IMPROVEMENT AND ANALYSIS OF TRESSFX REAL-TIME HAIR SIMULATION FRAMEWORK

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

DENIZ UĞURCA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
GAME TECHNOLOGIES

MAY 2015

Approval of the thesis:

IMPROVEMENT AND ANALYSIS OF TRESSFX REAL-TIME HAIR SIMULATION FRAMEWORK

submitted by **DENIZ UĞURCA** in partial fulfillment of the requirements for the degree of **Master of Science in Game Technologies Department, Middle East Technical University** by,

Prof. Dr. Nazife Baykal Director, **Informatics Institute, METU**	______________
Assist. Prof. Dr. Hüseyin Hacıhabiboğlu Head of Department, **Modelling and Simulation, METU**	______________
Prof. Dr. Veysi İşler Supervisor, **Computer Engineering, METU**	______________

Examining Committee Members:

Assoc. Prof. Dr. Alptekin Temizel Modeling and Simulation, METU	______________
Prof. Dr. Veysi İşler Computer Engineering Department, METU	______________
Assoc. Prof. Dr. Hüseyin Hacıhabiboğlu Modeling and Simulation, METU	______________
Assist. Prof. Dr. Murat Yılmaz Computer Engineering Department, Cankaya University	______________
Dr. Erdal Yılmaz Argedor, Ankara	______________

Date: **4 May 2015**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name:	DENIZ UĞURCA
Signature :	

ABSTRACT

IMPROVEMENT AND ANALYSIS OF TRESSFX REAL-TIME HAIR SIMULATION FRAMEWORK

Uğurca, Deniz
M.S., Department of Game Technologies
Supervisor : Prof. Dr. Veysi İşler

May 2015, 44 pages

One of the single largest challenges in today's game production is the simulation and rendering of realistic hair in real time. In most games, hair and fur are usually covered or simplified with textured meshes. TressFX real-time GPU hair framework, which is used in Tomb Raider (2013) game, includes realistic hair by utilizing parallel nature of GPUs. This framework, however, lacks one of the most distinctive properties of hair: Inter-hair interaction. Even though calculating this interaction in real-time is an expensive task, equalizing hair velocities gives the illusion of hair collision, thus, creating better visuals, at the expense of some performance loss in a cheaper way. In this study, an efficient way to address hair-hair collisions is implemented using uniform girds to improve realism of TressFX framework. Moreover, a user study is conducted to quantitatively measure the quality improvement. The results demonstrate that there is a significant difference in users' perception of simulation quality in support of the proposed method, while performance characteristics of simulation are not effected.

Keywords: Hair Simulation, Hair Physics, Verlet Integration, Direct Compute, GPU

ÖZ

GERÇEK ZAMANLI SAÇ BENZEŞİM PROGRAMI TRESSFX'İN İNCELEME VE GELİŞTİRİLMESİ

Uğurca, Deniz
Yüksek Lisans, Oyun Teknolojileri Bölümü
Tez Yöneticisi : Prof. Dr. Veysi İşler

Mayıs 2015 , 44 sayfa

Günümüzdeki oyun üretiminde en büyük zorluklardan biri, gerçek zamanlı ve gerçekçi saç benzetimi ve görselleştirmesidir. Çoğu oyunda saç ve tüy kapatılmakta ya da basitleştirilmiş dokulu ağlar kullanılmaktadır. Tomb Raider (2013) oyununda kullanılan TressFX sistemi, grafik işlemcilerinin paralel doğasından yararlanarak gerçek zamanlı saç kullanmaktadır. Fakat bu yöntemde saçın en karakteristik özelliklerinden biri olan saç-saç etkileşimlerine değinilmemiştir. Saç-saç etkileşimini gerçek zamanda hesaplamak oldukça pahalı olsa da, saçların hızlarını eşitletmek saç çarpışması yanılsamasını yaratarak daha iyi görselleri, bir miktar performans kaybıyla daha ucuza sağlar. Bu çalışmada, saç-saç çarpışması verimli bir şekilde, düzgün grid kullanılarak, TressFX sisteminin gerçekçiliğinin arttırılarak geliştirilmesi hedeflenmiştir. Buna ek olarak bir kullanıcı çalışmasıyla da sayısal olarak bu iyileştirme ölçülmüştür. Sonuçlar, önerilen metodda kullanıcıların benzetimdeki kalite algısında önerilen yöntem lehinde anlamlı bir fark olduğunu gösterirken, benzetimin performans özelliklerinde değişim olmadığını ortaya koymuştur.

Anahtar Kelimeler: Saç Simülasyonu, Saç Fiziği, Verlet Yöntemi, DirectCompute, GPU

To those who can't wait.

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor, Prof. Dr. Veysi İşler for invaluable guidance and support.

I would like to thank Dr. Erdal Yılmaz for his advices, patience and remarkable personality.

I would like to thank AMD for opening TressFX framework to general use and Kenneth Ellersdorfer for initial Unity3D implementation of the TressFX framework.

I would also like to thank Mustafa Mert Karaöz for his valuable technical insights.

Lastly, I would like to thank my mother Z. Günsel Uğurca and my father Rıfat Uğurca; for without them none of this would be possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CPU	Central Processing Unit
GPU	Graphics Processing Unit
GPGPU	General Purpose GPU
SoC	System on a Chip
FS	Fragment Shader
PS	Pixel Shader
VS	Vertex Shader
CS	Compute Shader
PBD	Point Based Dynamics
PBM	Point Based Methods
FEM	Finite Element Method
ODE	Ordinary Differential Equations
DAE	Differential Algebraic Equations
LOD	Level of Detail

CHAPTER 1

INTRODUCTION

Although it is currently common for the gaming industry to use characters with hair in realtime applications, it is still one of the most difficult issues in development. Usually textured meshes are used to yield plausible results, but these approaches are not well suited for realistic scenes. Since there is still a lack of fast and accurate method to simulate thousands of hairs in real time, static hair with texture-only methods are common in industry. Even games including hair simulation, will still limit the number of characters with long and interactive hair. Avatars and NPCs often have hats, helmets, or they can even be completely bald to reduce performance requirements considering simulating real time hair is a daunting task (See Figure: 1.1). Even so, most games and interactive applications are trying to include characters with interactive hair, as it is one of the defining characteristics of humans. For this reason in PC version of the Tomb Raider game, released in 2013, included TressFX hair simulation framework to simulate full head of hair in real time (See Figure: 1.2).

1.1 Motivation

Typically human hair includes 100.000 - 150.000 strands. Every hair strand is a thin structure with dynamic features. Even though a hair strand has elastic properties, as it can easily bend and twist, it is nearly inextensible and resist to shearing [1]. Hair strands are also notably small in diameter, which makes the simulation problem more difficult. To overcome this problem, a robust and efficient method is needed.

Currently, there is no widely accepted method to simulate hair. Since, every person has a different hair style (color, diameter and general shape) it is also quite a hard task to simulate hair for individuals considering every ethnic background have different characteristics of hair [2].

Having a deformable soft-body (such as cloth or fur) in a 3D scene greatly improves the reality in virtual environments but this improvement is usually limited with our current generation of computers. With some exceptions, simulating soft-bodies may be processed in parallel. Along with the rapid progression in multi-core hardware, especially in GPUs, they can now be utilized to perform these tasks. Also, with better simulation methodologies, it is now closer to become common in games.

Figure 1.1: Last of Us Remastered

1.2 Contributions

Although TressFX is a revolutionary technology in gaming, it still lacks one key aspect of the believable hair simulation: Inter hair collision. In this study, a visual improvement by adding inter-hair collision, over the existing framework is proposed. A user study is also conducted to determine the effectiveness of the proposed method. Furthermore, performance characteristics of this method is analyzed and discussed.

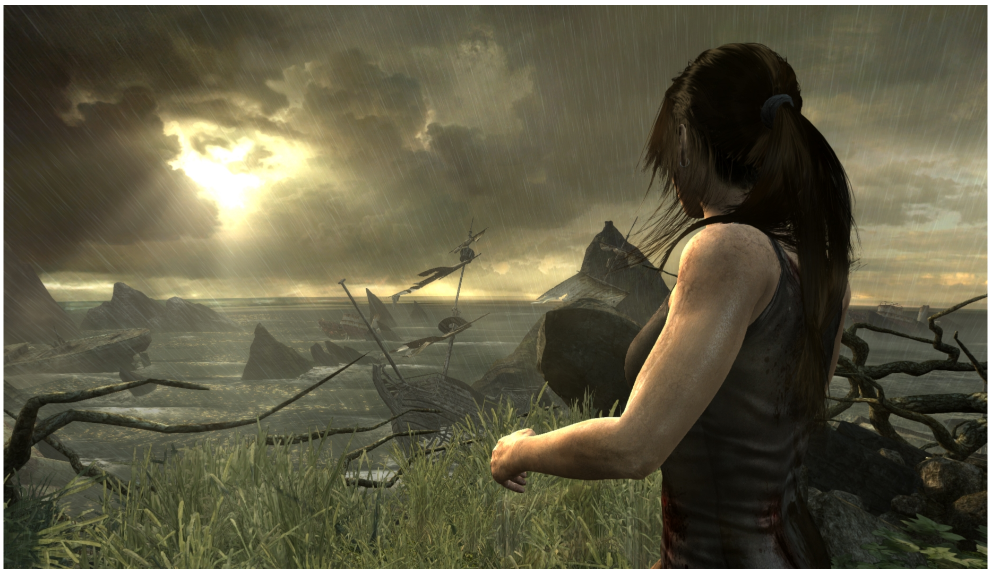


Figure 1.2: Tomb Raider -TressFX Hair

CHAPTER 2

BACKGROUND

In this section, background for physically based simulation and some useful information on GPU computing and DirectCompute is given. Additionally, some basic knowledge about hair in humans is provided.

2.1 Physically Based Simulation

In the early days of visualization and special effects, creating believable and plausible animations was a tedious task. Animators had to work long hours to create even for a few seconds of animation. To reduce the workload on animators, and create more plausible results, physically based animation has emerged. Today, physically based simulation is a particularly active research field in computer graphics [3]. Although it is computationally expensive to represent the physical phenomena accurately, recent rapid growth of computational power makes real time physically based animation a reality.

2.2 Mass-Spring Systems

Deformable bodies are hard to capture in simulation environments. To simplify this phenomena, mass-spring networks can be used. A mass-spring network is a simple approach to model deformable bodies [3]. Let's consider two particles with masses m_0 and m_1 , with positions p_0 and p_1, and with velocities v_0 and v_1. Forces acting on the first two particles (p_0, p_1) connected with spring s_0 rest length l_0, spring stiffness k and damping coefficient k_d, (f_0, f_1) with regards to will be:

$$f_0 = f^s(p_0, p_1) = k_s((p_1 - p_0)/(|p_1 - p_0|)) \tag{2.1}$$

$$f_1 = f^s(p_1, p_0) = -f(x_i, x_j) = -f_0 \tag{2.2}$$

Here, f^s spring force between particles i, j and k_s is the spring stiffness. As Hooke's law states, k_s is a constant factor, specific to that spring.

As the total momentum is be conserved ($f_i + f_j = 0$), its forces are proportional to the relative elongation:

$$f_i = f^d(x_i, v_i, x_j, v_j) = k_d(v_j - v_i)\frac{(x_j - x_i)}{|(x_j - x_i)|} \tag{2.3}$$

2.3 Simulation

Newton's second law of motion states:

$$f = ma \tag{2.4}$$

In this equation

$$a = \ddot{x} \tag{2.5}$$

is the rate of change in velocity, or second derivative of the position with respect to time. With this formula it is possible to compute the forces acting on a particle. Separating the second order differential equation results in:

$$\dot{v} = f(x, v)/m \tag{2.6}$$

$$\dot{x} = v \tag{2.7}$$

Where $\dot{x}$ is the first derivative of position which equals to v and $\dot{v}$ is the second derivative which equals to a.

There are analytical solutions of these equations:

$$v(t) = v_0 + \int_{t_0}^{t} f(t)/m \quad dt \tag{2.8}$$

$$x(t) = x_0 + \int_{t_0}^{t} v(t)/m \quad dt \tag{2.9}$$

These integrals sum the infinitesimal changes to time t from starting conditions:

$$v(t_0) = v_0 \tag{2.10}$$

and

$$x(t_0) = x_0. \tag{2.11}$$

Here, we're simulating (or time-integrating) $x(t)$ and $v(t)$ from starting time t_0. There are various ways to solve these equations by numerically approximating derivatives.

