REAL-TIME STRAND-BASED HAIR RENDERING USING GUIDE HAIR INTERPOLATION AND DEPTH PEELING

JONATHAN LIEW EU JIN

MATHEMATICS WITH COMPUTER GRAPHICS PROGRAMME
FACULTY OF SCIENCE AND NATURAL RESOURCES
UNIVERSITI MALAYSIA SABAH

2024

REAL-TIME STRAND-BASED HAIR RENDERING USING GUIDE HAIR INTERPOLATION AND DEPTH PEELING

JONATHAN LIEW EU JIN

THIS DISSERTATION IS SUBMITTED FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE WITH HONOURS

MATHEMATICS WITH COMPUTER GRAPHICS PROGRAMME

FACULTY OF SCIENCE AND NATURAL RESOURCES

UNIVERSITI MALAYSIA SABAH

2024

STUDENT DECLARATION

I hereby declare that this dissertation has been composed solely by myself, the work contained here is my own effort except for the citations and work that has been cited along with their respective sources.

[illegible]

JONATHAN LIEW EU JIN
(BS20110240)
19 JANUARY 2024

SUPERVISOR VALIDATION

SIGNATURE

SUPERVISOR
(RECHARD LEE)

[illegible]

ACKNOWLEDGEMENT

I would like to express my sincere appreciation to everyone that have assisted me in the completion of my Final Year Project, titled " Real-time Strand-Based Hair Rendering using Guide Hair Interpolation and Depth Peeling".

First, I would want to express my sincere gratitude for the assistance and support provided by my supervisor, Mr. Rechard Lee. He had supported me throughout this project, offering important input and nurturing my development as a researcher.

Secondly, I express my gratitude towards my examiner, Associate Professor Dr. Abdullah Bade for providing valuable assistance, support, and feedback on the errors I made in my dissertation writing.

I would also like to express my gratitude to all my Mathematics with Computer Graphics lecturers for all the information and skills passed down to me that has served as the basis for my project. Their unwavering instruction established the foundation for my achievement.

Furthermore, I would want to convey my utmost gratitude to my parents. Their endless encouragement and support throughout this project were my profound motivation.

Lastly, I would like to thank my friends for their vital efforts and endless support. Their propensity to exchange ideas and provide assistance throughout this thesis writing process had helped me a lot.

ABSTRACT

Hair is an important element in CGI and digital animation. However, it has been proven to be challenging to simulate hair properties when rendering hair. Lighting model that accounts for more than one light scattering direction renders high quality hair strands in terms of visual appearance at the cost of performance, while overlapping hair strands produce incorrect occlusion when accounting for hair transparency. The aim of this project is to render hair geometry in real-time and adapt order-independent transparency on hair strands. The objectives of this project are to render hair geometry in real-time using guide hair interpolation and to adapt order-independent transparency for correct occlusion of hair strands using depth peeling. These objectives were evaluated through two tests. Performance tests were carried out using different experiments, which were different number of guide hair strands with a fixed total number of hair strands rendered, different hair type setups with a fixed total number of hair strands, and different total number of hair strands with a fixed number of strands per guide patch. The results from the performance tests carried out have shown that guide hairs have reduced the number of hair strands simulated. This has allowed the hair-rendering performance to achieve the benchmark requirements of between 30 fps or 33.33 ms and 60 fps or 16.67 ms. The proposed system was also able to render different types of hairstyles that consists of up to about 50000 strands in real-time. The second test involved visual comparison. Visual results show depth peeling rendered hairball geometry with proper occlusion, thus visual results obtained are better than that of traditional alpha blending. Both objectives of this project were achieved. In conclusion, this project has provided a technique to improve hair-rendering performance and adapt order-independent transparency when rendering explicit hair strands.

PERSEMBAHAN HELAIAN RAMBUT WAKTU NYATA MENGGUNAKAN INTERPOLASI RAMBUT PANDU DAN PENGUPASAN KEDALAMAN

ABSTRAK

Rambut merupakan elemen yang penting dalam CGI dan animasi digital. Walau bagaimanapun, antara cabaran dalam bidang persembahan rambut adalah untuk menyelak sifat rambut. Model pencahayaan yang yang mempunyai lebih daripada satu arah penyerakan cahaya menghasilkan benang rambut berkualiti tinggi dari segi visual dengan kos prestasi system. Selain itu, helai-helaian rambut yang kerap bertindih menghasilkan oklusi yang salah ketika mempersembahkan rambut yang tabun. Tujuan projek ini adalah untuk mempersembahkan geometri rambut dalam masa nyata dan mengaplikasi ketelusan urutan bebas pada helaian rambut. Matlamat projek ini adalah untuk untuk mempersembahkan geometri rambut dalam masa nyata menggunakan interpolasi rambut panduan, dan untuk mengaplikasi ketelusan urutan bebas untuk mempersembahkan rambut dengan pertindihan helaian yang betul menggunakan pengupasan kedalaman. Kedua-dua objektif ini telah dinilai melalui dua ujian. Ujian prestasi dijalankan dengan menggunakan eksperimen yang berbeza, iaitu dengan menggunakan jumlah bilangan helaian rambut panduan yang berbeza dengan bilangan keseluruhan rambut malar, jenis rambut yang berbeza dengan jumlah total rambut malar, dan jumlah helaian rambut yang berbeza dengan jumlah helaian rambut yang tetap dalam setiap tompok pandu. Hasil daripada ujian tersebut telah menunjukkan bahawa rambut panduan telah mengurangkan bilangan rambut yang disimulasikan. Ini telah membolehkan prestasi persembahan rambut untuk mencapai keperluan penentu prestasi, iaitui antara 30 fps (33.33ms) dan 60 fps (16.67 ms). Sistem cadangan juga dapat menyembah pelbagai jenis gaya rambut yang terdiri daripada sebanyak 50000 helai dalam masa nyata. Ujian kedua melibatkan perbandingan hasil visual. Hasil penilaian visual menunjukkan bahawa model bebola rambut yang disembah dengan teknik pengupasan mempunyai pertindihan helaian yang betul. Oleh itu, hasil visual yang diperolehi adalah lebih baik daripada yang hasil teknik tradisional persebatian alfa. Kedua-dua objektif projek ini telah tercapai. Kesimpulannya, projek ini telah menyediakan teknik untuk meningkatkan prestasi persembahan geometri rambut dan mengaplikasi ketelusan urutan bebas ketika mempersembahkan rambut yang eksplisit.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Ψ	Marschner lighting model
$\Psi_{ambient}$	Ambient lighting component
$\Psi_{diffuse}$	Diffuse lighting component
$\Psi_{specular}$	Specular lighting component
Ψ_{R}	Primary (R path) specular component
Ψ_{TRT}	Secondary (TRT path) specular component
Ψ_{TT}	Strong Transmittance (TT path) specular component
k_a	Ambient reflection coefficient
k_d	Diffuse reflection coefficient
k_s	Specular reflection coefficient
θ	Angle of incidence
$\emptyset$	Rotation of hair strand around its axis
$\boldsymbol{l}$	Light vector
$\boldsymbol{l}'$	Projection of the light direction on the normal plane
$\boldsymbol{n}$	Surface normal vector
α	Angle of tilted cuticle scales
α_R	Longitudinal shift of R path
α_{TRT}	Longitudinal shift of TRT path
α_{TT}	Longitudinal shift of TT path
β_R	Surface roughness of fibre
β_R	Longitudinal width of R path
β_{TRT}	Longitudinal width of TRT path
β_{TT}	Longitudinal width of TT path
$\bar{C}_{result}$	Final blended image
α_s	Opacity (alpha) value of hair strand
$\bar{C}_{source}$	Final fragment colour vector of current hair strand
$\bar{C}_{destination}$	Colour vector stored in frame buffer
p	Kajiya-Kay Specular exponent coefficient
Z_i	Current depth value during generation of opacity map
Z_0	Depth value at which the head geometry begins from camera view

LIST OF ABBREVIATIONS

2D	Two-dimensional
3D	Three-dimensional
fps	Frames per second
ms	Millisecond
CGI	Computer-generated imagery
CPU	Central processing units
GPU	Graphics processing units
GUI	Graphical user interface
IDE	Integrated Development Environment
OIT	Order-Independent Transparency
R	Reflection
TT	Transmission-transmission
TRT	Transmission-reflection-transmission
UML	Unified Modeling Language

CHAPTER 1

INTRODUCTION

1.1 Overview

In this modern world of advancing technology, principles, and fundamentals of animation, form a successful design medium to catch the interest and attention of viewers through exaggerated and surreal artistic visual effects (Sheng, 2017). Animation is generally a combination of multiple pictures of drawings, items, or people in different situations moving through a series of frames that visually looks like constant modification (Jin, 2011; Lowe & Schnotz, 2008 and Dajani & Abu, 2019). However, according to Khong Kok Wai (2009), this medium of storytelling has come a long way as the transition in animation techniques from conventional hand-drawn methods to digital image rendering and computer-generated imagery (CGI) has substantially decreased the amount of time and work necessary for the creation of animation shots. With the aid of computer graphics fundamentals, two-dimensional (2D) and three-dimensional (3D) animations have been used in various industries, including film, television, advertising, and video games, to generate visual and cinematic effects, providing more realistic and immersive visual experiences.

As computer technology is rapidly advancing, 3D animations have become more accessible to digital artists and creators. This form of animation uses 3D computer models that have volume or mass, which makes the motion of these models depend on the influence of gravity (Au, 2014). Hence, a high set of skills to animate while having a basic knowledge of physics may be required to imitate the realistic effect from the real world to the 3D virtual environment. There has been a vast development in 3D animated films by Pixar from the film 'Toy Story' in 1995 to 'Turning Red' in 2022.

In animation, hair and fur are components attached to the mammal's skin surface and are animated using techniques like rigging. Rigging is the process of controlling a 3D model. It is important in animation because it involves the -

manipulation and modification of the characteristics of a virtual character such as their facial expressions (Orvalho *et al.*, 2012) and body language. However, rigging hair and fur strands individually through multiple iterations became a problem for digital artists. Wyndham Batton (2016) came up with a dynamic hair-rigging system to provide more freedom and control for artists when styling virtual hair.

Hair is a fibrous protein that develops from mammals' follicles in the dermis. It consists of three major components, which are the cortex, cuticle, and medulla. Realistic hair rendering has gained the interest of many artists and developers in today's world. Various industries such as video games and advertisements (Han, 2014; Joy *et al.*, 2022) have researched and applied hair rendering into their repertoire. Besides realistic dynamics, hair details are important in producing realistic visual representations of a virtual character.

In this study, hair geometry will be rendered in real-time and order-independent transparency will be adapted to account for correct occlusion of hair strands.

1.2 Problem Background

Hair is an important element in CGI and digital animation. It can express a fictional character's personality and emotion and contribute to its overall realism. Interactions of hair under external influence such as physics can make these simulations difficult. The demand for realistic simulations and renderings in the field of computer graphics has drastically increased as the state of technology today is rapidly developing. As we are living in a world where virtual, augmented, and extended reality are common technologies used in our daily lives, the demand for high-quality hair renders is increasing. Hair is one of the more complex parts of a mammal. Hence, modelling hair may be challenging for digital artists and back-end programmers. As mammals can have a minimum of thousand strands of hair, researchers and developers have been trying to obtain efficient ways to render hair models that are both cheap and visually appealing.

Studies have been done to obtain methods to render hair. Yang *et al.* (2000) and Kajiya and Kay (1989) have proposed to render hair strands as implicit geometry models. Allowing light properties to be simulated throughout the surface of hair based on hair densities (Andersen *et al.*, 2016) during the rendering process would give the rendered result a more volumized structure with a smaller number of hairs required (Yuksel & Tariq; 2010). The disadvantage of implicit-based techniques is that individual strands of hair are not represented well as it looks as if the hair strands are stuck as a single mesh (Figure 1.1). This technique also lacks explicit details to distinguish one hair strand from another (Kajiya & Kay, 1989).

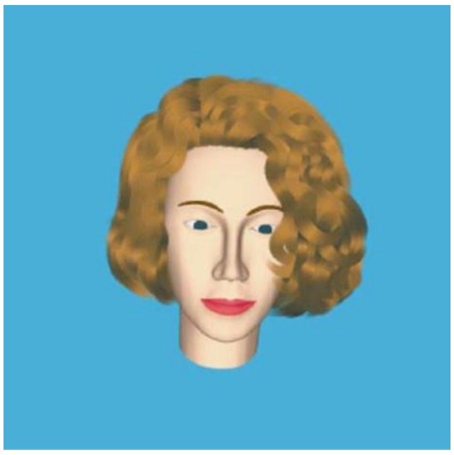


Figure 1.1 Cluster Hair model's render. Reprinted from "The cluster hair model", (Yang *et al.*, 2000)

One of the challenges faced by developers is to render hair strands explicitly. (Watanabe & Suenaga, 1991) suggested representing each individual hair strand as a trigonal prism. However, camera-facing triangles are expensive (Yuksel & Tariq, 2010). Instead of using 3D model representations, a common method used in multiple research (Ward *et al.*, 2007; Han, 2014) to overcome this challenge is to represent hair as an array collection of 2D line segments. This approach does not only have simple geometrical details, but they also allow styling variations such as curly or frizzy hair using Bezier interpolation (Kim, 2002). These simple details are easier to deal with and will reduce computational costs (Yuksel & Tariq, 2010). As lines are 2D, hair textures can only be mapped throughout the length of the polyline but not across the width. Polylines also result in rendering issues such as aliased edges and dull appearances (Ward *et al.*, 2007). This is not visually pleasing when the hairy model is rendered closer to the camera. Therefore, this approach on its own results in hair renders with lower quality.

Hair has complex properties, where individual strands are self-shadowing, translucent, and thin (Sintorn & Assarsson, 2008; Ward *et al.*, 2007). Accounting for light interaction on hair surfaces are also important during the rendering process (Marschner *et al.*, 2003). High-quality and physically accurate hair (Marschner *et al.*, 2003) can be rendered with these details. However, simulating these properties during the rendering process can be costly. For example, simulating light scattering on hair fibres can produce a more physically accurate representation hair model (Marschner *et al.,* 2003) but is costly when rendering hair geometry that comprises of a minimum of thousands of strands (Ou *et al.*, 2012; O'Reilly, 2022). Next, alpha blending, a common rendering technique to handle transparency, is used to produce translucent hair strands by lowering the alpha value at the ends of the fibre. However, as hair is highly overlapping with one another (Hunz, 2016), it makes the sorting process impractical (Sintorn & Assarsson, 2008) thus becoming a challenge when rendering hair strands when accounting for hair transparency.

In summary, rendering hair illumination and transparency in real time is still a challenge.

1.3 Problem Statement

Hair is an important element in the field of computer graphics as it is widely used in real-time rendering applications in many industries in today's world. Translucency and illumination are important properties to produce high-quality explicit hair strands (Sintorn & Assarsson, 2008; Ward *et al.*, 2007). However, researchers encounter many challenges in accounting for these properties when rendering hair in real time. Hair-rendering algorithms that account for more than one light scattering direction produces more accurate hair models but is very expensive to render (Marschner *et al.*, 2003). Next, alpha blending techniques produce incorrect occlusion when rendering the final image (Hunz, 2016) as hair strands are always overlapping. Hence, techniques are required to improve the performance when rendering hair geometry and sort the hair strands when accounting for hair transparency.

1.4 Aim

This project aims to render hair geometry in real-time and adapt order-independent transparency on hair strands.

1.5 Objectives

- To render hair geometry in real-time using guide hair interpolation.
- To adapt order-independent transparency for correct occlusion of hair strands using depth peeling.

1.6 Scope

- Hair will be rendered in dry conditions only.
- Hair strands are rendered as a hairball on a sphere model.
- Hair will be rendered using different hair type setups.
- Only one light source is set up in the scene.

1.7 Justification

As technology today is developing, the abundant availability of resources is an incentive for the initiation of this project. This project looks to improve virtual character designing and immersive and interactive experiences in digital content. This hair rendering technique could help revamp the quality of visual effects in many fields in our daily lives. Next, high-quality hair models require a long time to be rendered per frame as shown in (Andersen *et al.*, 2016). However, applications such as games and virtual reality require cheaper rendering algorithms along with simplified but detailed geometric representations of hair. This project aspires to contribute to the field of animation and computer graphics by providing a technique to render explicit hair models in real time.

Today, hair rendering methods are widely used, not just in the entertainment fields such as virtual character representations in real-time video games (Bertails *et al.*, 2008; Han, 2014), but as well as in the advertising field where the concept of metaverse is used as business opportunities to connect with tech-savvy and younger audiences (Joy *et al.*, 2022).

1.8 Project Scheduling

The table and figure below show a list of scheduled tasks along with their respective estimated time frames (refer Table 1.1 and Figure 1.2).

Table 1.1 List of scheduled tasks with their categories and time frames.

CATEGORY	ID	TASK	START	END	DURATION (DAYS)
Project Research	1	Researching on latest hair rendering issues	30.3.2023	24.4.2023	26

	2	Determining aim, objectives, problem statement, and problem background	11.4.2023	1.5.2023	21
	3	Literature review	4.5.2023	30.6.2023	58
	4	Framework design	24.6.2023	9.7.2023	16
Project Building	5	Setting up OpenGL Library	21.6.2023	30.8.2023	70
	6	Importing 3D models	31.8.2023	3.9.2023	4
	7	Basic renderer for models	4.9.2023	10.9.2023	6
	8	Placing guide strands	7.9.2023	21.9.2023	14
	9	Setting up lighting	22.9.2023	30.9.2023	8
	10	Setting up approximate strands	1.10.2023	25.10.2023	24
	11	Implement shadow mapping	26.10.2023	15.11.2023	20
	12	Setting up depth peeling	18.11.2023	30.11.2023	12
Project Analysis	13	System Analysis and Evaluation	1.12.2023	30.12.2023	29

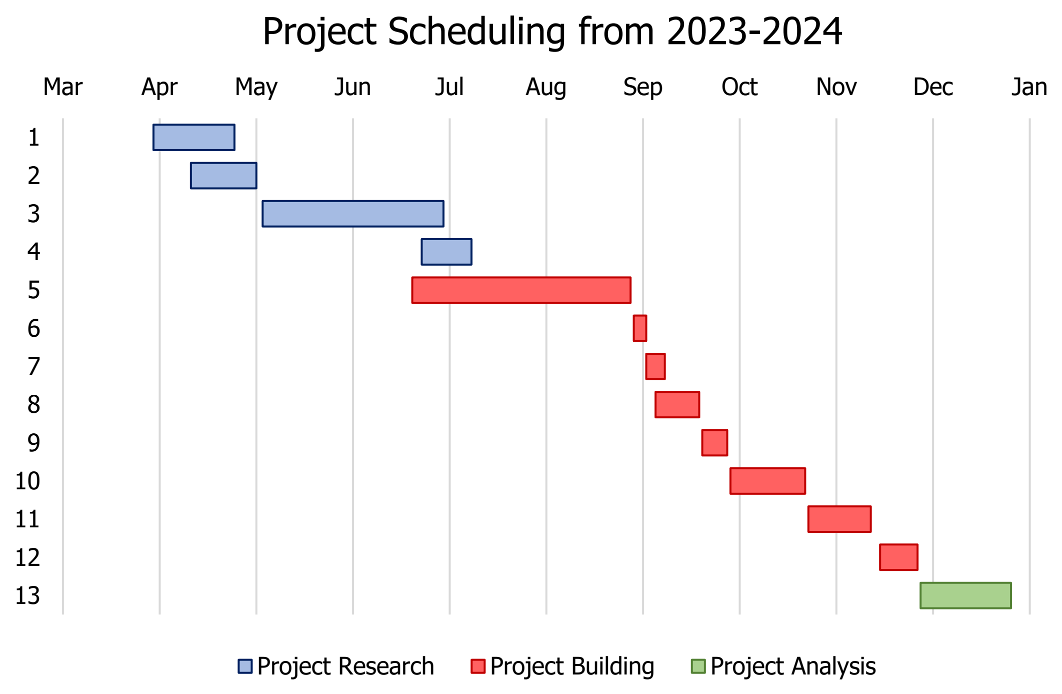


Figure 1.2 Graphical project scheduling representation.

1.9 Organisation Thesis

Chapter 1 introduces the background of hair rendering. This chapter also contains the overview and problem background leading into the problem statement that is focused on this project. This chapter also discusses the aim, objectives, scope and limitation, and justification for carrying out this project.

Chapter 2 will focus on past research on hair rendering. In-depth details on hair representation, illumination, shadowing, and transparency are also discussed in this chapter.

Next, chapter 3 will elaborate on the methodology used in this project. This chapter will analyse the framework, techniques used, and benchmarking of this project.

Chapter 4 will discuss the analysis and design of this project. Diagrams illustrating the flow of the project's design and the algorithm used in this project will be shown in this chapter.

Chapter 5 will show the result obtained from this project. An in-depth analysis of the results obtained are discussed in this chapter.

Chapter 6 will summarise and conclude this project. Lastly, contributions and future works will be discussed in this chapter.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter will discuss computer graphics and hair rendering techniques. The history and development of computer graphics will be further elaborated. Next, the background and techniques done by past researchers to render hair will be analysed. This chapter will review past research on hair representation, illumination models, shadowing, and transparency. This chapter will also discuss results and evidence by multiple scholars and researchers to further support information on these topics.

2.2 Computer Graphics

Marschner and Shirley (2018) defined the concept of computer graphics as a creation and manipulation process resulting in computer-based images. The origins of computer graphics can be traced back to the fields of arts and mathematics. Euclid, a Greek mathematician, laid the foundation for concepts of computer graphics with his formulation of geometry for two-dimensional (2D) and three-dimensional (3D) modelling (Fuller & Prusinkiewicz, 1989). René Descartes further contributed by discussing his concept on location of objects in space, which led towards the invention of the Cartesian xy-Plane (Vince, 2010).

In the early 1960s, there were already a few top manufacturers for digital computers such as International Business Machines Corporation (IBM) and Sperry Rand (Ekpenyong, 2009). However, Ekpenyong stated that the memory capacity of digital computers at that time only had a few kilobytes. In 1963, Ivan Sutherland achieved a milestone in the field of computer graphics when he introduced Sketchpad, a system that allowed a digital monitor to receive input from a digital pen (Johnson, 1963). Later, drawing algorithms for primitives such as lines and circles were introduced by Jack-

Bresenham (Ekpenyong, 2009).

Computer graphics is applied in many fields such as engineering, medicine, and art (Mortenson, 1999; Vidal *et al.*, 2006 and Lee *et al.*, 2021), due to the major developments in computer technology. In the past, computer graphics was used to assist many fields in decision making by summarising data in graphical representations such as tables, charts, and graphs (DeSanctis, 1984). Stronger computers were later able to model various objects using computer-aided design (CAD) methods (Tornincasa & Di Monaco, 2010). These methods were used to design various products such as buildings and vehicles. Today, there are many graphical editors, like Blender and AutoCAD, that are easily accessible to the public to carry out 3D rendering and modelling (Minnegalieva *et al.*, 2020).

The concept of virtual worlds, such as metaverse, and virtual characters have recently caught a lot of attention from our society (Ning *et al.*, 2023). Computer-generated imagery (CGI), an application of computer graphics, is being utilised to create virtual characters in many industries, for both business and entertainment purposes (Lee *et al.*, 2021). Academic literature such as (Brown & Cairns, 2004; Jennett *et al.*, 2008 and Nacke & Lindley, 2008) have proven that the appearance of a virtual character can influence users' attachment towards that character. These studies have also shown that characters that represent their user's emotions and personality traits enhance the user's immersion and overall experience.

As the demand for virtual characters increases, the importance of realistic virtual hair also grows. A study by LaFrance (2001) has shown that hairstyle is important in forming identity expressions. (Ducheneaut *et al.*, 2009; Turkay & Kinzer 2014) later showed that the visual appearance of computer-generated character's hair, such as hair colour and hairstyle variations, is one of the more important features for personality expression. Therefore, rendering high quality hair models is important for any application involving virtual characters, enhancing the overall realism in a scene.

