Hummingbird: DreamWorks Feather System

Nicholas Augello
DreamWorks Animation

David Tonnesen
DreamWorks Animation

Arunachalam Somasundaram
DreamWorks Animation

Figure 1: Feathers created using Hummingbird

ABSTRACT

This talk presents DreamWorks' Feather System *Hummingbird*, which is used for grooming body feathers and modeling scales interactively in real time. It is also used for feather motion such as secondary motion or wind, for feather special effects such as ruffling or puffing up, and for feather finaling. The system has been used in several shows at DreamWorks including *How to Train Your Dragon 2*, *Bilby*, and *How to Train Your Dragon: The Hidden World*.

CCS CONCEPTS

• **Computing methodologies** → **Shape modeling**; **Physical simulation**; **Procedural animation**.

KEYWORDS

feathers, grooming, simulation, procedural animation, finaling

ACM Reference Format:
Nicholas Augello, David Tonnesen, and Arunachalam Somasundaram. 2019. Hummingbird: DreamWorks Feather System. In *Proceedings of Siggraph 2019*. ACM, Los Angeles, CA, USA, 2 pages. https://doi.org/10.1145/8888888.7777777

1 INTRODUCTION

A bird typically consists of tens of large wing and tail feathers, and thousands of body feathers. This large quantity of body feathers in close proximity make feather grooming and feather motion a challenge. The feathers need to be interpenetration-free in their layout, wind effects need to be added during flight, and adding secondary motion enhances the beauty of the birds in motion.

https://doi.org/10.1145/8888888.7777777

2 GROOMING

The wing and tail feathers are modeled and rigged for animation, and the body feathers are created by *Hummingbird*. The interactive feather grooming is procedural in nature. First, the roots/follicles of the feathers are distributed uniformly as points on the skin based on painted density. Next, feather attributes are set on the follicles using either custom volumetric regions of influence or by painting onto the skin and transferring by proximity. A feather at this stage is represented as a simple mesh surface. The system currently takes up to three topologically similar feather model inputs from which the entire set of body feathers can be created. Any instanced feather is a blend [Seddon et al. 2008] between these models. The blending value (*Blend* attribute) is specified by painting a color (rgb) onto the skin surface; The *Direction* attribute specifies the initial direction of alignment for each feather; *Scale* controls the size (length and width); $Lift$ is used to raise or lower the feather away/onto to the growth skin; And $Loft$ is a function that defines how a feather is lofted, from root to tip, over the bird's skin.

Then, using the attributes on the follicles, feathers are initially instanced using a custom *'Feather Create'* node in the procedural graph. Any feather can also be interactively placed by moving its follicle or deleted by removing its follicle. The feather count can be further reduced via a custom *'Feather Prune'* node that prunes feathers based on an artist specified percentage of overlapping area. For each feather, its axis-aligned bounding box is projected along its normal axis to obtain a rectangle, which is used to approximate that feather's area. The overlapping area of the feathers in proximity is calculated using the overlap of its corresponding rectangles.

3 LAMINATION

The primary and secondary flight feathers are rigged and allow the animators expressive control. The body feathers, which cover bird's outer body, and the covert feathers, which cover the base of the flight feathers, are automatically layered using *Lamination*.

The main idea of lamination, as described in [Weber and Gornowicz 2009], is to avoid interpenetrations by construction. We start with a driver surface, feather root locations, and directions of

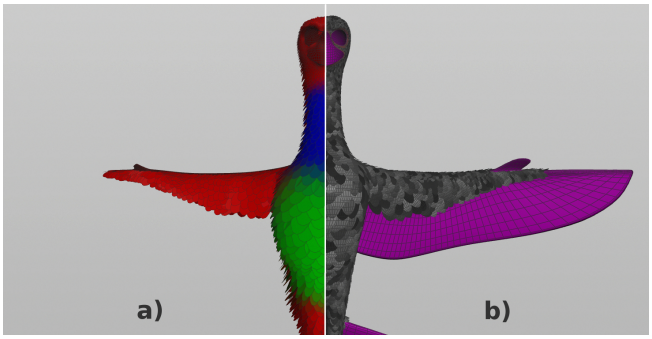

Figure 2: a) Initially instanced feather meshes, and b) Laminated feather meshes viewed along with the driver surface.

growth. The driver surface is an extension of the bird skin to include the volume occupied by the primary and secondary feathers. The directions of growth define a vector field that can be decomposed into a rotational and irrotational part. The irrotational part can be viewed as the gradient of a scalar potential, insuring well aligned directional vectors. The lofting function defines the height displacement of the feather above driver surface, so we can loft the feather from root to tip. The lofting function is created using a proprietary language, which is similar to a light weight version of Lisp. The combination of the irrotational vector field, the lofting function, and the driver surface allows us to define implicit constraint surfaces. We then grow one feather within each constraint surface, insuring the length and width of the feather is preserved. One of the features of the implicit constraint surface is that the normals do not cross within the domain, insuring intersection free feathers. However, when the driver surface is highly concave and the loft function values are large, this assumption no longer holds.