2.4 Explicit Euler Method

Where Δt is the time interval between two successive updates, an approximation for ordinary differential equations can be written as:

$$\dot{v} = \frac{v^{t+1} - v^t}{\Delta t} + O(\Delta t^2) \tag{2.12}$$

$$\dot{x} = \frac{x^{t+1} - x^t}{\Delta t} + O(\Delta t^2) \tag{2.13}$$

Substituting with (2.6) and (2.7) results in:

$$v^{t+1} = v^t + \Delta t \quad f(x^t, v^t)/m \tag{2.14}$$

$$x^{t+1} = x^t + \Delta t v^t \tag{2.15}$$

From here, change in velocity and position in Δt time frame is updated. This is called explicit Euler integration or Euler forward method as we're using the general formula:

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{2.16}$$

to advance the solution from x_n to x_{n+1}. Solution is being incremented through the interval h (which is time step t in this case). As the information used from beginning of the interval the step's error is $O(t^2)$.

2.5 Runge Kutta

Euler method is only moderately stable as it jumps through future, assuming the force is constant in the entire step. For large time steps this results in instability as particles can gain energy and overshoot their target. Runge-Kutta method (sometimes called RK2 or RK4, depending on the order) on the other hand, reduces these problems by sampling forces (to cancel out lower order error terms) more than one time in a time step.

Second order Runge-Kutta results in:

$$a_1 = hf(x_n, y_n) \tag{2.17}$$

$$a_2 = hf(x_n + \frac{1}{2}h, \quad y_n + \frac{1}{2}a_1) \tag{2.18}$$

$$y_{n+1} = y_n + a_2 + O(h^3) \tag{2.19}$$

From this point, fourth order Runge-Kutta can be calculated:

$$a_3 = hf(x_n + \frac{1}{2}h, \quad y_n + \frac{1}{2}a_2) \tag{2.20}$$

$$a_4 = hf(x_n + \frac{1}{2}h, \quad y_n + a_3) \tag{2.21}$$

$$y_{n+1} = y_n + \frac{1}{2}a_1 + y_n + \frac{1}{3}a_2 + \frac{1}{3}a_3 + \frac{1}{6}a_4 + O(h^5) \tag{2.22}$$

2.6 Verlet

Even though having a precise integration method is desired, its significance may be offset by performance in a real-time entertainment environment, such as games. One of the high performance and stable integration schemes, used in simulating molecular dynamics, is the Verlet integration [4]. It was first used in the game Hitman: Codename 47 (2000).

Being a "velocityless" method, Verlet stores only current position x and previous position $x*$. Keeping time step fixed, we can calculate the new position x'. With these two simple update rules.

$$x' = 2x - x * + a\Delta t^2 \tag{2.23}$$

$$x* = x \tag{2.24}$$

Also, velocity can easily be computed by:

$$v(t) = \frac{x(t) - x(t - \Delta t)}{\Delta t} \tag{2.25}$$

However, velocity here, is only first order accurate [5]. Since any error made at any given steps is expected to fade, Verlet algorithm is considered stable. Other finite difference algorithms don't guarantee this behavior [6]. These properties make Verlet an appropriate choice for real time hair simulation. This thesis is also based on Verlet algorithm, resuming Han and Harada's work [7].

2.7 GPU Computing

Hardware performance have increased tremendously in recent years with smaller transistors and multi-core processors yielding considerably more performance. Today, a common CPU includes 4-8 cores. GPUs however, include hundreds to thousands of cores. (See Figure: 2.1) Even though GPU cores are lightweight compared to their counterparts, they still offer more floating point operations per second compared to their CPU counterparts. In the current market, a 4.6 teraflop NVIDIA GTX 980 GPU including 4GB GDDR5 memory with 2048 CUDA cores including 1200 MHz clock rate will cost around $ 600 as of mid 2015 [8]. For more than $ 1000 Intel Haswell E Core i7 5960X offers theoretical peak of 384 gigaflops [9]. Definitely, CPUs's latency oriented nature differ from GPUs in this regard. They don't need to send and receive data from slow buses and as a result they're much faster for sequential and dependent calculations. On the other hand, throughput oriented processors process bulk data by processing in parallel at the expense of increased latency, but this latency can be justified by order of magnitude speed-ups in highly parallel workloads.

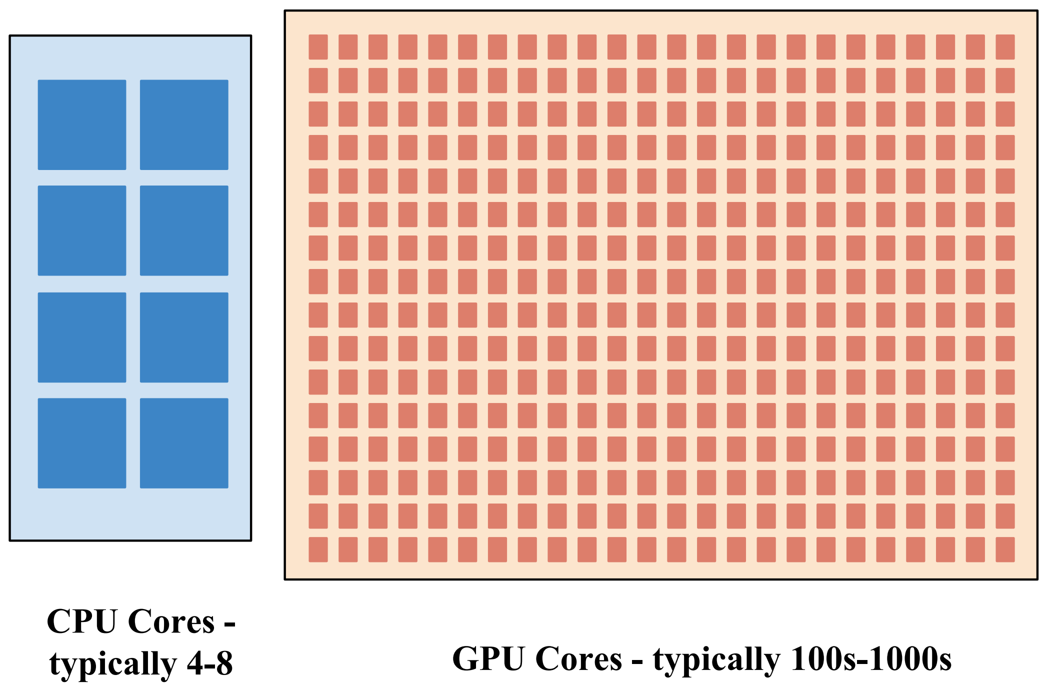


Figure 2.1: CPU GPU Comparison

GPGPU is General Purpose programming on Graphics Processing Units. In the late 90s, GPUs were specially designed for games and computer graphics. They were difficult to program and reason with. Today's GPUs are general-purpose parallel processors with support for accessible programming interfaces and standard languages such as C/C++ or C derived CUDA (NVIDIA), OpenCL (Khronos Group) or shader languages. Today not only games, but a wide range of applications can benefit from the throughput oriented programming. Developers who use parallelization with GPUs can achieve speedups of orders of magnitude vs. CPU implementations.

2.8 Compute Shader

The compute shader stage is a data-parallel step which runs on the graphics hardware. Compute shaders have different processing model than pixel or vertex shaders as they have cross thread data sharing and unordered access I/O operations. They can also be used with more general data structures such as irregular arrays, trees and structs [10]. In the context of games, complex effects such as fire and smoke simulations or soft body simulations such as cloth, hair and fur can be achieved easily without changing the render context to another framework such as OpenCL or CUDA.

2.9 DirectCompute

Microsoft DirectCompute is a compute shader counterpart of OpenGL compute shader. It supports general-purpose computing on GPUs, using same set of registers as the other programmable stages of Microsoft's DirectX10 and DirectX11 APIs. DirectCompute has a register based memory. Driver compiler selects and allocates registers automatically [11]. DirectCompute guarantees fairly consistent results across different hardware. DirectCompute also shares many concepts, idioms, algorithms and optimizations with the NVIDIA CUDA, Khronos OpenCL and Microsoft C++ AMP architectures.

In this study's context, DirectCompute API can also be utilized by soft body physics such as hair or cloth simulation. In mass spring models, bodies consist of masses and springs. They are updated by forces acting on them in parallel to speed up the computation.

2.10 Programming Compute Shaders

Decomposing parallel work is one of the main pillars of GPU computing. DirectCompute dispatches groups of threads (hundreds of thousands) to solve problems this way [12]. Assigning one thread to a part of the big computation will significantly speed up the process.

Compute Shader stage follows the same general principles as other programmable shader stages. However, as it is not tied to a particular shader stage, it doesn't need to pass or receive data from previous or next stages [13]. For this reason Compute Shaders are reasonably self-contained and can be used to do computing on its own.

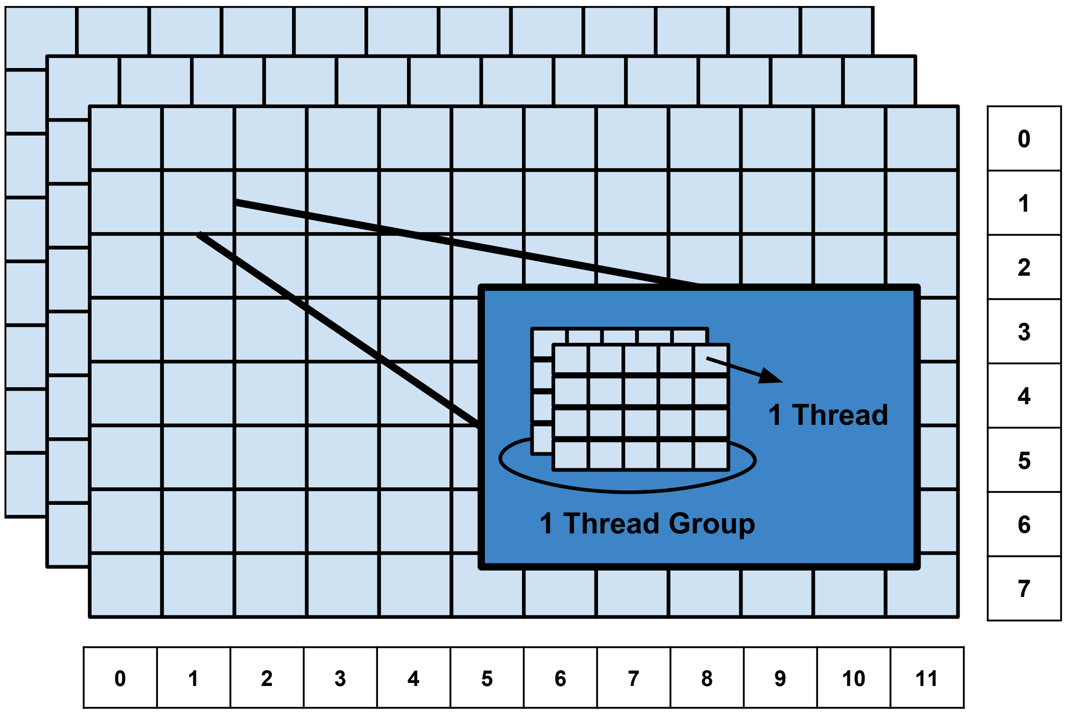


Figure 2.2: GPU Thread Groups

```
// CPU Side
//...
Dispatch(12, 8, 3);

//GPU Side
[numthreads(5, 4, 2)]
void MainCS(uint3 Gid : SV_GroupID, uint GI : SV_GroupIndex)
{
  //...
}
```

Figure 2.3: DirectCompute Kernel Call

A GPU is particularly efficient at processing parallel algorithms. It needs, however, to have a methodology that allows mapping different algorithms to run on many threads. Like other parallel programming specifications, DirectCompute uses a kernel based system. This is a considerably simple way to work with thousands of threads. Each thread will be tasked with executing one individual invocation of the kernel on a particular data element. In DirecCompute a shader program is executed via a Dispatch call. A dispatch call will consist of X, Y, Z elements, corresponding to thread group dimensions.

Figure 2.2 illustrates the threading model of DirectCompute API. Here, 12 x 8 x 3 = 288 thread groups are dispatched, each of them containing 5 x 4 x 2 = 40 threads. In this example a total of 11520 threads in parallel are used. After dispatch call, GPU executes every thread group with a "numthreads" attribute. This defines the number of threads in a thread group. Call to the GPU is instantiated by:

To operate on the data kernel must know the index of the relevant thread. For this reason DirectX specifies a number of built-in input variables as follows:

SV_GroupThreadID: XYZ indices of the individual thread in a thread group.
SV_GroupID: XYZ indices of a dispatched thread group.
SV_DispatchThreadID: XYZ indices of the thread in a thread group, combined with other threads and thread groups.
SV_GroupIndex: Individual index of a thread in a thread group.