2.3 Hair Strands

The geometry of a hair strand is very thin. It is said to be less than 0.1mm (Park *et al.*, 2023). Each strand can be segmented into a few parts. The first segment is the hair follicle. This active part of the hair is located in the dermal layer of the skin's surface layer (Nuutila, 2021) and is responsible for the production of keratin and sebum, structures responsible for the stability and shape of the hair fibre (Yoo *et al.,* 2010).

The other segment is the hair shaft, the visible part of the hair strand and is composed of the medulla, cortex, and cuticle (Shen *et al.*, 2020).

Hair strands exhibit various geometric properties across individuals, such as naturally straight, wavy, and curly hair. The almost elliptical cross-sectional geometry of hair ranges between 50 to 80 μm (Thozhur *et al.*, 2006). The geometric features of hair exhibit a certain level of consistency across individuals, allowing for a schematic differentiation between three categories, African, Asian, and Caucasian (Leerunyakul & Suchonwanit, 2020).

While hair strands have unique geometries, they have optical properties that mainly influenced by their translucent characteristics (Sintorn & Assarsson, 2008). Hair strands have refractive index of about 1.55 (Natarova *et al.*, 2019). The cortex of the hair strand includes melanin granules, which determine the hair's coloration (Shen *et al.*, 2020). There are two distinct forms of melanin: the predominant form is eumelanin, which imparts black or brown pigmentation to the hair, while the less prevalent form is pheomelanin, which gives the hair a yellow or red hue (Battistella *et al.*, 2020).

The hair's surface, which consists of overlapping scales, affects its interaction with light (Marschner *et al.*, 2003). Marschner *et al* states that some of the light that interacts with the semi-transparent cylindrical fibre is immediately reflected off the surface of the fibre. This results in a specular reflection that has the same colour as the incident light. A percentage of the light enters the fibre through transmission, gets partially absorbed, and then comes back out through scattering underneath the surface, resulting in a scattered reflection that has the same hue as the pigments inside the fibre. However, unlike a perfectly cylindrical shape, the overlapping cells that forms the hair strand's surface (Yang *et al*, 2014) causes some deviation on the reflected light rays (Marschner *et al.*, 2003). An illustration and further discussion on light scattering can be referred in Chapter 2.4.2b).

2.4 Hair Rendering

Rendering hair can be challenging due to the large number of strands a mammal has. This number can be a minimum of tens of thousands (Ward *et al.*, 2007). Hair strands also have complex properties, where individual strands are self-shadowing, translucent, and thin (Sintorn & Assarsson, 2008, 2009 and Ward *et al.*, 2007). Complex mathematical computations are required during the rendering process to ensure these properties are appropriately accounted for. This section discusses different hair representations, illumination models, and shadowing and translucent properties.

2.4.1 Hair Representation

The representation of hair in rendering can either be explicit or implicit. The choice of rendering algorithm depends on the type of geometric representation used for modelling the hair. Explicit models involve rendering hair using strand-based techniques that work with geometric primitives like lines or triangles. On the other hand, implicit models require algorithms capable of handling their volumized representation. The following three sub-sections will discuss more on strand-based technique, volume-based technique, and a proposed hybrid technique by Andersen *et al.* (2016) in the hair rendering field.

a) Strand-Based Technique

When explicitly representing hair, each individual hair strand is represented as a curved cylinder. In earlier works, (Watanabe & Suenaga, 1991, 1992) used trigonal prisms to represent hair, where each strand was modelled as a series of prisms connected by three faces. This method assigned a unique colour to each face of the prism, assuming minimal colour variation across the hair thickness. Another approach for hair representation involves a continuous strip of triangles (Yuksel & Tariq, 2010), where each triangle is always oriented towards the camera. However, Yuksel and Tariq have also stated that these proposed methods are too costly to render hair in real time.

Tessellation is a rendering technique that involves subdividing geometric primitives, resulting in more detailed geometry, for further processing in the rendering pipeline (Chang, 2018). Neulander and van de Panne (1998) proposed a less costly technique that involves dynamically tessellating the geometry of hair into polygons based on the hair's curvature and the distance from the camera. This approach allows for accurate rendering of strand cylinders using a reduced number of larger polygons. While this method incurs additional computational overhead compared to fixed tessellation, the benefits of accurately rendering hair strands with fewer polygons outweigh the cost. This also reduces cost for vertex transformations, rasterization, and antialiasing. Kong and Nakajima (1999) further optimised this technique by dynamically creating additional hairs at the boundary, effectively reducing the total number of hairs that need to be rendered.

Years later, academic literature like (LeBlanc *et al.*, 1991; Ward *et al.*, 2007 and Han, 2014) then represented hair strands as an array of 2D line segments. Line primitives are much cheaper to compute compared to triangles. Kim (2002) used B-spline interpolation to increase the density of individual hair strands and approximate the curves between each knot better. This allows one to manually shape the hair strands at any point throughout the length of the polyline for multiple hairstyling variations (Rankin & Hall, 1996).

When rendering an explicit representation of hair, the thin geometric nature of hair can cause aliasing issues, such as jagged lines, which become noticeable, especially when dealing with many hair strands. This problem arises because hair strands are of sub-pixel size. This results in abrupt colour changes or noisy edges around the hair. Antialiasing techniques can help overcome this issue by smoothening out the edges of hair strands.

One approach to address this problem, introduced by (LeBlanc *et al.*, 1991), involves using blender buffer techniques to mix the appropriate colours for each pixel. In this method, hairs are represented as broken lines, and the shading colour is blended within a pixel buffer. The order of drawing the polygons starts with those nearest to the camera until the one furthest from the camera. This ordering helps to reduce aliasing artifacts by ensuring that the colours of overlapping hairs are correctly blended, resulting in smoother edges. Kim and Neumann (2002) showed that the aliasing artifacts can be reduced by increasing the number of samples used to determine the depth of a pixel. However, this solution comes with a higher rendering cost.

Another approach is alpha blending (Sintorn & Assarsson, 2009). This technique involves blending the hair fragments to achieve approximate antialiasing effects. This approach will be explained further later on in this chapter (Chapter 2.4.4). Martin *et al.* (2018) proposed an antialiasing technique that is cheaper and suitable for rendering hair in real time. The first step involves calculating the percentage of a pixel that is covered by a strand. This percentage value obtained is then used to modify the alpha value of the pixel, resulting in increased transparency towards the edges of the hair strands, thus removing noise at the edges of the hair strands.

b) Volume-Based Technique

Kajiya and Kay (1989) introduced a volumetric approach that uses texture elements, also known as texels. Texels are 3D data structures containing frame and lighting information.

Rays are traced from the camera through the texels during the rendering process, contributing to the pixel's final colour. This approach resulted in high-quality renderings of fur as shown in Figure 2.1. Kajiya and Kay showed that their method was suitable for short and thick fur. However, Lansink (2010) later stated that Kajiya and Kay's approach was less successful for longer and finer hair strands as the cost of ray tracing became significantly more expensive to compute the hair geometry details from inside the hair volume. Collision detection among individual hair strands is also computationally intensive, which is not feasible for an interactive system. Lee *et al.* (2000) addressed this issue using a technique that increased the volume of hair without accounting for accurate collisions between individual strands. A study conducted by Choe *et al.* (2005) examines the hair density in space as a measure of collisions. When the density of a certain area in space exceeds a certain threshold, collision is identified, and the hair in that specific area is required to fill a larger volume.

Volumetric aliasing issues were solved using pre-filtered shading features. If hair animation is involved, the features of texels need to be updated for each frame, which outweighs the advantages of pre-filtering, thus making volume-based techniques costly and not suitable to be implemented in real-time applications.

Another implicit technique, introduced by (Yang *et al.*, 2000), uses volume density functions to approximate high-frequency details and generate anti-aliased hair clusters, treated as a single polygon. This makes the individual strands in the hair clusters lack detail and may look like a wig (refer Figure 1.1) as they are always stuck together because of the fixed density functions. (Wang *et al.*, 2022) constructed the hair volume using a set of guide hairs. Author chooses a subset of strands to be simulated and rendered. The selected subset is denoted as "guides" as they act as a reference to direct the pipeline in generating additional hair using interpolation to cover the remaining volume at a later stage.

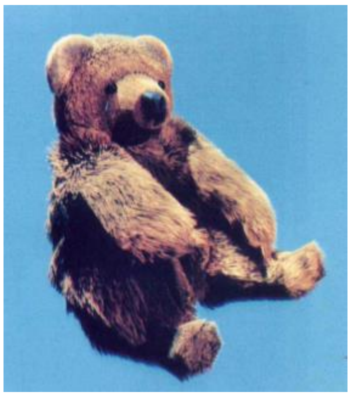


Figure 2.1 A furry teddy bear rendered using ray tracing through Kajiya-Kay's texels. Reprinted from "Rendering fur with three dimensional textures", (Kajiya & Kay, 1989).

c) Hybrid Technique

A study by Andersen *et al.* (2016) combined both strand and volume-based techniques to render fur. Hair fibres can be divided into two types, which are undercoat and guard hairs. The undercoat layers of hair strands are dense and thin, thus making it inappropriate to be represented using volumized models while explicit models are inappropriate to represent coarse guard layers. They successfully produced high-quality fur renders using their proposed algorithm, as shown in Figure 2.2. However, their results, shown in Table 2.1, showed that their algorithm is expensive. Their hybrid technique takes a few seconds to render a single frame, thus making their approach not well-suited to render hair in real time.

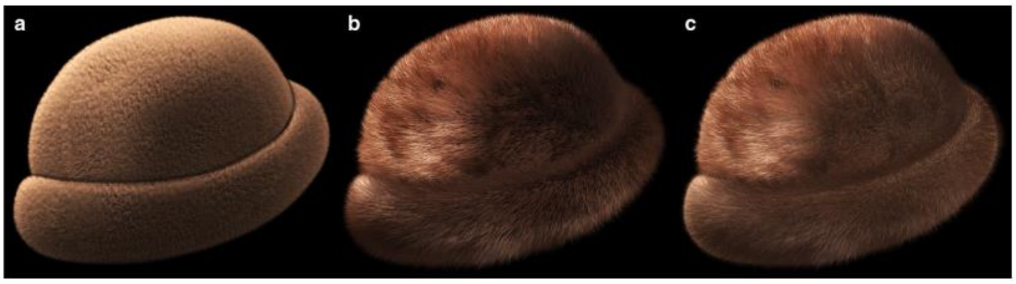


Figure 2.2 Comparison between renders of volume-based fur (left), strand-based fur (centre) and hybrid fur (right). Reprinted from "Hybrid fur rendering: combining volumetric fur with explicit hair strands", (Andersen et al., 2016).

Table 2.1 Total memory consumption and time taken to render a single-frame image of a furry cylinder, hat, and Stanford Bunny model.

	Cylinder	**Hat**	**Bunny**
Triangle count	3584	5632	69451
Total memory, GB	4.49	4.62	4.24
Total render time, ms	3813	4490	3476

Note: Retrieved from "Hybrid fur rendering: combining volumetric fur with explicit hair strands", (Andersen *et al.*, 2016).

2.4.2 Illumination

Apart from hair geometry representation, considering the optical properties of each individual hair fibre is important to achieve higher quality renders of each individual hair strand (Marschner *et al.,* 2003) and the interactions between each hair strand (Petrovic *et al.*, 2005).

In the field of computer graphics, an illumination model is used to compute the final colour of any point on an illuminated object in a scene (Currius, 2022). Hair illumination can be divided into two categories, local and global illumination. Local illumination properties focus solely on how an individual hair strand is illuminated while global properties involve the direct illumination from surrounding light sources as well as the indirect illumination produced from other hair surfaces.

There are two popular local illumination models that have been used in recent hair-rendering models such as (Jansson *et al.*, 2019), which are the Kajiya-Kay model, introduced by Kajiya and Kay (1989), and the Marschner model, introduced by Marschner et al. (2003). The following sub-sections will provide a comprehensive explanation of these models.

a) **Kajiya-Kay Model** (Kajiya & Kay, 1989)

Kajiya and Kay (1989) introduced the first illumination model that simulated light scattering to render hairy and furry models. In this study, they designed this technique to render a furry teddy bear model. Their technique involved categorising each hair strand into two lighting components, which are the diffuse and specular components. Diffuse reflection component estimates the light interaction on a surface by assuming incident ray is reflected from the surface in all directions while specular reflection component assumes incident ray is reflected from the surface in only one direction. Kajiya and Kay then used the sum of the two components to obtain the overall rendering equation for their scene.

Kajiya and Kay's diffuse component, $\Psi_{diffuse}$ is based on the physics principle, Lambert's Law (Lambert, 1760), which describes light behaviour on an ideal matte surface using the cross-product expression below.

$$k_d(\boldsymbol{l} \cdot \boldsymbol{n}) \tag{2.1}$$

where:

k_d is the diffuse reflection coefficient.

$\boldsymbol{l}$ is the light vector.

$\boldsymbol{n}$ is the surface normal vector.

The orthonormal basis (refer Figure 2.3) is defined by three vectors: the hair tangent, $\boldsymbol{t}$, the projection of the light direction on the normal plane, $\boldsymbol{l}'$, and an orthogonal vector, $\boldsymbol{b}$ where:

$$\begin{aligned} \boldsymbol{l}' &= \frac{\boldsymbol{l} - (\boldsymbol{t} \cdot \boldsymbol{l})\boldsymbol{t}}{||\boldsymbol{l} - (\boldsymbol{t} \cdot \boldsymbol{l})\boldsymbol{t}||} \\ \boldsymbol{b} &= \boldsymbol{l} \times \boldsymbol{t} \\ \boldsymbol{n} &= \boldsymbol{b}(cos\theta) + \boldsymbol{l}'(sin\theta) \end{aligned} \tag{2.2}$$

where:

θ is a position along the cylinder, where $0 \leq \theta \leq \pi$

$\Psi_{diffuse}$ was derived by applying Lambert's surface model to a light-facing cylinder through integration as shown in Figure 2.4 and the following equation is obtained.

$$\begin{aligned} \Psi_{diffuse} &= k_d \int_0^{\pi} (\boldsymbol{l} \cdot \boldsymbol{n})\, rd\theta \\ &= k_d rd\theta \\ &= k_d r\boldsymbol{l} \cdot \boldsymbol{l}' \int_0^{\pi} sin\theta\, d\theta \\ &= K_d \boldsymbol{l} \cdot \boldsymbol{l}' \\ &= K_d \boldsymbol{l} \cdot \frac{\boldsymbol{l} - (\boldsymbol{t} \cdot \boldsymbol{l})\boldsymbol{t}}{||\boldsymbol{l} - (\boldsymbol{t} \cdot \boldsymbol{l})\boldsymbol{t}||} \\ &= K_d \frac{1 - (\boldsymbol{t} \cdot \boldsymbol{l})^2}{\sqrt{1 - (\boldsymbol{t} \cdot \boldsymbol{l})^2}} \\ &= K_d(\sin(\boldsymbol{t}, \boldsymbol{l})) \end{aligned} \tag{2.3}$$

where:

K_d is the diffuse scalar coefficient that represents all quantities of $\boldsymbol{l}$ and $\boldsymbol{l}'$.

Next, Kajiya and Kay's specular component, $\Psi_{specular}$ was derived from Phong's specular-reflection model (Phong, 1975). Kajiya and Kay stated that the hair specular reflection follows a mirror angle along the tangent, regardless of the horizontal component of the observer's viewing direction. The hair strands' normals pointed in all the directions perpendicular to the tangent and as a result, the reflected light formed a cone-shaped pattern that opened at the same angle as the incident light strikes the hair surface as shown in Figure 2.5. The specular component is given by:

$$\begin{aligned} \Psi_{specular} &= k_s \cos^p(\theta - \theta') \\ &= k_s(\cos\theta\cos\theta' + \sin\theta\sin\theta')^p \\ &= k_s(\cos\theta\cos\theta' + \sin\theta\sin\theta')^p \end{aligned} \tag{2.4}$$

where:

k_s is the specular reflection coefficient.

θ is the angle of incidence.

θ' is the angle of reflection.

p is the Phong specular exponent coefficient.

Kajiya-Kay model is widely used as the fundamental for many hair rendering models such as (Marschner *et al.,* 2003; Jansson *et al.*, 2019) due to the model's simplicity (Marschner *et al.,* 2003). This model has also contributed to the field of computer graphics, including its application in the game Ryse (Schulz, 2014).

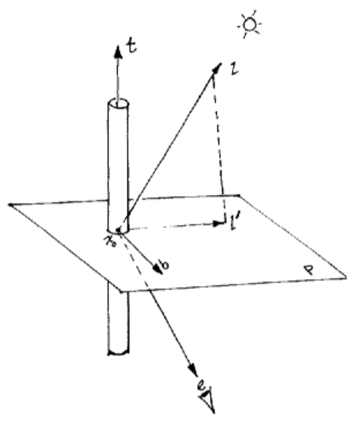


Figure 2.3 Orthonormal basis of diffuse model. Reprinted from "Rendering fur with three dimensional textures", by (Kajiya & Kay, 1989).

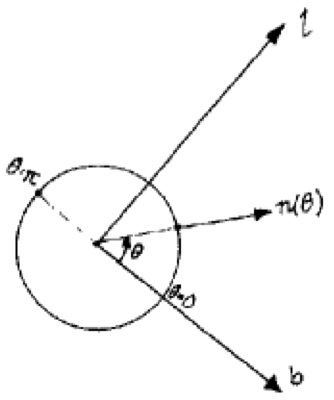


Figure 2.4 Integration along hemisphere. Reprinted from "Rendering fur with three dimensional textures", (Kajiya & Kay, 1989).

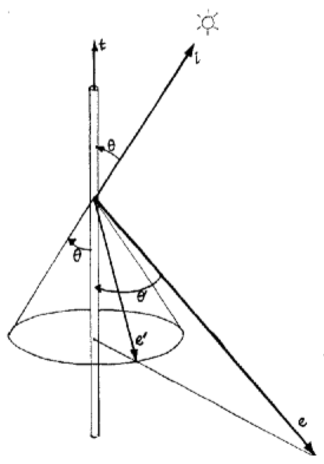


Figure 2.5 Kajiya and Kay's illumination model. Reprinted from "Rendering fur with three dimensional textures", (Kajiya & Kay, 1989).

b) Marschner Model (Marschner *et al.,* 2003)

In 2003, Marschner *et al.* developed a light-scattering model to render hair fibres. This model was improved upon the Kajiya-Kay model (Kajiya & Kay, 1989) where it used a more physically accurate representation of light scattering in hair. The Marschner model incorporated hair strand rotation about its axis to predict variations of reflection of light on hair strands by accounting for the elliptical cross section of hair strand, thus resulting in more accurate hair strand representations.

Marschner *et al.* accounted for the surface roughness of the overlapping cells, β mentioned in Chapter 2.3. Due to these scales or cells, reflected light rays are slightly shifted by a small angle constant, 2α, where the scales are tilted by α. Studies such as (Hunz, 2016) have defined a range of values of β and α as shown below, where both the values in the R lobe is between 5 and 10.

Table 2.2 List of defined values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$

Variable	**Definition**
Longitudinal shift of path R, α_R	$[5,10]$
Longitudinal shift of path TRT, α_{TRT}	$\frac{\alpha_R}{2}$
Longitudinal shift of path TT, α_{TT}	$\frac{3\alpha_R}{2}$
Longitudinal width of path R, β_R	$[5,10]$
Longitudinal width of path TRT, β_{TRT}	$\frac{\beta_R}{2}$
Longitudinal width of path TT, β_{TT}	$2\beta_R$

Note: Retrieved from "Interactive physically-based hair rendering", (Hunz, 2016).

Surface reflection and internal reflection in this lighting model occur in different directions, resulting in different peaks of reflection as seen in Table 2.2 and Figure 2.6. The specular reflection is the main reflection of strand and exhibits the same colour as the incident light. This acts as the primary highlight specular component. The internal reflection is a secondary reflection of the hair and adjusts its colour based on the pigments contained in the hair. This acts as the secondary highlight specular component in this lighting model. These peaks can be seen on real hair as shown in Figure 2.8. The result when accounting for these peaks is known as glints. According to Gray *et al* (1997) and Lemein (2020), the presence of glints are properties found on hair textures when light rays are scattered on the fibre.

Marschner *et al.* accounts for three reflective light components on the hair surface as illustrated in Figure 2.6, where the components are specular reflection (R), internal reflection (TRT) and diffuse reflection (TT). R component refers to the incident light ray that is reflected from the hair surface, TRT component refers to the incident light ray that undergoes transmission and internal reflection, and TT component refers to the strong transmittance component where the incident light ray that is completely transmitted through the hair strand without any reflection. These reflective components provide more accurate hair reflectance and thus allowing the rendered result to have a better representation to real hair. Comparisons between the Marschner model and Kajiya-Kay model can be seen in Figure 2.7. The bidirectional scattering Marschner model offers allows the resulting hair model to be more accurate to real hair than that compared to the Kajiya-Kay model.

However, Marschner *et al.* (2003) stated that their model is significantly more expensive than the Kajiya-Kay model. The high cost made the Marschner model more challenging to be implemented in real-time applications although the Marschner model produced higher quality hair renders.

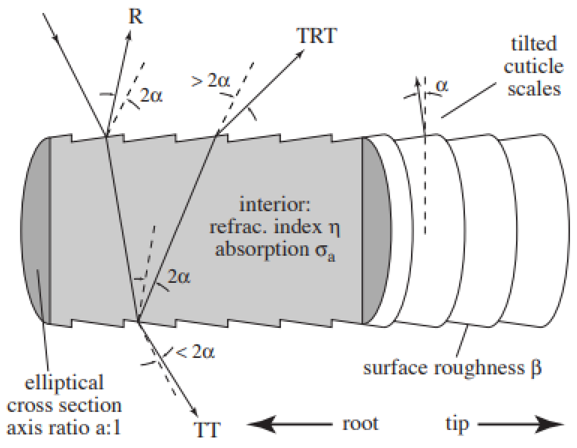

R
TRT
tilted cuticle scales
2α
$>2\alpha$
α
interior: refrac. index η absorption σ_a
2α
$<2\alpha$
surface roughness β
elliptical cross section axis ratio a:1
TT
root
tip

Figure 2.6 Marschner's illumination model. Reprinted from "Light Scattering from Human Hair Fibers", (Marschner et al., 2003).

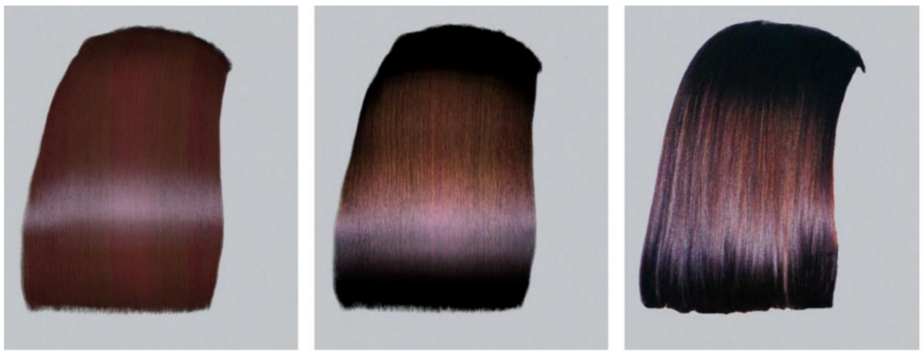


Figure 2.7 Comparison between Kajiya-Kay's hair model (left), Marschner's hair model (centre), and real hair (right). Reprinted from "Light Scattering from Human Hair Fibers", (Marschner et al., 2003).

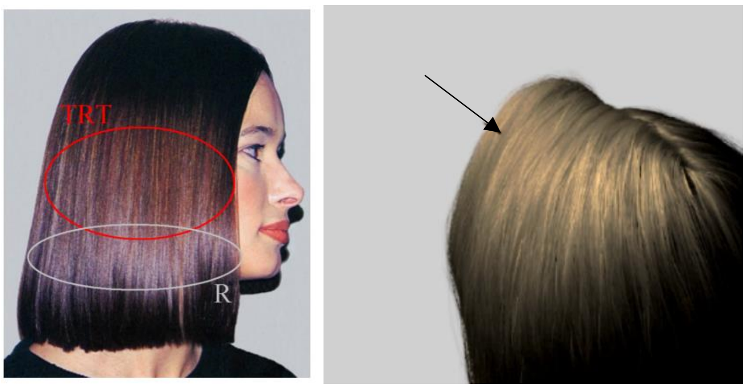


Figure 2.8 Visual representation of primary (R) and secondary highlight (TRT) effects on real hair. Reprinted from "The world of hair: a scientific companion", (Gray *et al.*, 1997) and "An artist friendly hair shading system", (Sadeghi *et al.*, 2010).

