Real-Time Dynamic Fur and Hair Simulation using Verlet Integration

Sachit Misra[*]

[*]Department of Computer Science, SRM Institute of Science and Technology

DOI: 10.29322/IJSRP.11.02.2021.p11053
http://dx.doi.org/10.29322/IJSRP.11.02.2021.p11053

Abstract- Throughout the history of game development, the physics behind the real-time hair simulation has continued to pose a challenge due to lack of availability of computational resources required by the system. Unlike rendering an animation, where the requirement of real-time simulation is absent, game hair physics needs more efficiency when it comes to utilization of computational resources. Generally, for making a hair strand mesh, a cylinder or a capsule mesh is an obvious choice despite its requirement of a higher number of draw calls or resources. This paper proposes the use of an innovative and highly efficient use of quad polygons, whose normals face the render in conjunction with the use of Verlet integration, which delivers optimal results by keeping the frames per second (FPS) stable. Additionally, the proposed physics also allows for physical forces, such as gravity and wind, to affect hair movement as well as simulate a natural curl in the hair strand.

Index Terms- Hair Simulation, Fur Simulation, Real-time hair simulation, Hair movement, Verlet integration.

I. Introduction

Hair is one of the few areas where most game developers struggle and have to make do by building a static textured mesh on the character's head, face or body which simply does not look realistic. To enhance the experience of the observer and adhere to the vision of the game character designer, it is possible to write a module which imitates hair physics closely. Hair simulation in an animation render gives a buffer time for calculating the position of all hair strands to produce an after render of the animation. On the other hand, video game hair simulation is challenging since heavy computation is required in real-time rather than an animation render. If hair simulation is done in real-time with minimal computational requirements and a reduced render time, it will aid to enhance the developer's capability in adding to the user experience.

3D hair physics was first displayed at the 1996 Tokyo Game Show, where the viewers were pleasantly surprised upon observing Aoi Umenokouji's ponytail in Virtua Fighter 3 for Sega's new Model 3 arcade system. Even though this was a huge initiative taken by the company, it did not seem to have been incorporated widely in the following years. For example, the popular Assassin's Creed franchise published by Ubisoft made across different game engines has been using a static mesh for hair till present day. The usage of hair physics escalated when AMD's TressFX and NVIDIA's HairWorks were released in 2014 and 2015 respectively. But Indie developers have not been able to fully embrace hair simulation softwares provided by these graphics companies as the developers face difficulties in setting up the environment because the process requires a 3D modeling software and not all developers have prior knowledge of its working. Furthermore, developers have to rely only on what is within the scope of the presently available softwares and are limited to only what they can offer. There are a few suggestions that developers can benefit from if they build a hair algorithm themselves. Firstly, they would be able to control the time complexity as well as the memory that this feature required by controlling specific aspects of hair simulation that they value in the game. Secondly, the manipulation of hair physics and its mechanism could be controlled by the developer. Thirdly, there would be reduced reliance on third-party dependencies. (Increased reliance on third party softwares can be problematic if support is discontinued, the code changes drastically or in the worst case, breaks). It is for these reasons that implementing this algorithm would benefit the developer.

II. Related Work

Most of the research done on hair simulation lies in three major categories: volume-based, which relies heavily on shaders and is GPU intensive, strand-based, which closely mimics the physics of multiple hair fibers and is CPU intensive and hybrid-based, which involves key hairs or wisps which direct the motion of the strands around them while maintaining multiple volumetric continuous meshes.

