REAL-TIME RENDERING AND SIMULATION OF FUR

TSBK03, ADVANCED GAME PROGRAMMING

Rebecca Cedermalm
rebce973@student.liu.se

Sunday 23rd December, 2018

Abstract

Real-time rendering and simulation of fur is an interesting topic. To get at good-looking and realistic result a lot of data is needed. Simulating hair is however quite easy to do on the GPU, it is easy to do in parallel where each thread can handle one hair strand. This project was done by one student for the course Advanced Game Programming at Linköping University. The project uses the method proposed by Han and Harada [2] in 2012 to simulate the hair strands of the fur. This method is stable enough for real-time purposes and preserves the initial style of the fur. The method worked very well and the result looks pretty good.

1 Introduction

Real-time rendering and simulation of fur has long been a problem because of the large amount of data that needs to be processed. Moving the implementation from the CPU to GPU makes it possible to reach the real-time marks even on common laptops.

Concerning the rendering there has overall been two different kind of methods, the fin and shell-method and a method where hair strands are rendered separately. The fin and shell-method was proposed by Lengyel et al. in 2001 [1]. It uses volume textures to create a series of concentric shells of a semi-transparent medium and adds fins to improve the visual quality near silhouettes. The fins are 2D texture maps that are placed normal to the surface. The result from using the fin and shell-method is pretty realistic when it is seen from a far. Up close it is however not that good-looking. To be able to get a good result up close another method needs to be used. Rendering fur as hair strands gives better looking result, both up close and a far, but is much more computationally heavy compared to the fin and shell-method. Fortunately only about 10-25% of the hair strands need to be processed, the others can the be interpolated between these. A pretty recent simulation method that uses this kind of method to render the hair strand is proposed by Han och Harada in 2012 [2]. There are a lot of different simulation methods for fur. The advantages of using this method is that it is stable enough for real-time purposes and preserves the initial hair shape. This makes it possible to groom the hair into certain hair styles which then will be somewhat kept through the simulation.

The reason that I wanted to do a real-time rendering and simulation of fur was that I wanted to play around with the different kind of shaders. Earlier I had only touched the concept of vertex shaders and fragment shaders. This made it very interesting to work with tessellation shaders, the geometry shader and compute shaders. The initial idea was to just work with the rendering of fur for real-time purposes. The simulation was added to make it more interesting.

2 Background

In this section the theory behind the implementation will be explained.

2.1 Rendering

Regarding the rendering of fur in real-time there would too much data to process if all the hairs were created beforehand, especially if the hair needs to be simulated. Therefore only about 10-25% of the hair is created, groomed and simulated. These hairs are called master hairs. More hairs will then be interpolated between these master hairs to get a fully fur covered object. The amount of interpolated hairs can be implemented to take into account the distance between the object and the camera. If the object is far away a high density of fur is not needed because it will not be seen anyway.

2.1.1 Lighting

There has been a lot of research in how the light gets traced through a hair strand. Unfortunately are most of them very computationally heavy which means that they are not so suitable for real-time purposes. The method proposed by Kajiya and Kay [3] in 1989 is pretty old by now but it is still used in games when it comes to rendering hair. Their model is divided into two parts, a diffuse and a specular component. The geometry for deriving the model can be seen in figure 1.

The diffuse component is obtained by integrating a Lambert surface model along the circumference of the half cylinder facing the light source. Through a few derivations equation 1 gets obtained. If the tangent of the hair is pointing towards the light source the hair gets dark, something that is observed in nature.

$$\Psi_{diffuse} = K_d sin(t, l) \qquad (1)$$

where K_d is a diffuse reflection coefficient, t is the tangent of the hair and l is the light vector.

