Final Project Technical Paper

Ryan Drury

Abstract– Here I'll outline my attempt at simulating hair. I'll try to cover every detail of my implementation, and the results that were yielded.

I. Introduction

I was hoping to get as far as I could in simulating hair that would dynamically and realistically interact with its outside environment. I was also hoping to explore the rendering component, but my progress in that area was limited. I made a decent amount of progress with respect to the physics, although they aren't quite stable enough to be used in practice. I tried to implement collision detection between the hairs and the model in which they were rooted, but the algorithm was very inefficient and it slowed down to a halt. It is also very unstable and unreliable. The hairs have a stiffness to them that can keep them away from the scalp, so this isn't that much of a problem. I gave the user the ability to add and remove hair from certain faces of the model by clicking on them. The method for picking was ray tracing, as it was quite simple to extend my collision detection code for ray tracing. I also added a wind effect.

II. Implementation Details

Here I'll cover my methods for the physics, and the rendering I implemented. As a disclaimer, I am not that familiar with the official terminology for physics, and I will frequently use the word acceleration when I should use force, because in my program, mass is ignored.

A. Rod Physics

The model I choose to use for the strands of hair was a linked chain of rods that would try to be rigid. That is, they would always try to not compress or stretch, given the forces exerted upon them by their outside environment. To implement this, I used an adhoc method that made use of the spring equation and spring dampening. For each rod, I would go to the end points and compute an acceleration pushing them towards the center with a magnitude of:

$$\frac{l - L}{2}$$

where L is the length the rod should maintain and l is the current length of the rod. If nothing else was acting upon the two points, and if the two points had no previous speed towards or away from each other, then it would be guaranteed that the rod would have length L in the next frame. However, the two endpoints would then continue rocketing towards each other in the next frame, as opposed to stopping. To solve this problem, I included spring dampening.

To put the process of spring dampening into words, the amount of velocity that each end point has towards or away from the other is calculated, and then eliminated. To do this, I first calculated the average velocity of the two points:

$$v_{\text{ave}} = \frac{v_1 + v_2}{2}$$

I then computed the direction to the other point with:

$$d = p_2 - p_1$$

I then computed the speed that each point had relative to the average velocity by taking the difference:

$$v_1' = v_1 - v_{\text{ave}}$$
$$v_2' = v_2 - v_{\text{ave}}$$

I then computed the projection of these velocities onto the directional vector:

$$\hat{v}_1 = \text{proj}_d(v_1')$$
$$\hat{v}_2 = \text{proj}_d(v_2')$$

It happens that $\hat{v}_1 = -\hat{v}_2$, so you can compute $\hat{v}_2$ more efficiently that way. These two velocities represent how much the two end points are towards or away from each other. To eliminate this movement, we can subtract these velocities out of the picture:

$$v_1 \leftarrow v_1 - \hat{v}_1$$
$$v_2 \leftarrow v_2 - \hat{v}_2$$

So that is how I dampen velocities. Once I had these routines implemented in functions, I applied them as follows. First, I used the dampening function to make it so that the points have no velocity towards or away from each other. Then I moved the points based upon their new velocities. Then, at this point in the future, I applied a spring with a constant of $\frac{1}{2}$. So that computed the new velocity to use. I then moved the points back to their previous positions. I found this experimentally, and it seemed to work reasonably well as long as the gravity was low enough.

So the previous mechanics were responsible for preserving the lengths of the hair rods. The next thing I implemented was the stiffness of the hair. That is, I wanted the rods to exert forces on one another so that they would try to maintain certain relative angles to one another. I'll now describe my implementation of this.