Global illumination properties were also implemented in past hair rendering research. Marschner *et al.* (2003) demonstrated that imitating the appearance of real hair requires a physically based hair shading model. Their research showed that the incident light ray undergoes both transmission and reflection on the hair surface. Moon *et al.* (2007) has also stated that volumetric objects such as clouds and translucent materials like hair are highly influenced by multiple light scattering within the volume. Therefore, multiple scattering is influential in determining the hair's appearance. Studies done by (Zinke *et al.*, 2004; Moon & Marschner, 2006) further showed that multiple light scattering is used to create light-coloured human hair, such as blond hair, that have low absorption values. Therefore, inter-reflections among the hair fibres, considering all possible paths which incident light can scatter from one fibre to another before reaching the observer's eye, is important to create realistic hair renders.

There are two global illumination models that are commonly used in hair rendering, which are path tracing and photon mapping. These models will be explained in the following sub-sections.

c) Path Tracing

Path tracing is a rendering technique in computer graphics that aims to compute light behaviour accurately. It extends the concept of ray tracing by sampling light rays from the virtual camera to compute multiple interactions from surrounding objects. Path Tracing had been implemented in multiple research such as in

(Kajiya & Kay, 1989; Gupta & Magnenat-Thalmann, 2005 and Yuksel *et al.*, 2007) to achieve realistic hair illumination models. Path tracing models also simulate effects, such as soft shadows, depth of field, ambient occlusion, and caustics, to be added into other methods within the algorithm (Jensen & Christensen; 2007). However, to obtain high-quality images with path tracing, many rays must be traced to minimize visible artifacts in the form of noise. This makes path tracing models very slow (Currius, 2022) and inappropriate to render hair in real-time despite being the simplest and most physically accurate.

The most accurate global illumination model to render hair is the Monte-Carlo path tracing model (Kajiya, 1986; Moon *et al.*, 2008). However, this model, like path tracing algorithms in general, is not suitable to be implemented in real-time applications as it is very expensive, required precomputation (Ou *et al.*, 2012), and has a slow convergence rate (Yu *et al.*, 2012).

d) Photon Mapping

In computer graphics, photon mapping is a two-pass rendering algorithm (Kang *et al.*, 2016) that involves tracing incident rays from the light source and the camera independently until a termination condition is met. These rays are connected to obtain the radiance value, allowing for accurate simulation of light interactions such as light refraction through transparent substances, diffusing inter-reflection between illuminated objects, and scattering in translucent materials.

Moon and Marschner (2006), and Zinke *et al.* (2008) are some of the researchers that applied Photon Mapping in their hair illumination study. Moon and Marschner dealt with light transport in volumes containing scattering objects. Unlike traditional approaches, this method traces paths directly through the hair volume, resulting in the need for a memory-intensive 5D photon map to handle and store the multiple scattered radiance.

Moon *et al.* (2008) further enhanced Moon and Marshner's technique for hair rendering by introducing a 3D grid of spherical harmonic coefficients to store scattered radiance distribution that depends on position and direction. This technique offers improved memory organization and compactness compared to traditional photon maps. Their implementation also outperformed Moon and

Marschner's approach by a factor of 20 in terms of performance while maintaining equivalent rendering quality.

Therefore, photon mapping has emerged as a successful technique for rendering hair. This method allows for the realistic approximation of multiple scattering in hair volumes through density approximation within the geometry. However, these approaches still require a significant amount of memory and can take several hours to compute just a single frame (Yu *et al.*, 2012).

2.4.3 Shadowing

Shadows are an important property of volumetric objects such as clouds and hair (Lokovic & Veach, 2000; Hunz, 2016). Sintorn and Assarsson (2008) have also stated that self-shadowing properties are influential in ensuring the rendering model produces realistic results (refer Figure 2.9). However, implementing self-shadowing properties on volumetric objects has been a challenge (Kim & Neumann, 2001).

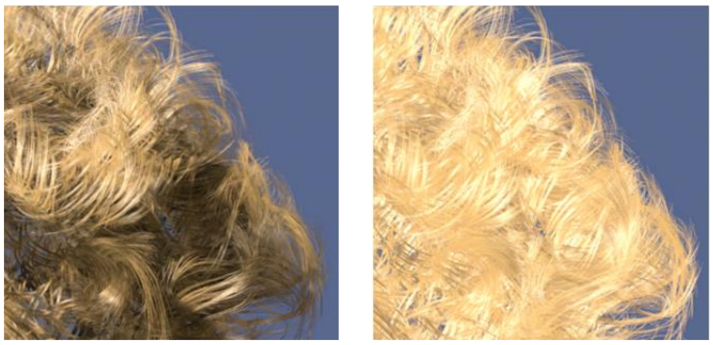


Figure 2.9 Comparison between hair models with self-shadowing (left) and without hair shadowing (right). Reprinted from "Deep shadow maps", (Lokovic & Veach, 2000).

Ray tracing is a technique where a ray is projected from a specific point in a scene towards a light source. As the ray traverses the scene, shadows are computed for the rendering process if it intersects with any objects along its path. Kajiya and Kay (1989) did not consider self-shadowing properties in their hair strand's diffuse component, unlike the Marschner model that used ray tracing for shadowing computations. This made Kajiya and Kay's hair model look flat while Marschner's hair model had more depth and a more accurate representation to real hair as shown in Figure 2.7. However, LeBlanc *et al.* (1991) stated that ray tracing produces accurate shadows but is expensive to be implemented in real-time hair rendering applications.

A common technique to calculate shadows in computer graphics is by using shadow maps which was introduced in (Williams, 1978) to cast shadows on curved surfaces. This technique initially involved a camera, placed at the origin of the light source, that captures an array of pixels that represent the depth of the nearest visible surfaces. Each point in the observer's view is then transformed to the coordinate system of the camera and its z-value is compared to the depth value stored in the array to determine if the point is in shadow.

However, this method required high resolutions to obtain accurate shadows and a larger number of depth samples during shadow lookups to make up for the high frequencies present in the shadow map. This algorithm is not suitable to be tested on translucent objects such as fur and hair as this method determines whether an object receives light using a binary decision (Williams, 1978). Therefore, these limitations make this traditional technique inappropriate to be used for hair rendering.

Years later, researchers have come up with different variations of shadow maps for hair rendering, which will be explained more in the following sub-sections.

a) Deep Shadow Map

Lokovic and Veach (2000) proposed a method using deep shadow maps to further improve the traditional depth-based shadow map technique by Williams (1978). Their goal was to cast shadows on translucent objects while overcoming aliasing issues in offline rendering.

Deep shadow maps differ from depth-based shadow maps where a piecewise linear approximation of visibility function, a function of transmittance, is stored for each pixel instead of their depth values to approximate the amount of light that passes through the hair fibres. This allows deep shadow maps to compute more accurate shadows than depth-based shadow maps. The equation of transmittance, $\tau(z)$ of light to any depth value, z within the hair volume is:

$$\tau(z) = \exp(-\Omega), \text{ where } \Omega = \int_0^z \sigma_t(z')dz' \tag{2.5}$$

where:

Ω is the opacity at a given point.

σ_t is the extinction function which is calculated by sampling the atmospheric density at time intervals, t along the light ray.

z' is the new depth value.

The transmittance function (equation 2.5) takes two hair properties into consideration. The first property is its visibility. When there are a greater number of hairs observed between the light source and the viewer, the light becomes more occluded, leading to less illumination and shadows. This can cause variations in visibility within a single pixel. Equation 2.5 ensures that the effects of occlusion are appropriately accounted for. The second property is transparency. As hair has translucent properties as stated in (Ward *et al.*, 2007; Sintorn & Assarsson, 2008), hair fibre absorbs, scatters, and transmits incoming light (Marschner *et al.,* 2003). Equation 2.5 also handles the translucency of hair fibres, given that the light rays are transmitted in a forward direction only.

The limitation to this method is that the initialisation of the shadow maps takes a long time and is expensive, and thus deep shadow maps are not suitable to be used in real-time applications despite being able to produce high quality shadows when rendering hair (Yu *et al.*, 2012).

b) Opacity Shadow Map

Kim and Neumann (2001) presented an alternative method to compute the transmittance function and self-shadowing using opacity shadow maps. This method uses parallel opacity shadow maps that are perpendicular to the incident light ray. These shadow maps contain line integral densities from the light source to each pixel instead of visibility function that was used in deep shadow maps. The opacity function, Ω values are sampled and stored in an array for the hair rendering process.

The hair volume is divided into multiple slices, and the hair geometry is rendered for each slice subsequently. The far-plane is then adjusted for each slice, moving it further back by the width of that slice. During the rendering process, the hair is given a constant colour and additive blending is used to determine the opacity at each slice, which can then be interpolated to approximate the opacity at any depth, l. This process can be represented as an illustration as shown in Figure 2.10. The opacity shadow maps are then used to compute the amount of light that passes through the hair fibres and determine the hair strands that are in the shadows.

Kim and Neumann stated that the number of shadow maps used influenced the quality of hair renders as layering artifacts were visible at brighter regions when a smaller number of maps were used. These artifacts were generated due to opacity interpolation between the hair layers. Nguyen and Donelly (2005) used 16 slices of opacity maps in real-time. Yuksel and Keyser (2008) have shown 16 slices is insufficient to render quality hair models due to the presence of artifacts as shown in Figure 2.12. Besides, each slice was encoded in one colour channel of one of the four Multiple Render Targets. This allowed for rendering the opacity of each slice in a single pass. However, to obtain the opacity for a specific fragment, the opacity values of all slices need to be summed in the fragment shader which could become a bottleneck, thus affecting the performance as more slices are used.

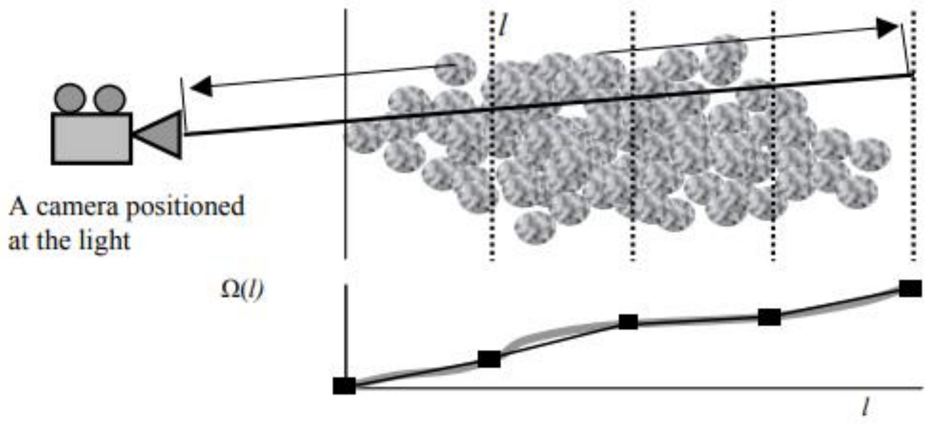


Figure 2.10 The opacity function $\Omega(l)$ represented as the grey curve is approximated using a set of parallel opacity maps. Reprinted from "Opacity shadow maps", (Kim & Neumann, 2001).

c) Deep Opacity Maps

Yuksel and Keyser (2008) proposed a technique that overcomes the limitations of opacity shadow maps. Their goal was to reduce the generation of layering artifacts produced by the shadow maps and compute detailed self-shadowing while using a smaller number of layers during the rendering process. The layering of opacity maps in this model are curved and follows the shape of the head model compared to the parallel ones used by Kim and Neumann (2001). This helps the model approximate the geometry of the hair model more accurately. Deep opacity maps also use two pass rendering algorithms, like opacity shadow maps, to compute the shadows of hair strands. The model then uses another render pass to render the scene with shadows.

First, the opacity layers are initiated by generating shadow maps from the perspective of the light source. This map determines the starting point of the hair geometry. The hair volume is then divided into successive layers that progressively move away from the light source (refer Figure 2.11). The layers are aligned along the hair structure by using the shadow map as a guide.

The shadow maps that were generated previously are then used to render the opacity maps using additive blending. The initial depth value is obtained from the shadow map and used to determine the depth values of each layer. The total opacity at any pixel is then determined by summing all the hair fragments that are within that layer and the layers behind it. Using a similar technique as in (Nguyen and Donelly, 2005), opacity maps can store up to three opacity layers using an RGBA texture. By using Multiple Render Targets, model can compute and store multiple layers simultaneously.

Lastly, the deep opacity maps are used during the rendering process to compute the shadows on the hair model. This technique computes the amount of light transmitting through the corresponding layer and thus the amount of shadow is on the hair, providing self-shadowing visual effect. An illustration of deep opacity maps can be seen in Figure 2.11, where the green area behind the last layer represents the region in total shadow. Figure 2.12 shows that deep opacity maps have rendered hair with a smaller number of layers used within the hair volume to compute shadows while improving the performance of the rendering process. The issue with visible layering artifacts has also been reduced as interpolation of opacities now occurs within the hair volume, thus hiding the layering artifacts.

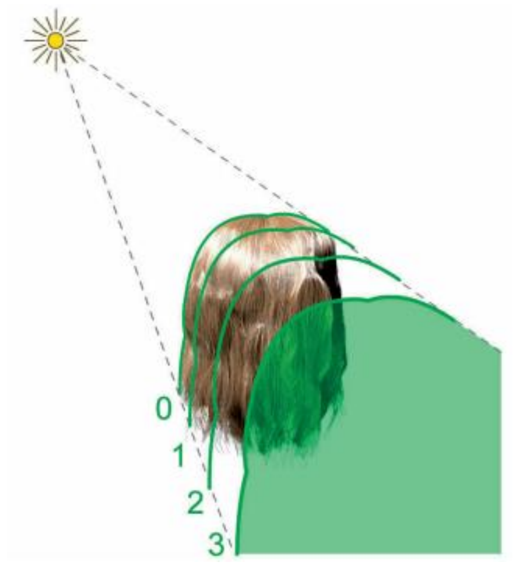


Figure 2.11 Hair model that applies deep opacity maps. Reprinted from "Deep Opacity Maps", (Yuksel & Keyser, 2008).

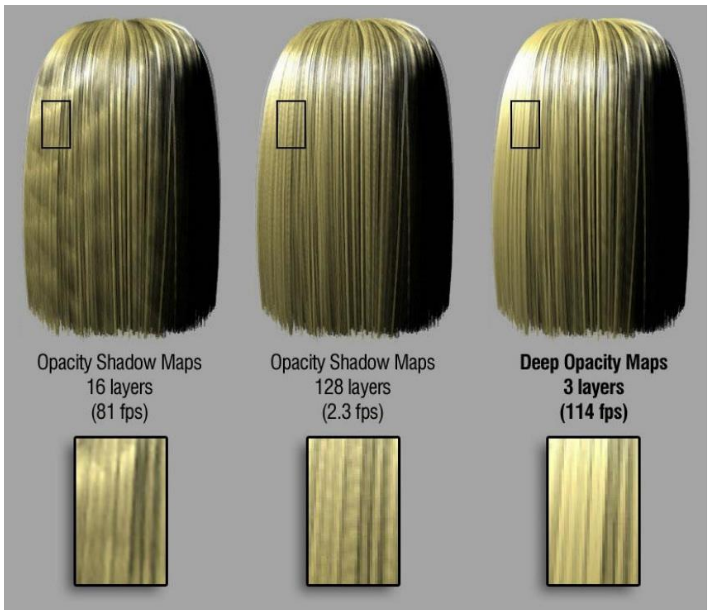


Figure 2.12 Comparisons between the quality of hair models rendered with Opacity Shadow Maps and Deep Opacity Maps. Reprinted from "Deep Opacity Maps", (Yuksel & Keyser, 2008).

2.4.4 Transparency

Transparency of hair strands is an important aspect of producing quality hair renders (Ward *et al.*, 2007). Sintorn and Assarsson (2008) have stated that hair strands are semi-transparent, which contributes to their visual appeal. Light transmission through the hair fibre is responsible for hair's translucent appearance (Yu *et al.*, 2012). Figure 2.13 shows that completely opaque hair models look rough, dull, and aliased.

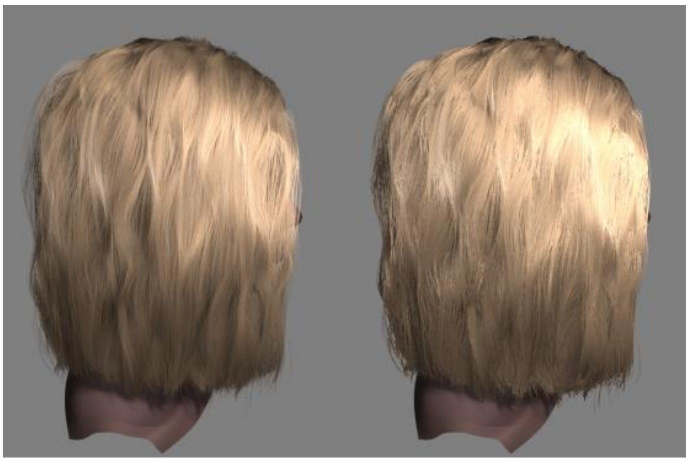


Figure 2.13 Comparison between translucent hair (left) and opaque hair (right) renders. Reprinted from "Interactive physically-based hair rendering.", (Hunz, 2016).

Alpha blending approach is widely applied in real-time computer graphics to compute and simulate translucent appearances when rendering sub-pixel geometry (Hunz, 2016). This technique renders geometry, usually, from back to front with low transparency. Appropriate alpha blending is required to accurately capture the transparency of hair, particularly lesser pigmented hair such as blonde due to their high transparent properties (Zinke *et al.*, 2004). However, multisampling techniques are also required during the rendering process as hair fibres are very thin to ensure the rendered output does not look too thick and visible aliasing artifacts are reduced. Alpha blending along with multisampling techniques are very expensive. Besides that, sorting the geometry by depth along the view direction is not well-suited for complex real-time applications like hair rendering as stated in (Sintorn & Assarsson, 2008; Hunz, 2016). Highly overlapping hair strands also make it hard for all the strands to be sorted. This motivated researchers to come up with an alternative for alpha blending.

A very popular approach over the years is the depth peeling algorithm, which was described in Everitt (2001) and Mammen (1989). This approach is an order-independent technique (OIT), meaning it computes translucent properties during the rendering process and sorting is done per fragment instead of the scene. The scene is rendered multiple times, with each pass using the depth buffer obtained from the previous pass for depth testing. Everitt suggested rendering 2-3 layers are used for this technique while Leshonkov and Frolov (2021) suggested there is only very little visual improvement to the final image after the second pass. Only the foremost fragment is stored during this process. This approach allowed sorting the pixels per fragment, allowing for correct occlusion when rendering the final image of scene.

However, depth peeling is expensive to render as it requires multiple passes to peel the layers of the transparent geometry. An extension to this technique, dual depth peeling (Bavoil & Myers, 2008) was introduced to render semi-transparent geometries in a single pass. However, it is not suitable for rendering hair as technique peels both front and back layers at the same time. Next, single-pass OIT techniques such as A-Buffer (Carpenter, 1984) and k-Buffer (Bavoil *et al.*, 2007) facilitate the rendering of several fragment, accounting for effects such as transparency and volume rendering, and sorting them using a single geometry pass. This resulted in improved performances compared to depth peeling when rendering a single frame. However, these techniques allocate memory size to store the fragments at the start of the system. This makes these single-pass techniques difficult to be implemented in real-time applications as dynamic memory allocation or tile sizing are required to overcome challenges of running out of memory when storing the fragments in real time. These techniques are still difficult to be implemented, even in modern GPU's (Enderton *et al.*, 2010).

2.5 Benchmarking

In the past, Elgenmann (2001) introduced a technique to analyse applications based on their performance by benchmarking their computation time and frames rates. Computation time is tested by using clock functions within computer libraries to obtain the execution time while frame rates are tested to know the number of frames is displayed per second. This performance testing technique was also used in studies such as in (Andersen *et al.*, 2016) to ensure their hair-rendering algorithms can run in real time. NVIDIA's Nsight Graphics tool (O'Reilly, 2022) is one of the tools used in past hair-rendering studies that allows frame profiling to measure the time taken for function calls. From benchmarking the computation time and frame rates, bottleneck of the algorithm can also be identified for further development.

Jansson *et al.* (2019) benchmarked hair properties such as self-shadowing and transparency through result's visual comparison. Graphical user interface (GUI) was used in those studies to enable and disable hair properties along with its values to compare the quality of hair models in their studies. The results obtained is able to show the importance of hair properties to render high-quality hair models.

2.6 Summary and Discussion

Hair representation is an important step as it influences the choices of the other hair-rendering techniques. Strand-based techniques are widely used in many studies and real-time applications such as the Frostbite game engine (Tafuri, 2019). This is because explicit hair models allow for detailed individual hair strands during the rendering process. However, representing hair as individual strands alone is not enough to render high quality hair.

Hair is proven to be a very complex material as there are many properties that should be considered when rendering hair models. These hair properties include illumination, self-shadowing, and transparency. There have been many techniques proposed by researchers in past years to handle these complex properties. However, even for the standards of today's available technology, these proposed techniques can be expensive due to the large number of hair strands required to be rendered in the scene. Thus, further research is required to further improve the quality and performance of hair-rendering algorithms.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter reviews the methods used to render explicit hair strands in real time using strand-based techniques. The project framework is created to manage the flow of the project. It is divided into two phases, which are the research and development phases. The system architecture of this project is designed upon the project's objectives proposed in Chapter 1 and research that was done in Chapter 2.

3.2 Project Framework

A framework is constructed for the hair rendering prototype before the beginning the implementation process when creating the prototype. The framework is divided into two phases, research phase and development phase.

Research phase covers Preliminary Research, Literature Review and Framework Design. Preliminary Research in Chapter 1 is to identify the project's problem background and statement, aim, objectives, and scopes. Literature Review in Chapter 2 is done to study the knowledge found by past researchers and understand the techniques implemented in their research. Framework Design in Chapter 3 is carried out after determining the technique applied to render hair.

In the development phase, technique implementation in Chapter 4 is where the selected method will be implemented. Then, the evaluation stage is where results and performance of prototype are evaluated. Lastly, the outcome of the project is concluded, and future work is discussed in Chapter 6. Figure 3.1 shows the constructed framework of research phase and development phase.

