,* pp. 608-611, 2004.

[Ng07] H. Nguyen, "GPU Gems 3," 1st ed., Pearson Education, Inc, 2006.

[NP98] I. Neulander and M. van de Panne, "Rendering Generalized Cylinders with Paintstrokes," in *Graphics Interface*, 1998.

[NT04] P. Noble and W. Tang, "Modelling and animating cartoon hair with NURBS surfaces," in *Computer Graphics International (CGI)*, pp. 60–67, June 2004.

[Nv08] Graphics Processing Unit (GPU), NVIDIA, http://www.nVidia.com/object/gpu.html, July 2008.

[Os07] M. Oshita, "Real-time hair simulation on GPU with a dynamic wisp model," in *Computer Animation and Virtual Worlds '07,* vol 18. pp. 583-593, July 2007.

[PB+02] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray Tracing on Programmable Graphics Hardware," in *Proceedings of SIGGRAPH 2002*, 2002.

[PH89] K. Perlin and E. Hoffert, "Hypertexture," in *Proc. of SIGGRAPH '89*, pp. 253–262, 1989.

[Pi08] Render for Maya, Pixar, https://renderman.pixar.com/products/tools/rfm_webinfopage.html , August 2008.

[PS+06] H. Pabst, J. Springer, A. Schollmeyer, R. Lenhardt, C. Lessig, B, Froehlich, "Ray Casting of Trimmed NURBS Surfaces on the GPU," in IEEE Symposium, on Interactive Ray Tracing, September 2006.

[Rob00] C.R. Robbins, "Chemical and Physical Behavior of Human Hair," fourth ed., *Springer-Verlag*, Dec. 2000.

[Ro04] R. –j. Rost, "OpenGL Shading Language," 1st ed., Pearson Education, Inc, 2004.

[Si67] R. T. Sims, "The measurement of hair growth," in British Journal of Nutrition, vol 22, pp 229-236, 1967.

[So08] cgKit Tutorial, SourceForge, http://cgkit.sourceforge.net/tutorials/materials/materials.html, May 2008.

[SY05] E. Sugisaki, Y. Yu, K. Anjyo, and S. Morishima, "Simulation-based cartoon hair animation," in *Proceedings of the 13th Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2005.

[To03] L. Tong, Harvard University, Mazur's Research Group, 2003, http://www.nsf.gov/od/lpa/news/03/pr03147.htm, July 2008.

[Ubi08] Assassin's Creed, Ubisoft, http://www.gamespot.com/pc/action/assassinscreed/index.html?tag=result;title, July 2008.

[Up92] S. Upstill, "The RenderMan Companion, A Programmer's Guide to Realistic," in Computer Graphics, Addson-Wesley, 1992.

[WaB+07] K.Ward, F. Bertails, T.-Y. Kim, S. R. Marschner, M.-P. Cani, and M. Lin, "A survey on hair modeling: Styling, simulation, and rendering," *IEEE Transactions on Visualization and Computer Graphics (TVCG), 13(2)*, pp. 213–34, Mar-Apr 2007.

[WBC07] J. Wither, F. Bertails, M.-P. Cani, "Realistic Hair from a Sketch," in *Shape Modeling International*, June 2007.

[Wh79] T. Whitted, "An illumination model for shaded display," in *ACM SIGGRAPH Computer Graphics,* vol 13, issue 2, 1979.

[WOQ05] Y.Wei, E. Ofek, L. Quan, and H.-Y. Shum, "Modeling hair from multiple views," in *Proceedings of ACM SIGGRAPH'05*, 2005.

[WS89] Y. Watanabe, Y. Suenaga, "Drawing human hair using the wisp model," in *Proceedings of Computer Graphics International*, pp 691- 700, 1989.

[WS92] Y. Watanabe and Y. Suenaga, "A trigonal prism-based method for hair image generation," *IEEE Computer Graphics and Applications*, vol. 12, no. 1, pp. 47–53, Jan 1992.

[XL+06] S. Xu, F. -C.M. Lau, H. Jiang, and Y. Pan, "A Novel Method for Fast and High-Quality Rendering of Hair," in *Proceedings of the 17th Eurographics Symposium on Rendering (EGSR'06)*, Nicosia, Cyprus, June 2006.