Volume-based approach is based on the observation that hair is perceived to be a bulk material in its interactions with the environment. The behavior of a hair body can be viewed as collective material. This material is characterized by resilience to shear and compression, and viscous damping. [11]

Strand-Based approach is simulating a single strand of hair as opposed to a collective. Dynamic models capture credible strand behaviors. The most common model used for simulating individual hair strands is the mass-spring system. The mass-spring system is used to simulate a hair strand which follows the kinematic equations of a spring given by:

$$F = K_S * (l - len_0) * norm(e_i) + C_S * (e_{i+1} - e_i) * norm(e_i)$$

Where, e_i is the edge, the segment connecting every two points is $e_i = p_{i+1} - p_i$ where p is point and the length is l, K_S is the spring coefficient and C_S is the damping coefficient. [8]

Verlet Integration modeling is another strand-based approach which is highly effective and is used widely. Verlet Integration is used to compute the movements of particles because it is more stable than Euler integration with large time steps and does not need to store particle velocities explicitly. The position of the kth particle, P_k, is updated based on the previous two frames' positions, P_k^{-1} and P_k^{-2}, gravity acceleration, g, and wind force, w, as follows:

$$P_k = (P_k^{-1} - P_k^{-2})\Delta t + g\Delta t^2 + \frac{|w \times t_k|}{|w|} w\Delta t^2$$

where w is a given wind force which has a direction and magnitude. We assume that wind acceleration to each particle is proportional to the angle between the hair direction (tangent vector) t_k at the particle and the direction of the wind force, w. [10]

Hybrid-based approach aims to simulate hair by roughly combining the aforementioned methods. The wisp and the NURBS surface model falls under the hybrid method. [6] The wisp model and hierarchical model of selectively subdivided generalized cylinders are used to control the position and motion of multiple hair strands following a key hair. In simulations produced clustering methods, the group construction of hair is often apparent.

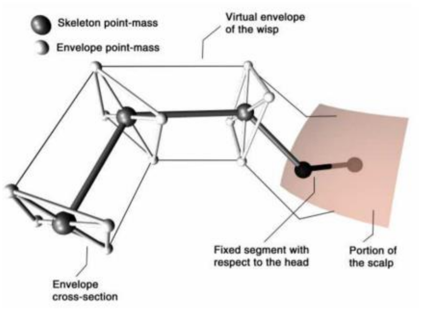

Figure 1 : The elements defining the skeleton and the envelop of a wisp, and their configurations [12]

Figure 1 shows elements defining the skeleton and the envelop of a wisp, and their configurations [12]. Previous researchers also studied the hair style and involve the statistical wisp model for the hair style generation approach. The hair style is still denoted by collision behavior between strands to maintain the realism of 3D virtual human or animal hair [2,4,5,14]. To improve efficiency of the wisp model, the former part of the hair strands are not simulated. [5]

A fourth approach relevant to the topic of discussion here is a particle-based dynamic simulation technique. Since hair and particle volume have some degree of connectivity, simulations of smoke have been devised which can, in some cases, pass roughly as hair simulations. Thus, a similar method can be used for creating volumetric forces in the simulation of connected hair particles. Taking the ideas from particle simulation, they can produce hair strands by linking the particles easily. [1]

Studies have signified the importance for having hair-on-hair collisions or interactions which has been studied by Hadap and Magnenat-Thalman. They take a radical approach by considering hair as a continuum. The continuum assumption states that the physical properties of a medium such as pressure, density and temperature are defined at each point in a specified region. [3]

III. Methodology and Experiment

A. *Location of hair root on mesh*

The vertices in mesh model act as the origin of the hair root. The mesh can be divided into areas where there is need for hair or fur. The density of hair on the mesh model can be controlled through a 3D modeling software, but an algorithm can be further devised to control and optimize the hair density on the mesh to make it developer friendly.

Algorithm 1: Hair Density Mapping

```
    Load Mesh Data
    Load Hair Data
    From Mesh get Vertices Array v[]
    From Hair get Hair root position Array hrp[]
    **Input:** Number of Hair Strands n
1   **if** $n \neq 0$ **then**
2       **for** $i = 0$ to length of v[] **do**
3           $hrp[i] = v[i \times (integer)(\text{length of } v[] / n)]$
4       end **for**
5   end **if**
```

The instantiation of the hair roots depends on the number of hair strands passed as an argument to the function. There have been models that adopt surface sampling method based on quasi-random numbers to position the hair root so that they have a uniform distribution. [8] This requires a considerable amount of computation at the first frame for instantiation. In Algorithm 1: Hair Density Mapping, if the mesh is modeled ensuring uniformity, the surface sampling may not be needed. Furthermore, if the mesh that the developer is working with does not have a satisfactory shape or uniformity, a new quad or a sphere can be imported and subdivided to replace the unsuitable mesh.