2.11 Hair Model

Human hair consist of very thin, semi-transparent inextensible strands which both interact with other hairs and outside objects. In the last century solids and fluids have been modeled with classic equations, but there is no widely accepted model for hair as its motion is considerably complex [14]. There are many different topics to address before discussing a realistic simulation of hair. In this part mechanical structure and background work will be explained.

2.12 Hair Structure

To achieve a realistic simulation of hair, requires a detailed understanding of hair structure. A human hair fiber is a 0.1 mm structure made up mainly from keratin (See Figure: 2.4 [15]). The Active (live) part of the hair lies under the skin and called follicle. Shape of the shaft varies by race. Africans usually have a flat hair cross section whereas Caucasians have ovoid or rounded hair cross section [2]. Hair simulations are almost solely interested with the dead part of the hair called the hair shaft.

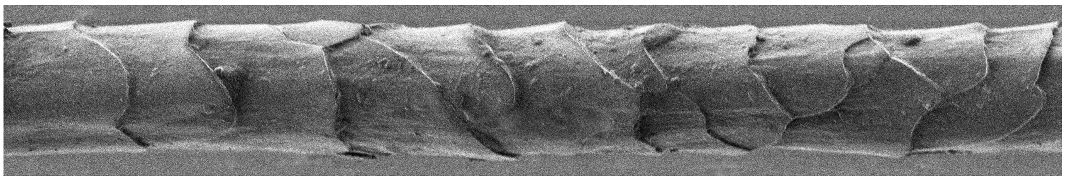


Figure 2.4: Human Hair

2.13 Hair Simulation

Keratin is the key structural protein material making up the outer layer of the skin. Even though keratin is considerably stiff, hair's cross section is remarkably small, thus it can easily be bent and twisted [14]. With this property on hand it is possible to virtually represent a hair strand as chained particles. In this representation, particles will have point masses and they will be connected by springs. This is called a mass-spring system. (See Figure: 2.5) Even though this representation have problems such as unwanted elongation and extreme twisting, it is a very easy model to understand and can be implemented without much effort. To address some of these problems, extra stretch springs and angular springs between particles (See Figure: 2.6) can be used.

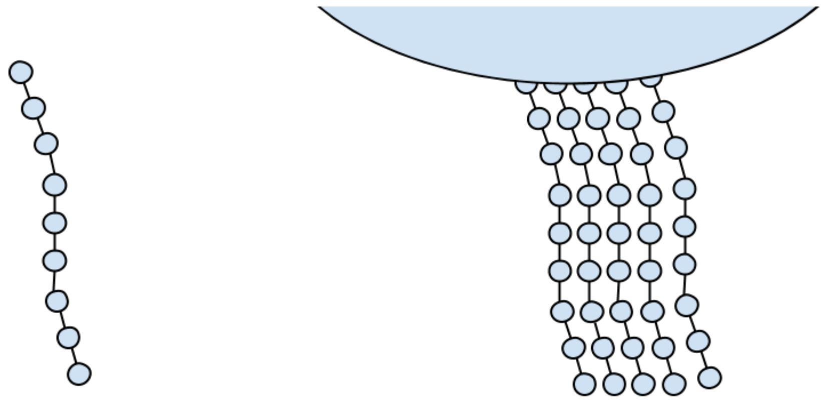


Figure 2.5: Mass Spring System

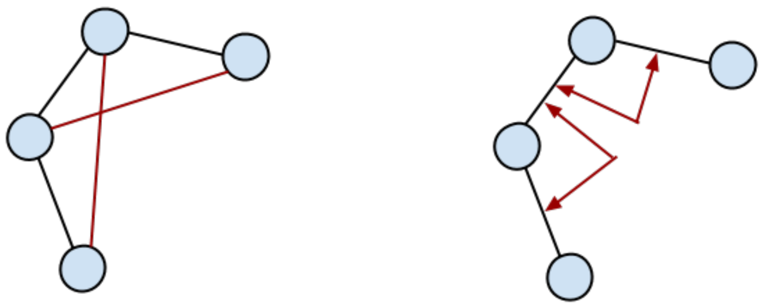


Figure 2.6: Angular and Stretch Springs

2.14 TressFX

In 2012, Dongsoo Han and Takahiro Harada published a paper called Real-time Hair Simulation with Efficient Hair Style Preservation. In this paper, a real-time hair simulation method is proposed. There were other real-time hair methods presented before such as position based dynamics (PBD) based simulations, ([16], [17]) but they were using a large portion of the GPU and thus are not suitable to be used in a game environment. TressFX, on the other hand, uses only a fraction of the available GPU power. For these properties it was later used in Tomb Raider game released in 2013.

TressFX has 2 parts which are hair simulation and hair rendering. As this study is about physical simulation, only physical simulation part of the framework is discussed.

2.14.1 Hair Constraints

TressFX works by utilizing constraints on hair strands. It uses length constraints to achieve inextensibility and also global constraints for maintaining overall hair shape and local constraints for hair bending and twisting effects.

To keep the hair in its starting shape and reduce unwanted entanglements, initial hair positions are set as goal positions. In every succeeding frame, hair particles are integrated towards these goal positions and consequently overall hair shape is maintained throughout the simulation, but this decreases the bending and twisting of the hair. For this reason, global constraints are integrated in moderation (usually 2 times).

Another constraint treatment used in TressFX is local constraints. In this step, hair strands move and rotate in global space to maintain goal rotations. Figure 2.7 shows a hair strand particles' positions initial (light blue), goal (red) and destination (green) positions a frame. This allows preserving of the hair style whilst sustaining believable bending and twisting effects. After shape constraints, "length inextensibility" step is applied.

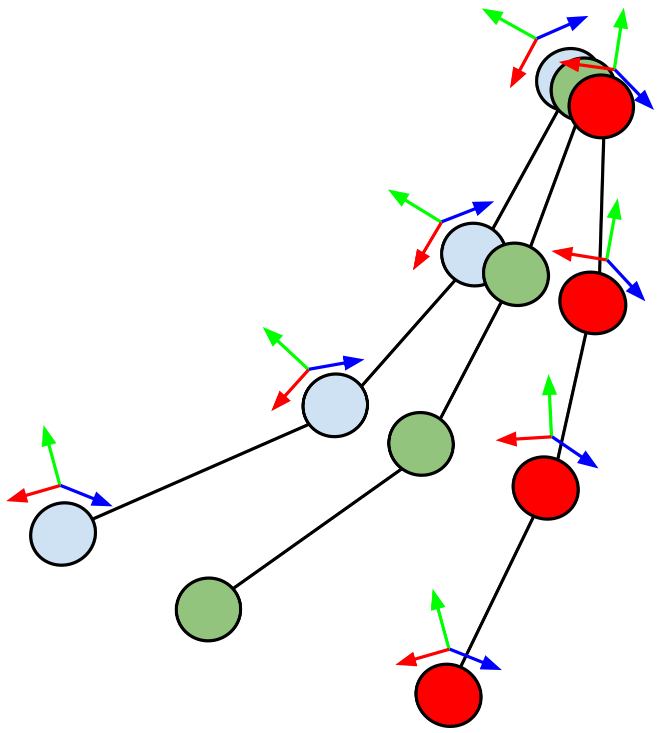


Figure 2.7: Local Constraints

CHAPTER 3

PREVIOUS WORK

There are various methods that have been proposed for rendering and simulating hair since late 1980s. Also in recent years with the advancement of the parallel architectures in both CPUs and GPUs, methods exploiting these parallel architectures have increased. This section discusses and reviews some of the most important hair simulation papers. In addition to hair simulation, some cloth and other soft-body simulation papers are also included since cloth and hair can easily be represented using same structures such as springs.

3.1 Simulation Methods

Rendering fur and furry surfaces was a largely unsolved problem even at the late 1980s. Kajiya and Kay's work is largely on rendering[18]. They model each hair as infinitesimally thin cylinders and render hair using texels and anisotropic lighting models. Banks generalized this approach to n dimensions[19]. Early physical hair models largely ignored torsion and inter-hair interaction forces, focused on non-aggregate methods using spring models for modeling each strandRosenblum [20]. Anjyo et al. focused on length preserving models [21]. Since strands don't have any interactions with other strands, they are easy to parallelize.

Plante et al. used layered wisp models to address visual defects caused by the lack of hair-hair interactions [22]. Baraff and Witkin proposed an implicit integration method with stiff springs [23]. Although this work was for cloth simulation it was later used for hair simulation by Ward and Lin and Choe et al. [24], [25]. Using stiff springs however is not enough, a correction method is needed. Teschner used this method with collisions in cloth simulation [26]. Lokovic and Veach proposed a technique called Deep Shadow Maps for approximating self shadowing of many thin strands [27]. In this work, instead of storing a single depth value, they created shadow maps which store a representation of the fractional visibility through a pixel at all possible depths.

Selle et al. proposed a new method to simulate the full geometry [28]. As this is one of the first works simulating full head of hair (about 100.000 strands) complex hair-hair and hair-body collisions requires considerable computing resources. Because of this, they failed to simulate 100.000 hairs, as self-collision costs are overwhelmingly expensive even for today's computers.

Hair can also be modeled as a continuum, to maintain hair volume. Hadap and Magnenat-Thalmann used a fluid model with chains integrated into volume [29]. This, being one of the first papers, mixing point chain hair model with fluid forces, increases fidelity, particularly in hair-hair and hair-air interactions. Bondo et al. also used a fluid continuum, but without connected particles [30]. This contrasts with Petrovic et al.'s work [31]. They used particle spring model to represent individual strands with volumetric methods. Their approach is based on that hair interacts as a bulk material. Utilizing ideas from Ward and Lin, by considering a small subset of hairs as keyhairs, reduces the vast number of hairs to deal with[24]. This model uses a Cartesian voxel grid for both rendering and simulation. Also, the grid based approach helps with simulation by enabling density and velocity to be averaged.

With the advancement in CPU and GPU hardware, simulated number of strands have increased considerably. In addition to hardware advancement some interesting alternatives emerged for simulating hair. Muller et al. introduced position based dynamics (PBD) [32]. PBD solves constraint dynamics problem in an iterative manner. This also enabled to manipulate objects directly during the simulation, with further improvement, Muller et al. addressed convergence issues [33]. As this technique is simple and stable, it is well suited for real-time applications like games. Tariq et al. demonstrated this technique for an NVIDIA demo with 166 simulated strands, 10220 rendered strands and 1.6 million triangles by utilizing vertex buffer and vertex shader [17]. Since in mass-spring systems, stretching is an undesired side effect, there are various methods to guarantee no-stretching. Extending the method, "follow the leader", Muller et al. used PBD technique efficiently visualize human hair and furry animals with single iteration per frame for simulation in real-time [16]. As this method is not as accurate as physically based techniques, it adds some artificial damping to hairs.

Daviet et al. argued that, friction effect plays a major role in hair dynamics like "stick-slip instabilities" [34]. They presented an iterative solver for Coulomb friction of tightly packed fibers like hair.

Real hair exhibits many fine details that is hard to capture in a simulation mostly because of the expensive computations required by complex collisions. Using incompressible fluid methods can overcome this problem efficiently. McAdams et al. created a technique which combines Lagrangian and Eulerian hair simulation techniques using fluid-like volumetric collision methods [35]. In this technique hair behaves similar to fluids, thus momentum and mass are conserved.

Hair may be counted as a super helix structure and the computing dynamics of super helical thin elastic structures efficiently would result in believable hair models. Bertails introduced a novel recursive scheme to simulate piecewise helical rods in linear time instead of quadratic [36].

Sueda et al. introduced a new hybrid framework combining Lagrangian and Eulerian approaches to simulations of thin bodies, elastic rods without fluid like forces. Instead they use a reduced node approach for reducing number of degrees of freedom allowed by the constraints [37].

Iben et al. proposed a method for simulating curly hair to be used in Disney/Pixar animation film Brave. To maintain curls helical shape they used extensible elastic rods in addition to bending and core springs [38]. Also their implementation includes a pruning mechanism to reduce hair-hair contacts and thus increasing parallelism. Even though many optimization techniques used in this work, with the sheer number of hairs on characters, simulating one frame of the hair took over 13 seconds on average on a 12 core Intel Xeon machine. But, since producing a film doesn't require real-time simulation, these costs are usually in the limits.

In 2012, Dongsoo Han and Takihiro Harada published a paper to solve the real-time hair simulation problem in games which this work is largely based on [7].

Guan et al. used a data-driven model for learning hair models for real-time visualizations [39]. They also introduced an efficient hair-body collision technique by using a form of iterative least squares in a reduced space.

Chai et al. used a data-driven technique with data-driven models simulating only a small set of hairs called guide hairs [40]. They argued that, previous works using interpolation of hair dynamics attenuates detailed motions of hair. With precomputed simulation data, optimizing interpolation weights for each particle by solving linearly constrained least square problem could overcome this shortcoming.