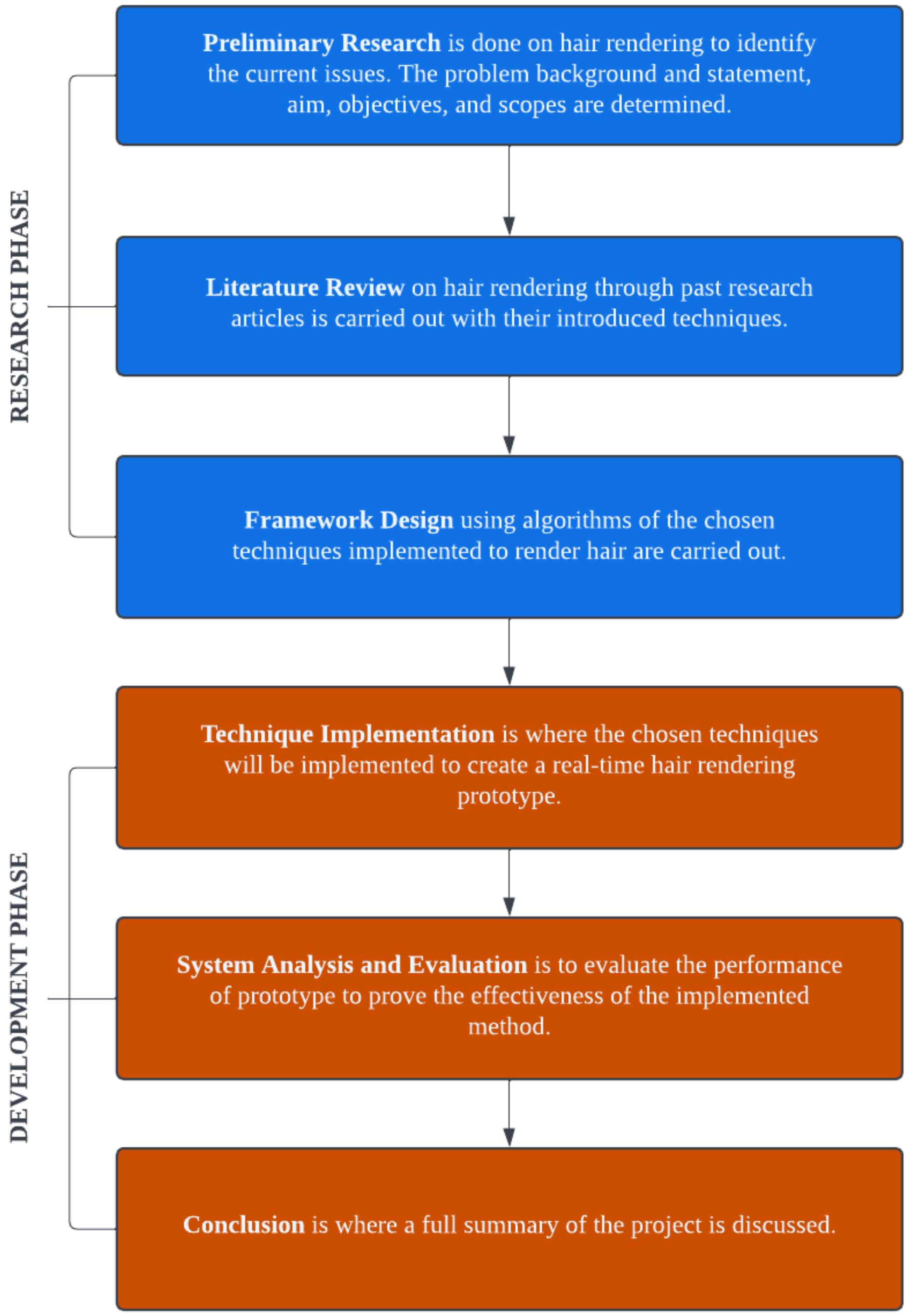

RESEARCH PHASE
Preliminary Research is done on hair rendering to identify the current issues. The problem background and statement, aim, objectives, and scopes are determined.
Literature Review on hair rendering through past research articles is carried out with their introduced techniques.
Framework Design using algorithms of the chosen techniques implemented to render hair are carried out.
DEVELOPMENT PHASE
Technique Implementation is where the chosen techniques will be implemented to create a real-time hair rendering prototype.
System Analysis and Evaluation is to evaluate the performance of prototype to prove the effectiveness of the implemented method.
Conclusion is where a full summary of the project is discussed.

Figure 3.1 Framework of research and development phase.

3.3 System Architecture

The system architecture (refer to Figure 3.2) of this project can be divided into five main components, which are input, asset loading, guide hair placement, shader programs, and depth peeling.

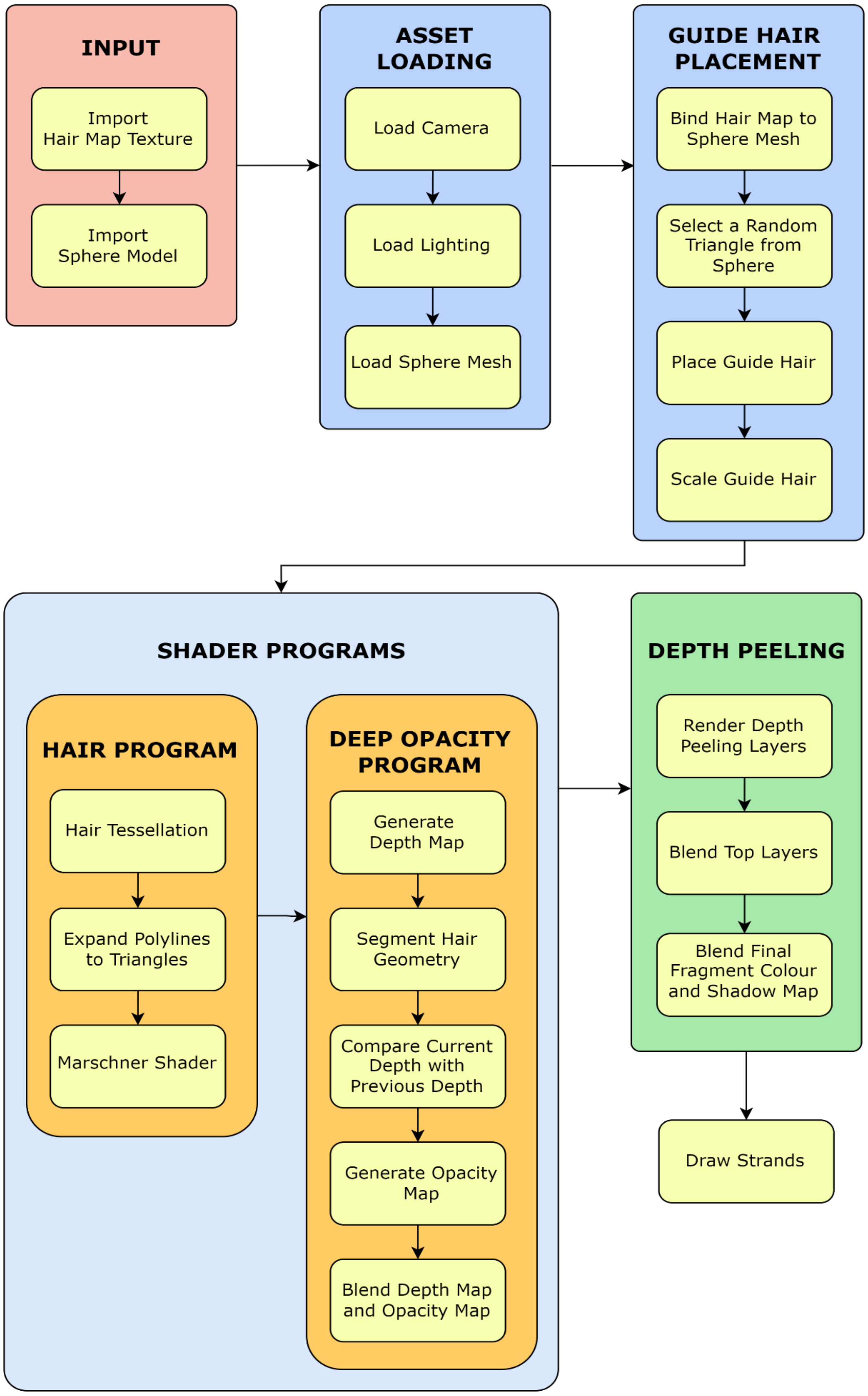


Figure 3.2 Overview of system architecture.

3.3.1 Input and Asset Loading

The system imports a hair map texture image and a three-dimensional (3D) OBJ-format sphere model. These dataset inputs are obtained under MIT License. These datasets can be retrieved from a Github repository published by Emma O'Reilly, https://github.com/elor1/HairRendering/blob/main/models/sphere.obj and https://github.com/elor1/HairRendering/blob/main/images/Noise.jpg. A lighting system and the imported sphere model are loaded into the scene along with a perspective camera looking at the sphere model.

3.3.2 Guide Hair Placement

Since hair strands inherit almost similar properties to one another, (Daldegan *et al.*, 1993; Chang *et al.*, 2002) used guide hairs to model a head of hair. Only guide hairs are fully simulated and the rest of the strands within the guide hair's neighbourhood can be interpolated.

The imported hair map texture is used to determine the placement of guide hair strands on the sphere model. Once the texture is mapped onto the sphere, the algorithm begins by selecting a random triangle on the sphere to place the generated guide hair. A random point within the triangle is chosen to place the guide hair is shown in Figure 3.3. The alpha values are then fetched from the imported hair map's texture coordinates. If the alpha value of the pixel colour is not zero, a hair is placed, and the alpha value is used to scale its length. The algorithm is shown in Figure 3.4.

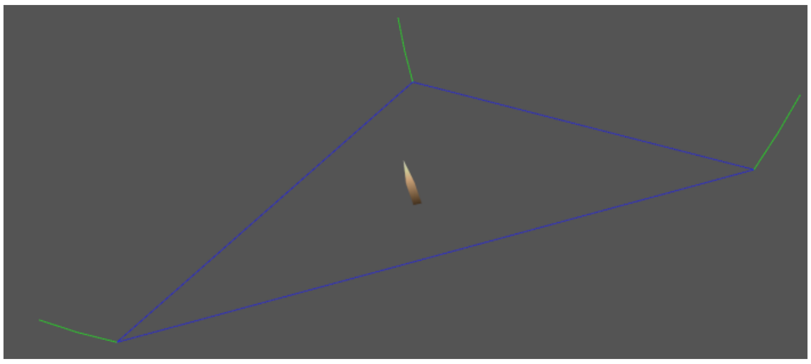


Figure 3.3 Placement of guide hair in triangles on mesh.

Algorithm 1: Guide Hair Placement

```
IN    :    Hair map image, $image$
           Sphere mesh, $mesh$
           Total number of guide hairs, totalGuide
           Maximum hair length, $maxLength$

1   $i \leftarrow 0$
2   while $i < numDispHair$ do
3       $triangle \leftarrow random(availableTriangles)$
4       $randomPoint \leftarrow random(triangle\_x, triangle\_y)$
5       $x \leftarrow min(max(randomPoint_x * image_{width}), 0), image_{width});$
6       $y \leftarrow min(max(randomPoint_y * image_{height}), 0), image_{height});$
7       $pixel \leftarrow image + y * image_{width} * channels + x * channels$
8       $alpha \leftarrow pixel[3]$
9       $alphaVal \leftarrow \frac{alpha}{255}$
10      if $alphaVal < 0.05$ then
11          $i \leftarrow i - 1$
12          continue
13      endif
14      $guideHair \leftarrow (maxLength * alphaVal,$
                    $randomPoint, randomPoint_{normal})$
15      $i \leftarrow i + 1$
16  endwhile
```

Figure 3.4 Algorithm of Hair Placement from Hair Map Texture.

3.3.3 Programs

Program objects are compilations of shaders. In this project, two shader programs will be used to add explicit details, consisting of Hair Program and Deep Opacity Program.

A hair program is used to represent the hair model as individual strands. The program begins by tessellating the line primitives into multiple segments and interpolating them as Catmull-Rom splines to create smooth curves between the knots. The polylines are then expanded into triangle strips as they add more details to the hair surface (Cedermalm, 2019) compared to 2D lines.

The Marschner lighting model is then used to compute light interactions with the hair fibre and obtain the final pixel colour. Further details on hair tessellation and the Marschner model will be explained in the following sub-sections.

a) Hair Tessellation and Triangle Expansion

Tessellation is applied to the hair lines to generate more hair strands through interpolation using a single guide hair. More hair strands can be generated through a single GPU pass using this technique as shown in Figure 3.5.

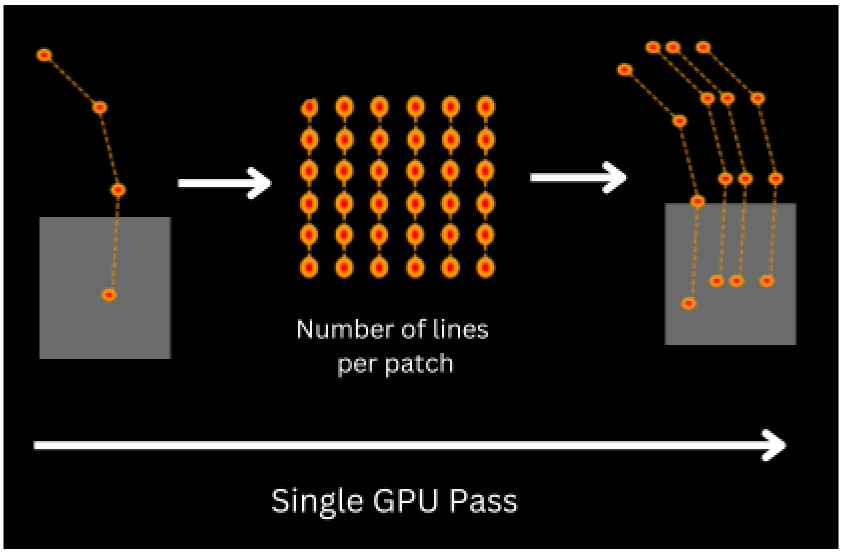


Figure 3.5 Illustration of guide hair interpolation.

This technique begins by generating guide hair patches of the guide hairs. The hair strands are then generated using Catmull-Rom spline interpolation (Pipenbrinck, 2013) as shown in Figure 3.6. The start and end points along with their respective tangents are required to compute the curve, $f(t)$. Tangents at any point iteration, i can be obtained using the previous and next vertex iteration as shown in Equation (3.2 where the tightness of hair strand, a is set to 0.5.

$$\mathrm{f(t)} = (2t^3 - 3t^2 + 1)P_0 + (t^3 - 2t^2 + t)T_0 + (-2t^3 + 3t^2)P_1 + (t^3 - t^2)T_1 \qquad (3.1)$$

where:

t is the current position in the tangent, where $t \in [0,1]$.

P_0 is the starting point of curve.

P_1 is the ending point of curve.

T_0 is the tangent vector at P_0.

T_1 is the tangent vector at P_1.

$$\mathrm{T_i} = \mathrm{a}(\mathrm{P}_{i+1} - P_{i-1}) \tag{3.2}$$

The added vertices are then used as knots to interpolate smooth curves using cubic B-spline interpolation (Medioni & Yasumoto; 1987) along the length of the hair strands.

The algorithm begins by clamping the x-tessellation coordinate and multiplying with the number of hair segments. The length is then used to determine the control points that are used for interpolation. The algorithm of hair tessellation is shown in Figure 3.4 below. The generated line segments generated from the tessellation process are then expanded into triangles.

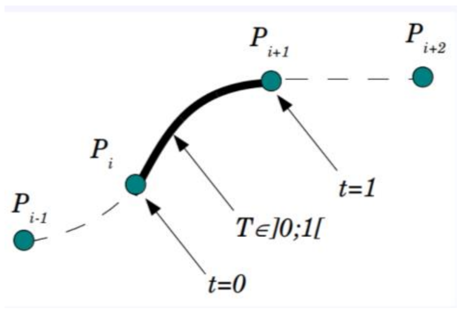


Figure 3.6 Illustration of Catmull-Rom spline interpolation.

Algorithm 2: Catmull-Rom Spline Interpolation

IN : Position within the hair patch, $tessCoordX$

OUT : New interpolated position

```
1    $length \leftarrow tessCoordX * numHairSegments$
2    $fractional \leftarrow length - floor(length)$
3    $index1 \leftarrow length$
4    $index0 \leftarrow \max\{index_1 - 1, 0\}$
5    $index2 \leftarrow \min\{index_1 + 1, \mathrm{numHairSegments}\}$
6    $index3 \leftarrow \min\{index_2 + 1, \mathrm{numHairSegments}\}$
7    $m_1 \leftarrow \frac{vertexData[index_2] - vertexData[index_0]}{2}$
8    $m_2 \leftarrow \frac{vertexData[index_1] - vertexData[index_3]}{2}$
9    $a \leftarrow vertexData[index1] + m1 * fractional$
10   $b \leftarrow vertexData[index2] + m2 * (1 - fractional)$
```

```
11        $c \leftarrow smoothstep(0, 1)$
12        **return** $a * (1 - c) + b * c$
```
Figure 3.7 Algorithm of Hair Spline Tessellation

b) Marschner Shader

Illumination properties are required to account for light interaction on the surface of the hair fibres and the final colour of the pixel in the rendered scene (Marschner *et al.,* 2003; Currius, 2022). In this project, only local illumination is involved as the cost for global illumination application is too high to be implemented in real time (Ou *et al.*, 2012; Yu *et al.*, 2012). The Marschner bidirectional light-scattering model (Marschner *et al.,* 2003) is chosen to compute due to the model being a more accurate model towards real hair (refer 2.4.2b)) than the Kajiya-Kay model (Kajiya & Kay, 1989). The Marschner model can compute two specular reflection peaks whereas Kajiya-Kay model only accounts for one, as Marschner model accounts for both surface and internal reflections. These properties are important when shading hair strands as hair is translucent, thus inheriting translucent properties. The structure of the Marschner model has been discussed in 2.4.2b).

The ambient component computes the effect of hair-hair interaction on indirect illumination, specifically in situations where the hairstyle is not directly exposed to light.

The specular component, $\Psi_{specular}$, of the Marschner shader consists of three lighting components, R, TRT, and TT, where R stands for reflection and T stands for Transmission. R component is the primary specular peak where light is reflected off the hair fibre as soon as the ray touches the surface. TRT component is the secondary specular peak, where this component undergoes internal reflection. The light ray transmits through the fibre, undergoes scattering, and exits from the same side it entered. The TT component is the specular component to compute the surface colour when light ray completely transmits and exits the hair fibre due to forward scattering. These components are dependent on the angle of tilted cuticle scales, α, which is responsible for the shifting of glints and the surface roughness, β, that increases the width of glints. These components are computed based on Phong reflection model (Blinn, 1977). All the specular components are then added together.

The diffuse lighting component is responsible for the light that scatters throughout the hair and reflects in all directions. The diffuse reflection is determined by the non-negative dot product between the hair normal and the incident light vector.

The final rendering equation, Ψ is obtained by computing the sum of the three components as shown below.

$$\Psi_{ambient} = k_a(C) \quad (3.3)$$

$$\Psi_R = \cos(\theta - \alpha_R)^{\beta_R} \quad (3.4)$$

$$\Psi_{TRT} = \cos(\theta - \alpha_{TRT})^{\beta_{TRT}} \quad (3.5)$$

$$\Psi_{TT} = \cos\emptyset \cos(\theta - \alpha_{TT})^{\beta_{TT}} \quad (3.6)$$

$$\Psi_{specular} = k_s(\Psi_R + \Psi_{TRT} + \Psi_{TT}) \quad (3.7)$$

$$\Psi_{diffuse} = k_d(\boldsymbol{l} \cdot \boldsymbol{n}) \quad (3.8)$$

$$\Psi = \Psi_{ambient} + \Psi_{diffuse} + \Psi_{specular} \quad (3.9)$$

where:

C is the base colour of hair strand.

$\emptyset$ is the rotation of hair strand around its axis.

The parameter values of longitudinal shifts and longitudinal widths for each lobe, $\alpha_R, \alpha_{TRT}, \alpha_{TT}, \beta_R, \beta_{TRT}$ and β_{TT} are set using the values in Table 2.2 where $\alpha_R = 5°$ and $\beta_R = 8°$. The lighting component intensities are set at $k_a = 0.5, k_d = 0.5$ and $k_s = 0.2$.

Algorithm 4: Marschner Shader

```
IN   :   4-component vertex position vector, $pos$
         3-component tangent vector, $tangent$
         3-component normal vector, $normal$
         4-component light position vector, $lightPos$
         4-component camera position vector, $camPos$
         3-component hair colour, $colour$
         3-component light colour, $lightColour$
OUT  :   3-component vector representing the final colour of pixel

1    $lightDir \leftarrow normalize(lightPos - pos)$
2    $viewDir \leftarrow normalize(camPos - pos)$
3    $hairColour \leftarrow colour * lightColour$
4    $\Psi_a \leftarrow hairColour$
```

```
5      $N \leftarrow normal.xyz$
6      $V \leftarrow normalize(viewDir.xyz)$
7      $L \leftarrow normalize(lightDir.xyz)$
8      $\Psi_d \leftarrow hairColour\ (N \cdot L)$
9      $\theta \leftarrow \sin^{-1}(N \cdot L) + \sin^{-1}(N \cdot V)$
10     $\cos\emptyset \leftarrow (L - (N \cdot L) \times \mathrm{N}) * (V - (N \cdot V) \times \mathrm{N})$
11     $\Psi_R \leftarrow \cos(\theta - \alpha_R)^{\beta_R}$
12     $\Psi_{TRT} \leftarrow \cos(\theta - \alpha_{TRT})^{\beta_{TRT}}$
13     $\Psi_{TT} \leftarrow \cos\emptyset * \cos(\theta - \alpha_{TT})^{\beta_{TT}}$
14     $\Psi_s \leftarrow k_s(\Psi_R + \Psi_{TRT} + \Psi_{TT})$
15     $\Psi \leftarrow \Psi_a * k_a + \Psi_d * k_d + \Psi_s * k_s$
16     return $\Psi$
```

Figure 3.8 Algorithm of Marschner Shader

c) Deep Shadow Maps

Shadowing properties are important to add depth to the hair model (Lokovic & Veach, 2000). Deep Opacity Mapping, introduced by Yuksel and Keyser (2008), is applied to compute shadow properties for translucent hair. Two render passes are required. The first pass is to generate hair shadow maps using depth testing, while the second pass generates the opacity maps.

Hair shadow maps are generated by rendering the depth texture of hair geometry from the point of view from the light source, where Z_0 is the pixel's depth value at which the hair geometry starts. The hair geometry is segmented into 4 layers, with each layer increasing in size from the light source as shown in Figure 2.11.

In the second pass, the deep opacity program calculates the values for opacity of each pixel on the image. Each colour channel of the opacity map is represented by a different layer of the shadow map. These layers influence the intensity values for red, green, blue, and alpha values of a pixel, where red is the first layer while alpha is the last. The pixels that fall in the last layer of the head geometry will have the darkest shadow. This step begins by obtaining Z_0 by performing a texture lookup on the shadow map and the depth values within each layer are determined. The value of layer size is set to 0.0005. This value is chosen after testing as any larger size makes the size for each layer too big. The current depth, Z_i is then compared to Z_0 to determine which layer the pixel belongs. Once the opacities for all the pixels are obtained, the cumulative opacity of a layer at a certain pixel is determined by adding together the transparency values of all the fragments that contribute to that pixel.

The shadow map and opacity map are then blended using additive blending to obtain the final shadow map.

Algorithm 5: Deep Opacity Mapping

```
1   **if** $Z_i < Z_0 + layerSize$ **then**
2           $opacityMap.r \leftarrow 0.01$
3   **else if** $Z_i < Z_0 + 3 * layerSize$ **then**
4           $opacityMap.g \leftarrow 0.01$
5   **else if** $Z_i < Z_0 + 7 * layerSize$ **then**
6           $opacityMap.b \leftarrow 0.01$
7   **else** $opacityMap.a \leftarrow 0.01$
8   **endif**
```

Figure 3.9 Algorithm of Opacity Mapping

3.3.4 Depth Peeling

Depth peeling, an order-independent transparency technique by Everitt (2001) is applied to simulate the semi-transparent properties of hair. This technique involves three rendering passes to render the hair model in layers. This technique also involves two depth buffers. In each pass, the current pixel's depth is compared to the stored depth from the previous pass, and if it is closer to the camera, its colour is remained, else, the pixel colour is discarded. This is to render the hair fragments in layers from front to back. The layers are then alpha blended to create the final image. Blending the layers allow hair to look translucent and acts as an antialiasing technique. The result of alpha blending equation, $\bar{C}_{result}$ is shown below.

$$\bar{C}_{result} = \alpha_{source}\bar{C}_{source} + (1 - \alpha_{source})\bar{C}_{destination} \quad (3.10)$$

where:

$\bar{C}_{source}$ is source colour vector (final colour of fragment shader).

$\bar{C}_{destination}$ is the colour vector already stored in frame buffer.

α_{source} is the alpha value of source colour vector.

Algorithm 6: Depth Peeling

```
1   if $currentDepth > previousDepth$ then
2       $fragDepth = fragCoord.z$
3   elseif $currentDepth < previousDepth$ then
4       $colour \leftarrow colour_{background}$
5       $fragDepth \leftarrow 1$
6   endif
```

Figure 3.10 Algorithm of Depth Peeling

3.4 Experimental Setup

A virtual environment is created using Microsoft Visual Studio 2019. The programming language used in this project is C++ together with OpenGL library. The hardware used for this project's implementation is ASUS TUF Gaming F15 with the following technical specifications:

- Processor : Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz
- Random-access Memory (RAM) : 16.0 GB (15.8 GB usable)
- Graphics : Intel(R) UHD Graphics, NVIDIA GeForce GTX 1650,
  where NVIDIA GeForce GTX 1650 is chosen during the testing phase.