I would begin a loop over the points in the chain of rods composing a strand of hair, and I would examine three points at a time. There is a middle point, p_0, and point to the left, p_1, and a point to the right, p_2. These three points determine two rods, which meet at a certain angle at the center point. Let $d_1 = p_1 - p_0$, $d_2 = p_2 - p_0$. These are the vectors emanating out from the middle point to the two neighboring points. The current angle at the meeting point between the two rods is calculated via:

$$\theta_C = \arccos\left(\frac{d_1 \cdot d_2}{||d_1||||d_2||}\right)$$

There is a specified ideal angle that the joint should obtain. Let this angle be θ_I. There is also a constant that behaves like a spring constant. Suppose this constant is denoted by k. Then the torque acting on the points is given by the spring equation:

$$\tau = k(\theta_C - \theta_I)$$

Once this is calculated, an appropriate acceleration needs to be applied to all three points. There are two intuitive notions that describe this acceleration. It must occur within the plane determined by the three points. The angle should only pull straight in, or push straight out. It should not try to twist the orientation of the points, or anything like that. Another thing is that it should act on the points perpendicular to the direction between the points. For this I defined a function, which I called antiProjection, and it just computed:

$$\text{antiproj}_u(v) = v - \text{proj}_u(v)$$

which is like the other component of the projection. That is, we can decompose v into the sum:

$$v = \text{proj}_u(v) + \text{antiproj}_u(v)$$

where $\text{proj}_u(v)$ is a multiple of u and $\text{antiproj}_u(v)$ is orthogonal to u.

So to take care of the acceleration being orthogonal to the relative positions, and for the acceleration to be within the plane determined by the three points, I computed:

$$c_1 = \text{antiproj}_{d_1}(d_2)$$
$$c_2 = \text{antiproj}_{d_2}(d_1)$$

Once obtained, c_1 is a vector that is orthogonal to d_1, and is within the plane spanned by d_1 and d_2. Similarly, c_2 is orthogonal to d_2, and is within this plane. The desired accelerations for p_1 and p_2 will be multiples of c_1 and c_2 respectively. I then normalized c_1 and c_2 and scaled them by the magnitude of the acceleration to apply:

$$c_1' = \frac{c_1}{||c_1||}$$
$$c_2' = \frac{c_2}{||c_2||}$$
$$a_1 = ||d_1||\tau c_1'$$
$$a_2 = ||d_2||\tau c_2'$$

I scaled the amount acceleration by the distance between the respective points because I had usually seen things like that when I had seen torque, but I can't completely justify or explain my use of it there. There may be a better, or more correct way. My strands became very unstable if they had more than around 15 links, so maybe this is related. Anyways, once the accelerations were calculated, I applied them as follows:

$$v_1 \leftarrow v_1 + a_1$$
$$v_2 \leftarrow v_2 + a_2$$

To keep it so that the net force on the system was still zero, I subtracted the velocity of the center point by these accelerations:

$$v_0 \leftarrow v_0 - a_1 - a_2$$

So that did it. A good value for the k was $-.4$. Anything smaller seemed to fold too easily and things higher than that would start strobing back and forth.

So those are the two main procedures I used to maintain the behavior of the hair. For the angular springyness, I did a single pass through the chain of rods, calling the above procedure on every triple of adjacent points. For the procedure that tries to maintain the lengths of the rods, I did n passes through a chain of length n, which yields an n squared algorithm. This seemed to dramatically improve the behavior though, so I kept it.

I'll now describe the physics involved for keeping the hair attached to the model. I only attached hairs to the vertices on the model. To maintain the attachment, I set the position of the first point in the chain to the location of its assigned vertex on the model. In this way, the chain would get pulled along if the model was to move.

One complication created by this was the fact that the top point in the chain was unaffected by gravity, while the rest of the points in the chain were. Since the length maintenance code makes use of the velocity, this created a strange effect. To compensate, I would treat the first point differently during these calculations, accelerating it upwards by negative gravity before so that it would appear to be moving away from the other points, and then reseting its velocity back to zero afterwards. This seemed to help.

To get the hairs to go off the scalp and fight gravity, I used the angular spring procedure to give the first rod a tendency to try to align itself with the vertex normal at the vertex it was attached to. When gravity was sufficiently low, this worked pretty well and collision detection between the scalp and the hair wasn't needed too badly. It would be feasible to have the chain use an arbitrary vector to align itself with, and this would accomplish "combing" of the hair, but I did not implement such a feature.