Liu et al. proposed a completely new method for integration of Hookean mass-spring systems which uses a solver based on block coordinate descent [41]. Using spring directions as auxiliary variables, they cast time-stepping as an integration problem. Although it converges slower than Newton's method, obtaining visually acceptable results are much faster. Being an iterative solver in its nature, this technique can not be parallelized efficiently and thus uses only a single core of a CPU.

Table 3.1 shows a matrix of previous approaches.

Table 3.1: Related Work Table

Method / Hardware	CPU	GPU
Verlet	-	[7]
Semi-implicit Euler	[36], [38], [23], [37]	-
Point-Based	-	[32], [17], [**?**]
Data-Driven	[34], [40], [39]	-
Volumetric	[29], [30], [31]	-
Mathematical	[41]	-

Starting from late 1980s, hair simulation has come a long way. Today, it is possible to simulate thousands of hair strands in an interactive game environment with the rapid succession of faster hardware with every generation. Furthermore, algorithms also improved remarkably to enable this visual affinity. In these works, there is a recognizable pattern. Hair simulations are becoming more parallel by using multiple CPU cores and GPUs, also more data driven by using statistical learning algorithms. In coming years these two techniques may merge to create lower training times for learning models and better runtime performance for multi-core models.

3.2 Hair in Computer Games

Due to the chaotic nature of realistic human hair, it was a hard task to simulate hair realistically in games even a few years ago. Given the history of the video games, only a small percentage includes characters with hair. In 1997 Tomb Raider 2 was released. It was one of the first games to include interactive hair braids. As it included braids with just a few moving parts in the game, it allowed developers to increase the fidelity and immersion without much performance penalty (See Figure: 3.1).

Figure 3.1: Tomb Raider 2

Including physically based hair simulation in games has increased in recent years. Apart from the TressFX, latest examples include Call of Duty: Ghosts (2014) and The Witcher 3:Wild Hunt (2015). Both of these games are using NVIDIA's Hairworks technology for simulating real-time fur. (See Figure: 3.2) HairWorks is open to developers to download and experiment with, through plugins on different versions of Autodesk 3DS Max and Maya but implementation details are not exactly known. Though it is possible to speculate that it is using Point Based Simulation as NVIDIA affiliated authors published papers explaining and implementing this technique. An interesting note in Witcher 3 game is that in the game character′s beard grows with time[42]. As facial hair also delineates the differences between characters this enhancement would add increased immersion to the game (See Figure: 3.3).

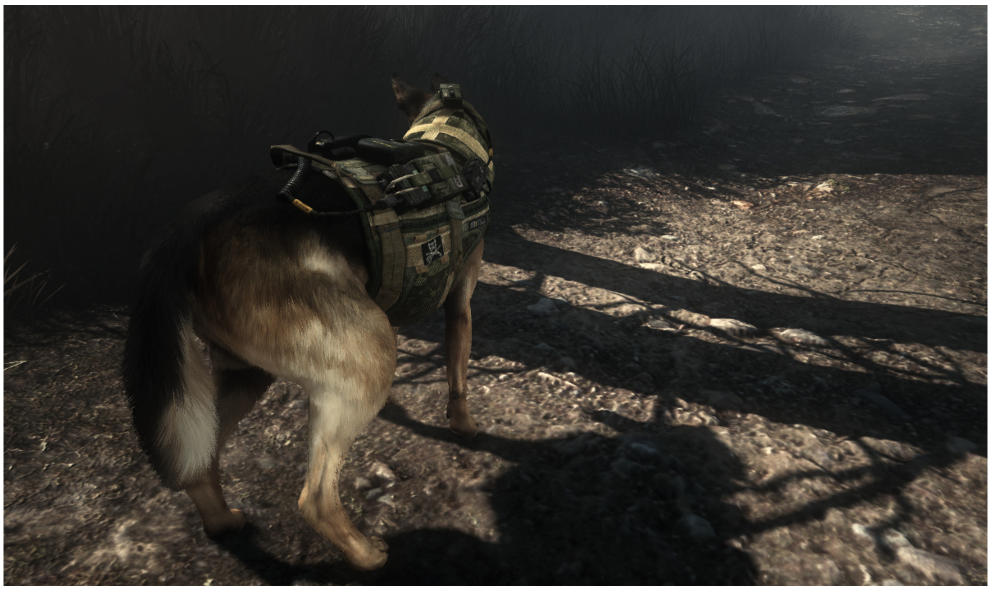


Figure 3.2: Call of Duty: Ghosts

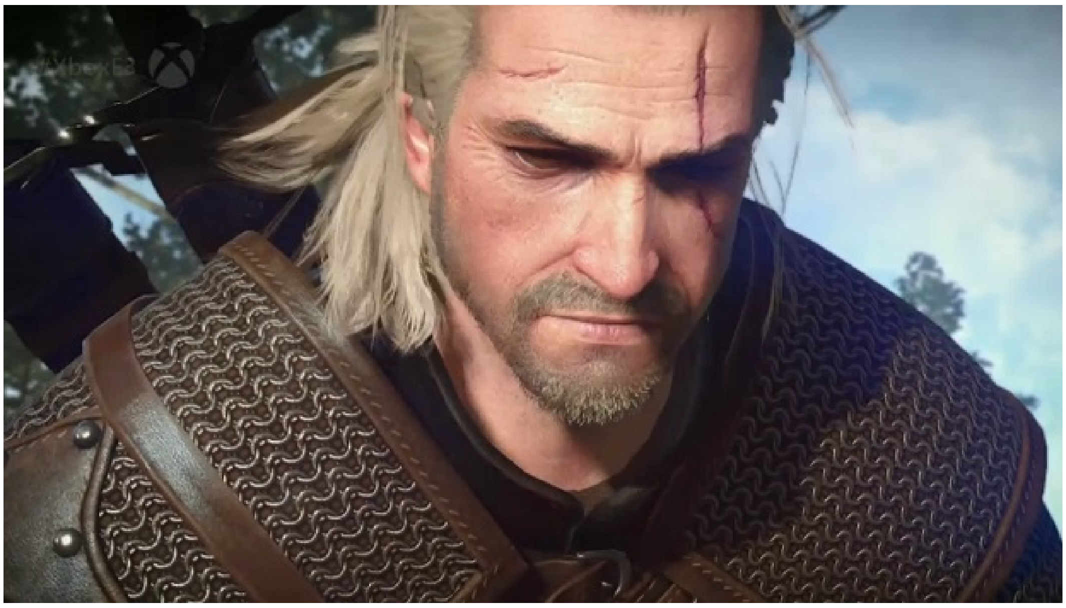


Figure 3.3: The Witcher 3: Wild Hunt

CHAPTER 4

PROPOSED METHOD

As described in previous chapters, this work aims to introduce hair-hair interaction to the existing TressFX framework with the same spirit as in Petrovic et al. [31]. Han and Harada's [7] work, form a basis for TressFX hair simulation framework. In their work, compute capabilities of modern GPUs are utilized by exploiting the parallel nature of hair. This parallel nature however, is prone to inter-hair interactions. Since a GPU thread group is assigned to a strand and a thread is assigned to a vertex, having collisions between individual hairs is non trivial task because of the GPU's inter-communication issues. With thousands of hair strands, using $O(n^2)$ complexity algorithm is not feasible, especially in a fast local memory constrained architecture such as a GPU. In this study, an improvement is proposed over the original TressFX method by obtaining better visuals by reason of inter-hair interaction. As will be discussed in following chapters, this improvement is also evaluated by a user study.

4.1 Uniform Grid

Petrovic et al. by taking ideas from Hadap et al., proposed a volumetric method for efficiently solving hair-hair collision problem. A hair strand is surrounded by other hairs and air and both of these damp hair movements [29]. Thus, it is safe to assume that adjacent hair strands move with same or similar velocities and this allows to model hair as a continuum. With this assumption, hair model becomes similar to fluid models hence reducing the chaotic complexity of hair to a more manageable scale. One of the easy and effective ways to represent this phenomena in virtual environment is to use 3D uniform grids as a helper data structure (See Figure: 4.1). First proposed by Franklin et al., [43] uniform voxel grids provide a fast solution to collision problem. One of the advantages of using a uniform grid is that uniform grids are well suited for multi-core architectures such as GPUs. The parallel nature of many-core systems allow mapping a group of threads to objects in a voxel grid and thus yielding better performance. Liu et al. also used voxel grids on GPUs to improve liquid simulation performance [44].

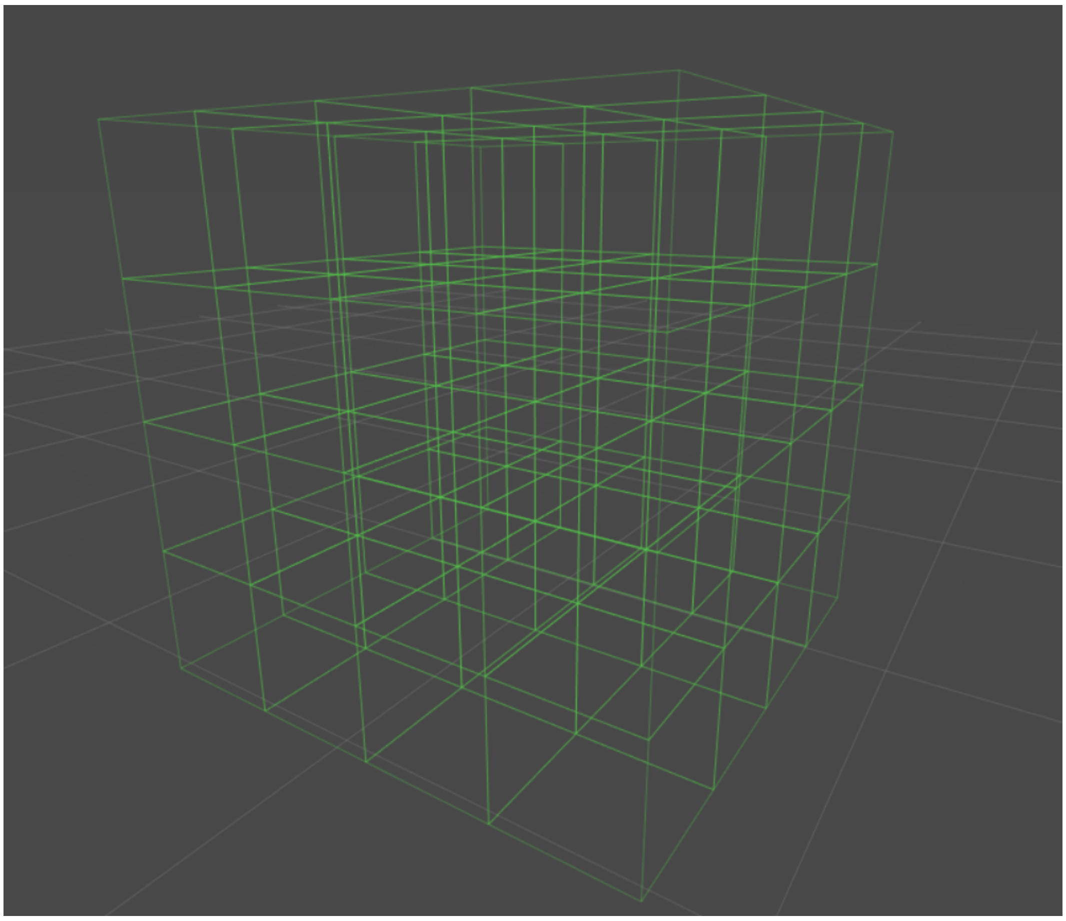


Figure 4.1: A 4x4x4 Uniform Voxel Grid

4.2 Uniform Grid with DirectCompute

To equalize the velocities, each hair vertex must be assigned to a voxel of the uniform grid. In order to achieve this, first a call is made to every particle to find grid positions. In this call, an integer is assigned to hair vertices from 0 to $NumberOfVoxelsPerEdge^3 = TotalNumberOfVoxels$ (See Figure: 4.1).