3.5 Benchmarking

Benchmarking tests are carried out in two parts. This is to validate this project's prototype system based on the objectives of this study. The first test involves analysing the influence of guide hair interpolation on the hair-rendering performance. Performance tests are carried out for three setups, which are on different number of guide hair strands with a fixed total number of hair strands rendered, different hair type setups with a fixed total number of hair strands, and different total number of hair strands with a fixed number of strands per guide patch. Performance tests are benchmarked using two parameters, which are frame rate and frame time.

The second test involves comparing the visual results of translucent hair strands using depth peeling and alpha blending. The tests are carried out to compare the visual appearance of hair stands with and without sorting the hair fragments per pixel.

Further details on the setups used for testing will be explained in Chapter 5. Table 3.1 shows the descriptions of parameters used in the type of benchmarking test.

Table 3.1 List of parameters used to carry out their benchmarking tests.

Parameters	**Description**	**Test**
Frame time (ms)	Frame time is a measurement to determine the time taken for a computer algorithm to complete its task for each frame.	Performance Test
Frames per second (fps)	Frame rate is a measurement that is used to determine how many frames is being displayed per second. A frame rate test is conducted on the prototype to determine its ability to render hair in real time.	
Rendered frame	The visual results of hair renders are evaluated by using the rendered frame to compare the appearance of hair strands using depth peeling and traditional alpha blending techniques.	Visual Comparison Test

3.6 Summary

To conclude this chapter, there are two phases in this project, research phase, which was carried out in Chapter 1 and 2, and development phase, which will be carried out in the implementation section (Chapter 4) and the creation of this project's prototype. This chapter also discusses on the methods used to render hair and benchmarking techniques to evaluate the results obtained from this project.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

4.1 Overview

This chapter elaborates on the design and implementation of the proposed system, outlining the overall workflow of the system framework and the algorithms employed. The system is developed using the C++ programming language within the Visual Studio Community Edition 2019 IDE. External libraries, such as ImGUI, Graphics Library Framework (GLFW), OpenGL Extension Wrangler (GLEW), OpenGL Mathematics (GLM), Simple OpenGL Image Library 2 (SOIL2), and Open Asset Import Library (Assimp), are imported to assist the accomplishment of the proposed algorithm.

The Unified Modeling Language (UML) serves as a modelling language, providing illustrations to visually represent the design of the proposed system. Various UML diagrams, such as the Use Case Diagram, Activity Diagram, and Class Diagram are used to illustrate the overall system design.

4.2 Use Case Diagram

Based on Figure 4.1, the primary actor interacting with the proposed system is the user. The user can adjust hair attribute parameters, acting as control factors, through sliders. Additionally, user can also manipulate the virtual camera's properties by zooming and rotating. These use cases allow the manipulation of the final hair colour. Once the hair strands are rendered and displayed, the system computes and updates the current frame, and frame rate and frame time values in the GUI.

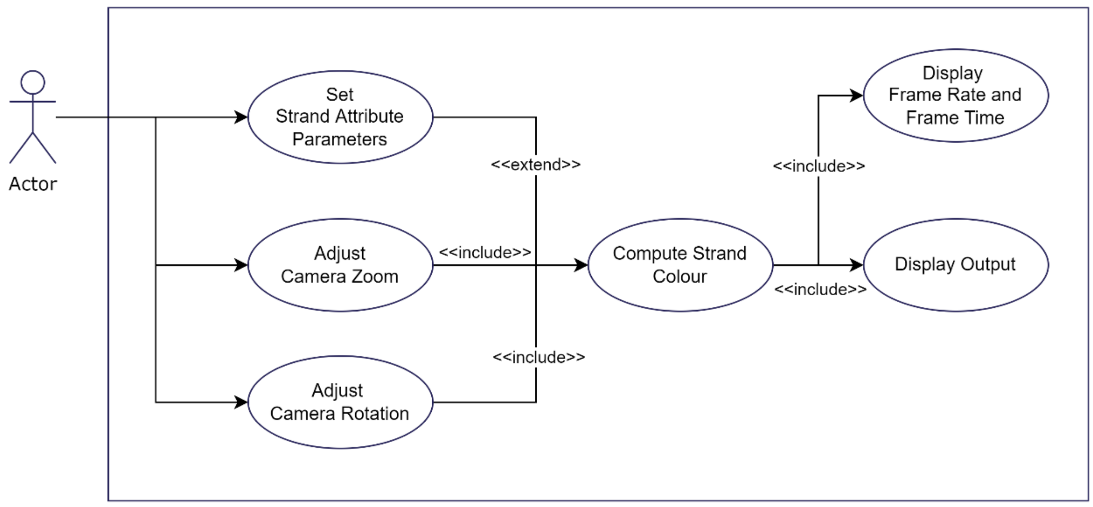


Figure 4.1 UML Case Diagram of proposed system.

4.2.1 Use Case Description

Use cases are further described in Table 4.1-4.5. Table 4.1 shows description of use case diagram for Set Strand Attribute Parameters, where user defines or manipulates values for parameters of variables such as the longitudinal shift and width of hair strand in the Marschner lighting model.

Table 4.1 Use case description for Set Strand Attribute Parameters.

Use Case	**Description**
Actor	User
Pre-condition	All assets have been uploaded.
Basic Flow of Events	User adjusts parameter slider value. Update system's parameter values. Attributes of hair strands are recomputed and updated.
Post-condition	Strand attributes are ready to be updated.

Table 4.2 shows description of use case diagram for Display Frame Rate and Frame Time, where the benchmark values are computed and displayed within GUI for analysis.

Table 4.2 Use case description for Display Frame Rate and Frame Time.

Use Case	**Description**
Actor	System
Pre-condition	Hair strand has been shaded.
Basic Flow of Events	Performance of shader is obtained. Update display values.
Post-condition	Performance values are displayed.

Table 4.3 shows description of use case diagram for Adjust Camera Zoom, that allows user to adjust the zoom value of virtual camera in scene.

Table 4.3 Use case description for Adjust Camera Zoom.

Use Case	**Description**
Actor	User
Pre-condition	All assets have been uploaded.
Basic Flow of Events	User adjusts scroll wheel on mouse. Get zoom sensitivity slider value. Camera matrix is updated.
Post-condition	Camera matrix is ready to be updated

Table 4.4 shows description of use case diagram for Adjust Camera Rotation, that allows user to rotate virtual camera to pan around hair ball model.

Table 4.4 Use case description for Adjust Camera Rotation.

Use Case	**Description**
Actor	User
Pre-condition	All assets have been uploaded.
Basic Flow of Events	User holds down left mouse button and drags. Camera matrix is updated.
Post-condition	Camera matrix is ready to be updated

Table 4.5 shows the use case description for Compute Strand Colour, where system computes the hair strand colour during the rendering process.

Table 4.5 Use case description for Compute Strand Colour.

Use Case	**Description**
Actor	System
Pre-condition	All input has been set by user.
Basic Flow of Events	Get strand attributes and camera details. Update uniforms. Compute strand colour. Compute performance.
Post-condition	Rendered strand is ready to be displayed. Rendering performance is ready to be displayed.

4.3 Activity Diagram

Figure 4.2 shows the activity diagram for Compute Strand Colour use case. This use case is carried out once user has set or updated hair attributes and camera details. Diagram begins with the system getting the input of strand parameters and camera details. The hair strand uniforms are updated to compute the strand colour using Marschner shader. The frame rate and frame time values are then calculated. The rendered frame, frame rate and frame time are then ready to be displayed. Finally, the user can choose to exit the system or manipulate input to recompute the strand colours.

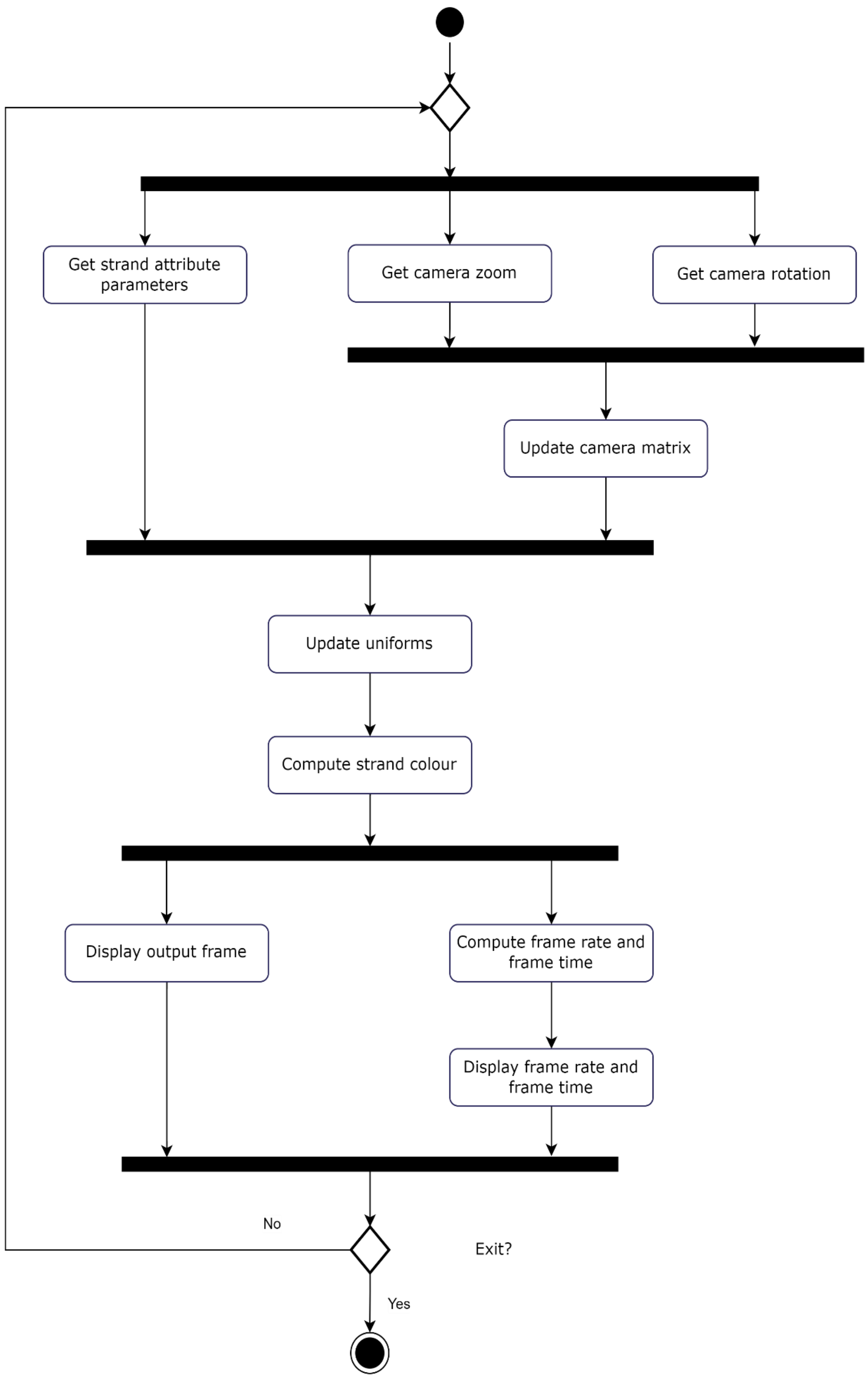


Figure 4.2 Activity diagram of Compute Strand Colour use case.

4.4 Class Diagram

As shown in Figure 4.3, the main class, *Application*, initialises the hair rendering techniques and integrates actions with GUI, which will be executed to run the proposed rendering system. The initialised GUI is used to allow user interaction for resulting output manipulation. *Application* class also has a main virtual camera created from the *Camera* class. Attributes of guide hairs are initialised in the *Hair* class and undergo further rendering techniques in the *Strand* class. The uniforms for the hair rendering shaders are created in the *Uniforms* class and all the shader programs are compiled in the *ShaderProgram*.

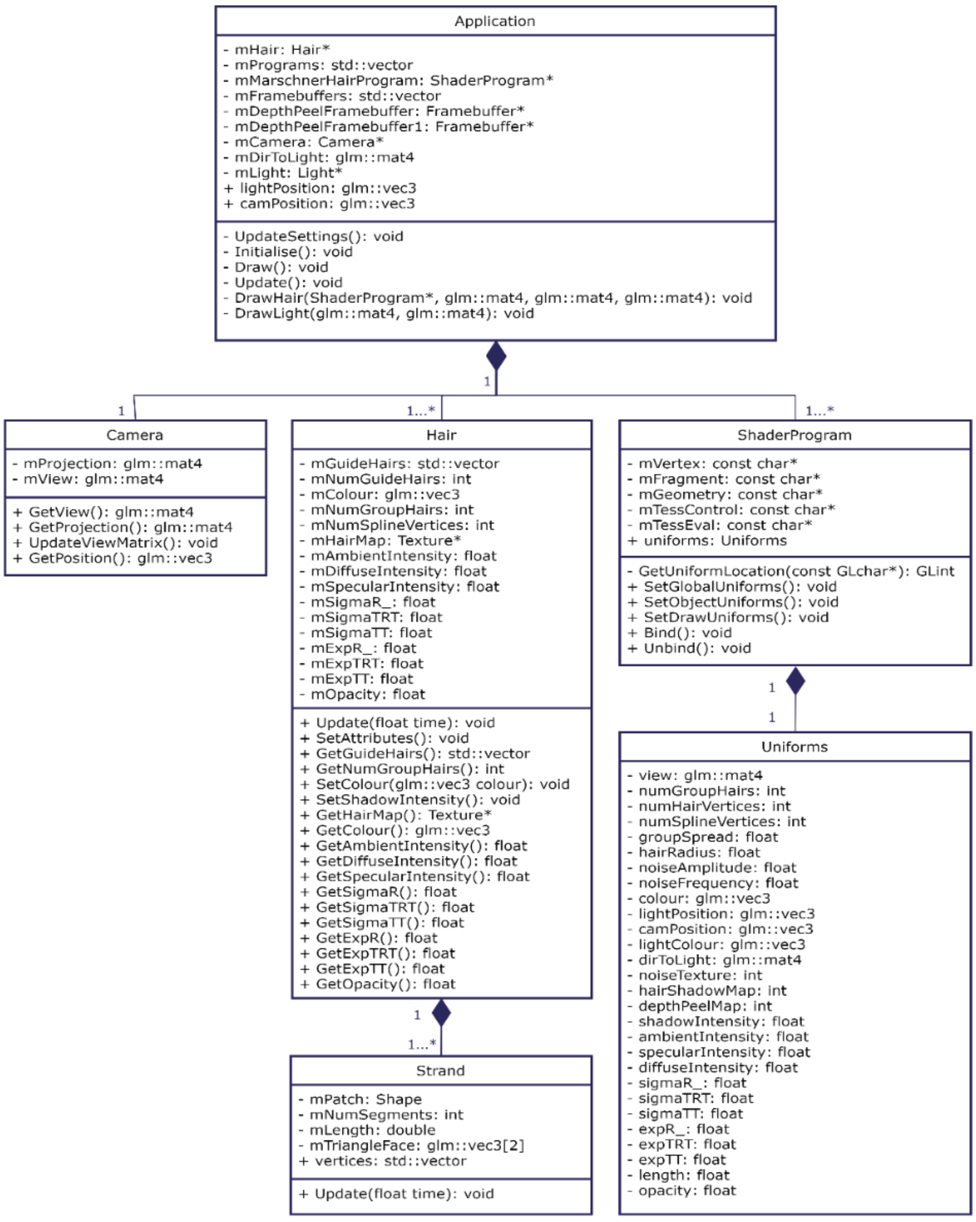


Figure 4.3 UML Class Diagram of proposed system.

4.5 Sequence Diagram

A sequence diagram graphically illustrates the interactions of a use case in a chronological order. It describes the objects and classes involved in a certain scenario when carrying out a function. Figure 4.4 depicts the Sequence Diagram of the Compute Strand Colour use case. The proposed system starts updating the camera position and generating the guide hair strands once the user sets the parameters using application's interface. The hair uniforms are also stored during the rendering process in the hair program. Once the guide hairs are generated, the strands are sent to the hair program to be interpolated and shaded to obtain the final image.

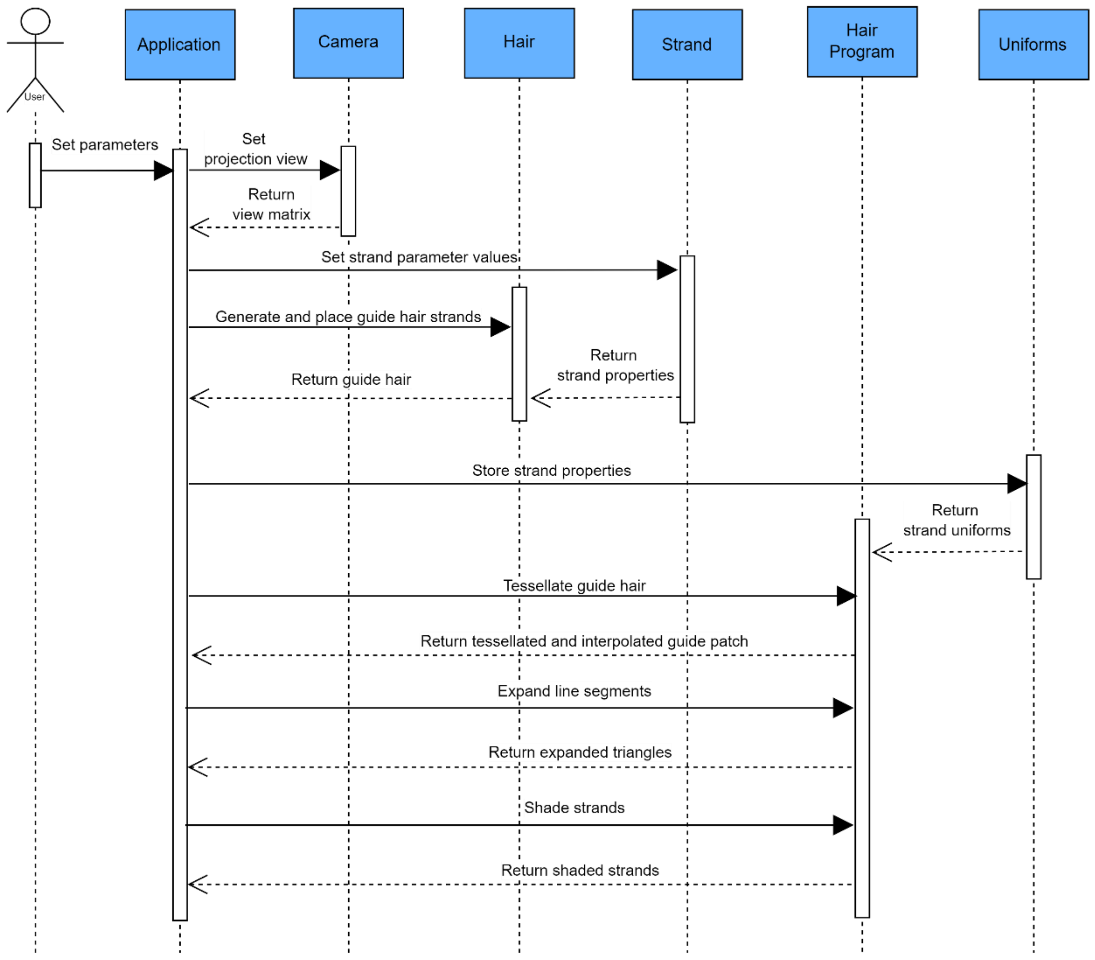
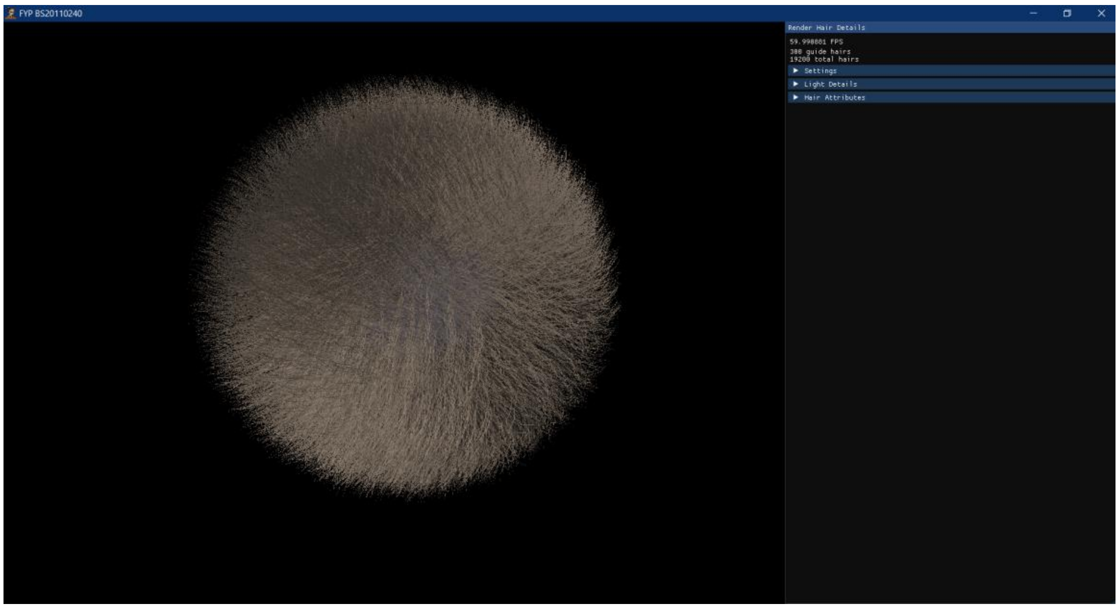


Figure 4.4 UML Sequence Diagram of Compute Strand Colour use case.

4.6 Graphical User Interface (GUI)

A simple user interface has been implemented for this system. The design involves two horizontal frames in a full-screen window. This is so every function can be analysed and controlled by the user in just a single frame within the window. Blue, black and grey are chosen as the base colours of the user interface. The choice of colours allows for simple colour palette's that are not too striking while having enough contrast to account for a good user experience as shown in Figure 4.5.



```
59.998001 FPS
300 guide hairs
19200 total hairs
```

Figure 4.5 GUI initial landing page.

The left frame begins by displaying a hairball where the hair strands are rendered using Marschner's fragment shader while the right frame is an overlay interface that allows user to interact and manipulate the system's output.

User is able to interact with the viewport by rotating the virtual camera by holding down the left mouse button and moving the mouse. User can also adjust the virtual camera's zoom value by scrolling up or down the mouse wheel.

As shown in the right frame (Figure 4.6), users can access displayed output details, which are the frame rate, the number of guide hairs, and the total hair strands count used to render the resulting hairball displayed on the left frame. The subsequent section is the *Settings* dropdown. This section has a checkbox that allows users to hide or display the sphere model of the hairball, and a slider to adjust the sensitivity multiplier value when user zooms in and out.

Light Details dropdown section contain sliders to allow user to adjust properties of virtual light in the. User can manipulate the x, y and z coordinate values, and the colour of virtual light by adjusting the red, green, and blue sliders or by choosing a specific colour on a colour palette. Lastly, there is a checkbox that allows user to enable or disable the display of shadows in the scene.

The last section is the *Hair Attributes* dropdown. This section begins by allowing users to select between Kajiya-Kay and Marschner shader type and its transparency type, which are either traditional alpha blending method or depth peeling method, to be rendered. User can set the number of total hair strands, guide hairs and hair strands interpolated from each guide hair to be rendered through an input field. Lastly, user can manipulate the visual properties for each hair strand. These properties include maximum hair length, radius, spread for each guide hair group, strand colour, noise amplitude and frequency, shadow intensity, and intensities of ambient, diffuse and specular properties. The changes for these variables are updated in real time and output is displayed. All controls are detailed and tabulated in Table 4.6 for sliders in GUI. A sample output is captured as shown in Figure 4.7.