A final feature I added near the end was wind resistance. I considered this as a function that operated on the end points of a single rod. The input to the function is a vector giving the wind direction and its magnitude. I compute the dot product between the normalized wind vector and direction vector for the current rod. If the two vectors were orthogonal, the I wanted maximum force. If they were parallel, then I wanted no force. So I defined the equation for the force to be:

$$a = (1 - \text{abs}\,(w' \cdot d'))\, w$$

And this seemed to work pretty well. I now realize that this is incorrect, as wind does not accelerate things in this way, but rather, averages the objects velocity with its own wind velocity. In that sense, something traveling the same speed as the wind will not undergo any acceleration, while my model above will continue to accelerate such an object. So that is a flaw.

B. Hair Rendering

A chain of points is placed at each vertex in the model. These chains emanating from the vertices are used as values for interpolation for the hair strands coming off the faces. To interpolate two chains of points, I interpolate the positions of each of the points along the respective chains. The result is a new chain of the same length, where each point is somewhere in the middle. I then performed a variant of bilinear interpolation to generate hair strands across faces. It is a little different because my code only does the interpolation over triangles, rather than four sided objects, so there are only three boarder values to interpolate between. I first interpolate between two of the border values. I then interpolate between this calculated interpolation and the third boarder value. A side effect of using this is that many hairs tend to bunch up near the third value. This could be fixed by using indexing for the parameters of the interpolation that compensated for this effect. But I did not implement this correction.

So that is how I perform the interpolation along the faces. I will now describe how I render a particular chain of points (generated by the interpolation) as a strand of hair. I was hoping to try many different techniques for this, but I ended up with sticking to a pretty minimalistic approach. I did not try to employ any kind of curved, parametric line. Each rod in the chain is rendered as a textured rectangular prism. It is a cheap rectangular prism, where only 3 quads are drawn for each rod. Two quads go down the center of the rod, intersecting perpendicularly such that their intersection is the rod itself (as a line in 3D space). The third quad is drawn at the roof of the segment. If the third quad was not there, then the hair would look like plus signs if you happened to be looking straight down the end of a hair. The quads were textured via a ppm image loaded at run time. The texture coordinates for each quad are determined completely by its position in the chain. Each chain uses the same texture coordinates, so every chain looks exactly the same, as far as the texturing goes.

For the lighting of the hair, I followed advice given to me by Professor Pang. Pang mentioned that it can be difficult to specify normals to hair, because you don't have a surface to work with. He went on to say that you can just use the tangent line of the hair, and that it will look alright. After implementing this, I am surprised that it actually did turn out pretty well. When the hair curls and oscillates, the lighting gives the impression of glossy, shiny hair.

To control where the hair was, I simply stored the memory to handle the hair being everywhere, but then also had an array that stored which faces had active or inactive hair. Inactive hair was not processed for the physics, and it is not displayed.

C. Collision Detection

I was able to implement functional collision detection, but the collision resolution was not very stable or reliable. The collision detection, although correct, was also incredibly slow for large amounts of hair, leading to speeds along the lines of 1 frame for every 5 - 7 seconds. I will first describe my methods for collision detection and the role that the code ended up playing in my program.

I planned to implement collision detection by looking for rods that intersected with triangles from the model. So I wrote a function that found the intersection point between a line and a plane. This function took, as its inputs, two points on the line, l_0 and l_1, a point on the plane, p, and the normal vector to the plane, n. The intersection

point is then calculated as:

$$z = \frac{n \cdot (p - l_0)}{n \cdot (l_1 - l_0)}(l_1 - l_0) + l_0$$

where no intersection exists when we have that the denominator, $n \cdot (l_1 - l_0)$ is zero. To derive this, you can consider the parameterized line:

$$l(t) = t(l_1 - l_0) + l_0$$

and then solve for a t such that $l(t)$ will be on the plane. That is, you can solve the equation:

$$n \cdot (l(t) - p) = 0$$

and then plug that t back into $l(t)$. So it wasn't too difficult to determine the intersection point between the infinite plane and the infinite line. To check to see if the line with end points, l_0 and l_1 had collided with the plane, all you had to check was if $t \in [0, 1]$, where t is the parameter mentioned in the above paragraph. The intersection point will be on the line between its endpoints, l_0 and l_1, if and only if $0 \leq t \leq 1$. I had to mess with this to get it so that rods wouldn't collide with faces that they were directly attached to at their root point. I never did get that completely stable and usable.