Figure: 4.1: World To Voxel

```
int World2Voxel(float4 vertex)
{
    float3 pos = vertex.xyz - VoxelGridStartPos;
    int nv = NumVoxelsPerEdge;
    int voxelNo =
      int(pos.y / EdgeLengthOfAVoxel) * nv * nv +
      int(pos.z / EdgeLengthOfAVoxel) * nv +
      int(pos.x / EdgeLengthOfAVoxel);

    return voxelNo;
}
```

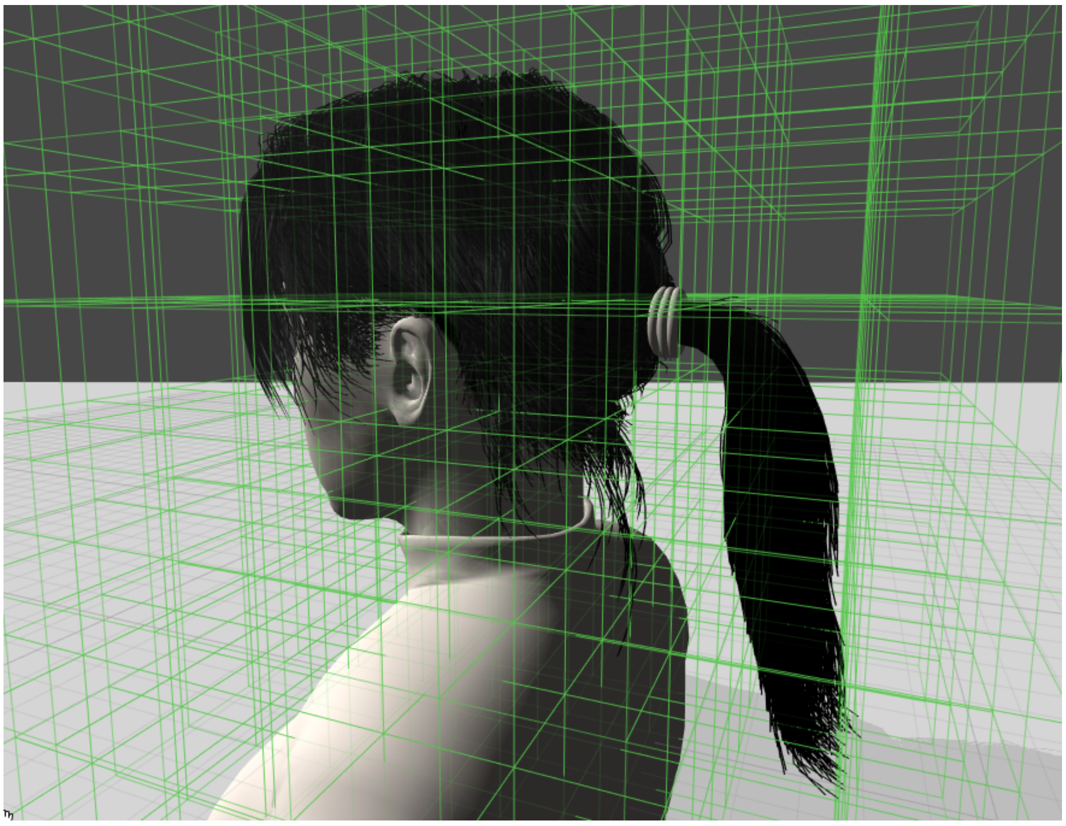


Figure 4.2: Hair in Uniform Grid

This call is made by an "embarrassingly parallel" workload. Each vertex of every strand is calculated fully in parallel. Vertex position is localized by subtracting start position of voxel grid and normalized by dividing edge length of a voxel. Positions of the vertices are read from a RWStructuredBuffer. A RWStructuredBuffer is a data buffer residing in GPU, which enables both read and write operations which differs from StructuredBuffers in this regard. With this data structure, it is markedly easy to assign hair vertices to voxels without going back to CPU [45].

4.3 Velocity Diffusion

Due to representation of hair as a volume in this study, density plays an important role in hair movement. After the construction of the voxel grid, influence of a hair vertex in each voxel is found via a 3D tent function. Particle influences linearly increases toward the center and at very center influence is 1 and becomes 0 when vertex coordinates are at the edges of the voxel (4.1).

$$D_{xyz} = \sum_i (1 - |P_x^i - x|)(1 - |P_y^i - y|)(1 - |P_z^i - z|) \tag{4.1}$$

Here, D is density and P_x^i is the x position of the ith particle.

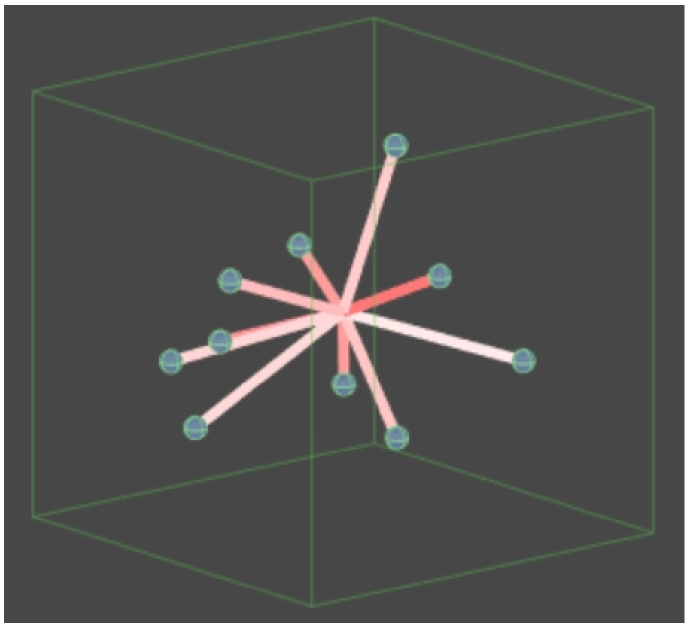


Figure 4.3: Tent Function

Figure 4.3 shows a representation of influences in a voxel. Redder lines show more influence whereas whiter lines shows less influence.

After finding the average density of the voxel, average velocity of the voxel should be found. This process is very similar to finding average density. Using the same 3D tent function, this time influence of every particle is multiplied with that particles' velocities v^i and all of these values are summed and divided to the density (D_{xyz}) of the voxel (See Figure: 4.2).

$$V_{xyz} = \frac{\sum_i (1 - |P^i_x - x|)(1 - |P^i_y - y|)(1 - |P^i_z - z|)v^i}{D_{xyz}} \quad (4.2)$$

After finding the average velocity, Verlet integration continues normally. But if a particle in a voxel is moving faster than the average velocity of that voxel, its velocity is diffused by introducing a drag term. (See Figure: 4.4) With this treatment, degenerate fast moving hair particles are slowed down to prevent unwanted collisions. This, also allows hair to move like a contiguous structure and outputs more visually pleasing results.

4.4 Diffusion in Compute Shader

In the shader, a global buffer (linear array) is used to hold these indices. At every frame, every hair particles voxel address is calculated and added to this buffer in parallel (See Figure: 4.1). This is essentially a scatter operation (See Figure: 4.5).

In DirectCompute, however, buffers have fixed size. In this implementation

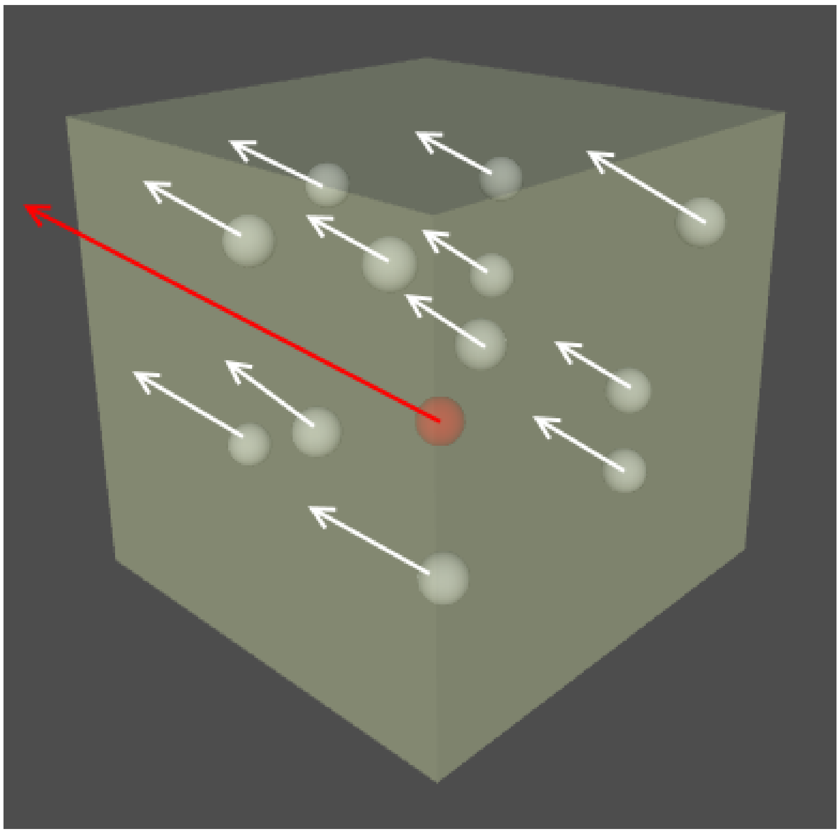


Figure 4.4: Fast Moving Particle

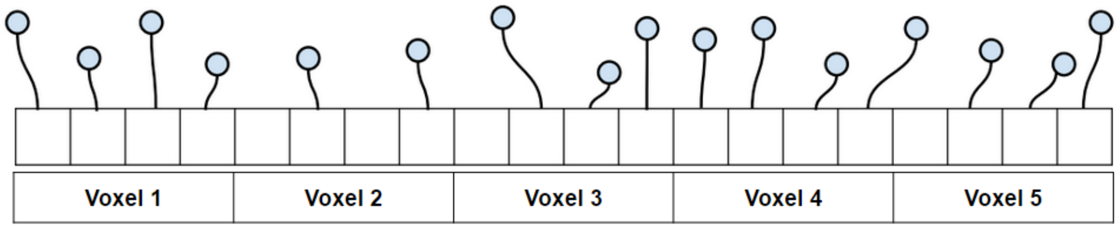


Figure 4.5: Set Particles to Voxels

512 (8x8x8) voxels are used with a maximum capacity of 1024 in each voxel. This pre-allocation results in 2MB of extra memory. Sometimes there may be more than 1024 particles in a voxel. To cope with this, extra voxels may be used or capacity of a voxel may be increased but it will also increase memory requirements. In this work randomness is used to cope with capacity problem. Particles will be added to voxel buffer randomly without checking if that place is used by another hair particle via Figure 4.6.

This function outputs a value between 0 and 1 when seeded by two floating point numbers. Here, floating parts of the vertex x and y coordinates are used. As floating parts are changing rapidly, functions output is random.

Figure 4.7 shows hair vertices' states between two successive frames. White particles represent particles which are not added to the voxel buffer and red particles represent the ones added to the voxel buffer. In the first frame (left

```
static const float2 r = float2(
  // e ^ pi (Gelfond's constant)
    23.14069263277,
  // 2 ^ sqrt(2) (Gelfond-Schneider constant)
     2.66514414269);

float Random(float2 p)
{
  return frac(cos(fmod(123456789., 1e-7 + 256. * dot(p, r))));
}
```

Figure 4.6: Random Voxel Index

cube) red particles' velocities are calculated and diffused. White particles are not added to this calculation due to space constraints, but in second frame (right cube), because of the randomness, some of white particles are turned to red; hence, added to calculations and their velocities are diffused. In the course of a few hundred frames eventually all particles will almost surely be added to voxel buffer and their velocities will be diffused if their speed is more than 2 times the average voxel velocity in some direction. For example, if a particle's Z coordinate velocity is more than two times the average voxel velocity, Z component of its velocity vector is reduced by ten percent. Even for pathologically fast moving particles, (five to ten times more than the average velocity) this effect adds up in consequent frames and reduces that particle's velocity to near average voxel velocity.

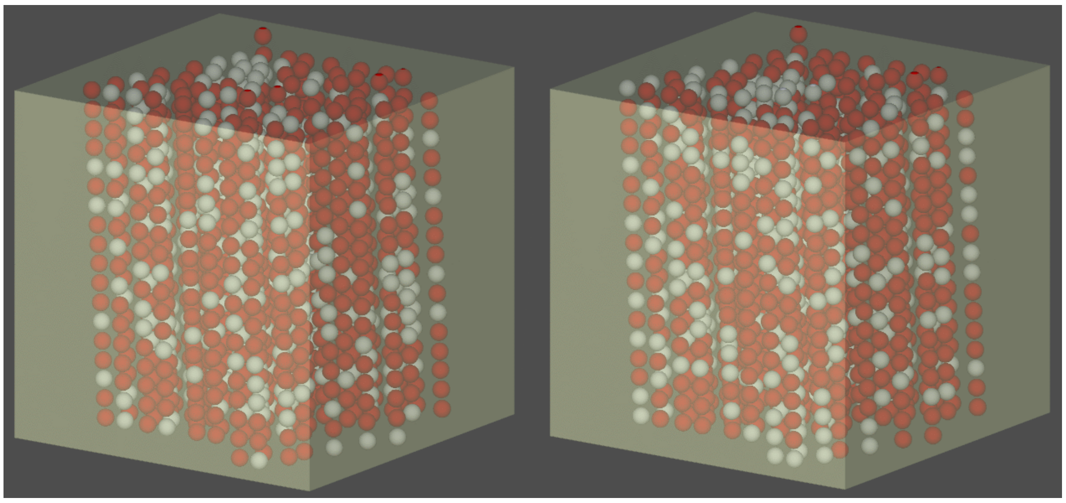


Figure 4.7: Random Particles Between Successive Frames

After this treatment, hair in simulation looks more realistic and inter hair collisions are much less pronounced. In Figure 4.8 an image of interlocked hair is shown.