The specular component is a specular model in the same spirit as the Phong model modified

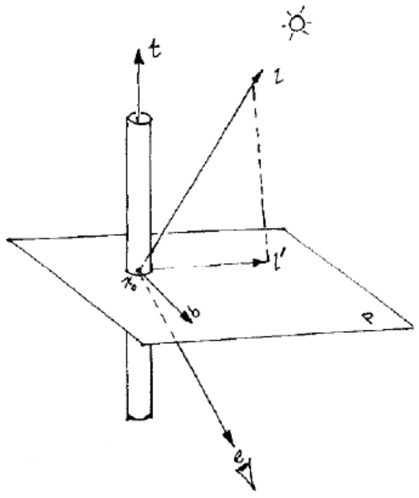

Figure 1: The geometry for deriving the light model.

to approximate the diffraction around hair. The equation can be seen in equation 2.

$$\Psi_{specular} = K_s(t * (l * t) * e + sin(t, l)sin(t, e))^p \qquad (2)$$

where K_s is a specular reflection coefficient, t is the tangent of the hair, l is the light vector, e is the eye vector and p is the Phong exponent specifying the sharpness of the highlight.

2.2 Simulation

The simulation of the hairs are done using a method proposed by Han and Harada [2] in 2012. The method is position based which means that the positions are handled directly instead of using forces. The simulation is done for each master hair strand and is made up of five steps: integration, global constraints, local constraints, edge/length constraints and wind.

The integration is used to apply external forces such as gravity and is further explained in section 2.2.1.The global and local constraints are used to keep the initial shape of the hair, similar to shape matching. These constraints are applied so that an artist can create an initial groom of the fur that will be somewhat kept during the simulation. The global constraint is explained in section 2.2.2 and the local in section 2.2.3. The edge/length constraint is used to keep the initial length of the hair and is explained in section 2.2.4. To make the simulation more interesting, wind was applied. The wind method is explained in section 2.2.5.

2.2.1 Integration

The integration step takes into account external forces, such as gravity. It is applied to the new position using verlet integration seen in equation 3.

$$\begin{aligned} P_{i+1} &= P_i + V + accel * \Delta t^2 \\ V &= P_i - P_{i-1} \end{aligned} \tag{3}$$

where P_{i+1} is the next position to compute, P_i is the current position, V is the velocity, $accel$ is the acceleration from the applied forces and Δt is the time step.

2.2.2 Global constraints

The global constraint is as mentioned used to keep the initial groom of the hair. The idea is to push the hair to the position it would be at if it was transformed rigidly. This constraint is handled in world-space. The constraint is illustrated in figure 2.

The constraint was calculated by following equation 4.

$$P_i = S_G(H * P_i^0 - P_i) \tag{4}$$

where P_i is the position of hair vertex i. S_G is the strength of the constraint which should be a value between 0 and 1. H is the head transform and P_i^0 is the rest position of hair vertex i.

2.2.3 Local constraints

The local constraint has the same meaning to it as the global constraint, it is needed to keep the shape of the hair. It is the most complex constraint of the ones used for this method and the goal for it is to apply a correction to each segment based on the local deformation and not the overall as was done for the global constraint.

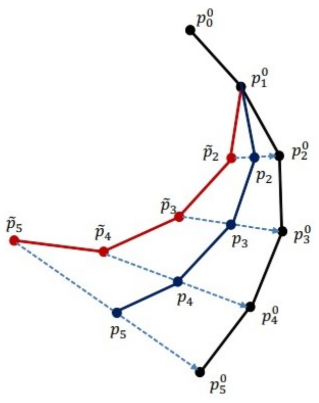

Figure 2: Illustration of how the global constraint works. The black line is the hair strand if it would be transformed rigidly. The red line is the simulated hair strand and the blue line is the final hair strand after the constraint has been applied to the simulated red line. Depending on the strength of the constraint the blue line can go from the red line to the black line.

As was seen in figure 2 the hair was bent quite uniformly from the original position using the global constraint. However the position of the hair vertex gets more corrected the more down the curve it is located. This is because the rotation of the segments is sort of cumulative. The local constraint, that can be seen in figure 3, tries to find how much a segment has been rotated relative to the original hair segment instead of how much it has been rotated overall.