B. *Hair/Fur Grain Mapping*

Since the direction of the hair would always be pointing straight outward from the mesh, it is reasonable to assume that hair would be normal to the mesh. This is not always the case when allowing for the natural hair grain pattern [7]. For the natural grain of the hair/fur resting on the mesh, a point inside of the mesh can be defined and projected as a ray passing through the vertices to give a direction vector. The direction vector can be computed as follows:

$$\vec{v} = \frac{b - a}{||b - a||}$$

Where a and b are two points in 3D space, $||\cdot||$ denotes Euclidean norm, and $\vec{v}$ is the direction vector.

This direction vector represents the direction of the hair extending out of the vertex and is added as a constraint on the hair. When external force vector is applied on the hair, the motion of the hair is in the resultant vector direction of the external force and the natural grain direction.

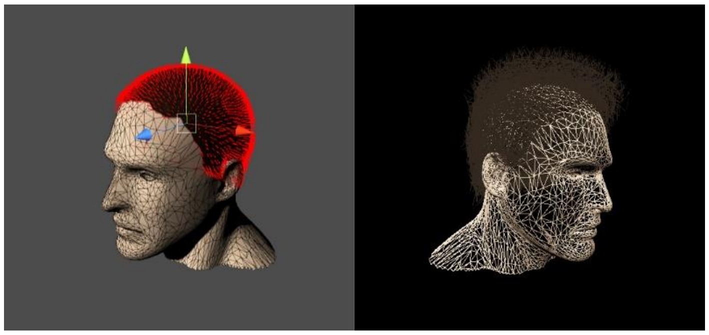

Figure 2: Hair grain projected from a centered point inside the mesh (left); Wireframe of the rendered hair with hair grain (right)

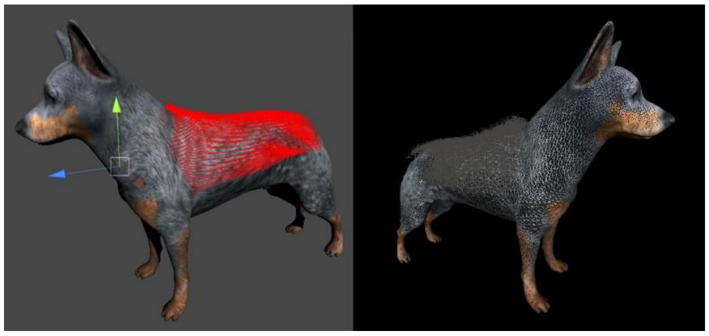

Figure 3: Hair grain direction vectors (red) projected from an off-centered point inside the mesh (left); Wireframe of the rendered hair with hair grain (right)

Figure 2 and 3 show hair grain projected from a centered point inside the mesh (left); Wireframe of the rendered hair with hair grain (right) and hair grain direction vectors (red) projected from an off-centered point inside the mesh (left); Wireframe of the rendered hair with hair grain (right) respectively.

C. *Hair/Fur Physics Simulation*

For physics simulation, Verlet Integration provides better numerical stability, is efficient and easy to implement from the standpoint of a developer. External physical forces like wind drag and gravity can also be accommodated in this process. A linearly interpolated (lerp) damping method is introduced to represent a damping effect. Figure 4 below shows the architecture of the hair strand in this study uses a partitioned/segmented strand.