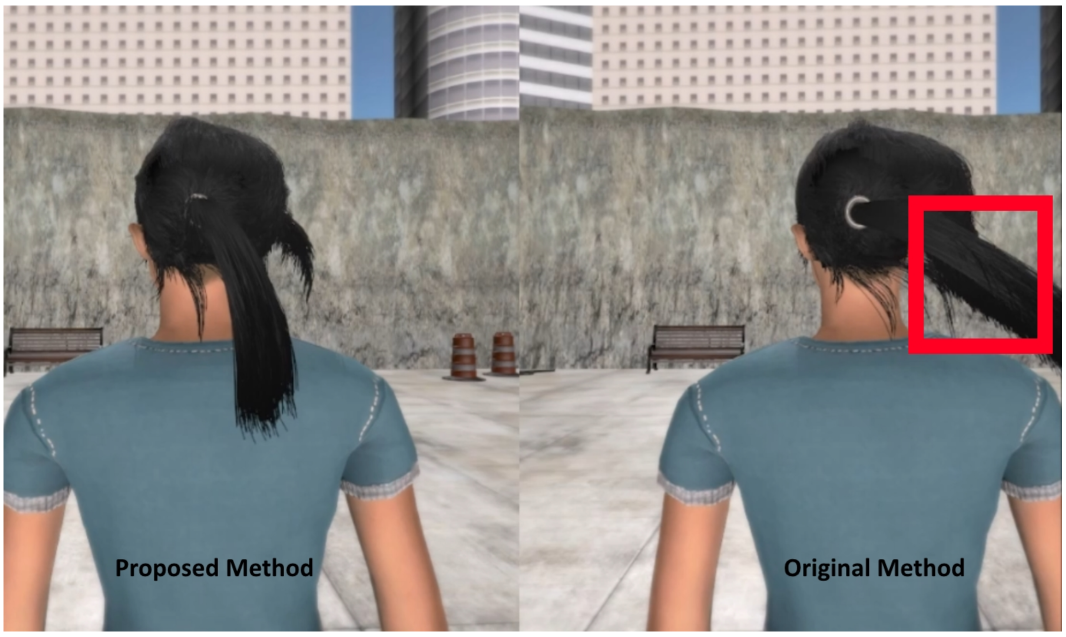


Figure 4.8: Hair Comparison

CHAPTER 5

RESULTS AND DISCUSSION

After adding the new step, some performance loss is introduced to the simulation. In this section this loss is justified via a user study and performance characteristics of the original and proposed method on various architectures are discussed.

5.1 Performance On Different Architectures

Performance of the braid hair simulation have been tested on two major graphics card vendors. In tests one Intel and five NVIDIA based chipsets are used. Every test has been done with compiled release versions in 1024x768 windowed resolution with Unity's "Fantastic" quality setting and ran for 20 seconds five times. Arithmetic mean of the fastest three observations are taken and the following results are obtained.

Table 5.1: Performance Tests

Graphics Card	Original Shader FPS	Proposed Shader FPS
Intel(R) HD Graphics 4000	9.62	9.36
NVIDIA GeForce GT 540M	17.64	17.06
NVIDIA GeForce GT 650M	36.91	30.51
NVIDIA GeForce GTX 760	90.24	74.66
NVIDIA GeForce GTX 660 Ti	110.21	87.02
NVIDIA GeForce GTX 960	152.65	124.36

As shown in Table 5.1, it is possible to see with lower end cards, performance is about the same. This is mainly because of the lower computing unit count. Intel HD Graphics 4000 GPU has 16 execution units and NVIDIA GT540M graphics card has 96 shader cores whereas GTX 660Ti has 1344 cores, GTX 760 has 1152 cores and GTX 960 has 1024 cores with varying clock speeds. In the case of lower end chipsets, performance is stalled at the process of integrating individual hair strands, therefore adding an additional step would not introduce extra cost. However, after a certain point in graphics card core count, a performance gap appears. It is possible to say that in this implementation, core count is important until all hair vertices can be integrated without "core starvation". After this point, additional performance penalties from the new implementation becomes pronounced.

5.2 User Study

In testing this interactive 3D physical simulation, users are presented with two applications for each hair style (braid and long) which are exactly same in appearance and interaction properties, but differ in simulation characteristics. Figure 5.1 shows a screen shot of the application in the administered user study. Also, questions asked to participants can be found in the appendix A.1.

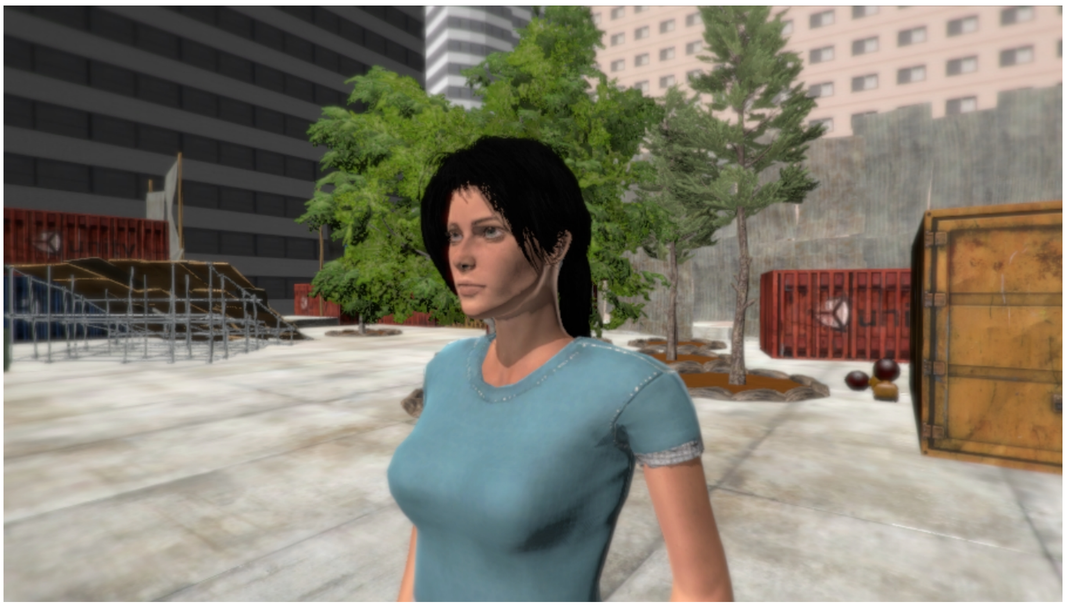
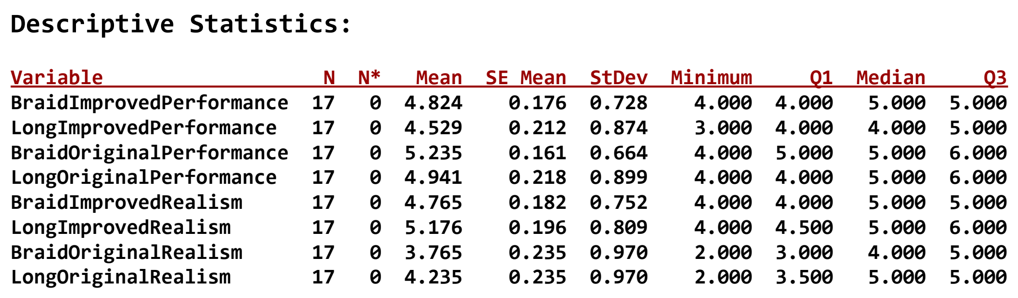


Figure 5.1: Screenshot of User Study

All tests are conducted with NVIDIA 580M graphics card and Intel Core i7 3630QM CPU. Users interacted with two applications in random order and after this, they were presented with a questionnaire which asks them to rate realism and performance of two simulations. A 6-point Likert scale is used to measure differences in hair simulations. With a total of 17 participants, a statistical analysis is conducted to gathered data to test proposed hypotheses. The average age of the participants was 29.4 and ages were ranged from 24 to 37. 4 of the participants were female and 13 were male. 10 of the participants had technical backgrounds related to information technologies and all of them had at least a BSc degree in a field of science. To have a better idea about the given answers and summarize the data descriptive statistics is given in 5.2.

To test hypotheses Wilcoxon signed-rank test is used. This test is appropriate



```
Descriptive Statistics:

Variable                   N  N*   Mean  SE Mean  StDev  Minimum     Q1  Median     Q3
BraidImprovedPerformance  17   0  4.824    0.176  0.728    4.000  4.000   5.000  5.000
LongImprovedPerformance   17   0  4.529    0.212  0.874    3.000  4.000   4.000  5.000
BraidOriginalPerformance  17   0  5.235    0.161  0.664    4.000  5.000   5.000  6.000
LongOriginalPerformance   17   0  4.941    0.218  0.899    4.000  4.000   5.000  6.000
BraidImprovedRealism      17   0  4.765    0.182  0.752    4.000  4.000   5.000  5.000
LongImprovedRealism       17   0  5.176    0.196  0.809    4.000  4.500   5.000  6.000
BraidOriginalRealism      17   0  3.765    0.235  0.970    2.000  3.000   4.000  5.000
LongOriginalRealism       17   0  4.235    0.235  0.970    2.000  3.500   5.000  5.000
```

Figure 5.2: Descriptive Statistics

in this scenario, as 17 users are a relatively small sample size and may not have normality assumption fulfilled.

In Wilcoxon signed-rank test, median values are compared with testing whether two samples' differences originates from a distribution with zero median. For all of the 4 tests, hypotheses are similar and tests if there is a statistically significant difference between two groups.

H_0 : *Median difference between pairs of observations is zero.*
H_1 : *Median difference between pairs of observations is not zero.*

5.2.1 Performance Test of Braid Hair Style

H_0 : *Median performance differences between proposed and original method for braid hair style is zero.*
H_1 : *Median performance differences between proposed and original method for braid hair style is not zero.*

In testing of performance with braid hair style, p value is:

$$p = 0.071 > 0.05 \tag{5.1}$$

Therefore, null hypothesis is failed to be rejected. Users didn't perceived a difference in performance between two applications.

5.2.2 Performance Test of Long Hair Style

H_0 : *Median performance differences between proposed and original method for long hair style is zero.*
H_1 : *Median performance differences between proposed and original method for long hair style is not zero.*

In testing of performance with long hair style, p value is:

$$p = 0.154 > 0.05 \tag{5.2}$$

Therefore, the null hypothesis is failed to be rejected. With the long hair style users, again, didn't perceived a difference in performance between two applications.

5.2.3 Realism Test of Braid Hair Style

H_0 : *Median perceived realism differences between proposed and original method for braid hair style is zero.*
H_1 : *Median perceived realism differences between proposed and original method for braid hair style is not zero.*

Testing for realism, however shows different results. With p value:

$$p = 0.005 < 0.05 \tag{5.3}$$

Null hypothesis is rejected. In consequence, there is a statistically significant difference in realism of the two simulations. As median value of results from proposed simulation is higher than the original simulation values, it can be said that proposed simulation for braid hair style looks more realistic to tested users.

5.2.4 Realism Test of Long Hair Style

H_0 : *Median perceived realism differences between proposed and original method for long hair style is zero.*
H_1 : *Median perceived realism differences between proposed and original method for long hair style is not zero.*

P value for realism test of long hair style is:

$$p = 0.027 < 0.05 \tag{5.4}$$

Therefore, null hypothesis is rejected. In this case, even though median values are same (5.0), the distribution of LongOriginalRealism is skewed towards the left side. Hence we conclude that proposed simulation appears more realistic on tested subjects.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, a GPU based hair simulation method TressFX is presented and an improvement to this technique is proposed. TressFX is currently the state of the art in hair simulation by allowing real-time and realistic hair simulation in games. But it doesn't have hair-hair collision. With this study, a volumetric effect to simulate inter hair-collision is added to the TressFX simulation, utilizing ideas from [31]. A uniform grid is used to equalize hair velocities and therefore creating the image of hair-hair interaction. Alongside the extra computation required to make the collision effect, some performance loss is added to the system. As computer hardware keeps getting better at every generation, especially GPUs, this loss will be pronounced less and less.

With conducted user study, two hairstyles are tested regarding performance and realism of simulation. In both tests, users didn't perceived a difference in performance characteristics for either of the simulations. In realism survey, however, users found proposed hair simulation scheme more realistic.