Table 4.6 Description for parameter sliders in GUI.

Parameter	**Label Name**	**Description**	**Range**
-	Zoom Sensitivity	Adjust sensitivity multiplier value of zoom.	$[0.1,2]$
-	Light Position	Adjust coordinate values of key light.	$[-10,10]$
-	Light Colour	Adjust RGB values of key light.	$[0,255]$
-	Strands per guide hair	Set number of hair strands interpolated from guide hair in a group.	$[0,64]$
-	Vertices per strand	Set number of vertices per strand.	$[2,50]$
-	Max length	Set maximum length of hair strands.	$[0.01,2]$
-	Hair radius	Set radius of hair strands.	$[0,0.1]$
-	Group spread	Set the radius of group.	$[0,1]$
-	Hair colour	Set colour of hair strand.	$[0,255]$

-	Noise amplitude	Set strength of random variations along the hair strands.	[0,1]
-	Noise frequency	Set rate of random variations along the hair strands.	
-	Shadow intensity	Set intensity of shadow.	[0,50]
ka	Ambient intensity	Set intensity of ambient lighting.	[0,5]
$\mathrm{k}s$	Specular intensity	Set intensity of specular lighting.	
$\mathrm{k}d$	Diffuse intensity	Set intensity of diffuse lighting.	
αs	Opacity	Set opacity value of hair strand.	[0,1]
-	Colour variation	Set colour variation of hair strand.	[0,5]
α_R	Longitudinal shift	Set angle between the view and reflected light vector.	[5,10]
β_R	Longitudinal width	Set sharpness of specular reflections.	[0,30]

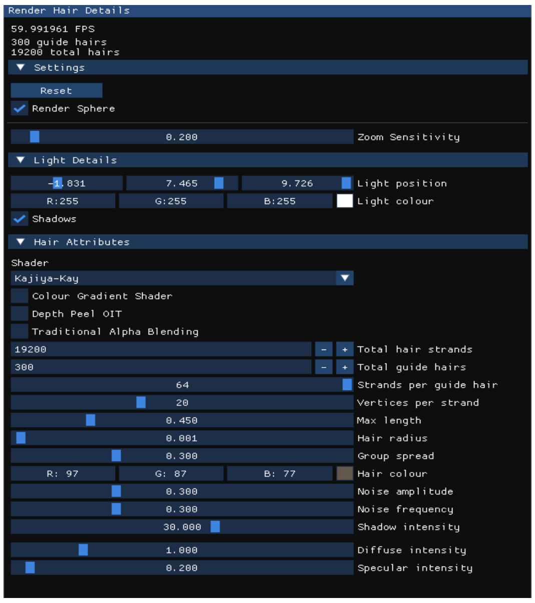
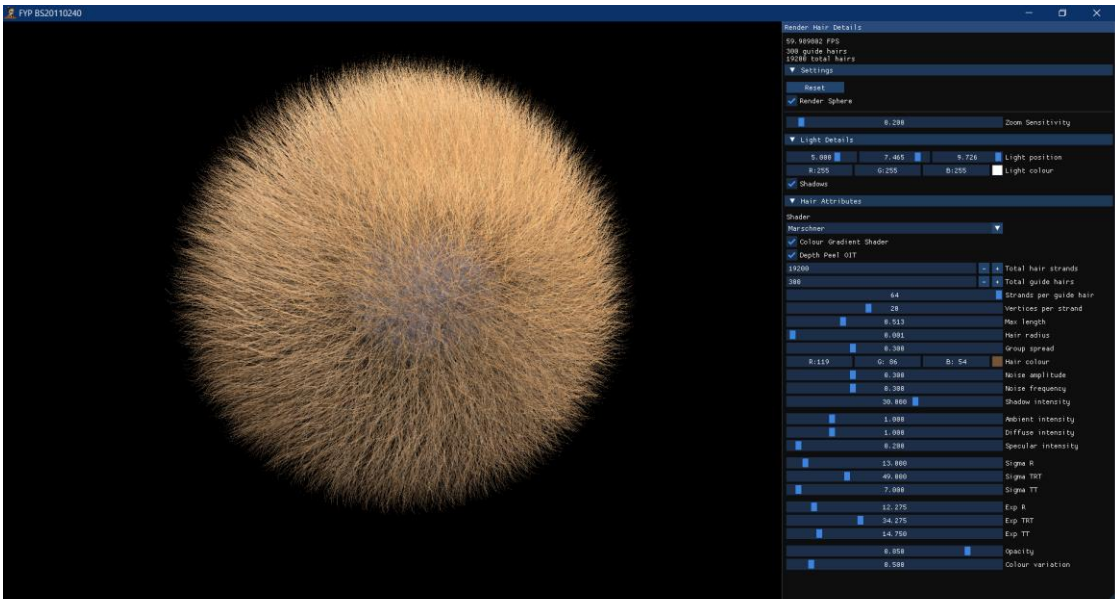

```
Render Hair Details
59.991961 FPS
300 guide hairs
19200 total hairs
▼ Settings
  Reset
  ✓ Render Sphere
  0.200                              Zoom Sensitivity
▼ Light Details
  -1.831     7.465     9.726         Light position
  R:255      G:255     B:255         Light colour
  ✓ Shadows
▼ Hair Attributes
  Shader
  Kajiya-Kay
  Colour Gradient Shader
  Depth Peel OIT
  Traditional Alpha Blending
  19200                     - +      Total hair strands
  300                       - +      Total guide hairs
  64                                 Strands per guide hair
  20                                 Vertices per strand
  0.450                              Max length
  0.001                              Hair radius
  0.300                              Group spread
  R: 97      G: 87     B: 77         Hair colour
  0.300                              Noise amplitude
  0.300                              Noise frequency
  30.000                             Shadow intensity
  1.000                              Diffuse intensity
  0.200                              Specular intensity
```

Figure 4.6 Interface displays system output details in the first section (from top) followed by *Settings, Light Details* and *Hair Attributes* dropdown section after those sections have been expanded in the right frame.



```
FYP BS20110240

Render Hair Details
59.909002 FPS
300 guide hairs
19200 total hairs
▼ Settings
Reset
✓ Render Sphere
0.200                                   Zoom Sensitivity
▼ Light Details
5.000        7.465        9.726         Light position
R:255        G:255        B:255         Light colour
✓ Shadows
▼ Hair Attributes
Shader
Marschner
✓ Colour Gradient Shader
✓ Depth Peel OIT
19200                               - + Total hair strands
300                                 - + Total guide hairs
64                                      Strands per guide hair
28                                      Vertices per strand
0.513                                   Max length
0.001                                   Hair radius
0.300                                   Group spread
R:119        G: 86        B: 54         Hair colour
0.300                                   Noise amplitude
0.300                                   Noise frequency
30.000                                  Shadow intensity
1.000                                   Ambient intensity
1.000                                   Diffuse intensity
0.200                                   Specular intensity
13.000                                  Sigma R
49.000                                  Sigma TRT
7.000                                   Sigma TT
12.275                                  Exp R
34.275                                  Exp TRT
14.750                                  Exp TT
0.850                                   Opacity
0.500                                   Colour variation
```

Figure 4.7 A sample output of GUI.

4.7 Algorithm

This subtopic discusses pseudocodes for algorithms used in the proposed system along with GUI actions to get a conceptual understanding of how these techniques are implemented in this project.

4.7.1 Place Guide Hairs

System only renders a small subset of guide hairs and the rest of the strands are interpolated in the Tessellation Shader (Chapter 4.7.4) due to the large number of hair strands. These guide hairs are placed randomly in triangles on sphere model. Their lengths are determined by the alpha values from the hair map.

```
START
        Input model.
        Apply imported hair map image texture onto sphere model.
        Get user input for number of guide hairs to be displayed.
        Store all triangles of input model.
        For each guide hair to be placed:
            Select a random triangle.
            Select a random point within the selected triangle.
            Scale length of guide hair using alpha value of the selected
            point.
        Store created guide hair in an array.
```

```
END
```

Figure 4.8 Pseudocode for placing guide hairs.

4.7.2 Draw Hair Strands

Colour and shadows of hair strands are rendered in this section. Shadowing process takes two rendering passes. First pass generates a shadow map with depth values for each respective pixel, while the second pass is to generate the deep opacity map.

Next, depth peeling technique (Chapter 4.7.11) is applied to render translucent hair strands. A total of three render passes are used to generate the final colour for the hair strands, where the next closest pixel to the camera is used for every successive render pass. The rendered layers are then blended to generate the final image. The pseudocode for this section is as illustrated below.

```
START
        If user chooses to display shadows:
            Bind hair shadow map frame buffer.
            Disable depth testing.
            Enable additive blending.
            Bind frame buffer for Deep Opacity .
            Draw shadows.
            Clear colour and depth buffers.
            Unbind frame buffers.
            Enable depth testing.
            Disable blending.
        Bind first layer of Depth Peeling frame buffer.
        Clear colour and depth buffers.
        Draw Hair Shader Program.
        Bind second layer of Depth Peeling frame buffer.
        Clear colour and depth buffers.
        Draw Hair Shader Program through Depth Peeling.
        Unbind second layer.
        Clear colour and depth buffers.
        Disable depth testing.
        Render the colour texture of second layer.
        Enable alpha blending.
```

```
                Render the colour texture of first layer.
                Enable depth testing.
                Disable blending.
END
```

Figure 4.9 Pseudocode for drawing hair strands.

4.7.3 Hair Shader Program

In this section, a shader program pipeline is used to render and visualise guide hair strands. Shader program begins by compiling all shaders. Next, program takes attributes of guide hairs and stores them as uniforms to be accessed during the rendering process.

```
START
                Input guide hair attributes.
                Initialise uniforms.
                Generate topology through Tessellation Shader.
                Expand line segments into triangles through Geometry Shader.
                Obtain the final pixel colours through Fragment Shader.
END
```

Figure 4.10 Pseudocode for hair shader program.

4.7.4 Tessellation Shader

Tessellation shader has two sections, tessellation control shader (TCS) and tessellation evaluation shader (TES). TCS adjusts the outer tessellation level using the total number of hair strands that are to be generated and the vertex count within each strand. TES then processes the isoline patches generated and converts them into 2D line segments.

```
START
                For each invocation in the TCS
                If the invocation ID is 0:
                    Get user input of number of strands in a group, numGroupHairs
                    and number of vertices per strand, numSplineVertices.
                Set the outer tessellation level for the first edge to value of
                numGroupHairs.
                Set the outer tessellation level for the second edge to the value of
                numSplineVertices - 1.
```

```
        For each vertex in the TES:
            Get current vector coordinate values through Spline Offset.
            Get previous and next vector coordinate values through Spline
            Offset.
            Calculate and store current tangent vector.
            Stream out tangent value.
END
```

Figure 4.11 Pseudocode for tessellation shader.

4.7.5 Geometry Shader

Geometry shader is used to convert the line segments generated in Chapter 4.7.4 into camera-facing triangles as illustrated below.

```
START
        For each vertex in the input geometry:
            Retrieve the vertex position from the input array.
            Compute the cross product of the normalised tangent and
            position vectors.
            Scale the resulting offset direction for vector by the hair radius.
            Adjust the offset based on a tapering factor.
            Update global variables.
            Get first displaced position by adding offset to vertex position.
            Get second displaced position by subtracting offset to vertex
            position.
            Apply the projection matrix to both displaced positions.
            Emit both resulting vertices.
            Stream out position vector, and tangent and normal vectors.
END
```

Figure 4.12 Pseudocode for geometry shader.

4.7.6 Fragment Shader

Fragment shader is used to compute the final colour of pixels within a fragment.

```
START
        For each fragment in the fragment shader:
            Get light space position.
            Get pixel colour using Marschner Shading.
            Get user input of light properties.
            Get final fragment colour by adding fill light to resulting colour.
            Stream out final fragment colour.
END
```

Figure 4.13 Pseudocode for fragment shader.

4.7.7 Spline Offset

Hair strands are offset in a circle around guide hair's position based on the adjustable group spread variable and noise to add randomness.

```
START
    Interpolate hair strand spline through Catmull-Rom Curve
    Interpolation.
    Add random noise from imported noise texture onto position of hair
    strand.
    Return position.
END
```

Figure 4.14 Pseudocode for spline offset.

4.7.8 Catmull-Rom Curve Interpolation

Catmull-Rom curve interpolation is applied to generate a smooth and continuous hair strand. The algorithm processes each tessellation coordinate in the input geometry. The algorithm then performs interpolation between control points of current and next segments of the hair spline by using neighbouring control points and its tangents.

```
START
    For each tessellation coordinate in the input geometry:
        Compute the length along the spline.
        Extract the fractional part of the length.
        Set the second control point as the integer part of the length.
        Determine index of neighbouring control points.
        If index $< 0$:
            index $= 0$
        If index > number of hair segments:
            index = number of hair segments
        Calculate the first tangent between the first and third control points.
        Calculate the second tangent between the second and fourth control points.
        Interpolate between control points of current and next segments of hair spline
        Return the resulting positions along the spline.
END
```

Figure 4.15 Pseudocode for Catmull-Rom curve interpolation.

4.7.9 Marschner Shading

Marschner lighting model is used during the shading process. Algorithm begins by shifting normal vector. Three lighting components, R, TRT and TT are used to compute the specular component with reflection effect. The overall illumination is obtained by adding result of ambient, diffuse, and specular elements. Algorithm ends by returning the final colour obtained.

```
START
    Shift vector normal, N from vertex position.
    Compute normalised vectors for light direction, L and view direction, V.
    Calculate the dot product of $N \cdot V$ and $N \cdot L$.
    Compute hair colour by multiplying the base colour and light colour.
    If user chooses to apply colour gradient to overall hair colour:
        Initialise a colour scale based on resulting hair colour.
        Make the roots end of the colour scale darker than the tips.
        Update the hair colour based on the modified colour scale.
    Calculate diffuse reflection using product of hair colour with $N \cdot L$.
    Calculate $\theta$ using the angles between $N$ and $L$.
    Compute term1, the normalised reflection of the light direction with respect to the normal.
    Compute term2, normalised reflection of the view direction with respect to the normal.
```

```
            Compute $\cos\phi$ by obtaining the dot product between term1 and
            term2.
            Compute primary highlight component, R.
            Compute secondary highlight component, TRT.
            Compute tertiary component, TT.
            Calculate the specular reflection using sum of R, TRT and TT
            components.
            Combine ambient, diffuse, and specular components.
            Return final colour.
END
```

Figure 4.16 Pseudocode for Marschner shader.

4.7.10 Deep Opacity Maps

Deep opacity mapping algorithm divides hair volume into three layers, where each layer increases in size. Algorithm then determines which layer the current depth value lies to determine its corresponding colour channel and value.

```
START
            Get current vector coordinate.
            Get current fragment depth in clip space.
            Split depth of hair volume into 3 layers.
            Let RGBA colour of current coordinate = (0,0,0,0).
            If current depth lies in first layer:
                Set red colour channel = 0.01
            Else if current depth lies in second layer:
                Set green colour channel = 0.01
            Else if current depth lies in third layer:
                Set blue colour channel = 0.01
            Else:
                Set alpha channel = 0.01
            Stream out RGBA value.
END
```

Figure 4.17 Pseudocode for deep opacity maps.

4.7.11 Depth Peeling

Depth peeling algorithm involves two depth buffers for every render pass required. These depth buffers are required to compare the previous closest point with the current one. Algorithm discards fragment colour if depth value for current iteration is closer to the camera than the previous.

```
START
        Input current fragment colour and current depth value.
        Get previous stored depth value.
        If current depth > previous depth:
            Current fragment colour remains unchanged.
        Else:
            Replace Current fragment colour with background colour.
        Update final fragment colour and depth.
        Update previous stored depth value.
        Stream out previous stored depth value.
END
```

Figure 4.18 Pseudocode for depth peeling.

4.8 Summary

In conclusion, this chapter provides an overview of the development system. UML diagrams shown in this chapter have explained the structure and relationships on class functions implemented in this project's system. The algorithms of proposed model used are also explained in this chapter.

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Overview

In this chapter, tests will be carried out to analyse the proposed system. Analysis of performance tests and visual results of rendered hairball geometry will be carried out in this chapter.

5.2 Experimental Setup

The default parameters used in this chapter are as stated in Table 5.1. Some parameters will be modified for performance comparisons, such as different noise amplitude and maximum hair length values for different hair type setups. The scene is lit up using only one key light.

Table 5.1 Parameters used for testing phase.

Parameters	**Value**
Light intensity	1.5
Opacity, α_s	0.7
Total hair strands	25600
Total guide hairs	400
Strands per guide patch	64
Maximum hair length	0.5
Hair radius	0.001
Group spread	0.3
Hair colour	(97, 87, 77)
Noise amplitude	0.2

Noise frequency	0.5
Ambient intensity, k_a	0.5
Specular intensity, k_s	0.2
Phong specular exponent, p	7
Diffuse intensity, k_d	0.5
Longitudinal shift of R component, α_R	5
Longitudinal width, β_R	8
Light Colour	(255,255,255)
Light Intensity	1.5
Light Position	(-1.831, 7.465, 9.726)

During performance testing, hairball geometry is shaded using Marschner shader. Different hair types are used to add stress to the proposed model. The hair types along with their condition parameters are as stated below, where the default hair type is normal hair.

i. **Normal hair**

Maximum hair length is set to 0.5 and noise amplitude is set to 0.1.

ii. **Curly hair**

Maximum hair length is set to 0.5 and noise amplitude is set to 0.5.

iii. **Long hair**

Maximum hair length is set to 1.0 and noise amplitude is set to 0.1.

iv. **Long and curly hair**

Maximum hair length is set to 1.0 and noise amplitude is set to 0.5.

5.3 Results and Discussion

This section discusses the results obtained from the testing phase of the proposed system. This section has two subtopics. The first subtopic discusses the hair-rendering performance while the second subtopic discusses the visual result of depth peeling.

All performance tests carried out in Chapter 5.3.1 are repeated for 5 iterations. All tests are done for 60 seconds as a constant variable to obtain the average frame rate and frame time for each iteration. The timing measurements have been captured with Nvidia's Nsight Graphics tool on an experimenting device that runs on NVIDIA GeForce GTX 1650 and Microsoft Windows 10 as its operating system. The captures have been done in the proposed system's editor with a scene viewport resolution of 1920x1080. The ideal and minimum benchmarking values are set to the general performance requirement for real-time simulations or applications.

Table 5.2 Performance benchmark values.

Requirement	**Frame rate (fps)**	**Frame time (ms)**
Minimum	30	33.33
Ideal	60	16.67

5.3.1 Hair-rendering Performance

The performance testing for this section involves testing the influence of hair interpolation on the performance of Marschner shader. Tests involve different number of guide hair strands with a fixed total number of hair strands rendered, different hair type setups with a fixed total number of hair strands, and different total number of hair strands with a fixed number of strands per guide patch. Tests are done with self-shadowing enabled as hairball geometry has better visual appearance as discussed in Section 4.3 of Chapter 2.

The first performance test is carried out on hairball-rendering simulation in 60 seconds using 400, 800, 1600 and 3200 guide hair strands respectively, where total hair strands are kept constant at 25600 hairs.

Table 5.3 Frame rate and frame time of hairball simulation in 60 seconds using 400 guide hair strands and 64 hair strands per guide patch.

Time (s)	**Frame Rate (fps)**	**Frame Time (ms)**
0	56.20	17.79
5	55.00	18.18
10	57.10	17.51
15	56.80	17.61
20	55.60	17.99
25	55.45	18.03
30	56.90	17.57
35	57.00	17.54
40	54.80	18.25
45	57.00	17.54

50	55.85	17.91
55	54.70	18.28
60	55.90	17.89

Table 5.4 Frame rate and frame time of hairball simulation in 60 seconds using 800 guide hair strands and 32 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	29.35	34.07
5	29.75	33.61
10	29.95	33.40
15	32.20	31.06
20	29.82	33.54
25	29.49	33.91
30	30.11	33.21
35	31.66	31.59
40	30.75	32.52
45	29.92	33.42
50	30.22	33.09
55	30.48	32.81
60	30.20	33.11

Table 5.5 Frame rate and frame time of hairball simulation in 60 seconds using 1600 guide hair strands and 16 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	15.10	66.23
5	15.62	64.02
10	15.80	63.29
15	15.30	65.34
20	15.95	62.70
25	15.80	63.29
30	16.58	60.31
35	15.03	66.53
40	16.90	59.17

45	15.85	63.09
50	16.00	62.50
55	15.42	64.85
60	14.75	67.80

Table 5.6 Frame rate and frame time of hairball simulation in 60 seconds using 3200 guide hair strands and 8 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	7.12	140.45
5	8.12	123.15
10	8.56	116.82
15	8.12	123.15
20	8.50	117.64
25	8.13	123.00
30	9.32	107.30
35	8.56	116.82
40	6.45	155.04
45	8.72	114.68
50	8.00	125.00
55	7.8	128.21
60	7.9	126.58

Table 5.7 Average frame rate and frame time of simulation under different total number of guide hair strands used in 60 seconds. Total number of hair strands are kept constant at 25600 strands.

Total Number of Guide Hairs	Frame Rate (fps)	Frame Time (ms)
400	56.02	17.85
800	30.30	33.00
1600	15.70	63.69
3200	8.10	123.46

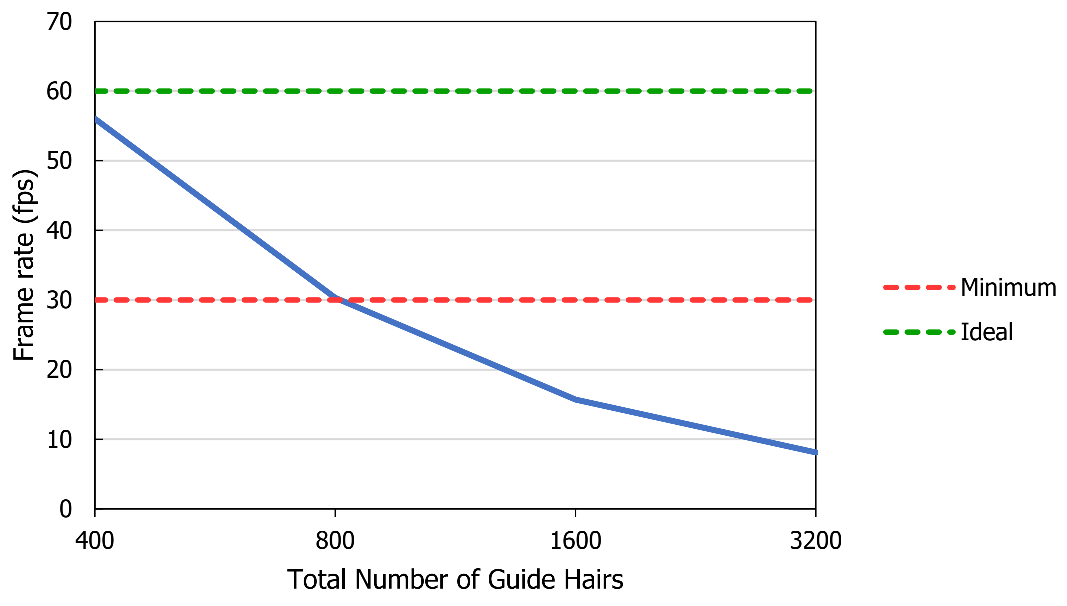


Figure 5.1 Rendering performance with increasing number of guide hairs used. Total number of hair strands are kept constant at 25600 strands.

