Enhancing the Realism of Procedural Wind on Curves with Collision, Shielding, and Gusts

Arunachalam Somasundaram
DreamWorks Animation

Figure 1: Procedural wind blowing from screen right to left a) without collisions (causes the fur to crash through the skin), b) with only collisions and no shielding, and c) with both collisions and shielding (shielded areas are tinted with brown color).

ABSTRACT

Curves are traditionally used to represent assets such as fur and grass in CG. Applying noise fields to create procedural wind on curves has been an attractive method for its speed and variability. However, this makes the wind appear procedural, lacking in both its dynamic nature and its realistic interaction with objects. In this talk, techniques are presented to enhance the realism of procedural wind with collisions, shielding, and gusts. These techniques are in use in DreamWorks' productions providing significant realistic control over the wind.

CCS CONCEPTS

• **Computing methodologies** → **Procedural animation**; **Collision detection**.

KEYWORDS

wind, gust, curves, fur, procedural animation, realism, collision

ACM Reference Format:
Arunachalam Somasundaram. 2019. Enhancing the Realism of Procedural Wind on Curves with Collision, Shielding, and Gusts. In *Proceedings of Siggraph 2019*. ACM, Los Angeles, CA, USA, 2 pages. https://doi.org/10.1145

1 INTRODUCTION

There are several types of 3-D noise fields (such as Perlin, Billow, or Wavelet) [Lagae et al. 2010] that can be applied to curves to add wind effects. This procedural approach is enticing for a variety of reasons including speed, low memory, control, variability, and evaluation at any time frame independent of other time frames.

However, using noise fields makes the wind look procedural overall. Realistic wind is dynamic in nature and it can occur as varying gusts. Also, the curves must physically and realistically collide with collision meshes, such as the growth skin. Several fur curves will also be shielded from the wind by collision objects amounting to receiving less wind. Adding collision, shielding, and gusts enhances the realism of procedural wind on curves.

2 BASIC PROCEDURAL CURVE WIND

Applying noise field values to directly offset curve point positions has issues. Further post processing is needed, such as applying length correction to the curve segments. Also, the curve topology needs to be taken into account to make the curves appear more like they are acted upon by wind than just a noise field offset.

In our approach, we use a custom node that can rotate the curve segments hierarchically based on calculated directions, starting from the root segment. Let P_{tip} be a curve's tip point and P_1 be a segment's start point. At that segment, we calculate a direction $dir_{seg} = dir_{wind} + dir_{noise}$, where dir_{wind} is the wind's main direction, and dir_{noise} is the noise field vector. Then, a quaternion Q is calculated from rotation of dir_{tip} to dir_{seg}, where $dir_{tip} = P_{tip} - P_1$. Q is used to rotate all the points of the curve after P_1, pivoting at P_1. Parameters $global_{rotate}$ and $ramp_{rotate}$ are used to scale the rotation angles globally and along the segments respectively. The noise of any curve point P is sampled at a point P_{noise}, where $P_{noise} = P_{root} + (dir_{wind} * len_P) * waviness$. P_{root} is the curve's root point, len_P is the curve's length up to the point P, and the *waviness* parameter can be used to offset the noise field sampling position from P_{root} for all the points along that curve. An appropriate *waviness* value will cause a ripple or wavy effect to travel through the length of the curve. The hierarchical curve segment rotational approach, along with the $ramp_{rotate}$ and *waviness* parameters, take into the account the curve topology and provide more artist control along the length of the curve. A per point *envelope* attribute (defaults to 1.0) is provided to scale the rotation amount.

3 MESH COLLISION

The technique described below visually emulates the effect of the curves bending around colliders (like wind would flow around) to reach towards the wind deformed positions.

To resolve collisions, the curve segments are incrementally rotated (using quaternions) from their original position towards the wind deformed position until the stop condition ($condition_{stop}$) is reached, which corresponds to either a) a collision object hit, or b) the wind deformed position is reached. For a curve segment Seg_a (with vertices V_{a1} and V_{a2}), vertex V_{a2} and all vertices further down the curve are rotated about vertex V_{a1} until the $condition_{stop}$ is reached for that segment. While this resolves collision for Seg_a, this can cause curve segments further down Seg_a to interpenetrate the collision object. For a curve segment Seg_b (with vertices V_{b1} and V_{b2}) that already interpenetrates the collision object because of a prior segment rotation, vertex V_{b2} and all vertices further down the curve are rotated out of the collision object until Seg_b collision is resolved. This also uses the fact that when one segment is finished resolving collision, the next segment further down has its starting vertex (which is the ending vertex of the prior collision resolved segment) to be collision free. The algorithm iterates over each curve segment starting from the root segment and ending at the tip segment. The root of the curve is slightly offset (using the $offset_{root}$ parameter) from the growth skin surface, so the first segment can also undergo collisions similar to other segments. The number of steps to incrementally rotate can be controlled using the $rotation_{steps}$ parameter. An offset scale parameter, $offset_{scale}$, is also provided to offset the wind deformed curve vertices from the growth skin along the closest point skin normal. This can be used to try to maintain the original offset (volume) of the curves ($offset_{scale} = 1.0$) or produce compression effects such as when the wind pushes the curves against the skin ($offset_{scale} < 1.0$).