The equation for the local constraint can be seen in equation 5.

$$\begin{aligned} d_i &= P_i^0 - P_i \\ P_{i-1} &= P_{i-1} - \frac{1}{2} S_L d_i \\ P_i &= P_{i-1} + \frac{1}{2} S_L d_i \end{aligned} \tag{5}$$

It works in the same way as the global constraint except that it is written in the local frame and moves two positions P_{i-1} and P_i instead of only one, P_i. The constraint is applied multiple times to achieve a stable convergence.

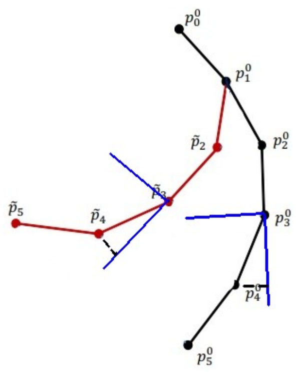

Figure 3: Illustration of how the local constraint works. The black line is the hair strand if it would be transformed rigidly and the red line is the simulated hair strand.

2.2.4 Edge/Length constraints

The edge/length constraints is used to make the hair strand keep its initial length. The distance between each hair vertex should always stay the same, which means that if it is not it needs to be fixed accordingly. The constraint is applied to each segment of the hair strand. Each segment is made up of two vertices, P_1 and P_2. These vertices need to be moved to be able to fulfill the constraint. It is done by computing the delta positions that need to be added to the positions through equation 6.

$$\begin{aligned}\overrightarrow{n} &= (P_1 - P_2)/|(P_1 - P_2)| \\ \Delta d &= |(P_1 - P_2)| - d \\ \overrightarrow{\Delta P_1} &= -\frac{w_1}{(w_1 + w_2)} * \Delta d * n \\ \overrightarrow{\Delta P_2} &= \frac{w_2}{(w_1 + w_2)} * \Delta d * n \end{aligned} \tag{6}$$

where P_1 and P_2 are the position of the vertices, d is the initial length between them and w_1 and w_2 are the inverse masses of the two vertices. Since all the hair vertices have the same mass w_1 and w_2 can be simplified to 1, which means that the vertices will move the same amount towards or away from each other.

Since the constraint is applied to each segment one at a time, the same vertex will be pushed twice which can make the constraint unfulfilled. This is fixed by using the constraint as an iterative process, which hopefully converges to a solution where the constraint is fulfilled.

2.2.5 Wind

The wind is applied as a random change in the positions of the hair vertices. A wind direction is chosen. This vector then gets separated into four wind vectors as seen in figure 4. This is done to generate random directions and avoid hair clumping.

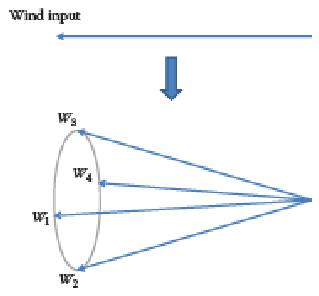

Figure 4: Illustration of how a single wind direction is divided into four wind vectors to be able to generate random directions and avoid hair clumping.

The wind magnitude is calculated using equation 7.

$$\text{wind magnitude *=}\ sin^2(\text{frame} * 0.05) + 0.5 \tag{7}$$

The wind direction is calculated using equation 8.

$$\begin{aligned} a &= ((\text{strand index})\%20)/20), \\ W &= a * W_1 + (1-a) * W_2 + \\ &+ a * W_3 + (1-a) * W_4, \\ V &= P_i - P_{i+1}, \\ f &= -(VxW)xV, \\ P_i &= P_i + f * \Delta t^2 \end{aligned} \tag{8}$$

where a is only used to get some randomness into the direction between the hair strands.

20 is only a arbitrarily chosen number. W_1, W_2, W_3 and W_4 are the four wind vectors seen in figure 4. V is the tangent of the hair and Δt is the time step.