Table 5.8 Frame rate and frame time of hairball simulation in 60 seconds using 25600 guide hair strands and 1 hair strand per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	0.95	1238.72
5	1.10	1262.68
10	0.75	1235.16
15	0.50	1250.30
20	0.85	1243.80
25	0.70	1260.05
30	1.00	1230.73
35	0.32	1255.92
40	0.90	1222.60
45	0.80	1245.40
50	0.65	1232.18
55	1.05	1265.70
60	0.83	1240.85

Table 5.3 to Table 5.6 show the performance result of the first performance test. The influence of hair interpolation on the rendering performance can be seen in the results of Table 5.7 and Figure 5.1. Based on the results in Table 5.7 and Figure 5.1, 400 guide hairs used produced the highest performance of 56.02 fps and 17.85 ms while 3200 guide hairs produced the lowest performance of 8.10 fps and 123.46 ms. The results show that the frame rate decreases and frame time increases as the total number of guide hairs used increases. This is because as more guide hairs are used when total hairs are kept constant, more hair strands are required to be simulated and less hairs are interpolated within each guide patch, thus requiring more GPU passes. This leads to an increase in computational time and decrease in frame rate. From the graph in Figure 5.1, the rendering performance drops below minimum real-time performance after the 800 guide hairs (32 hair strands per guide patch) mark, where 800 guide hairs recorded performance of 30.30 fps and 33.00 ms.

The influence of hair interpolation can also be observed when comparing the average frame rate and frame time values in Table 5.3 and Table 5.8. The average frame rate and frame time values when simulating only 400 out of 25600 strands in Table 5.3 achieves 56.02 fps and 17.85 ms respectively. These performance values with interpolation are much higher than compared to the performance when rendering hair geometry without interpolation as shown in in Table 5.8, which achieved only an average of 0.80 fps and 1244.93 ms respectively. This is because more GPU passes are required without interpolation, where 25600 passes are required to fully simulate every strand geometry in the scene. The values in Table 5.3 are much lower because only 400 passes are required, and the remaining strands are interpolated to fill the remaining volume in the respective patches.

The next test is done using different hair type setups, normal, curly, long, and long and curly hair strands. This test is done to test the performance of Marschner shader on different hair types. These setups act as stress on the proposed system. Results can be seen in Table 5.9 to Table 5.12 and Figure 5.2.

Table 5.9 Frame rate and frame time of curly hairball simulation in 60 seconds using 400 guide hair strands and 64 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	47.66	20.98
5	47.62	21.00

10	49.63	20.15
15	47.13	21.22
20	50.43	19.83
25	48.90	20.45
30	51.02	19.60
35	48.19	20.75
40	52.88	18.91
45	50.66	19.74
50	44.11	22.67
55	52.08	19.20
60	49.88	20.05

Table 5.10 Frame rate and frame time of long hairball simulation in 60 seconds using 400 guide hair strands and 64 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	53.42	18.72
5	56.02	17.85
10	52.69	18.98
15	52.60	19.01
20	54.05	18.50
25	52.60	19.01
30	54.50	18.35
35	48.26	20.72
40	56.21	17.79
45	52.85	18.92
50	60.00	16.67
55	57.34	17.44
60	49.43	20.23

Table 5.11 Frame rate and frame time of long and curly hairball simulation in 60 seconds using 400 guide hair strands and 64 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	43.55	22.96
5	41.98	23.82
10	45.35	22.05
15	46.73	21.40
20	45.09	22.18
25	43.96	22.75
30	45.66	21.90
35	44.25	22.60
40	45.98	21.75
45	45.21	22.12
50	46.62	21.45
55	44.72	22.36
60	41.24	24.25

Table 5.12 Average frame rate and frame time of simulation under different hair type setups in 60 seconds.

Hair Type	Frame Rate (fps)	Frame Time (ms)
Normal	56.02	17.85
Curly	49.14	20.35
Long	53.68	18.63
Long and Curly	44.58	22.43

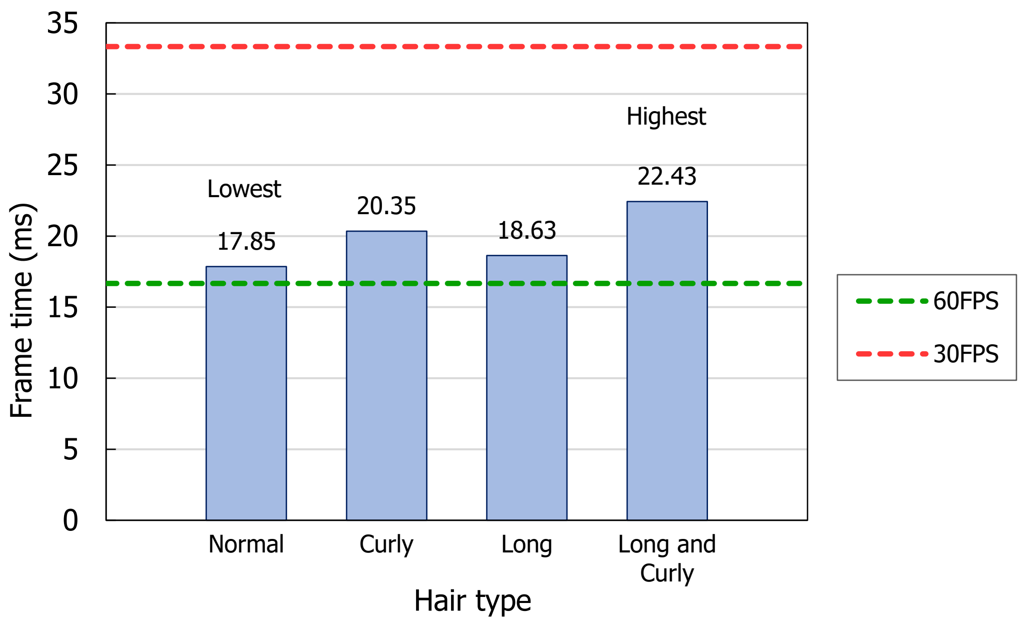


Figure 5.2 Average frame times of different hair type setups.

From Figure 5.2, all hair type setups achieved real-time performance. Normal hair achieved highest frame rate of 56.02 fps and long and curly hair achieved the lowest frame rate of 44.58 fps. In terms of computation time, normal hair type renders the fastest at an average of 17.85 ms while long and curly hair renders the slowest at an average of 22.43 ms between frames. Rendering times for the curly hair and long hair setups are longer than that of normal hair (20.35 ms and 18.63 ms respectively). Longer hair requires the generation and rendering of longer line segment geometries, while the noise used to create curly hair adds disturbance to the geometric structure. From Table 5.12 and Figure 5.2, rendering curly hair requires a longer computation time than rendering longer hair. The combination of curly and long hair setups adds the most stress to the proposed system, thus achieving the lowest average frame rate and highest average frame time.

The last test is done with increasing number of guide hair strands of 400, 500, 600, 700 and 800 to increase the total number of hair strands to 25600, 32000, 38400, 44800 and 51200 hairs respectively, where number of guide hairs within each guide patch are kept constant at 64 strands. Results can be seen in Table 5.13 to Table 5.17 and Figure 5.3. The average performance results from Table 5.13 to Table 5.16 are obtained as shown in Table 5.17 and analysed using a graph as shown in Figure 5.3.

Table 5.13 Frame rate and frame time of hairball simulation in 60 seconds using 500 guide hair strands and 64 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	46.20	21.65
5	47.60	21.01
10	47.55	21.03
15	46.80	21.37
20	48.10	20.79
25	47.80	20.92
30	47.35	21.12
35	46.93	21.31
40	47.08	21.24
45	47.90	20.88
50	46.70	21.41
55	46.80	21.37
60	46.40	21.55

Table 5.14 Frame rate and frame time of hairball simulation in 60 seconds using 600 guide hair strands and 64 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	39.75	25.16
5	40.20	24.88
10	39.04	25.61
15	38.80	25.77
20	40.00	25.00
25	38.50	25.97
30	39.13	25.56
35	40.57	24.65
40	39.80	25.13
45	38.95	25.67
50	39.90	25.06
55	40.03	24.98
60	38.70	25.83

Table 5.15 Frame rate and frame time of hairball simulation in 60 seconds using 700 guide hair strands and 64 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	35.13	28.47
5	34.00	29.41
10	35.20	28.41
15	34.62	28.89
20	33.75	29.63
25	34.50	28.99
30	34.88	28.67
35	34.95	28.61
40	35.33	28.30
45	34.20	29.24
50	33.82	29.57
55	34.94	28.62
60	35.00	28.57

Table 5.16 Frame rate and frame time of hairball simulation in 60 seconds using 800 guide hair strands and 64 hair strands per guide patch.

Time (s)	Frame Rate (fps)	Frame Time (ms)
0	30.81	32.46
5	32.14	31.11
10	33.02	30.28
15	31.25	32.00
20	31.37	31.88
25	31.89	31.35
30	31.69	31.56
35	32.73	30.55
40	31.98	31.26
45	30.91	32.35
50	32.95	30.35
55	31.15	32.10
60	30.47	32.82

Table 5.17 Average frame rate and frame time of simulation under different total number of strands used in 60 seconds. Total number of hair strands within each guide patch are kept constant at 64 strands.

Total Number of Hair Strands	Frame Rate (fps)	Frame Time (ms)
25600	56.02	17.85
32000	47.17	21.20
38400	39.49	25.32
44800	34.64	28.87
51200	31.72	31.53

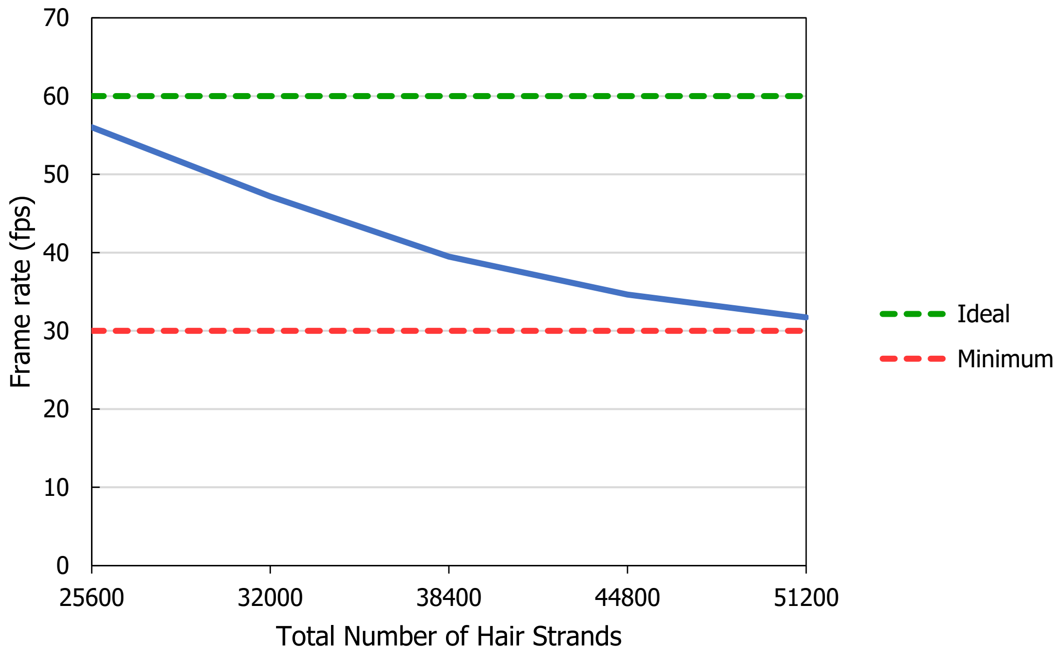


Figure 5.3 Rendering performance with increasing total number of hair strands used. Total number of strands per guide patch are kept constant at 64 strands.

From Table 5.17 and Figure 5.3, the frame rate decreases and frame time increases as more total strands of hair are rendered in the scene, where 25600 total strands achieved the highest performance of an average 56.02 fps and 17.85 ms while 51200 strands achieved the lowest performance of an average 31.72 fps and 31.53 ms. This is because more GPU passes are required as more hair strands are required to be rendered, thus dropping the performance of the shader. However, the proposed system still achieves real-time performance, even when rendering 51200 hair strands. This allows leeway for proposed system to account for hair transparency.

Therefore, the implementation of hair interpolation improves the performance of Marschner shader, where interpolation has allowed the system to render hair strands in real-time with different hair type setups and with different total hair strands of up to about 50000 strands.

5.3.2 Visual Result of Depth Peeling

This section discusses the visual result of the inclusion of hair transparency within the proposed system and the differences between traditional alpha blending technique and depth peeling technique.

Figure 5.4 and Figure 5.5 show the visual results of rendered hairball geometry, with and without accounting for transparency properties. The visual appearance of hair without proper transparency looks dull and lacks depth, as illustrated in left figures in Figure 5.4 and Figure 5.5. In contrast, rendered models that incorporate transparency, like in the right figures, produce improved visual representation.

Depth peeling also acts as an anti-aliasing technique in the algorithm to overcome aliasing edges caused by sub-pixel-sized hair geometry. This can be shown as the right figures in Figure 5.4 and Figure 5.5 have smoother edges that look less aliased or jagged as compared to the left figures. Rough and aliased edges are reduced through the blending effect when layers produced from the geometry peeling (refer Figure 5.6) are blended to form the final image.

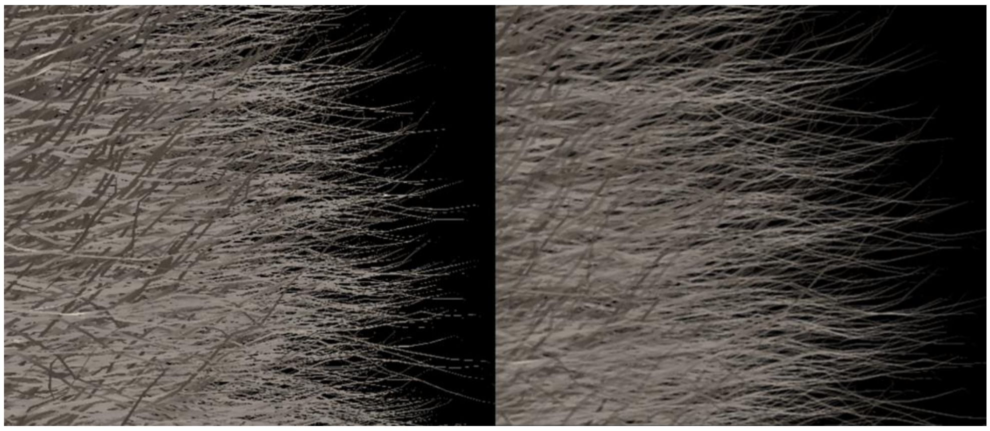


Figure 5.4 Close-up view of Marschner hair rendered without (left) and with (right) depth peeling.

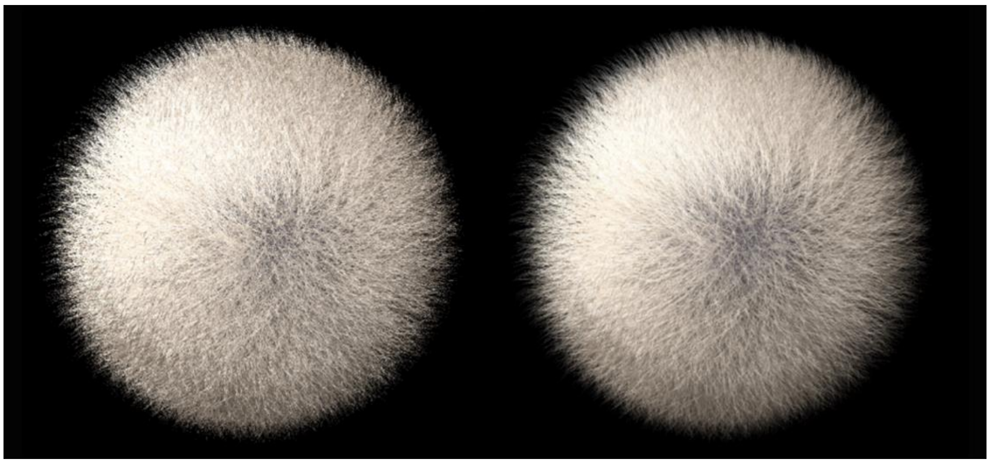


Figure 5.5 Full Marschner hairball geometry rendered without (left) and with (right) depth peeling.

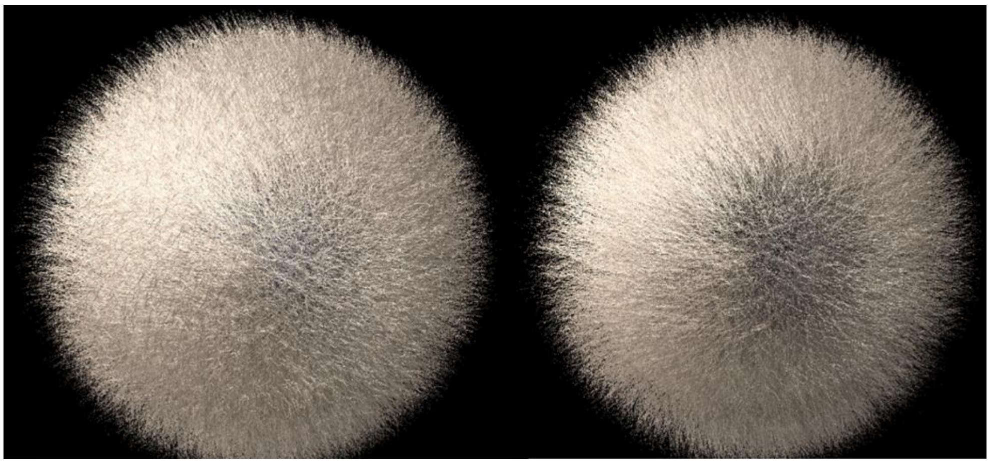


Figure 5.6 Sample output of first (left) and second (right) hairball geometry layer using depth peeling.

Figure 5.7 shows a close-up look of hairball geometry without hair transparency while Figure 5.8 and Figure 5.9 shows hairball geometry with alpha blending and depth peeling transparency respectively. Result in Figure 5.8 shows why traditional alpha blending is not suitable to be implemented when rendering hair strands. This is because hair strands are highly overlapping with one another as discussed in Chapter 2. This property of hair strands leads to renders with incorrect occlusion as strands are only sorted solely based on the root's z-depth values, rendered, and then blended together, resulting in an average of all the contributing hair fragments.

Multi-pass rendering approach, depth peeling technique results in visually better hair renders as shown in Figure 5.9. The final rendered image accounts for transparency along with the sorting of hair strands. This is because the algorithm sorts the scene per fragment along the view direction by peeling the hair geometry in layers from front to back, and then blending them together. This technique allows a full sorting and blending of all strand fragments.

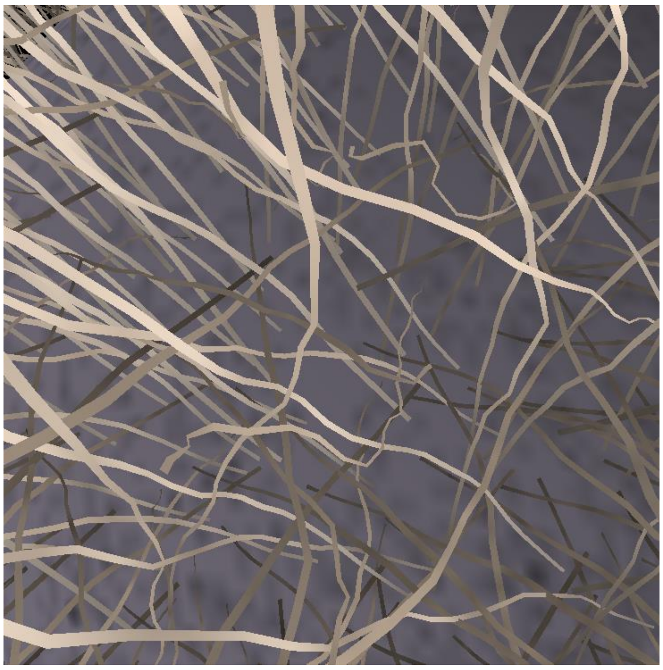


Figure 5.7 Zoomed in frame of rendered hairball geometry without translucent effect.

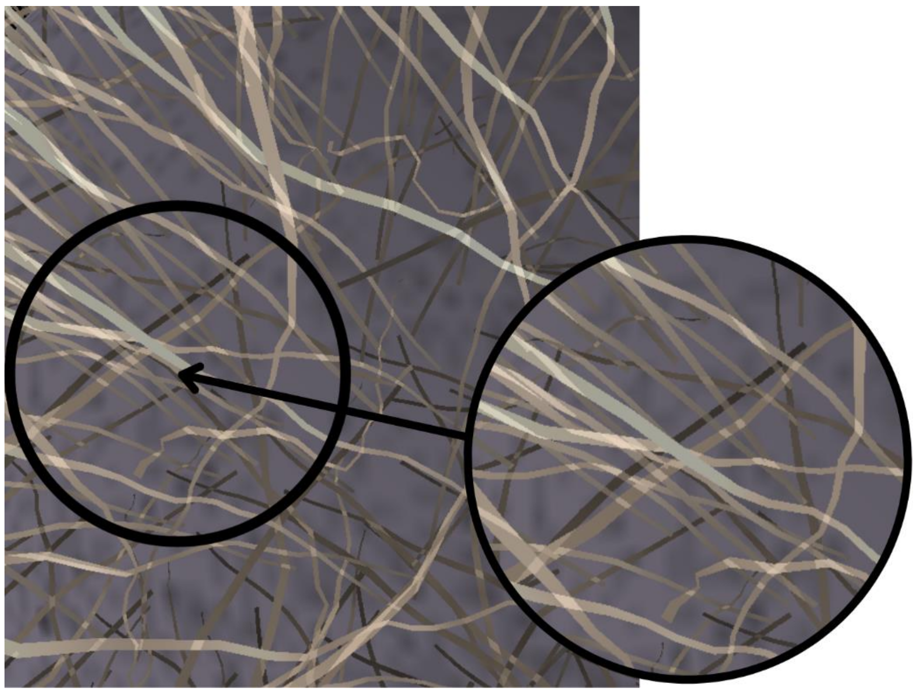


Figure 5.8 Zoomed in frame of rendered hairball geometry with alpha blending transparency. Hair geometry is rendered with incorrect occlusion as hair fragments are not sorted per pixel.

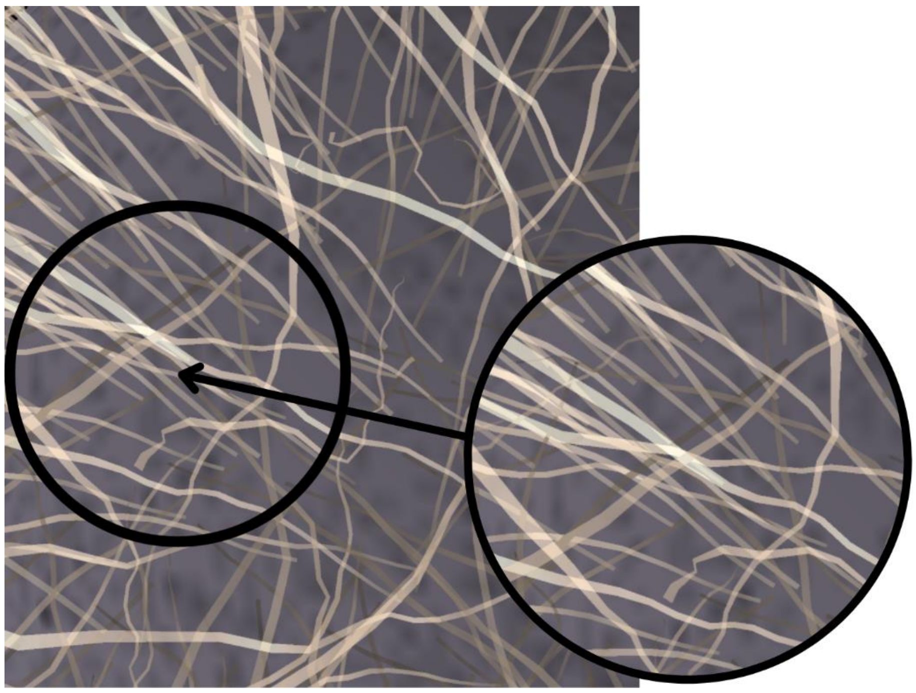


Figure 5.9 Zoomed in frame of rendered hairball geometry with depth-peeling transparency. Hair geometry is rendered after sorting hair fragments per pixel.

Therefore, depth peeling has improved the visual result of hair transparency by producing correct occlusion of overlapping strands by sorting the hair fragments per pixel.