A novel use of the lofting function is to include additional values such as such as lift, wind, and noise. For example we have set up controls for the animator to easily fluff the breast feathers. We paint a scalar map on the driver skin to define areas for fluffing, and the animators key-frames a scalar fluff multiplier value over time. The product is fed into into the lofting function. We also have a *'Feather Collide'* node which computes distances between the driver and collision meshes, and is used to limit the final lofting values.

Grooming is often done in an iterative feedback loop workflow, where feathers are groomed, laminated, and then saved to disk. In the next grooming session, instead of generating new feathers via the *'Feather Create'* node, the previously laminated feathers can be used for the initial feather placement and orientation.

4 ART-DIRECTED MOTION

For art-directed feather motion in a shot, a custom *'Feather Rotation'* node is then used to rotate each laminated feather surface geometry locally about its axes pivoting at its root, either rigidly or with a bend. Rotation into the skin is restricted. Rotation attributes can be set for each feather by processes described below.

4.1 Wind

3-D procedural noise fields such as *Perlin* noise or *Wavelet* noise [Lagae et al. 2010] are used to drive the rotation values of the feathers. Two levels of noise fields, low and high frequencies, are applied to produce global wind and per-feather variation. Feathers are rotated slightly outwards away from the skin first to allow space for the new orientations. The noise amplitude is restrained to lie within that space to minimize interpenetrations.

4.2 Secondary Motion

Performing a full feather mesh physical simulation for secondary motion can be prohibitive. In our system, in a shot, the root points of the feathers are tracked per frame. To create secondary motion, a custom physical spring simulation is run on those points with the animated roots being the goal and with the spring lengths set to zero. For each feather, the difference vector between its simulated root point position and its original root point position is projected onto its local axes to determine its local rotation. The relative neighboring similarities of the simulation minimizes interpenetrations. The interpenetrations that can happen in fast moving areas, such as wings, are not noticeble due to speed and motion blur. The few interpenetrations that can occur elsewhere may need minor cleanup work. This simulation is cost effective in speed and memory.

4.3 Finaling and Special Effects

Special effects such as ruffling and puffing can be controlled by setting appropriate rotation attributes. A *'Feather Mod'* node was also built to specify the rotation attributes with a falloff from a point for manipulating feathers. The user can also directly modify the feather mesh vertices, if needed, to manipulate the feathers.

5 IMPLEMENTATION

The entire system is implemented in a node based procedural third party package. The grooming is interactive in real-time and the feather meshes can be visualized in motion in the 3-D viewer. The lamination process in a shot is frame independent and can be run in parallel across frames. The original feather lamination process was written as a set of libraries, command line executables, and integrated into our proprietary rigging system. Two years ago we ported the lamination process to a third party package. This consolidated the character effects artist's work into a single package, enabling the creation of the *Hummingbird* system.

6 CONCLUSION AND RESULTS

The *Hummingbird* feather system was used successfully in productions at DreamWorks for grooming feathers on twenty plus birds, with up to 25,000 feathers per bird. It was also used for grooming the scales on the dragon *CloudJumper*. For the approximately 25% of the shots that are hero shots the character effects artists would add in art directed motion. The remaining 75% of the shots were automatically laminated in batch.

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

Daniel Seddon, Martin Auflinger, and David Mellor. 2008. Rendertime Procedural Feathers Through Blended Guide Meshes. In *ACM SIGGRAPH 2008 Talks (SIGGRAPH '08)*. ACM, NY, USA, Article 76, 1 pages. https://doi.org/10.1145/1401032.1401130

Andrew J. Weber and Galen Gornowicz. 2009. Collision-free Construction of Animated Feathers Using Implicit Constraint Surfaces. *ACM Trans. Graph.* 28, 2, Article 12 (May 2009), 8 pages. https://doi.org/10.1145/1516522.1516523