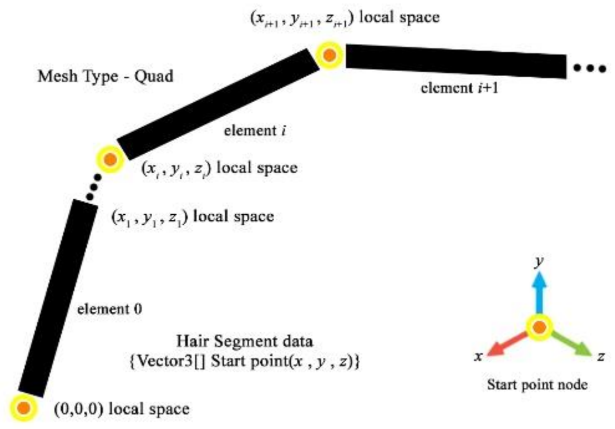
$(x_{i+1}, y_{i+1}, z_{i+1})$ local space
Mesh Type - Quad
element i+1
element i
(x_i, y_i, z_i) local space
(x_1, y_1, z_1) local space
element 0
Hair Segment data
{Vector3[] Start point(x , y , z)}
(0,0,0) local space
y
x
z
Start point node

Figure 4: Hair architecture

Verlet integration is a solution to the kinematic equation for the motion of an object which is given by:

$$x = x_0 + v_0 t + \frac{1}{2}at^2 + \frac{1}{6}bt^3 + \cdots \qquad (\ldots 1)$$

For calculating the next time-step, a Taylor Series Expansion about $x(t \pm \Delta t)$ yields:

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 + \frac{1}{6}b(t)\Delta t^3 + O(\Delta t^4) \qquad (\ldots 2)$$

Where x is the position in 3D space, v denotes velocity of segment, a is the acceleration, b is the jerk term, O is the error value and t is the time.

And for the previous time-step, also yields,

$$x(t-\Delta t) = x(t) - v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 - \frac{1}{6}b(t)\Delta t^3 + O(\Delta t^4) \qquad (\ldots 3)$$

Solving equations (2) and (3) for $x(t \pm \Delta t)$ yields:

$$x(t \pm \Delta t) = 2x(t) - x(t-\Delta t) + a(t)\Delta t^2 + O(\Delta t^4) \qquad (\ldots 4)$$

This implies that the value x or the next 3D coordinate is independent of v. So programmatically, the values stored in the previous frame are compared with the current frame to get the coordinates without having to calculate the velocity of each segment in a hair strand.

Verlet Integration describes the motion of hair simulation using quad polygons, introduces a change in velocity and places some constraints on the relative positions of the polygons to enhance hair simulation considerably. The constraints [9] that are needed in this process are:

- Damping forces determine the amount of time taken for the hair to return to its mean position.
- Gravitational force, a direction vector, is a constant acceleration in a particular direction vector.
- Hair grain amount is the extent to which the hair grain affects the movement of hair.
- Wind speed adds a small random force value using Perlin Noise to each hair strand for a smooth movement multiplied by a direction vector.

The amount of elasticity of the hair can also be simulated by Verlet Integration which can be closely compared to a mass-spring model.

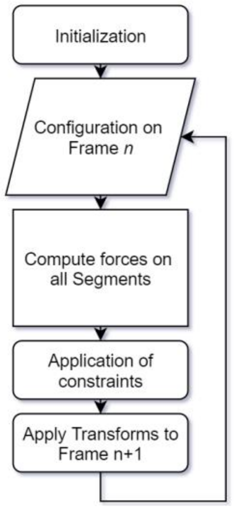
Initialization
Configuration on Frame n
Compute forces on all Segments
Application of constraints
Apply Transforms to Frame n+1

Figure 5: Implementation of Verlet integration in game engine

Figure 5 here illustrates the implementation of Verlet integration in game engine. With Verlet Integration, each segment is linked with the values of other segments, hence, this process reveals a smooth looking inertial "string" like object. However, we observe that while hair fibers do have physics that somewhat represent a string, hair fibers in the real world have a natural curve to them which is discussed in the next section.