5.4 Summary

In this chapter, two tests were carried out, which were performance test on hair-rendering with hair interpolation and visual comparison test on depth peeling. All the results obtained were discussed in detail. Overall, the results from the experiments conducted have shown that the proposed system has achieved the objectives of this project. Implementation of hair interpolation when rendering hair geometry achieved real-time performance, and depth peeling handled incorrect occlusion caused by overlapping hair strands.

CHAPTER 6

CONCLUSION

6.1 Overview

This chapter includes conclusion of the contributions and results of this project. Future works are also discussed in this project.

6.2 Conclusion

In summary, this project is coded in C++ and C++ library. The project's executable file is generated using Visual Studio 2019. The aim of this project is to render hair geometry in real-time and adapt order-independent transparency on hair strands. There are two objectives, where both had been achieved.

The first objective is to render hair geometry in real-time using guide hair interpolation. Performance tests were carried out using different experiments, which were different number of guide hair strands with a fixed total number of hair strands rendered, different hair type setups with fixed total number of hair strands, and different total number of hair strands with a fixed number of strands per guide patch. The results from the performance tests carried out have shown that guide hairs have reduced the number of GPU passes, thus reducing the number of hair strands simulated. This has allowed hair-rendering to achieve real-time performance when rendering different types of hairstyles.

The second objective is to adapt order-independent transparency for correct occlusion of hair strands using depth peeling. Visual comparison tests were carried out for this objective. Visual result of inclusion of hair transparency within the proposed system and the differences between traditional alpha blending technique and depth peeling technique were obtained and discussed in this project. Results show depth peeling rendered hair geometry with proper occlusion, thus visual results obtained are better than that of traditional alpha blending.

Therefore, we can conclude both objectives of this project were achieved.

The contributions to this study are by improving the performance of hair-rendering by reducing the number of simulated hair strands and adapting order-independent transparency technique to handle incorrect occlusion and improve visual results of translucent hair strands.

6.3 Future Works

Sustainability is an important factor for a proposed system since it presents potential aspects or fields for additional enhancement. This provides opportunities for researchers to further enhance a system, with the goal of achieving better results after every study made. Some suggested areas for future works to improve the proposed framework are:

i. **Level-of-Detail (LOD)**

   Implementation of technique to switch between a rasterizer to simulate explicit hair strands when camera view is close to the hair model and a raymarcher to approximate the hair geometry as volume model during minification can help improve the performance of proposed system.

ii. **Culling**

   Implementation of culling can help improve the performance of hair rendering as polygons that are not visible to the viewer are removed and not included in the rendering process.

iii. **Application of Hair Physics**

   Hair physics in the field of hair rendering allows the simulation of hair movement and behaviour. Physics techniques such as mass-spring systems and dynamic follow the leader (Müller *et al.*, 2012) are some dynamic techniques to simulate hair physics properties.

6.4 Summary

This project has provided a technique to improve hair-rendering performance and adapt order independent transparency when rendering explicit hair strands. As discussed earlier, there is still room for improvement in this proposed system. Therefore, more study is required in the field or hair-rendering to improve visual results while maintaining real-time performance.

REFERENCES

Andersen, T. G., Falster, V., Frisvad, J. R., & Christensen, N. J. (2016). Hybrid fur rendering: combining volumetric fur with explicit hair strands. *The Visual Computer*, **32**, 739-749.

Au, K. C. (2014). Animation: 2D versus 3D and their combined effect (Doctoral dissertation, Massachusetts Institute of Technology).

Barringer, R., Gribel, C. J., & Akenine-Möller, T. (2012). High-quality curve rendering using line sampled visibility. *ACM Transactions on Graphics (TOG),* **31**(6), 1-10.

Battistella, C., McCallum, N. C., Gnanasekaran, K., Zhou, X., Caponetti, V., Montalti, M., & Gianneschi, N. C. (2020). Mimicking natural human hair pigmentation with synthetic melanin. *ACS Central Science*, **6**(7), 1179–1188. https://doi.org/10.1021/acscentsci.0c00068

Batton, W. (2016). A Dynamic Hair Rigging System for Maya.

Bavoil, L., & Myers, K. (2008). Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, 1, 12.

Bavoil, L., Callahan, S. P., Lefohn, A., Comba, J. L. D., & Silva, C. T. (2007). Multi-fragment effects on the GPU using the k-buffer. *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*.

Bertails, F., Hadap, S., Cani, M. P., Lin, M., Kim, T. Y., Marschner, S., & Kačić-Alesić, Z. (2008). Realistic hair simulation: animation and rendering. In *SIGGRAPH Core: Realistic hair simulation: animation and rendering*, 1-154. https://doi.org/10.1145/3260727

Blinn, J. F. (1977). Models of light reflection for computer synthesized pictures. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, 192-198.

Brown, E., & Cairns, P. (2004). A grounded investigation of game immersion. In *CHI'04 extended abstracts on Human factors in computing systems,* 1297-1300.

Carpenter, L. (1984). The A -buffer, an antialiased hidden surface method. *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*.

Cedermalm, R. (2019). Real-time Fur Deformations for Film-Quality Characters.

Chang, J. T., Jin, J., & Yu, Y. (2002). A practical model for hair mutual interactions. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 73-80.

Chang, W. (2018). Application of tessellation in architectural geometry design. In *E3S Web of Conferences,* **38**, 03015.

Choe, B., Choi, M. G., & Ko, H. S. (2005). Simulating complex hair with robust collision handling. *In Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation,* 153-160.

Currius, R. R. (2022). Realistic Real-Time Rendering of Global Illumination and Hair Through Machine Learning Precomputations. Chalmers Tekniska Hogskola (Sweden).

Dajani, D., & Abu Hegleh, A. S. (2019). Behavior intention of animation usage among university students. *Heliyon*, **5**(10). https://doi.org/10.1016/j.heliyon.2019.e02536

Daldegan, A., Thalmann, N. M., Kurihara, T., & Thalmann, D. (1993). An integrated system for modeling, animating and rendering hair. In *Computer Graphics Forum,* **12**(3), 211-221.

DeSanctis, G. (1984). Computer graphics as decision aids: Directions for research. Decision Sciences, **15**(4), 463-487.

Dietrich, C. A., Comba, J. L., & Nedel, L. P. (2006). Storing and accessing topology on the gpu: A case study on mass-spring systems. *ShaderX 5-Advanced Rendering Techniques*, 565-578.

Ducheneaut, N., Wen, M. H., Yee, N., & Wadley, G. (2009). Body and mind: a study of avatar personalization in three virtual worlds. In *Proceedings of the SIGCHI conference on human factors in computing systems,* 1151-1160.

Eigenmann, R. (2001). Performance Evaluation and Benchmarking with Realistic Applications. MIT Press, 40-48.

Ekpenyong, F. E. (2009). Introduction to computer graphics and animation. 3-140.

Enderton, E., Sintorn, E., Shirley, P., & Luebke, D. (2010). Stochastic transparency. *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*.

Everitt, C. (2001). Interactive order-independent transparency. **2**(6), 1-11.

Fuller, N., & Prusinkiewicz, P. (1989). Applications of Euclidean constructions to computer graphics. The Visual Computer, **5**, 53-67.

Gray, J., Dawber, R., & Gummer, C. (1997). The world of hair: a scientific companion. Macmillan.

Gupta, R., & Magnenat-Thalmann, N. (2005, December). Scattering-based interactive hair rendering. In *Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*, IEEE. https://doi.org/10.1109/cad-cg.2005.75

Han, D. (2014). Hair Simulation in TressFX. GPU Pro, **5**, 407-418.

Han, D., & Harada, T. (2012). Real-time hair simulation with efficient hair style preservation. *VRIPHYS 2012 - 9th Workshop on Virtual Reality Interactions and Physical Simulations*, 45–51. https://doi.org/10.2312/PE/vriphys/vriphys12/045-051

Heitz, E., Dupuy, J., Crassin, C., & Dachsbacher, C. (2015). The SGGX microflake distribution. *ACM Transactions on Graphics (TOG),* **34**(4), 1-11.

Hunz, J. (2016). Interactive physically-based hair rendering. https://kola.opus.hbz-nrw.de/index.php/frontdoor/index/index/year/2016/docId/1302

Jansson, E. S. V., Chajdas, M. G., Lacroix, J., & Ragnemalm, I. (2019). Real-Time Hybrid Hair Rendering. In *EGSR*, 1-8. https://diglib.eg.org/bitstream/handle/10.2312/sr20191215/001-008.pdf?sequence=1&isAllowed=n

Jennett, C., Cox, A. L., Cairns, P., Dhoparee, S., Epps, A., Tijs, T., & Walton, A. (2008). Measuring and defining the experience of immersion in games. International journal of human-computer studies, **66**(9), 641-661.

Jensen, H. W., & Christensen, P. (2007). High quality rendering using ray tracing and photon mapping. In *ACM SIGGRAPH 2007 Courses*. https://doi.org/10.1145/1281500.1281593

Jiang, J., Sheng, B., Li, P., Ma, L., Tong, X., & Wu, E. (2020). Real-time hair simulation with heptadiagonal decomposition on mass spring system. Graphical Models, **111**, 101077.

Jin, C. H. (2011). The role of animation in the consumer attitude formation: Exploring its implications in the tripartite attitudinal model. In *Journal of Targeting, Measurement and Analysis for Marketing*, **19**(2), 99–111. https://doi.org/10.1057/jt.2011.8

Johnson, T. E. (1963). Sketchpad III: a computer program for drawing in three dimensions. In *Proceedings of the May 21-23, 1963, spring joint computer conference*, 347-353.

Joy, A., Zhu, Y., Peña, C., & Brouard, M. (2022). Digital future of luxury brands: Metaverse, digital fashion, and non-fungible tokens. *Strategic change*, **31**(3), 337-343.

Kajiya, J. T. (1986). The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 143-150.

Kajiya, J. T., & Kay, T. L. (1989). Rendering fur with three dimensional textures. *ACM Siggraph Computer Graphics*, **23**(3), 271-280.

Kang, C. M., Wang, L., Xu, Y. N., & Meng, X. X. (2016). A survey of photon mapping state-of-the-art research and future challenges. *Frontiers of Information Technology & Electronic Engineering*, **17**(3), 185-199.

Kim, T. Y. (2002). Modeling, rendering and animating human hair. University of Southern California. https://www.researchgate.net/profile/Tae_Kim20/publication/2394902_Modeling_Rendering_and_Animating_Human_Hair/links/0046352992b4514a39000000.pdf

Kim, T. Y., & Neumann, U. (2001). Opacity shadow maps. In *Rendering Techniques 2001: Proceedings of the Eurographics Workshop in London, United Kingdom*, 177-182. Springer Vienna. https://doi.org/10.2312/egwr/egwr01/177-182

Kim, T. Y., & Neumann, U. (2002). Interactive multiresolution hair modeling and editing. *ACM Transactions on Graphics (TOG),* **21**(3), 620-629.

Kong, W., & Nakajima, M. (1999). Visible volume buffer for efficient hair expression and shadow generation. In *Proceedings Computer Animation 1999*, 58-65. IEEE.

Lafrance, M. (2001). First impressions and hair impressions. Yale University, Department of Psychology, New Haven, Connecticut.

Lambert, J. H. (1760). Photometria sive de Mensura et gradibus Luminis. Colorum et Umbrae, Eberhard Klett, Augsberg, Germany.

Lansink, M. (2010). Hair simulation and rendering. University of Twente.

LeBlanc, A. M., Turner, R., & Thalmann, D. (1991). Rendering hair using pixel blending and shadow buffers. *The Journal of Visualization and Computer Animation*, **2**(3), 92-97. https://doi.org/10.1002/vis.4340020305

Lee, D.-W., & Ko, H.-S. (2001). Natural hairstyle modeling and animation. *Graphical Models*, **63**(2), 67–85. https://doi.org/10.1006/gmod.2001.0547

Lee, L. C., Lin, Z., Hu, R., Gong, Z., Kumar, A., Li, T., Li, S., & Hui, P. (2021). When Creators Meet the Metaverse: A Survey on Computational Arts. https://doi.org/10.48550/arxiv.2111.13486

Lee, M., Hyde, D., Bao, M., & Fedkiw, R. (2018). A skinned tetrahedral mesh for hair animation and hair-water interaction. *IEEE transactions on visualization and computer graphics*, **25**(3), 1449-1459. https://doi.org/10.1109/tvcg.2018.2808972

Leerunyakul, K., & Suchonwanit, P. (2020). Asian hair: A review of structures, properties, and distinctive disorders. Clinical, *Cosmetic and Investigational Dermatology*, **13**, 309–318. https://doi.org/10.2147/ccid.s247390

Lemein, J. (2020). Hair rendering: importance sampling of dual scattering approximation

Leshonkov, A., & Frolov, V. (2021). *Real-time rendering of small-scale volumetric structure on animated surfaces*. Ceur-ws.org. Retrieved January 11, 2024, from https://ceur-ws.org/Vol-3027/paper6.pdf

Lokovic, T., & Veach, E. (2000). Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 385-392. https://doi.org/10.1145/344779.344958

Loubet, G., & Neyret, F. (2017). Hybrid mesh-volume LoDs for all-scale pre-filtering of complex 3D assets. *Computer Graphics Forum*, **36**(2), 431–442. https://doi.org/10.1111/cgf.13138

Lowe, R., & Schnotz, W. (2007). Learning with Animation: Research Implications for Design. In *Cambridge University Press eBooks*. http://ci.nii.ac.jp/ncid/BA84736673

Mammen, A. (1989). Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. IEEE Computer graphics and Applications, **9**(4), 43-55. https://doi.org/10.1109/38.31463

Marschner, S. R., Jensen, H. W., Cammarano, M., Worley, S., & Hanrahan, P. (2003). Light scattering from human hair fibers. *ACM Transactions on Graphics (TOG*), **22**(3), 780-791. https://doi.org/10.1145/882262.882345

Marschner, S., & Shirley, P. (2018). Fundamentals of computer graphics. CRC Press.

Martin, T., Engel, W., Thibieroz, N., Yang, J., & Lacroix, J. (2018). TressFX: Advanced Real-Time Hair Rendering. In *GPU Pro 360 Guide to Lighting*, 281-298. AK Peters/CRC Press. https://doi.org/10.1201/b16721-14

Medioni, G., & Yasumoto, Y. (1987). Corner detection and curve representation using cubic B-splines. *Computer vision, graphics, and image processing*, **39**(3), 267-278. https://doi.org/10.1016/s0734-189x(87)80181-0

Minnegalieva, C. B., Gabdrakhmanov, R. I., Khambelov, A. I., Khairullina, L. E., Bronskaya, V. V., & Kharitonova, O. S. (2020). 3D modeling in the study of the basics of computer graphics. In *Journal of Physics: Conference Series,* **1515**(2), 022045. https://doi.org/10.1088/1742-6596/1515/2/022045

Moon, J. T., & Marschner, S. R. (2006). Simulating multiple scattering in hair using a photon mapping approach. *ACM Transactions on Graphics (TOG)*, **25**(3), 1067-1074. https://doi.org/10.1145/1141911.1141995

Moon, J. T., Walter, B., & Marschner, S. (2008). Efficient multiple scattering in hair using spherical harmonics. In *ACM SIGGRAPH 2008 papers*, 1-7. https://doi.org/10.1145/1399504.1360630

Moon, J. T., Walter, B., & Marschner, S. R. (2007). Rendering discrete random media using precomputed scattering solutions. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, 231-242. https://diglib.eg.org/xmlui/bitstream/handle/10.2312/EGWR.EGSR07.231-242/231-242.pdf?sequence=1

Mortenson, M. E. (1999). Mathematics for computer graphics applications. *Industrial Press Inc.*.

Nacke, L., & Lindley, C. (2008). Boredom, immersion, flow: A pilot study investigating player experience. In *IADIS International Conference Gaming 2008: Design for engaging experience and social interaction*.

Natarova, A. O., Kokodii, N. G., Pogorelov, S. V., & Priz, I. A. (2019). Physical methods of analysis of the human hair properties. *2019 IEEE 8th International Conference on Advanced Optoelectronics and Lasers* (CAOL).

Neulander, I., & van de Panne, M. (1998, June). Rendering generalized cylinders with paintstrokes. In *Graphics Interface*, **98**, 233-242.

Nguyen, H., & Donnelly, W. (2005). Hair animation and rendering in the nalu demo. GPU Gems, **2**, 361-380.

Ning, H., Wang, H., Lin, Y., Wang, W., Dhelim, S., Farha, F., Ding, J., & Daneshmand, M. (2023). A Survey on the Metaverse: The State-of-the-Art, Technologies, Applications, and Challenges. *IEEE Internet of Things Journal*, 1. https://doi.org/10.1109/jiot.2023.3278329

Nogueira, A. F., & Joekes, I. (2004). Hair color changes and protein damage caused by ultraviolet radiation. *Journal of Photochemistry and Photobiology B-biology*, **74**(2–3), 109–117. https://doi.org/10.1016/j.jphotobiol.2004.03.001

Nuutila, K. (2021). Hair follicle transplantation for wound repair. *Advances in Wound Care*, **10**(3), 153–163. https://doi.org/10.1089/wound.2019.1139

O'Reilly, E. (2022). *HairRendering: An OpenGL application demonstrating realistic physics and lighting for hair*. GitHub. https://github.com/elor1/HairRendering

Orvalho, V., Bastos, P. P. Z., Parke, F. I., Oliveira, B., & Alvarez, X. (2012). A Facial Rigging Survey. In *Eurographics*, 183–204. https://doi.org/10.2312/conf/eg2012/stars/183-204

Ou, J., Xie, F., Krishnamachari, P., & Pellacini, F. (2012). ISHair: Importance Sampling for Hair Scattering. *Computer Graphics Forum*, **31**(4), 1537–1545. https://doi.org/10.1111/j.1467-8659.2012.03150.x

Park, J. H., Ho, Y. H., & Manonukul, K. (2023). Hair diameter measurement methods: micrometer caliper versus phototrichogram. *Archives of Aesthetic Plastic Surgery*, **29**(2), 97–101. https://doi.org/10.14730/aaps.2022.00423

Petrovic, L., Henne, M., & Anderson, J. (2005). Volumetric methods for simulation and rendering of hair. *Pixar Animation Studios*, **2**(4), 1-6.

Phong, B. H. (1975). Illumination for computer generated pictures. *Communications of the ACM*, **18**(6), 311–317. https://doi.org/10.1145/360825.360839

Pipenbrinck, Nils. (2013). Hermite curve interpolation. Cubic.org. Retrieved July 13, 2023, from http://cubic.org/docs/hermite.htm

Rankin, J. H., & Hall, R. L. (1996). A simple naturalistic hair model. *ACM SIGGRAPH Computer Graphics*, **30**(1), 5–9. https://doi.org/10.1145/232845.232847

Sadeghi, I., Pritchett, H., Jensen, H. W., & Tamstorf, R. (2010). An artist friendly hair shading system. ACM Transactions on Graphics, **29**(4), 1–10. https://doi.org/10.1145/1778765.1778793

Schulz, N. (2014). The rendering technology of ryse. https://advances.realtimerendering.com/s2014/crytek/Sigg14_Schulz_Mader_Ryse_Rendering_Techniques.pptx

Scott, D. W. (2011). Box–Muller transformation. *Wiley Interdisciplinary Reviews. Computational Statistics*, **3**(2), 177–179. https://doi.org/10.1002/wics.148

Shen, Yuliang., Liu, Hualong., Li, Wangyan. (2020). Hair follicle implantation device with hair suction component.

Sheng, W. (2017). Research on the Beauty of Art Animation. In *6th International Conference on Social Science, Education and Humanities*, 220-226. https://doi.org/10.2991/ssehr-17.2018.52

Sintorn, E., & Assarsson, U. (2008). Real-time approximate sorting for self shadowing and transparency in hair rendering. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, 157-162. https://doi.org/10.1145/1342250.1342275

Sintorn, E., & Assarsson, U. (2009). Hair self shadowing and transparency depth ordering using occupancy maps. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, 67-74. https://doi.org/10.1145/1507149.1507160

Tafuri, S. (2019). Strand-based Hair Rendering in Frostbite. In *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games Course*.

Thozhur, S. M., Crocombe, A. D., Smith, P. A., Cowley, K., & Mullier, N. (2006). Structural characteristics and mechanical behaviour of beard hair. *Journal of Materials Science*, **41**(4), 1109–1121. https://doi.org/10.1007/s10853-005-3648-2

Tornincasa, S., & Di Monaco, F. (2010). The future and the evolution of CAD. In *Proceedings of the 14th international research/expert conference: trends in the development of machinery and associated technology*, **1**(1), 11-18.

Turkay, S., & Kinzer, C. K. (2015). The Effects of Avatar-Based Customization on Player Identification. In *IGI Global eBooks*, 247–272. https://doi.org/10.4018/978-1-4666-8200-9.ch012

Vidal, F., Bello, F., Brodlie, K., John, N. W., Gould, D. A., Phillips, R. J., & Avis, N. J. (2006). Principles and Applications of Computer Graphics in Medicine. *Computer Graphics Forum*, **25**(1), 113–137. https://doi.org/10.1111/j.1467-8659.2006.00822.x

Vince, J. (2010). Cartesian Coordinates. In *Springer eBooks*, 27–34. https://doi.org/10.1007/978-1-84996-023-6_5

Wai, K. K. (2017). Rethinking Animaton: Technology, Artistic Expression, and Realism. *Westminster*. https://www.academia.edu/17590955/Rethinking_Animaton_Technology_Artistic_Expression_and_Realism

Wang, Z., Nam, G., Stuyck, T., Lombardi, S., Zollhöfer, M., Hodgins, J., & Lassner, C. (2022). Hvh: Learning a hybrid neural volumetric representation for dynamic hair performance capture. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 6143-6154.

Ward, K. A., Bertails, F., Kim, T. Y., Marschner, S., Cani, M., & Lin, M. (2007). A Survey on Hair Modeling: Styling, Simulation, and Rendering. *IEEE Transactions on Visualization and Computer Graphics*, **13**(2), 213–234. https://doi.org/10.1109/tvcg.2007.30

Watanabe, Y., & Suenaga, Y. (1989). Drawing Human Hair Using Wisp Model. In *Springer* eBooks, 691–700. https://doi.org/10.1007/978-4-431-68093-2_45

Watanabe, Y., & Suenaga, Y. (1992). A trigonal prism-based method for hair image generation. *IEEE Computer Graphics and Applications*, **12**(1), 47–53. https://doi.org/10.1109/38.135883

Williams, L. A. (1978). Casting curved shadows on curved surfaces. *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, 270-274. https://doi.org/10.1145/800248.807402

Yang, F.-C., Zhang, Y., & Rheinstädter, M. C. (2014). The structure of people's hair. PeerJ, 2, e619. https://doi.org/10.7717/peerj.619

Yang, X. D., Xu, Z., Yang, J., & Wang, T. (2000). The cluster hair model. *Graphical Models*, **62**(2), 85-103. https://doi.org/10.1006/gmod.1999.0518

Yoo, B.-Y., Shin, Y.-H., Yoon, H.-H., Seo, Y.-K., & Park, J.-K. (2010). Hair follicular cell/organ culture in tissue engineering and regenerative medicine. *Biochemical Engineering Journal*, **48**(3), 323–331. https://doi.org/10.1016/j.bej.2009.09.008

Yu, X., Yang, J. C., Hensley, J., Harada, T., & Yu, J. (2012). A framework for rendering complex scattering effects on hair. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 111-118. https://doi.org/10.1145/2159616.2159635

Yuksel, C., & Keyser, J. (2008). Deep Opacity Maps. *Computer Graphics Forum*, **27**(2), 675–680. https://doi.org/10.1111/j.1467-8659.2008.01165.x

Yuksel, C., & Tariq, S. (2010). *Advanced techniques in real-time hair rendering and simulation*, 1 -168. https://doi.org/10.1145/1837101.1837102

Yuksel, C., Akleman, E., & Keyser, J. (2007). *Practical Global Illumination for Hair Rendering,* 415-418. https://doi.org/10.1109/pg.2007.53

Zinke, A., Sobottka, G., & Weber, A. (2004). Photo-Realistic Rendering of Blond Hair. In *VMV*, 191–198.