4 WIND SHIELDING

Curves shielded from the wind by colliders receive less wind. For every point P of a curve with n vertices, a ray R is cast from a point P_{offset} in the direction of the wind $wind_{dir}$ (normalized vector), where $P_{offset} = P - wind_{dir} * offset$. The $offset$ parameter can be used to set the distance from which the wind is blowing from. If the ray R, starting at point P_{offset} hits a mesh geometry collision object (currently curve collisions are ignored) on its way to its corresponding curve point P, a counter $count_{hit}$ is incremented for that curve. The proportion of the curve vertices that are shielded, $alpha_{curve}$, is calculated, where $alpha_{curve} = count_{hit}/n$. Then, $alpha_{curve}$ is scaled by an artist specified *shield* parameter to obtain env_{shield}. The procedural wind node's *envelope* attribute is scaled by env_{shield} to obtain the shielding effect.

5 WIND GUSTS

The appearance of wind gusts can be obtained procedurally or manually (if precise artistic direction is needed). In the procedural method, randomized spheres of influence are sent in the direction of the wind from a plane perpendicular to the wind direction and offset at an artist specified distance. A sphere's influence is maximum (1.0) at its center and falls off to 0.0 at its boundary, and each sphere corresponds to a gust of wind. Parameters such as *num_gusts*, *radius_range*, *radiusNoise_range*, *speed_range*, *startFrame_range*, *life_range*, and *decay_range* control the number of gusts and the range of values that control the spheres' size, shape, speed, origin time, life, and decay respectively. A dynamic and organic wind motion can be obtained using this method. In the manual method, the artist is provided with the ability to send gusts of wind where the influence of any gust is controlled by control objects such as spheres or in and out cubes which can be rotated, translated, and scaled in any direction and precisely keyed by the artist. The influence within and from the boundary of the control objects can be set by an artist specified ramp. Highly art directed wind gusts can be achieved using this method. The influence of the control objects is used to add to the curve point's env_{gust} value which is then used to scale the procedural wind's *envelope* attribute (clamped to 1.0). If needed, a simple curve jiggle simulation is run after the application of wind to soften the influence of the passing gust control objects.

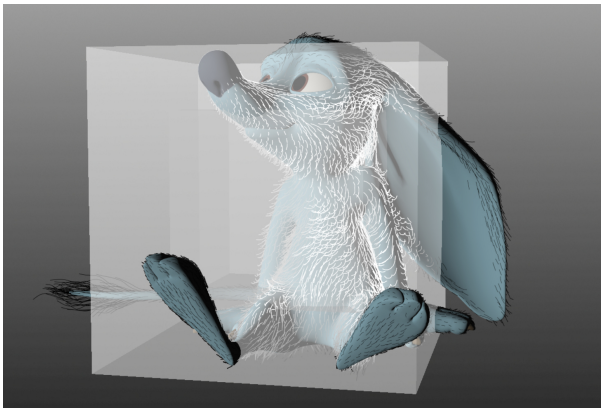

Figure 2: A manual gust control and *envelope* visualized

6 IMPLEMENTATION

The procedural wind and nodes that add realism were implemented in a node based procedural commercial package. The nodes were built using C++ and the package's expression language, and are multi-threaded. *Curve_Wind* node produces the actual curve wind. *Curve_Wind_Gusts* and *Curve_Wind_Shielding* nodes set the *envelope* attribute on the curve points before feeding them to the *Curve_Wind* node. If needed, the *Curve_Jiggle* node is then used to run the jiggle simulation during gusts. The *Curve_Wind_Collision* node finally performs the collision on the wind deformed curves.

7 CONCLUSION AND RESULTS

Adding collision, shielding, and gusts to procedural wind is an attractive approach to capture the dynamic nature of the wind and the physical collision aspects of it while also benefiting from the advantages of proceduralism. This has provided the artists with significant control over the wind. For 10,000 curves (≈50,000 CVs) , the entire process runs at about 5 fps on 30 CPUs. This technique has been used in DreamWorks' productions *Bilby* and *How to Train Your Dragon: The Hidden World*, and is in use in current productions.

REFERENCES

A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D.S. Ebert, J.P. Lewis, K. Perlin, and M. Zwicker. 2010. A Survey of Procedural Noise Functions. *Computer Graphics Forum* (2010). https://doi.org/10.1111/j.1467-8659.2010.01827.x