[XY01] Z. Xu and X. D. Yang, "V-hairstudio: an interactive tool for hair design," *IEEE Computer Graphics & Applications*, vol. 21, no. 3, pp. 36–42, May/June 2001.

[YCJ02] Y. Yu, J. Chang, J. Jin, "Efficient and realistic hair modeling and animation using sparse guide effect," *University of Illinois at Urbana-Chapaign*, 2002.

[YX+00] X. D. Yang, Z. Xu, T. Wang, and J. Yang, "The cluster hair model," *Graphics Models and Image Processing*, vol. 62, no. 2, pp. 85–103, Mar. 2000.

[ZW06] R. Zhang and B. C. Wuensche, "Interactive Styling of Virtual Hair," in *Proceedings of IVCNZ '06*, Great Barrier Island, New Zealand, pp. 143-148, November 2006.

APPENDIX A: RENDERING COMPARISON

Here, the rendering hours, hair rendering techniques and some additional information of some famous animations are compared.

Animations	Hair Rendering Technique	Rendering Hours	Budget	Gross Revenue	Company	Year
Toy Story[1]	Volume shaders[2]	0.8M	$30M	$354M	Pixar	1995
Toy Story 2	RenderMan for Maya[3]	2M	$90M	$485M	Disney/ pixar	1999
Monsters, Inc.	Mass-Spring Model	25M	$115M	$525M	Pixar	2001
Final Fantasy	Splines + lighting	22.5M	$137M	$85M	Square	2001
Ice Age	Ray Tracing	19.5M	$60M	$383M	Blue Sky	2002
Incredibles	Volumetric Method	13M	$92M	$631M	Disney	2004
Shrek 3	Ray Tracing	20M	$160M	$797M	Dreamwork	2007
Ratatouille	RenderMan for Maya	13.4M	$150M	$620M	Pixar	2007
Kong Fu Panda	Ray Tracing	24M	$130M	$363M*	Dreamwork	2008

1-The first completely computer-generated animated movie

2-Volume shader introduce by [Up92].

3-Full information can be found in [Pi08].

* Till now

APPENDIX B: GPU LIMITATION

GPU resources in both vertex shader and pixel shader are limited as you can see in the following tables.

Vertex Shader				
Value Description	*Value for given Shader Model*			
	1.1	*2.0*	*2.x*	*3.0*
Total maximum number of instruction slots.	128	256	256	≥ 512
Maximum static flow control instructions per shader‡.	–	16	16	∞
Maximum static flow control nesting depth.	–	viz. static flow count‡	viz. static flow count‡	24
Maximum dynamic flow control nesting depth.	–	–	≤ 24	24
Maximum loop/rep flow control nesting depth.	–	1	≥ 1; ≤ 4	4
Maximum subroutine call nesting depth.	–	1	≥ 1; ≤ 4	4

Pixel Shader				
Value Description	*Value for given Shader Model*			
	1.x/1.4	*2.0*	*2.x*	*3.0*
Total maximum number of instruction slots. For lower versions, the instruction count is divided between texture (first number) and arithmetic (second number) instructions.	4 + 8/6 + 8	32 + 64	≥ 96	≥ 512
Maximum static flow control nesting depth.	–	–	24 or 0	24
Maximum dynamic flow control nesting depth.	–	–	≤ 24	24
Maximum loop/rep flow control nesting depth.	–	–	≤ 4	4
Maximum subroutine call nesting depth.	–	–	≤ 4	4

[Ha05]

APPENDIX C: BEZIER CURVES

Figure C-1: Bézier Curves

Bézier curve is all the points created by the following formula [HB04]:

$$\mathbf{P}(u) = \sum_{k=0}^{n} p_k B_{k,n}(u), \qquad 0 \le u \le 1$$

$$B_{k,n}(u) = C(n,k)u^k(1-u)^{n-k} = \frac{n!}{k!(n-k)!}u^k(1-u)^{n-k}$$

Where, $p_k = (x_k, y_k, z_k)$ and p_0 through p_n are the n+1 control-point positions that control the shape of the final curve, P(u). One of the properties of any Bézier curve is that $P(0) = p_0$ and $P(1) = p_n$. Thus, the curve P(u) always connects the first and last control points.

