Hair Simulation Using Cuda and OpenGL

Abdallah Dandashli & Salah Mneimneh

ayd06@mail.aub.edu / sm20@mail.aub.edu

American University of Beirut / EECE 696 Parallel Programming

Abstract

We intend on simulating the movement of hair threads, taking into consideration different factors that affect their movement such as air flow, gravity, and collision interaction. The applications that use the current implementation of hair simulation are vast, ranging from animations, video games, to CGIs in movies. The use of this simulation can extend to other applications, such as aiding the analysis done in aerodynamics and it might even allow creation of new technologies that could use hair (or fur) on them for various benefits that our simulation provides. Since this is a simulation, we have many different elements in play, the high number of hair strands we need to simulate and constant collision detection.

Introduction:

Our project aims to enhance the speed of hair simulation by primarily identifying key functions that bottleneck performance, then recognizing which of its aspects can be parallelized to benefit from the GPU's capabilities. Finally, to implement a solution to increase performance.

Benefits:

Increasing performance of hair simulation would allow researchers in the field of hair care to add more complex attributes to their hair models, in order to create more accurate prototypes that require exact physical and chemical models for virtually testing and developing products.

Modeling hair is essential to computer graphics, it is important for creating convincing virtual humans for many diverse CG applications, interactive systems, such as virtual environments, videogames and the entertainment industry.

Hair Modeling Categories:

Hair modeling includes three phases: Hair styling, simulation, and rendering.

1. Hair styling is attaching hair to the scalp,giving it an overall or global shape,and managing finer hair properties.

   Many methods proposed include 2D distribution mapping to a 3D model. A 3D attaching system and an automatic distribution system.

   It is also desirable to be able to give a specific shape using parametric equations, or manually shape bulks of hair or generating hair from images or fluid mechanics. It is also required to give the hair specific properties which simulate the appropriate physical and chemical responses.

2. It is difficult to provide a realistic model for dynamic hair because each individual hair strand has a complex mechanical behaviour. Animation of a full head of hair is computationally expensive.As a consequence,existing hair animation methods propose a tradeoff between realism and efficiency, depending on the intended application.

   Hair simulation is the actual calculations being performed whenever needed,such as the force of each hair on another, or external forces acting on the hair as well as the movement of the animated object with the hair. In our project we specifically tackle the effect that wind has on hair strands.

3. Hair rendering deals with the final visual representation of hair,this is more important in the entertainment industry where visual fidelity is most important; in a research, hair rendering is as important as the other categories since scientists care more about the numeric results. The most worked on method in creating realistic hair rendering is by multiple or single scattering of light in hair and self-shadowing of hair.

Code:

We will first explain how the code works then dive into the details of each function.

In general, the way our hair simulation model works is that we start by introducing some starting points for the strands on an allocated area for the head only in ¾ of the region of the circle representing the head. After that we need to put the interpolated points,if they exist, and the end points. The idea behind the interpolated points is that they will partition the hair in order for us to be able to move individual partitions to simulate the free movement of a hair strand. Now we need to check for collisions after setting the interpolated points and after ensuring all points are in the correct locations we can apply wind.

In our implementation, the horizontal axis is the z-axis and the vertical is the y-axis due to the rendering implementation and to get a sideway view of the person.

SetStartPoint:

First step is setting the starting point of the hair strand. We sequentially iterate over each hair strand to calculate its x, y and z coordinates. Even though our model is 2D only, however a 3rd dimension is needed in order to get a 3D feel.

The function starts by randomly assigning a value to the z-axis which is bounded by [-headradius, headreadius]. The function rand() gives a value between 0 and RAND_MAX, there we divide rand() by RAND_MAX to get a random value between 0 and 1 then multiply by 2*headradius and subtract by headradius, which effectively gives a random value between - headradius and headradius. A -10 is also introduced to the equation in order to prevent hair from starting on the face.

In order to calculate the y-axis coordinate, we use pythagoras to give us the maximum distance we can get from the z point we have to the head perpendicularly. Then assign randomly a value between that [-distance, distance] except if z is greater than 0, then only between [30%distance, distance], this is done so that we leave space for the face.

The x-axis is then limited between the pythagoras equation for a 3rd dimension triangle in 2 directions. The value is in front or in the back (-ve or +ve) randomly.