So we know how the find the intersection point, and whether to see if the intersection point is on the given line. Now we need to test to see if it is actually within the triangle. So suppose that the triangle is given as the three points, p_0, p_1, and p_2. We can let:

$$b_1 = p_1 - p_0$$
$$b_2 = p_2 - p_0$$

and then try to find x, y such that:

$$xb_1 + yb_2 = z - p_0$$

where z was the intersection point. If we could get the x and the y, then we would have that z was on the triangle if and only if $x, y \geq 0$ and $x + y \leq 1$. I calculated y as:

$$y = \frac{||b_1||^2 b_2 \cdot z - (b_1 \cdot z)(b_1 \cdot b_2)}{||b_1||^2||b_2||^2 - (b_1 \cdot b_2)^2}$$

and then once y was known, x could be calculated as:

$$x = \frac{b_1 \cdot z - b_1 \cdot b_2 y}{||b_1||^2}$$

Note that I really don't think this is the best way to do this, and that there probably is a better way. I just got this to work so I stuck with it. When I derived this, I started out with the initial equation and then I dotted both sides with b_1 and then another copy of it with b_2, solved for x in terms of y in one equation, applied the substitution in the other equation, and then came down to a single result.

So that completely covers my collision detection. When it runs, it sees if every rod is colliding with every face on the model, which takes forever when the hair has many subsegments to it.

For the collision resolution, I tried to reflect the velocity of the point sticking inside the model out so that it would bounce back out. This may work in certain cases, but it wasn't enough for when gravity was trying to push a hair through a face.

Although the collision handling more or less failed, I was able to use the code for the collision detection to implement a ray tracer, which I used for picking the model's faces with the mouse cursor. I used the position of the camera as l_0 and then used information about the camera (where it was looking and it's orientation) as well as the mouse's position on the screen to create a point l_1 that was on the plane one unit away from the camera such that l_1 was always directly underneith the mouse cursor. I then relaxed the condition from $t \in [0, 1]$ to just $t \geq 0$. I then ran the intersection point calculation code for every face on the model, and returned the face that minimized t. The face that minimizes t is closest to the camera. This picking was used for spawning and deleting hair off of faces by changing the flags that affect their hair status.

III. Conclusion

Using the blocks for the hair, as well as the interpolation made the hair appear to be pretty nappy. The wind effect looks very interesting if blown in the opposite direction of gravity. The movement is almost similar to that of fire, but the movements are at a slower pace. I wasn't able to get as far as I would have liked. It would have been interesting to see what a successful implementation of collision detection would have looked like. I was unable to handle twisting of the hair stands. That component of the physics is ignored completely. It would have been very interesting to explore more about the rendering component of the project. I feel like I touched the surface of all three of the areas, but not quite enough to create something that would be useful for an artist. But, still, it is interesting and fun to play with.

References

[1] Florence Bertails, Basile Audoly, Marie-Paule Cani, Bernard Querleux, Frederic Leroy, Jean-Luc Leveque. "Super Helices for Predicting the Dynamics of Natural Hair." ACM Transactions on Graphics (Proceedings of the SIGGRAPH conference) - August 2006. Web. 31 Jan. 2011. http://www.lmm.jussieu.fr/ audoly/research/hair06/index.html

[2] Florence Bertails, Basile Audoly, Marie-Paule Cani, Bernard Querleux, Frederic Leroy, Jean-Luc Leveque. "Super Helices for Predicting the Dynamics of Natural Hair." ACM Transactions on Graphics (Proceedings of the SIGGRAPH conference) - August 2006. Web. 31 Jan. 2011. http://www.lmm.jussieu.fr/ audoly/research/hair06/index.html

[3] Florence Bertails, Basile Audoly, Bernard Querleux, Frederic Leroy, Jean-Luc Leveque, Marie-Paule Cani "Predicting Natural Hair Shapes by Solving the Statics of Flexible Rods." Eurographics (short papers) - August 2005. Web. 31 Jan, 2011. http://www-evasion.imag.fr/Publications/2005/BAQLLC05/

[4] Bertails, F. (2009). Linear Time Super-Helices. Computer graphics forum, 28(2), 417-426.

[5] Bruderlin, A. (2000). A method to generate wet and broken-up animal fur. The Journal of visualization and computer animation, 11(5), 249-259.