3 Implementation details

The application created was built using OpenGL, GLSL and C++. Details regarding the rendering can be found in section 3.0.1 and regarding the simulation in section 3.0.2.

3.0.1 Rendering

The implementation follows the theory explained in section 2.1. The master hairs was created on the CPU. The first hair vertex of a hair strand was positioned at a vertex that belonged to the object. The second hair vertex was positioned a specified length away from the first vertex following the normal and so on. The hair data was saved into textures and sent to the GPU.

The fur shader program included a vertex shader, tessellation shaders, a geometry shader and a fragment shader. The vertex shader was mostly just a pass-through shader. In the tessellation shaders the amount of new interpolated hairs were decided. The number of tessellations depends on the area of the triangle and the distance to the camera. The most advanced shader was in this case the geometry shader where the hair geometry was extracted. Using the information from the textures created and the barycentric coordinates from the tessellation shaders the hair vertices of each hair strand was added. Some randomness was added to each new hair vertex to make it look more realistic. Lighting calculations were finally added in the fragment shader.

3.0.2 Simulation

The implementation followed the theory explained in section 2.2. It was implemented in a compute shader. Hair data was ping ponged between different textures to make writing and reading easier to follow.

4 Result

The resulting application runs in real-time on a laptop with the following specifications: Intel Core i7-6500U CPU @ 2.50GHz, 8 GB RAM and a dedicated GPU, Nvidia Geforce GTX 960M. A running example can be found on Youtube [4].

4.0.1 Rendering

Figure 5 shows the master hairs in their initial setup. Figure 6 shows how the fur looks like after the interpolation step has been done and figure 7 shows how the fur looks like after lighting is applied.

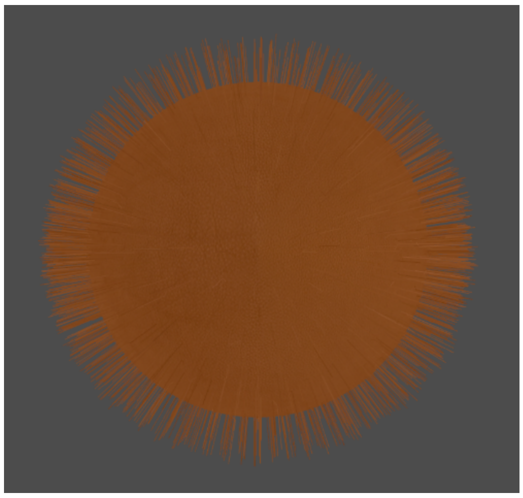

Figure 5: Illustration of the master hairs in their initial groom.

4.0.2 Simulation

Figure 8 show the fur when simulation is applied and figure 9 shows it when wind is added as well.

5 Conclusion

The methods chosen worked well to get a good looking real-time result. The method works for both simple and pretty complex models. Unfortunately it does not work with objects made

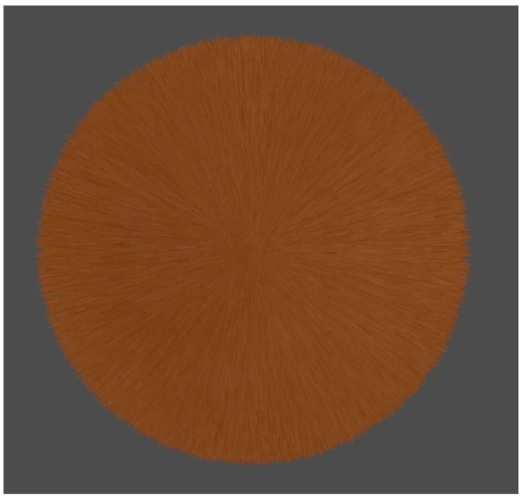

Figure 6: Illustration of how the fur looks like after the interpolation.