We tried to perform a GPU implementation of this function by giving each thread 32 hair strands to set their starting points, however we faced several errors and problems and commented it out due to insufficient time.

SetEndPoint and Interpolated points:

Hair grows curving back and down from the head. First we set the endpoint for each hair strand at a 45° angle above and behind the starting point.After this if the hair strand has interpolated points we set them in order to give us a better curvature for the hair. Here we face the problem that an interpolated point might appear inside the head which we solve by introducing a collision detection function explained in the next paragraph. Finally we need to drop the hair down to give it a natural look. This is done by using spherical linear interpolation (slerp).

The idea is we want to rotate the hair until the first interpolated point or end point reaches near the head. The algorithm is split into two steps ,the first step is to set the first interpolated point in the correct position and if there are no interpolated points set the end point. The second step is in the case we have interpolated points, set them all after setting the first one and then set the end point. The bending is calculated from 2 rotation angles, the first rotation angle depends on three edges, the first one is between either of the first interpolated point (if it exists) or the end point and the starting point set previously. The second edge is between the origin and the starting point and finally the third edge is between the origin and the new point,the origin being the center of the head. After getting these distances the angle could be calculated by the following equation:

$$Angle(rad) =$$

$$acosf(\frac{SecondEdge^2 + ThirdEdge^2 - FirstEdge^2}{2 * SecondEdge * ThirdEdge})$$

Where first edge = the length of each hair partition
Second edge = the radius of the head
Third edge = Second edge + a factor 0.1f to give volume to hair.

The second step is similar to the first one, but here the angle also depends on the new location we calculated in step 1 since the interpolated points will follow from the first one so the equation here is

$$Angle2(rad) =$$

$$acosf(\frac{2 * SecondEdge^{2} - FirstEdge^2}{2 * ThirdEdge^2})$$

Where first edge = the length of each hair partition
Second edge = the radius of the head + a factor 0.1f
Third edge = Second edge.

After getting these angles and given the angle between starting point and Z axis call it angle0 we are able to determine the coordinates of these points depending on whether the sum of the angles exceeds the value of pi or not for both cases of points.We repeat this for all interpolated points and the end point.Finally we need to do collision detection in order to check if anything went wrong with the points and put them in the correct position if so.

The math here was difficult for us given we weren't familiar with slerp, but we were lucky enough to find plenty of helpful libraries and code to help us in this process.

Wind:

The basic idea here is that since we assume the wind is coming from the +Z direction so the hair must move upwards in the Y direction and to the left in the Z direction (Y vs Z dimension). The maximum distance Y could increase is bounded by the starting point of the hair and the strength of the wind.

We start by setting the end point to its correct position relative to the position of the starting point as well as the strength of the wind.Then we have that each interpolated point has a vector which we are able to move by using mathematical relations similar to those used in curving the hair(slerp). After getting the final positions of the interpolated points we apply collision detection to ensure everything is in the right place.

Collision Detection:

In order to detect collision, we must check if a hair point and its interpolated points are inside the head or not. So we calculate the hypotenuse (A) between the starting point of the hair strand and the origin point of the head as well as the hypotenuse (B) between the interpolated points and the origin. We then see if B is less than A + 0.1 (since we've put the hair above the head by 0.1 distance). If it isn't then a collision was detected.

After detection of any collision, we start fixing the point's position and it depends on whether that point is the first interpolation point, a point between the first then we check the endpoint. This is because changing the position of the conflicting point requires keeping the same distance from it and the points before it and after it as well as keeping the angle with the origin, since we don't want the hair to take another form, just move it upwards.

To fix them, we used mathematical relations that find the appropriate place for them with restrictions. First step is calculating the angle between the previous point and the origin (this is why we need to make a special case for the first interpolated point). Second step is to get the angle between the previous point and the conflicting point.

Then check if the hair should go directly down, or go around the head to correctly adjust the rest of the points of the hair. Finally, check the end point of the hair strand similarly as for the interpolation points (The first loop doesn't reach the end point).

GPU optimization:

It is evident that the problem is inherently parallel since we are applying the same process on each strand of hair. Our kernel code makes use of this by basically making the wind blow kernel be handled by every thread with a number of threads equal to the number of hair strands. We also made the collision detection a separate kernel to be launched along with the wind blow kernel and handle every hair strand.