For simulating fur, another parameter can be added which controls the width of each strand. Empirically, it is observed that fur strands are usually thick and tapering in width towards the end of the strand. Having a start and an end width for each segment inside a hair strand would require a lot of data to be entered manually by the programmer. This problem can be solved mathematically. There is a set start and end width that the developer can alter. The developer feeds in the values that are shared by all the strands. Each segment receives the value of the width from the previous segment. This calculation can be performed for all segments given by the experientially derived formula,

$$w_i = w_n - \left(\frac{w_n - w_0}{n}\right) i$$

where w_0 is the start width, end width is denoted by w_n, number of hair segments by n and hair length by l. The bottom segment, which starts from the base width, has $i = 0$ which goes up to $i = n$.

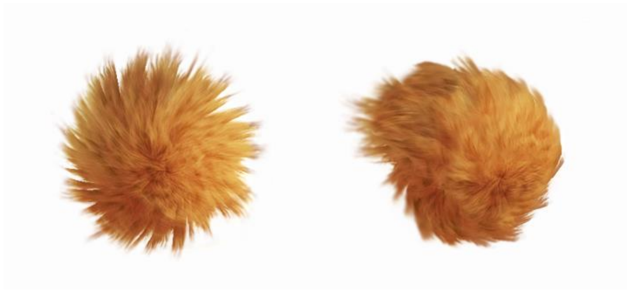

Figure 6: Fur simulation using Verlet integration on static ball (left) and rotating ball (right)

Figure 6 shows fur simulation using Verlet integration on static ball (left) and rotating ball (right). Another interesting dimension of realism, a random additional length and width parameter can be added along with another feature to cap the root vertices to make them smooth. This feature will simulate realistic fur. Increasing the starting width, w_0, will not only visually increase the hair density, but also decrease the time taken to render the frame, as the relatively thicker hair strands occlude the thinner ones behind them from the renderer.

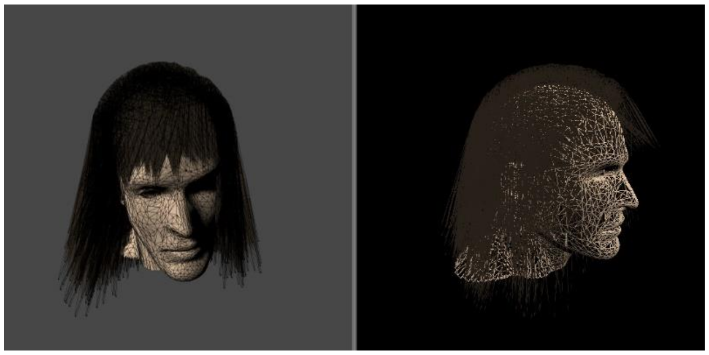

Figure 7: Straight long hair strands simulated with Verlet Integration in a shaded mesh geometry (left) and wireframe (right)

Figure 7 shows straight long hair strands simulated with Verlet Integration in a shaded mesh geometry (left) and wireframe (right).

D. Replicating the natural curl

A hair strand of any origin curls naturally in a particular direction most of the times. These curls are caused by the asymmetrical forces exerted on the fiber by the hHa8 keratin produced in the body. These symmetrical forces may be produced by the shape of the follicle and the amount of keratin coated on the hair. This irregularity can be programmatically introduced in the generated hair strand as a force. Most of the observed isolated hair strands lie in three categories namely; straight, wavy and curly. A straight hair strand would have no asymmetrical forces acting to give a natural curvature, whereas a wavy or a curly hair would have this asymmetry to varying extents along the edges.