For future work, different LOD techniques may be introduced into simulation as with increasing distance from the character with hair would greatly reduce the costs whilst possessing same appearance with a smaller number of hairs in simulation. Also in scenes with many characters, some of the same simulation data may be used in different characters to lower overall simulation costs. Furthermore, basic simulation data may be preprocessed and can be added to characters which are distant to the player thus giving the illusion of an actual simulation is in progress. A user study, similar to one, introduced in this work, can be conducted to verify results.

REFERENCES

[1] Jamie Snape. Simulating Hair Dynamics. http://www.cs.unc.edu/~lin/COMP768-F07/LEC/hair.pdf, 2003. Online; accessed 10-May-2015.

[2] B Lindelöf, Bo Forslind, Mari-Anne Hedblad, and U Kaveus. Human hair form. morphology revealed by light and scanning electron microscopy and computer aided three-dimensional reconstruction. *Archives of dermatology*, 124(9):1359–1363, 1988.

[3] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: Class notes. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 88:1–88:90, New York, NY, USA, 2008. ACM.

[4] Thomas Jakobsen. Advanced Character Physics. http://web.archive.org/web/20080410171619/http://www.teknikus.dk/tj/gdc2001.htm, 2001. Online; accessed 10-May-2015.

[5] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: Class notes. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 88:1–88:90, New York, NY, USA, 2008. ACM.

[6] Hannes Johnson. Classical dynamics Lecture Notes. https://notendur.hi.is/hj/EE4-05/Verlet.pdf, 2003. Online; accessed 10-May-2015.

[7] Dongsoo Han and Takahiro Harada. Real-time hair simulation with efficient hair style preservation. In *Workshop on Virtual Reality Interaction and Physical Simulation*, pages 45–51. The Eurographics Association, 2012.

[8] NVIDIA. GeForce GTX 980. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-980/specifications, 2015. Online; accessed 15-May-2015.

[9] Intel. i7-5960X. http://goo.gl/RmoS9V, 2015. Online; accessed 15-May-2015.

[10] Boyd, Chas. SIGGRAPH 2008: Beyond Programmable Shading. http://s08.idav.ucdavis.edu/, 2008. Online; accessed 10-May-2015.

[11] Wolfgang Engel. Microsoft® DirectCompute on Intel® microarchitecture Code Name Ivy Bridge Processor Graphics. https://software.intel.com/sites/default/files/m/d/4/1/d/8/DirectCompute_on_DirectX_11.pdf, 2012. Online; accessed 10-May-2015.

[12] Tianyun Ni. Direct Compute Bring GPU Computing to the Mainstream. http://www.nvidia.com/content/GTC/documents/1015_GTC09.pdf, 2009. Online; accessed 10-May-2015.

[13] Jason Zink, Matt Pettineo, and Jack Hoxley. *Practical Rendering and Computation with Direct3D 11*. A. K. Peters, Ltd., Natick, MA, USA, 1st edition, 2011. Chapter: The Computation Pipeline, Page: 289.

[14] Sunil Hadap, Marie-Paule Cani, Ming Lin, Tae-Yong Kim, Florence Bertails, Steve Marschner, Kelly Ward, and Zoran Kačić-Alesić. Strands and hair: Modeling, animation, and rendering. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 1–150, New York, NY, USA, 2007. ACM.

[15] Scienceimage. HUMAN HAIR AND MERINO WOOL FIBRE. http://www.scienceimage.csiro.au/library/textile/i/8115/human-hair-and-merino-wool-fibre/, 2008. Online; accessed 10-May-2015.

[16] Matthias Muller, Tae-Yong Kim, and Nuttapong Chentanez. Fast simulation of inextensible hair and fur. *VRIPHYS*, 12:39–44, 2012.

[17] Sarah Tariq and Louis Bavoil. Real time hair simulation and rendering on the GPU. *ACM SIGGRAPH 2008 talks on - SIGGRAPH '08*, page 1, 2008.

[18] James T Kajiya and Timothy L Kay. Rendering fur with three dimensional textures. In *ACM Siggraph Computer Graphics*, volume 23, pages 271–280. ACM, 1989.

[19] David C Banks. Illumination in diverse codimensions. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 327–334. ACM, 1994.

[20] Robert E Rosenblum, Wayne E Carlson, and Edwin Tripp. Simulating the structure and dynamics of human hair: modelling, rendering and animation. *The Journal of Visualization and Computer Animation*, 2(4):141–148, 1991.

[21] Ken-ichi Anjyo, Yoshiaki Usami, and Tsuneya Kurihara. A simple method for extracting the natural beauty of hair. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 111–120. ACM, 1992.

[22] Eric Plante, Marie paule Cani, and Pierre Poulin. A layered wisp model for simulating interactions inside long hair. In *Proc. of Eurographics Workshop on Animation and Simulation*, pages 139–148, 2001.

[23] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH 98 Conference Proceedings*, pages 43–54, 1998.

[24] Kelly Ward and Ming C Lin. Adaptive grouping and subdivision for simulating hair dynamics. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 234–243. IEEE, 2003.

[25] Byoungwon Choe, Min Gyu Choi, and Hyeong-Seok Ko. Simulating complex hair with robust collision handling. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 153–160. ACM, 2005.

[26] Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, M-P Cani, François Faure, Nadia Magnenat-Thalmann, Wolfgang Strasser, et al. Collision detection for deformable objects. In *Computer Graphics Forum*, volume 24, pages 61–81. Wiley Online Library, 2005.

[27] Tom Lokovic and Eric Veach. Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 385–392. ACM Press/Addison-Wesley Publishing Co., 2000.

[28] Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *Visualization and Computer Graphics, IEEE Transactions on*, 15(2):339–350, 2009.

[29] Sunil Hadap and Nadia Magnenat-Thalmann. Modeling dynamic hair as a continuum. In *Computer Graphics Forum*, volume 20, pages 329–338. Wiley Online Library, 2001.

[30] Yosuke Bando, Bing-Yu Chen, and Tomoyuki Nishita. Animating hair with loosely connected particles. In *Computer Graphics Forum*, volume 22, pages 411–418. Wiley Online Library, 2003.

[31] Lena Petrovic, Mark Henne, and John Anderson. Volumetric methods for simulation and rendering of hair. *Pixar Animation Studios*, 2005.

[32] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[33] Matthias Muller. Hierarchical position based dynamics. *VRIPHYS*, 8:1–10, 2008.

[34] Gilles Daviet, Florence Bertails-Descoubes, and Laurence Boissieux. A hybrid iterative solver for robustly capturing coulomb friction in hair dynamics. *ACM Transactions on Graphics*, 30(6):1, December 2011.

[35] Aleka McAdams, Andrew Selle, Kelly Ward, Eftychios Sifakis, and Joseph Teran. Detail preserving continuum simulation of straight hair. In *ACM Transactions on Graphics (TOG)*, volume 28, page 62. ACM, 2009.

[36] Florence Bertails. Linear time super-helices. In *Computer Graphics Forum*, volume 28, pages 417–426. Wiley Online Library, 2009.

[37] Shinjiro Sueda, Garrett L Jones, David IW Levin, and Dinesh K Pai. Large-scale dynamic simulation of highly constrained strands. In *ACM Transactions on Graphics (TOG)*, volume 30, page 39. ACM, 2011.

[38] Hayley Iben, Mark Meyer, Lena Petrovic, Olivier Soares, John Anderson, and Andrew Witkin. Artistic simulation of curly hair. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 63–71. ACM, 2013.

[39] Peng Guan, Leonid Sigal, Valeria Reznitskaya, and Jessica K Hodgins. Multi-linear data-driven dynamic hair model with efficient hair-body collision handling. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 295–304. Eurographics Association, 2012.

[40] Menglei Chai, Changxi Zheng, and Kun Zhou. A reduced model for interactive hairs. *ACM Transactions on Graphics, (Proc. of SIGGRAPH 2014)*, 33(4):to appear, 2014.

[41] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):214, 2013.

[42] EuroGamer. Geralt's beard grows as you play Witcher 3. `http://www.eurogamer.net/articles/2015-03-24-geralts-beard-grows-as-you-play-witcher-3`, 2015. Online; accessed 10-May-2015.

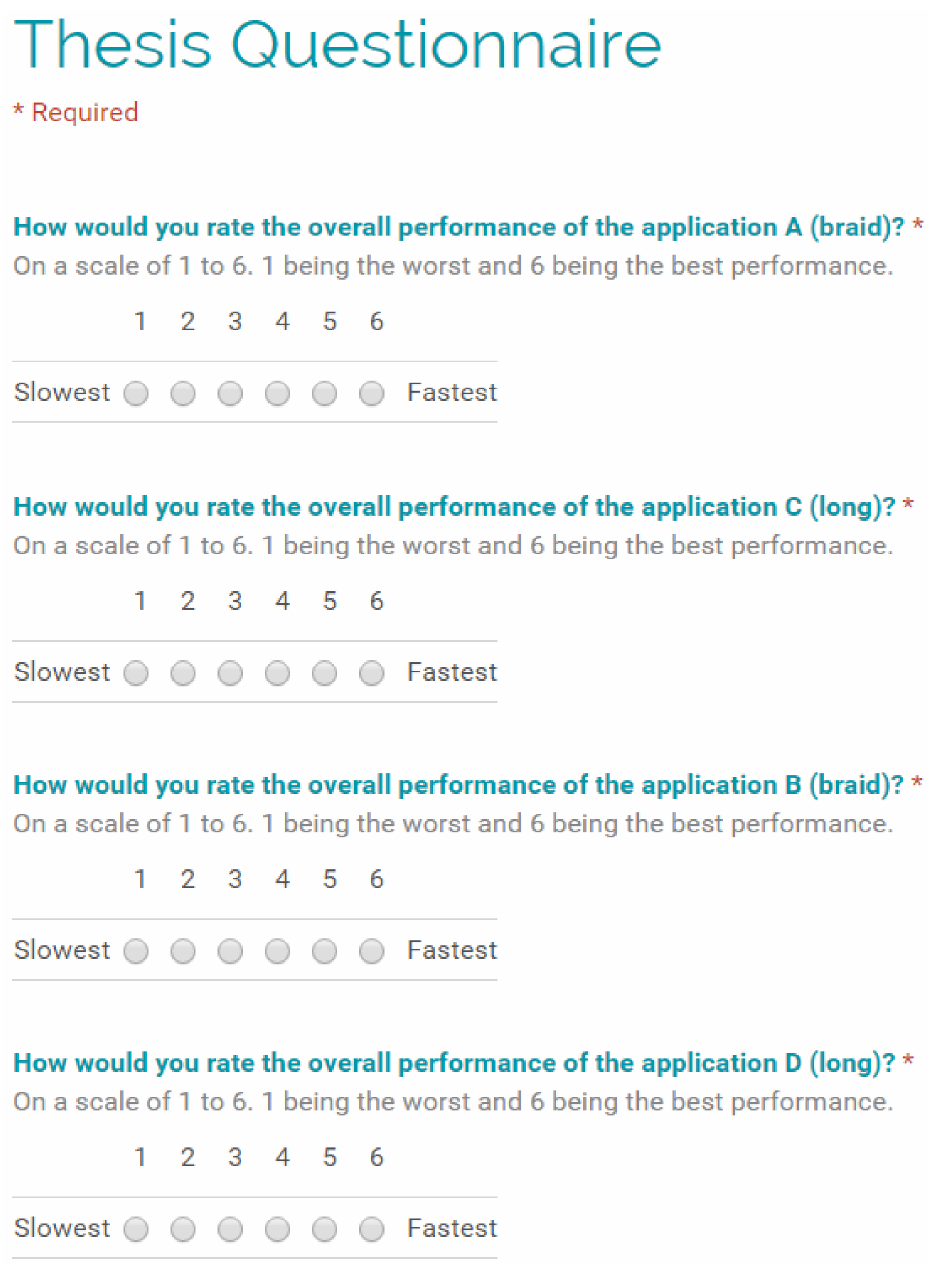
[43] Wm Randolph Franklin, Chandrasekhar Narayanaswami, Mohan Kankanhalli, David Sun, Meng-Chu Zhou, and Peter YF Wu. Uniform grids: A technique for intersection detection on serial and parallel machines. In *Proceedings of Auto-Carto*, volume 9, pages 100–109, 1989.

[44] Youquan Liu, Xuehui Liu, and Enhua Wu. Real-time 3d fluid simulation on gpu with complex obstacles. In *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*, pages 247–256. IEEE, 2004.

[45] Microsoft Dev Center. RWStructuredBuffer. `http://bit.ly/1Fi64Z4`, 2013. Online; accessed 10-May-2015.