The kernel's could be optimized to be applied to every interpolated point instead of every hair strand however this optimization is not worth the hassle since we would have alot of control divergence, and we would need alot of global memory accesses which is decreased by taking the strand instead of each thread taking a point.

Results:

The reported results at first showed that the time of CPU execution was varying a lot whereas GPU execution timing was fairly consistent. This leads to the CPU sometimes being a bit faster and this is due to the element of randomness in the collision detection function (Hair initialization is random and can cause varied collision) since with low collision the CPU will execute slightly faster. However running the simulation several times showed faster GPU execution on average.

After tinkering with the GPU configurations and using "Visual Profiler" to find efficiency problems, we were able to achieve consistent speed up even when there aren't many collisions. This was done by setting block size to 64 and adjusting the block count accordingly.

Conclusion:

This project achieves a speed up and gives quite interesting visual results and experimenting more gives different kinds of hairstyles. Since kernel implementations were only on wind and collision detection which are the bulk of the code, implementing other kernels on the other parts which initialize the hair could benefit the program.

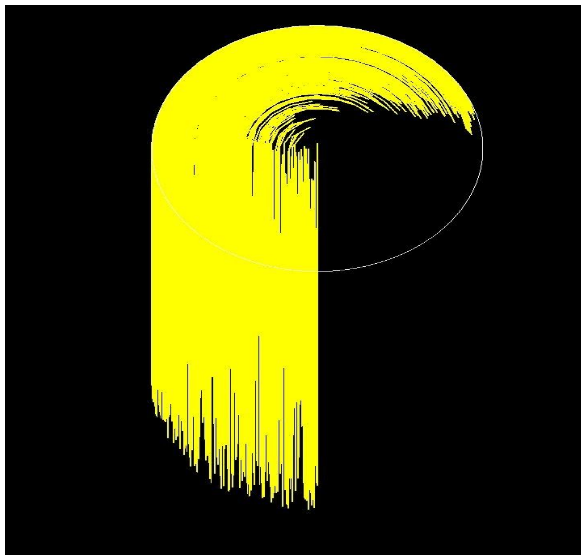


Fig.1 Hair Initialized

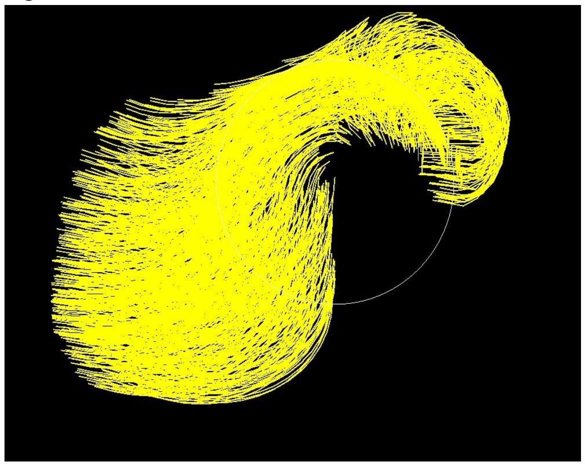
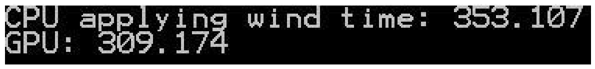
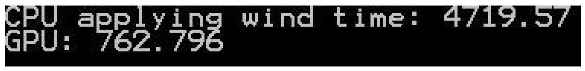


Fig.2 Hair After Wind


```
CPU applying wind time: 353.107
GPU: 309.174
```
Fig.3 Performance in msec in Release Mode (32 block size, GPU time went down to 255 msec with 64 block size)


```
CPU applying wind time: 4719.57
GPU: 762.796
```
Fig.4 Performance in Debug Mode (32 block size)

References:

NVidia HairWorks
https://developer.nvidia.com/hairworks

Pixar Animation Studios
https://graphics.pixar.com/library/Hair/paper.pdf

Kurihara T., Anjyo K., Thalmann D. (1993) Hair Animation with Collision Detection. In: Thalmann N.M., Thalmann D. (eds) Models and Techniques in Computer Animation. Computer Animation Series. Springer, Tokyo

H. Zhang, P. Zhang and J. Hou, "An Improved Collision Detection Algorithm for Hair Simulation Study," *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, Chongqing, China, 2019, pp. 1408-1411.

Pelechano N, Bull L, Slater M. "Fast collision detection between cloth and a deformable human body". Technical report, Department of Computer Science, University College London. 2002.