Using trigonometric functions, we can introduce a curve in the hair. The hair strand is made up of multiple segments that are interconnected. A variable force can be applied to each successive segment in a hair strand. The force on each hair segment is determined by a trigonometric function and the strength of the effect is variable and controlled. When the applied strength is increased, the hair strand starts taking the shape of the function more accurately and stiffly. For the hair to form a complete circle the parametric equation a circle can be applied in the form of an algorithm.

```
Algorithm 2: Natural Hair Curve
    Load Hair Data
    From Hair get Hair segment Array $hs[]$
    **Input:** Curve Strength Multiplier position vector $c$
1   **for** $i = 0$ to length of $hs[]$ **do**
2       x coordinate of Segment start point += $\sin(c.y \times i) + \cos(c.z \times i)$
3       y coordinate of Segment start point += $\sin(c.z \times i) + \cos(c.x \times i)$
4       z coordinate of Segment start point += $\sin(c.x \times i) + \cos(c.y \times i)$
5       $hs[i]$ = Segment start point
6   end **for**
```

The above algorithm needs to be offset by the local position and rotation initialization. This offset integrates the simulation with the game engine in a neat way, since the trigonometric functions will always have fixed values and would not account for the initial positions.

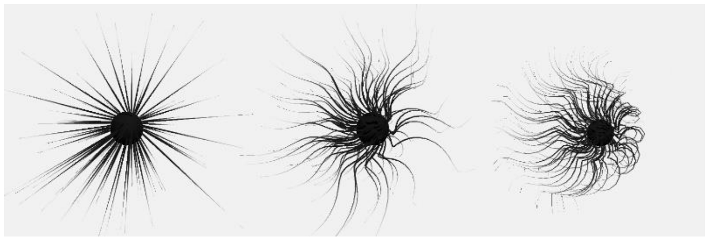
Figure 8: Zero gravity simulation of 100 strands of straight hair (extreme left); wavy hair (middle); curly hair (extreme right) on a sphere

Figure 8 shows Zero gravity simulation of 100 strands of straight hair (extreme left); wavy hair (middle); curly hair (extreme right) on a sphere.

IV. Result and Discussion

From the aforementioned algorithms, a complete visual representation of hair can be achieved with a relatively less CPU intensive simulation.

As the number of hair strands increase, the load on the CPU also increases with a higher chance of it causing a bottleneck. This causes the hair simulation to retard and be visually glitchy. Table 1 shows geometrical data of the simulation based on the amount of hair strands and segments generated. The algorithms presented in the paper can be very easily combined and a hair simulation model can be created on any game engine. C# programming language in Unity3D offered by Unity Technologies has been used for all simulations provided in this paper.

Number of strands (s)	Number of segments per strand (n)	$n \times s$	Vertices (including 10.3k vert main mesh model) unlit
100	4	400	11.1k
500	4	2000	14.3k
1000	4	4000	18.3k
2000	4	8000	26.3k
4000	2	8000	26.3k

Table 1: Geometrical data with defined variables

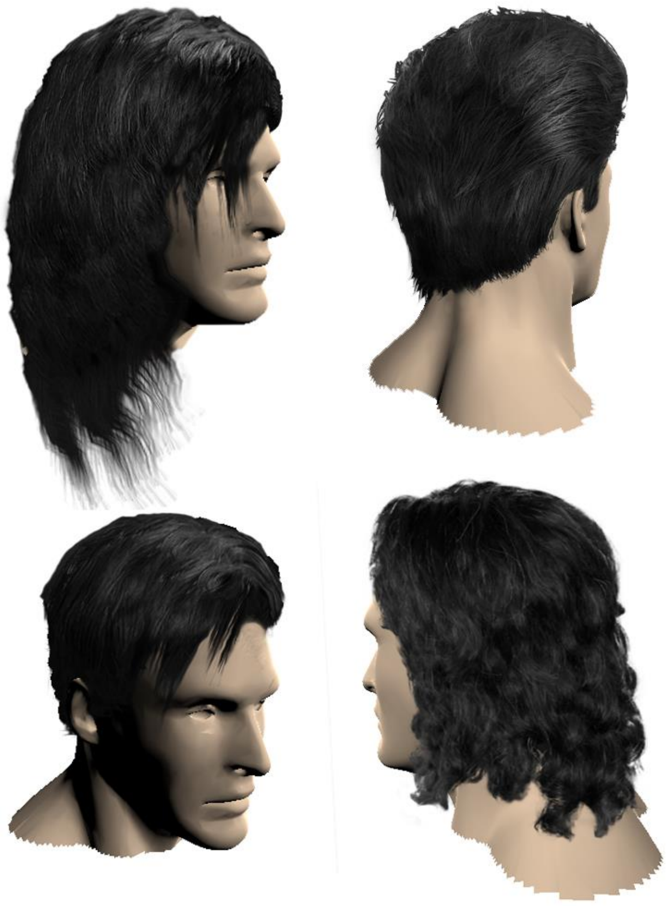
Figure 9: Output of hair simulation algorithms on a scalp containing 112k vertices having 10k hair strands