APPENDIX D: A FAST VOXEL TRAVERSAL ALGORITHM FOR RAY TRACING

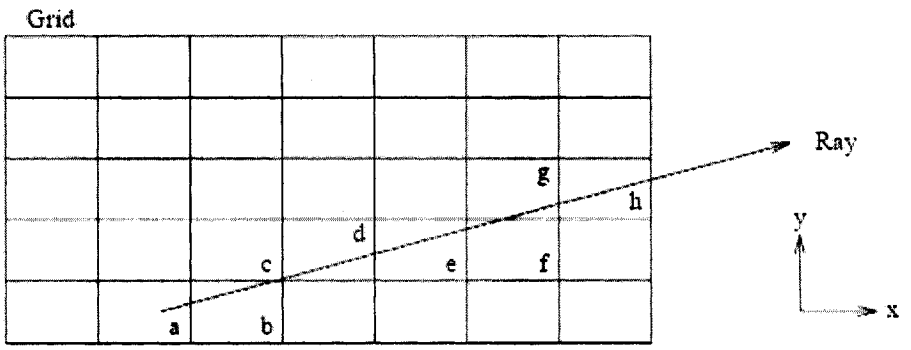


Figure D-1: 2D Traversal for Ray Tracing
[AW87]

In 1987, Amanatides and woo introduced a fast traversal algorithm for tracing the rays. Let us first consider the 2D version of this problem. The correct traverse of the grid in Figure 7-1 is to visit voxels in the order: a, b, c, d, e, f, g, and h, considering the fact that the ray originates from a.

Their traversal algorithm consists of two phases: initialization and incremental traversal. The initialization phase is to first find the voxel in which the ray origin is found. If the ray origin is outside of the grid, the point in which the ray enters the grid will be found and the corresponding voxel will be considered as the first voxel. The starting voxel coordinates are specified by the integer variables X and Y. Moreover, the variables stepX and stepY are initialized to either 1 or -1

indicating whether X and Y are incremented or decremented as the ray crosses voxel boundaries. If we consider the ray as:

r = t*rD + rO,

where rO is the ray origin and rD is the ray direction value of **t** at which the ray crosses the first vertical voxel boundary is tMaxX and the value of **t** at which the ray crosses the first horizontal voxel boundary is tMaxY. Minimum of tMaxX and tMaxY will indicate how much we can travel along the ray and still remain in the current voxel. Here is the 2D traversal algorithm:

```
loop {
        if(tMaxX < tMaxY) {
                tMaxX= tMaxX + tDeltaX;
                X= X + stepX;
        } else {
                tMaxY= tMaxY + tDeltaY;
                Y= Y + stepY;
        }
        NextVoxel(X,Y);
}
```

Where tDeltaX (or tDeltaY) indicates how far along the ray we must move for the horizontal (or vertical) component of such a movement to equal the width (or height) of a voxel.

We loop until either we find a voxel with a non-empty object list or we fall out of the end of the grid.

Extending the algorithm to three dimensions simply requires that we add the appropriate z variables and find the minimum of tMaxX, tMaxY and tMaxZ during each iteration. This results in:

```
list= NIL;
do {
        if(tMaxX < tMaxY) {
                if(tMaxX < tMaxZ) {
                        X= X + stepX;
                        if(X == justOutX)
                                return(NIL); /* outside grid */
                        tMaxX= tMaxX + tDeltaX;
                } else {
                        Z= Z + stepZ;
                        if(Z == justOutZ)
                                return(NIL);
                        tMaxZ= tMaxZ + tDeltaZ;
                }
        } else {
                if(tMaxY < tMaxZ) {
                        Y= Y + stepY;
                        if(Y == justOutY)
                                return(NIL);
                                tMaxY= tMaxY + tDeltaY;
                        } else {
                                Z= Z + stepZ;
                                if(Z == justOutZ)
                                        return(NIL);
                        tMaxZ= tMaxZ + tDeltaZ;
                        }
                }
                list= ObjectList[X][Y][Z];
        } while(list == NIL);
return(list);
```