Supplemental Talk Outline
Furry, Floppy, Fuzzy: *Once Upon a Monster's* Fur Pipeline

Peter Demoreuille* Oliver Franzke† Lydia Choy‡
Double Fine Productions

Figure 1: *Raw simulation data, interpolated strand curves, raw geometry, diffuse lighting, final result.*

1 Introduction

Sesame Street: Once Upon a Monster required Double Fine Productions to faithfully portray well-loved Sesame Street characters along with a set of furry monster companions. The resulting authoring tools, simulation, tessellation, sorting, shading, lighting and rendering techniques created an efficient and flexible fur pipeline that was used to create fur for every character in the title.

2 Overview

The vast majority of fur rendering in videogames can be classified as a "shells and fins" approach, where a fur-like appearance is generated by rendering successive overlapping layers of textured geometry ("shells"), coupled with camera-facing geometry along the silhouette ("fins"). This approach often leads to a fairly distinctive and uniform look, and can be expensive to render due to large amount of overdraw. The approach used in *Sesame Street: Once Upon a Monster* instead renders thousands of strands of fur, where the appearance and simulation of each strand can be individually customized by artists. This allows creating a wide variety of looks, makes continuous level-of-detail trivial, and can greatly reduce overdraw. Fur dynamics are inspired by hair simulation techniques, where a sparse field of key hairs are used to control groups of fur strands. Rendering uses an ad-hoc hair shader, spherical harmonic lighting, and a simplified opacity map to provide self-shadowing.

2.1 Authoring

The fur on each character consists of an arbitrary number of *fur layers*, which generally correspond to spatial regions of the character (eg, belly, arms, head). Each layer is rendered independently, and as such has complete control over the shader variables, textures, and other data which influences final fur appearance. While strands within each layer are sorted by depth, layers can influence layer-layer sorting order to help minimize transparency sorting issues.

Customizability and localized control are a priority when authoring fur, and thus a large number of variables have been exposed to control the appearance of individual fur strands. A single layer is built using data from three primary sources. A set of textures defined using the model's UV space controls per-strand properties including length, width, curvature, twist, stiffness, and offset from the skin. *Curvature* influences the degree to which the strand curves towards or away from the body, *twist* controls a strand's rotation around the normal, and *stiffness* controls the degree that each strand reacts to the simulation. A per-layer data file specifies material properties such as the shader, diffuse color textures, fur shader attribute textures, fake-shadowing controls, and specular parameters. The final dataset provides a vector "combing" field used to determine the layout direction of each strand. This data is generated using Maya, where artists may create a set of vector markers on the surface of the rigged model. When creating strands, this field is interpolated taking into account both distance between markers and the difference between the normal at the marker and strand root, allowing fine control in tricky areas such as faces and armpits.

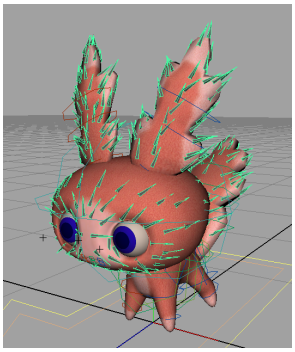

Figure 2: *Combing vector field in Maya*

2.2 Simulation

A simple two-segment chain simulation using several custom constraints drives the underlying fur movement. The simulation is updated by software-skinning the vertices to which each chain root is constrained, updating each chain vertex, then updating each fur strand by interpolating the nearest three simulation vertices. The key constraints providing fur-like movement are those limiting the angle between each segment and preventing penetration of the skin. Artist control is provided by allowing each fur layer to specify simulation parameters that influence the gross behavior of the simulation (such as damping and constraint strength), while the per-strand stiffness texture gives precise local control by dampening or exaggerating the movement of individual strands. To lower simulation cost without noticeable loss in fidelity, the artist can use a lower-

*e-mail:pbd@pod6.org
†e-mail:oliverfranzke@doublefine.com
‡e-mail:lydiachoy@mac.com

lod model to create the fur simulation data. This will allow a larger number of strands derive their motion from each key-hair, reducing memory and CPU consumption.

2.3 Transparency and backface culling

After mesh vertices have been skinned and the simulation updated, individual fur strands must be updated. Each strand's location is computed using barycentric interpolation of the mesh vertices, and tangents for a cubic curve used to deform geometry in the vertex shader are computed using the simulation vertices.

