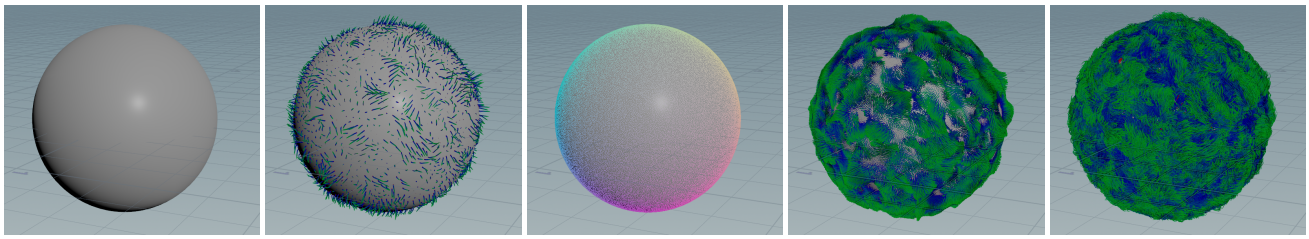
Improving Groom Interactivity in Houdini

Curtis Andrus
curtis.andrus@animallogic.ca
Animal Logic
Vancouver, BC, Canada

Beau Parkes
beau.parkes@al.com.au
Animal Logic
Sydney, NSW, Australia

Don Boogert
don.boogert@gmail.com
Animal Logic
Vancouver, BC, Canada



(a) Skin Geometry **(b) Guide Curves** **(c) Full Density Scatter** **(d) Length and Width** **(e) Scraggle**

Figure 1: Different stages in an Alfro groom, starting from the initial geometry (a) up to the full groom in (e).

ABSTRACT

To improve performance and interactivity working with our Houdini-based Grooming Tools, Animal Logic developed a set of custom nodes to control and optimize the process of evaluating our groom generation networks.

CCS CONCEPTS

• **Computing methodologies** → **Procedural animation**.

KEYWORDS

hair, grooming, workflows, interactivity, animation

ACM Reference Format:
Curtis Andrus, Beau Parkes, and Don Boogert. 2022. Improving Groom Interactivity in Houdini. In *The Digital Production Symposium (DigiPro '22), August 7, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3543664.3543678

1 INTRODUCTION

Animal Logic's Grooming Toolset, *Alfro*, has historically been built on top of a proprietary node-based procedural geometry system called *ALF*. However, creation/modification of nodes was limited to R&D developers, and it had limited support for our USD-based pipeline. SideFX's Houdini, with Solaris, consistently improving Grooming Tools, and high-levels of flexibility, was in a good position to solve these problems. Because of this, we've transitioned the artist-facing component of Alfro into a Houdini-based system over the past several years, with an initial version on *DC League of Super Pets* (2022).

Alfro fits into our larger Houdini-based Surfacing workflow built around USD. At a high-level, artists work in a LOP Network to bring in models, materials and other assets from our USD-based pipeline. This is then connected into a SOP workarea, where the low-level Alfro operations take place. Finally, the groom is brought back into the USD stage to be rendered.

One of the goals of this transition was to ensure that Alfro could be used by artists whose Houdini experience varies widely. To enable groom construction without the need for a complex Houdini network, we built our tools around the concept of a "stream," where a single connection in the network brings along all the information commonly used in a groom (e.g. groom curves, guide curves, skin geometry, etc.). We provide a collection of HDAs that perform common grooming operations (such as "Spiral" or "Clump") that work with the stream concept, as show in Figure 2.

Another requirement for our tools was the concept of a Groom Rig. Building bespoke grooms for each character is impractical at scale, so artists aim to build rigs that act as templates for a certain kind of style (e.g. double-coated animal fur). Most artists work with rigs instead of the low-level Alfro operations, quickly grooming a character by plugging guide curves and attribute maps into the rig's parameters.

We can represent groom rigs in Houdini directly with an HDA. However, getting sufficient interactive behavior from our groom rigs in this environment has proven been a challenge, for several reasons:

- Working in multiple contexts (Solaris, SOPs, etc.) can easily trigger a recook, especially if the artist hasn't carefully set up their Houdini workspace.
- Each cook evaluates the groom rig's entire network, which can get very heavy.
- Switching between different settings (such as preview and render) requires an extra recook to switch back.