Appendix A

QUESTIONNAIRE



Thesis Questionnaire

* Required

How would you rate the overall performance of the application A (braid)? *

On a scale of 1 to 6. 1 being the worst and 6 being the best performance.

	1	2	3	4	5	6	
Slowest	○	○	○	○	○	○	Fastest

How would you rate the overall performance of the application C (long)? *

On a scale of 1 to 6. 1 being the worst and 6 being the best performance.

	1	2	3	4	5	6	
Slowest	○	○	○	○	○	○	Fastest

How would you rate the overall performance of the application B (braid)? *

On a scale of 1 to 6. 1 being the worst and 6 being the best performance.

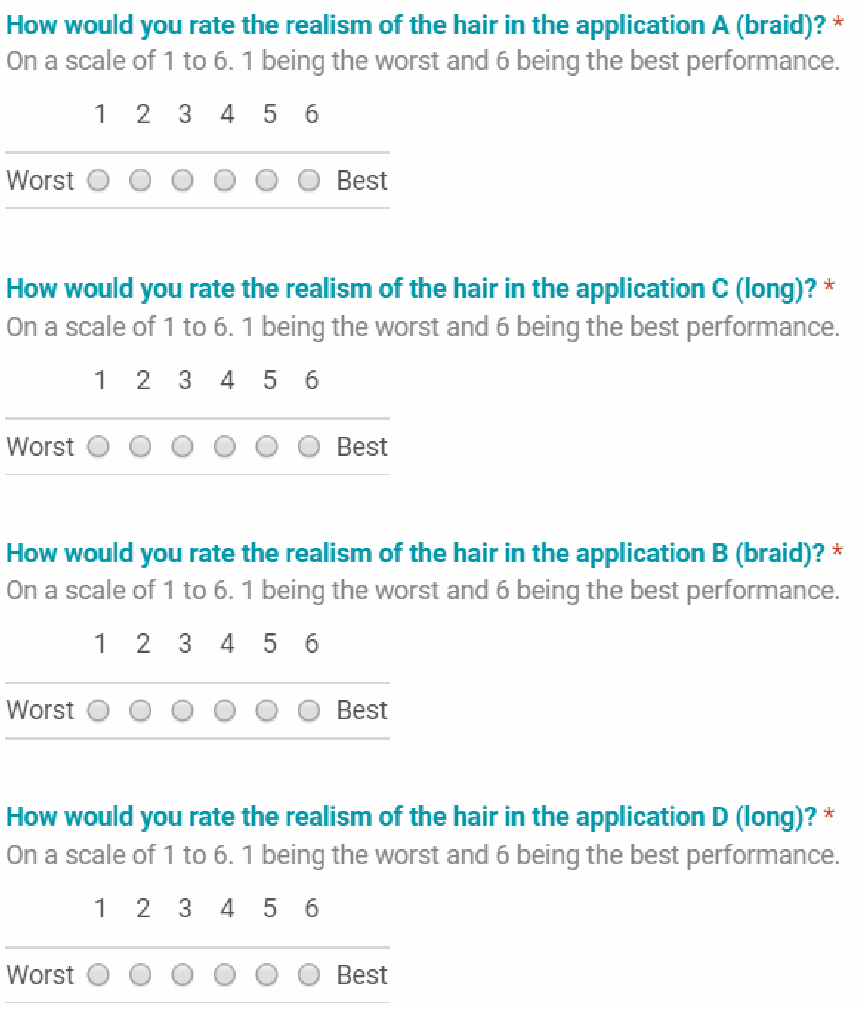
	1	2	3	4	5	6	
Slowest	○	○	○	○	○	○	Fastest

How would you rate the overall performance of the application D (long)? *

On a scale of 1 to 6. 1 being the worst and 6 being the best performance.

	1	2	3	4	5	6	
Slowest	○	○	○	○	○	○	Fastest

Figure A.1: Hair Performance Questionnaire



How would you rate the realism of the hair in the application A (braid)? *
On a scale of 1 to 6. 1 being the worst and 6 being the best performance.

	1	2	3	4	5	6	
Worst	○	○	○	○	○	○	Best

How would you rate the realism of the hair in the application C (long)? *
On a scale of 1 to 6. 1 being the worst and 6 being the best performance.

	1	2	3	4	5	6	
Worst	○	○	○	○	○	○	Best

How would you rate the realism of the hair in the application B (braid)? *
On a scale of 1 to 6. 1 being the worst and 6 being the best performance.

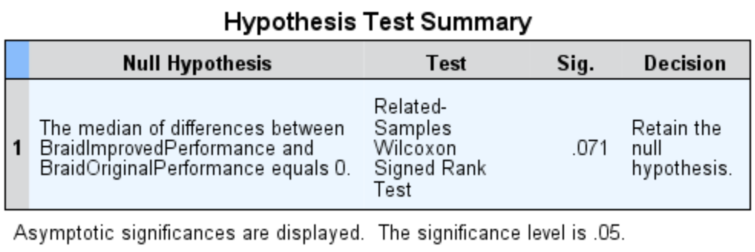
	1	2	3	4	5	6	
Worst	○	○	○	○	○	○	Best

How would you rate the realism of the hair in the application D (long)? *
On a scale of 1 to 6. 1 being the worst and 6 being the best performance.

	1	2	3	4	5	6	
Worst	○	○	○	○	○	○	Best

Figure A.2: Hair Realism Questionnaire

A.1 Performance Test of Braid Hair Style

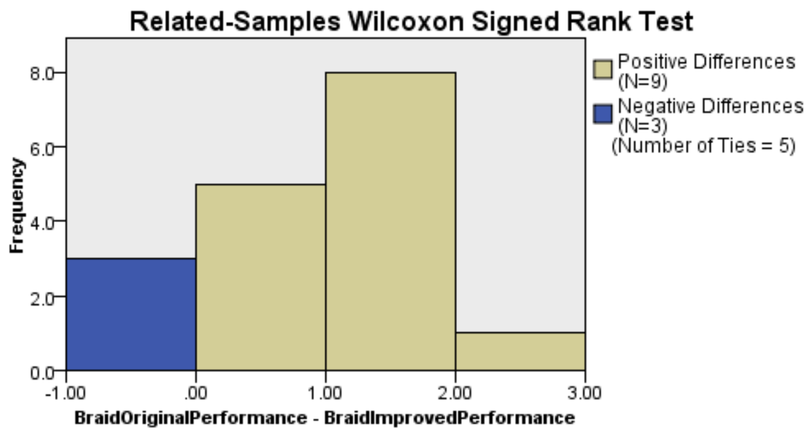


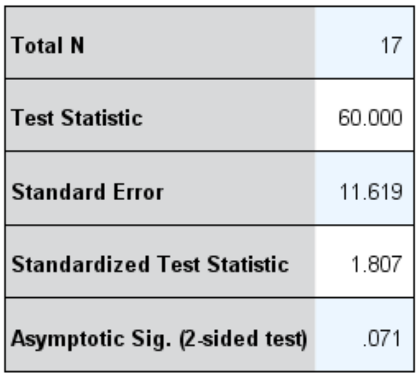
Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The median of differences between BraidImprovedPerformance and BraidOriginalPerformance equals 0.	Related-Samples Wilcoxon Signed Rank Test	.071	Retain the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

Figure A.3: Braid Hair Performance Test

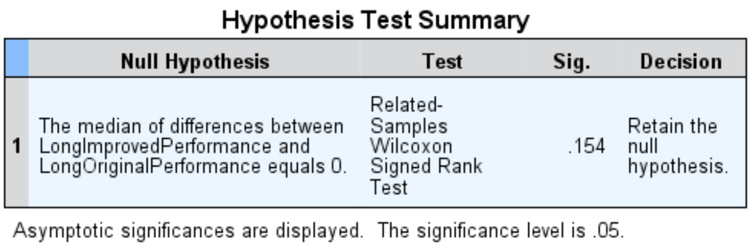




Total N	17
Test Statistic	60.000
Standard Error	11.619
Standardized Test Statistic	1.807
Asymptotic Sig. (2-sided test)	.071

Figure A.4: Braid Hair Performance Test Graph

A.2 Performance Test of Long Hair Style

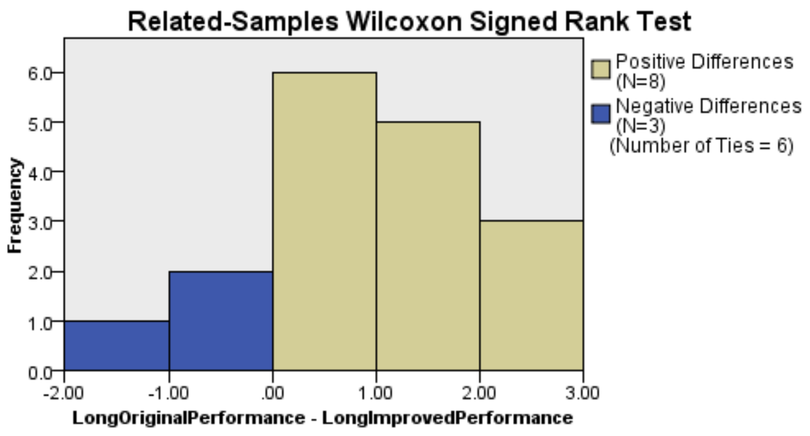


Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The median of differences between LongImprovedPerformance and LongOriginalPerformance equals 0.	Related-Samples Wilcoxon Signed Rank Test	.154	Retain the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

Figure A.5: Long Hair Performance Test



Total N	17
Test Statistic	48.500
Standard Error	10.874
Standardized Test Statistic	1.425
Asymptotic Sig. (2-sided test)	.154

Figure A.6: Long Hair Performance Test Graph

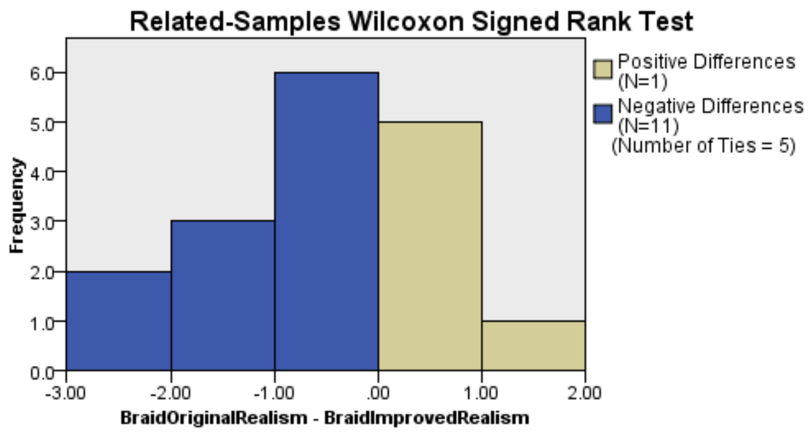
A.3 Realism Test of Braid Hair Style

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The median of differences between BraidImprovedRealism and BraidOriginalRealism equals 0.	Related-Samples Wilcoxon Signed Rank Test	.005	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

Figure A.7: Braid Realism Test



Total N	17
Test Statistic	4.000
Standard Error	12.445
Standardized Test Statistic	-2.812
Asymptotic Sig. (2-sided test)	.005

Figure A.8: Braid Realism Test Graph

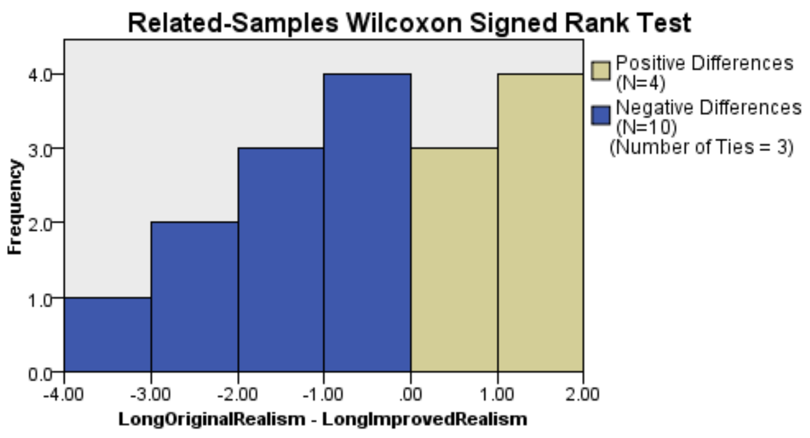
A.4 Realism Test of Long Hair Style

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The median of differences between LongImprovedRealism and LongOriginalRealism equals 0.	Related-Samples Wilcoxon Signed Rank Test	.027	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

Figure A.9: Long Hair Realism Test



Total N	17
Test Statistic	18.000
Standard Error	15.576
Standardized Test Statistic	-2.215
Asymptotic Sig. (2-sided test)	.027

Figure A.10: Long Hair Realism Test Graph