As fur uses alpha blending to create a soft and fuzzy look, strands must be drawn in a stable back-to-front order. To accomplish this, an approximation of the vertex-shader logic calculates a single depth-value per strand. A fast integer radix sort is used to sort the strands and generate an index array to determine draw order. Back-facing strands in layers with backface-culling enabled generate a sort key that forces them to the back of the index array, allowing these strands to be skipped trivially in several subsequent stages.

While strand draw order is fixed to use depth values, the draw order of layers is more customizable. By default, layer bounds are calculated by their member strands and sorted back-to-front, but additional controls are provided to bias and offset the final draw order. These controls have proven critical to address edge cases where layer sort order must be cheated. Attempts to eliminate this problem by sorting every strand regardless of layer have been made, but this adds significant overhead: it may require an arbitrarily large number of draw calls, sorting is slower due to increased cache utilization, parallelization is reduced, and peak memory usage is increased (making it difficult to implement this system in low-memory situations, such as on the PlayStation(R)3 SPUs).

2.4 Tessellation and level-of-detail

After each strand has been updated with current position and simulation data, sorted, and backface-culled, a vertex buffer is generated containing compact per-strand data. Geometry instancing (or custom vertex-fetch on the Xbox 360) is used to create strip geometry for each strand on the GPU, where the number of subdivisions in each strip is adjusted to reduce vertex and fragment shader utilization. Additional level-of-detail is provided by stochastically pruning the strands that are updated and rendered each frame. A uniform distribution of strands may be removed from the character by randomly permuting strand data offline, and prematurely ending updates after a certain percentage of the strands have been processed.

2.5 Shading and Lighting

Since the fur system simulates individual hair strands, we initially implemented shading model for hair fibers described in [Marschner et al. 2003]. However, its computational complexity and unintuitive artistic controls quickly presented issues. Due to the stylized art direction of *Once Upon a Monster* and the desire for a shader with more intuitive behavior, we decided to use the phenomenological model described in [Kajiya and Kay 1989], as the basis of our fur shading model. Because of the placement and length of fur strands, removing the additional calculations from the Marschner shading model did not cause a significant visual change while providing a significant speed boost.

Our final shading model supports one shadow casting key light supplemented with an unlimited amount of fill lights representated using spherical harmonics. A rim-lighting term based on the direction of the key light rounds out the lighting and provides a subtle glow to the fur's edges. Balancing the shader workload between per-vertex and per-pixel operations is critical for good performance. The vertex shader calculates the position of the tessellated vertex and several lighting terms. Some degree of per-vertex lighting provided an excellent tradeoff between quality and performance, as fur strands are tessellated enough to make interpolated lighting values look convincing. When computing final fragment color, the pixel shader first retrieves information about diffuse darkening, glossiness and transparency from an attribute texture; evaluates the shadow attenuation; and computes a final result by combining all of these values with the lighting calculated in the vertex shader. Our shader did not support per-pixel color maps for fur, instead the texture of the character was sampled at the root of the strand in the vertex shader. This made it also possible to "clean" the characters by dynamically modifying its texture (using a GPU-based volume painting approach).

While [Lokovic and Veach 2000] or [Yuksel and Keyser 2008] would provide high-quality fur self-shadowing, in practice they were too computationally demanding to provide acceptable performance. Our solution augments a traditional depth shadow-map with accumulated opacity information by rendering to a 4-channel color texture. The *RGB* channels have alpha-blending disabled and encode the depth value of the sample, while the *A* channel is additively blended, and accumulates opacity as the strands are rendered back-to-front into the shadow-map. This data can then be used to provide the characteristic semi-transparent edges of shadows cast from fur. Standard filtering techniques may be used when sampling this map, and provide an additional level of detail when shader complexity must be reduced due to high numbers of characters.

3 Conclusion

The fur pipeline described has been flexible enough to implement a wide varity of fur styles, efficiently utilizes the CPU and GPU, and is extremely artist-friendly while providing high fidelity results.

References

KAJIYA, J. T., AND KAY, T. L. 1989. Rendering fur with three dimensional textures. In *SIGGRAPH*, ACM, J. J. Thomas, Ed., 271–280.

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 385–392.

MARSCHNER, S. R., JENSEN, H. W., CAMMARANO, M., WORLEY, S., AND HANRAHAN, P. 2003. Light scattering from human hair fibers. In *ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, SIGGRAPH '03, 780–791.

YUKSEL, C., AND KEYSER, J. 2008. Deep opacity maps. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008) 27*, 2.