Figure 9 is the output of hair simulation algorithms on a scalp containing 112k vertices having 10k hair strands with a varying number of segments per main divisions (Top, Sides and Back) in the scalp mesh. These can be performed on any mesh which implies that the hair grain can be adjusted to different parts of the sub-divided scalp mesh. In addition, to improve the appearance of the hair strands, a texture can be applied with a directional shader which can simulate real-time global lighting on to the hair strands. There are a few more extensions and applications as to what the above-mentioned algorithms are capable of on a game engine. For example, a linear string of hair strands can be interconnected horizontally to form a cloth. Real-time grass can also be simulated. For this, a grass texture can be applied on a plane, and the width of each strand can be increased to accommodate the texture. This results in a grassy texture covering a terrain mesh. Since quad used for a terrain mesh will typically contain only four vertices and two triangles, the terrain mesh should be sub-divided to ensure there are enough roots for the grass. Figure 10 shows a wireframe of the terrain and grass blades containing 1000 hair strands with 4 segments having a total of 10.3K Tris (left) and rendered hair strands with grassy texture (right).

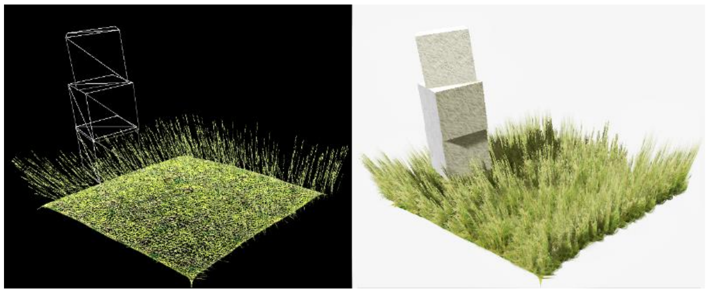
Figure 10: Wireframe of the terrain and grass blades containing 1k hair strands with 4 segments having a total of 10.3K Tris (left) Rendered hair strands with grassy texture (right).

Figure 10 here shows a wireframe of the terrain and grass blades containing 1k hair strands with 4 segments having a total of 10.3K Tris (left) Rendered hair strands with grassy texture (right). Since Unity3D has been used for the development of this simulation, it can be integrated into augmented and virtual reality applications with minimal changes. This project can be exported as a module which can additionally help indie developers with character design/terrain design as this is highly scalable and customizable. Post-processing features, for instance, anti-aliasing can be used on hair strands to reduce the serrated appearance. Another aspect to enhance this model is the use of shaders. A semi-transparent shader would be required on each hair strand and that texture would be continuous through all the segments in the hair. This texture can either be tiled or stretched throughout the length of the hair. There are still more flexible options that the developer can use for mapping a texture to the strand when using a shader.