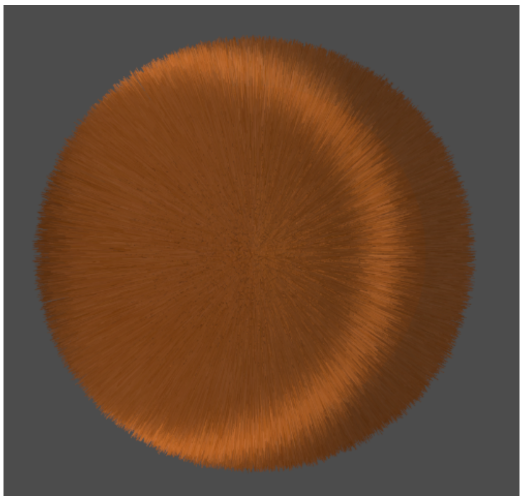

Figure 7: Illustration of how the fur looks like with the correct lighting applied.

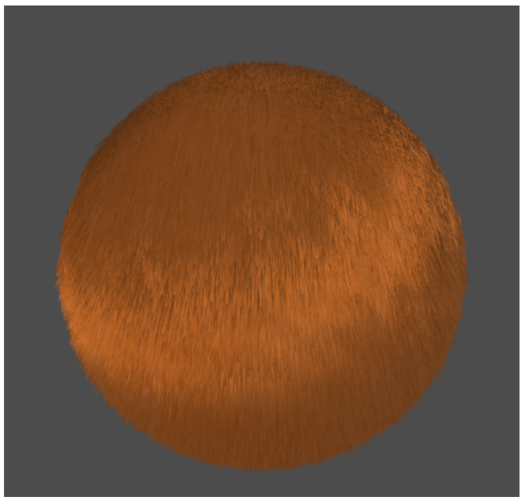

Figure 8: Illustration of how the fur looks like after simulation has been applied.

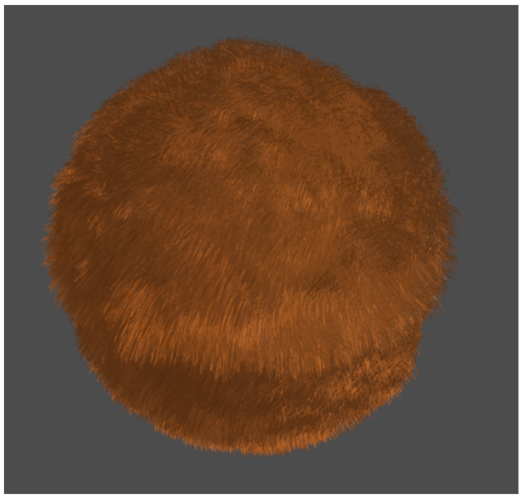

Figure 9: Illustration of how the fur looks like when wind is applied to the simulation.

up of too many vertices because of the choice of saving the hair data in textures. This could however be fixed if the hair data would be divided into several textures instead of only using one. Currently the implementation does not include any collision handling either, because it introduced some instability for the hairs on top of the object. This could have been because the initial groom of the hairs was unrealistic or a parameter set wrongly. Regarding the lighting of the fur, a few improvements would be nice to implement, such as shadows between the hairs and secondary high-lights which is something that has been observed in real hair.

References

[1] Jerome Lengyel, Emil Praun, Adam Finkelstein and Hugues Hoppe. ”Real-Time Fur over Arbitrary Surfaces”. In proceedings of the 2001 symposium on Interactive 3D graphics, 2001.

[2] Dongsoo Han and Takahiro Harada. Real-Time Hair Simulation with Efficient Hair Style Preservation. In VRIPHYS, edited by Jan Bender, Arjan Kuijper, Dieter W. Fellner, and Eric Guérin, pp. 4551. Aire-la-Ville, Switzerland: Eurographics Association, 2012.

[3] James T. Kajiya and T. L. Kay. Rendering Fur with Three Dimensional Textures. In Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques, pp. 271280. New York: ACM, 1989.

[4] Rebecca Cedermalm. ”Real-Time Rendering and Simulation of Fur”. Video of resulting application: https://youtu.be/UTkMCiZSpkA