A drawback worth mentioning in these algorithms is that they do not account for hair-on-hair collisions, static friction or cohesion and, therefore, there is interpenetration in hair strands. However, empirical observations show that hair-on-hair collision calculations do not make a huge impact on the visual appearance of the simulation, but they do get pronounced when viewed up close. [13]

V. Conclusion

In this research, a practical solution on the game engine Unity3D is proposed to carry out a realistic looking real-time hair simulation. This technique could be employed by game developers to create realistic straight, long, short, curly and wavy, wet or dry hair without the model taking a toll on the CPU resources. Although, if the number of strands are excessive, it could cause a bottleneck in the simulation. As a suggestion for future research, the technique could include clump of textured hair with pre-defined collision bodies, rather than having strands, to speed up the process even further. The head model could have a provision for inverse kinematics as a part of the hair rig and also include bones. A valuable addition to this model could be inclusion of collision in between hair strands to add depth to the realism of this simulation.

Appendix

This project has been uploaded on Github (https://github.com/SachitMisra/Real-Time-Dynamic-Fur-and-Hair-Simulation-using-Verlet-Integration) and is under review in the Unity Asset Store and will retail free of cost in support of open source research.

Acknowledgment

Unity v2019.1.1f1 offered by Unity Technologies was used for this simulation. The researcher is thankful to printable_models from free3D.com for providing the Australian Cattle Dog model and textures and 3dtextures.me for the grass and concrete texture.

References

[1] Choe, B., and Ko, H.-S. 2005. A statistical wisp model and pseudophysical approaches for interactive hairstyle generation. IEEE Trans. on Vis. and Comput. Graph. 11, 2, 160–S170

[2] D. Patrick, S. Bangay, and A. Lobb, Modelling and rendering techniques for african hairstyles, in Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa. ACM Press, 2004.

[3] Hadap, S. and N. Magnenat-Thalmann (2001). "Modeling Dynamic Hair as a Continuum." Computer Graphics Forum 20(3): 329-338.

[4] 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. 10.2312/PE/vriphys/vriphys12/045-051.

[5] Jung, S. and S.-H. Lee, Hair Modeling and Simulation by Style. Computer Graphics Forum, 2018. 37(2): p. 355-363.

[6] L. Chen, S. Saeyor, H. Dohi, and M. Ishizuka, A system of 3d hairstyle synthesis based on the wisp model, The Visual Computer, 1999.

[7] Landauer, W. (1925). On the Hair Direction in Mammals. Journal of Mammalogy, 6(4), 217-232. doi:10.2307/1373408

[8] Li, L., Li, R., &amp; Yu, J. (2016). A mass spring based 3D virtual hair dynamic system for straight and curly hair. 2016 35th Chinese Control Conference (CCC). doi:10.1109/chicc.2016.7554457 W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[9] Müller, Matthias & Kim, Tae & Chentanez, Nuttapong. (2012). Fast Simulation of Inextensible Hair and Fur. VRIPHYS 2012 - 9th Workshop on Virtual Reality Interactions and Physical Simulations. 10.2312/PE/vriphys/vriphys12/039-044.

[10] Oshita, M.(2007) Real-time Hair Simulation on GPU with a Dynamic Wisp Model. Computer Animation and Virtual Worlds, Vol. 18, Issue 4, Wiley, 2007. International Conference on Computer Animation and Social Agents 2007 (CASA 2007). June 2007. Belgium

[11] Petrovic, L., M. Henne and John Anderson Pixar. "Volumetric Methods for Simulation and Rendering of Hair." (2006).

[12] Plante, Eric et al (2002). Graphical Models (GMOD), Volume 64, Number 1, page 40-58.

[13] Poyart E., Faloutsos P. (2010) Real-Time Hair Simulation with Segment-Based Head Collision. In: Boulic R., Chrysanthou Y., Komura T. (eds) Motion in Games. MIG 2010. Lecture Notes in Computer Science, vol 6459. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-16958-8_36

[14] Z. Xu and X. D. Yang, V-hairstudio: an interactive tool for hair design, IEEE Computer Graphics & Applications, vol. 2001.

Authors

First Author – Sachit Misra, 2nd Year Computer Science Engineering, SRM Institute of Science and Technology, sachitmisra01@gmail.com