DigiPro '22, August 7, 2022, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9418-5/22/08…$15.00
https://doi.org/10.1145/3543664.3543678

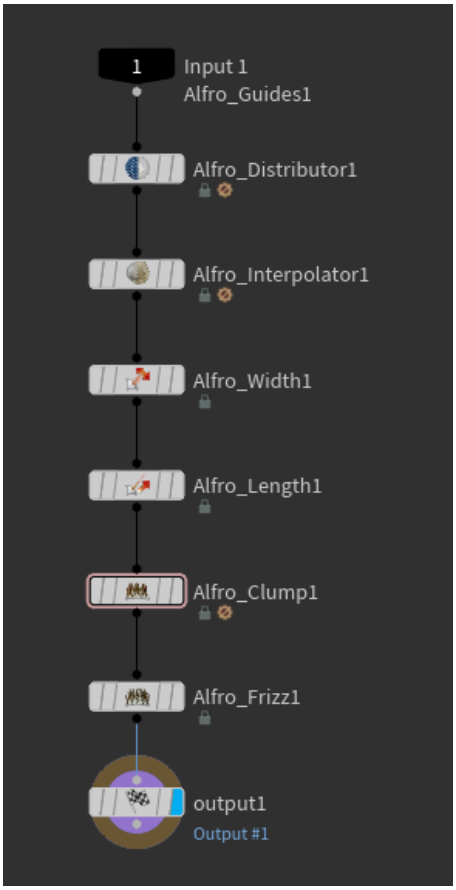


Figure 2: Example of a groom rig graph with Streams.

Some improvements can be made through better organization of our Houdini scenes, as well as directly optimizing Alfro's nodes. However, we wanted to decouple performance from the artist's set-up as much as possible. To do this, we built a couple custom nodes to wrap around important networks and managing the cooking of their contents. These custom nodes are described in the following sections.

2 CACHE NETWORK

With artists diving into inner SOP Networks to do their geometry work, then back out to the USD context to render the result, it is very easy to trigger a recook of a potentially expensive graph. In many cases, the cooks can be triggered from changes that have nothing to do with the groom itself. Houdini can often avoid recooking through the use of Data IDs [SideFX 2022], but in many cases the data ID can change even if the associated geometry has not, leading to a potentially unnecessary recook. Other cases, such as doing a render, involve the artist changing from some preview parameter set (e.g. 10% density) to higher quality settings (100% density). This results in one heavy cook before the render, followed by another cook to go back to the preview settings.

Houdini does provide several options to store results and avoid cooking (stashing / save to disk / etc.), but they largely depend on the artist knowing when something should be saved or regenerated. Other nodes, like the Cache SOP, are meant for caching results over time for faster playback. Neither of these options are practical for the problems described above.

We solve this problem with a node we call the *Cache Network SOP*, which stores a history of previous results (and the inputs/parameters used to generate them). If an internal node needs to cook with a previously seen parameter set and inputs, the Cache Network simply returns the stored result and avoids any internal cooking. To store the result history, we build a 128-bit hash (using the xxHash library [xxHash 2022]) from the parameter values in its internal subnetwork as well as the input geometry, and use that as keys into a cache table (shared between all nodes in the Houdini process). To avoid excessive memory usage, we limit the number of items in the cache and evict with a least-recently-used strategy.

As an example, we placed our point scatter node into a Cache Network and ran it on Houdini's Rubber Toy geometry. Uncached, the operation takes 33.5s to generate 7.9 million points. Subsequent cooks, however, hit the cache return the result in 0.005s.

The cache network node is placed inside the component nodes mentioned earlier. This hides any additional complexity from artists, and enables caching of all intermediate grooming operations.

3 SOLO VIEW NETWORK

Aside from run-time performance, there are many cases where a full evaluation isn't necessary for an artist to get useful visual feedback. One example would be working with clump curves (lower density curves that drive the shape of denser hair clumps). Artists can get feedback they need just by seeing the clump curves, and don't need to wait for the full density groom to be evaluated (which can be quite expensive). Other examples of useful intermediate stages can be seen in Figure 1, where each stage gives meaningful feedback. However, since our groom rigs always cook the full result, seeing the intermediate steps is not possible without unlocking and diving into the rig's details. Enabling intermediate steps would not reduce the full evaluation time, but would reduce the time to feedback.

One possible approach to this "solo" behavior would be to use a Switch node at the end of the Rig's network that chooses which node's result to output based on some rig-level parameter. However, this would effectively require artists to remember to build in this functionality manually, which could be troublesome depending on the their Houdini experience level.

To solve this, we built the *Solo View Network SOP*. When cooked, our node locates a child node (specified by a string parameter), cooks it and returns the result. The appropriate child node can also be automatically detected based on which rig parameters have changed (we call this "auto" mode).

To use auto mode, the artist creating the Groom Rig sets a "solo_path" tag on each parameter, indicating which child node is relevant. When that parameter changes, the Solo View node uses the tags value to find the relevant child node. This gives a fairly seamless experience for the user of a groom rig. A similar approach could be taken with parameter Python callbacks, but this would add a huge amount of extra complexity for the artist, and wouldn't handle cases like expressions.

Unlike the Cache Network node, this node wraps around the entire groom rig. Manually setting parameter tags can be time-consuming, so we provide a shelf tool to do a "best guess" tagging based on channel references. The artist can then make additional modifications where desired.

4 IMPLEMENTATION DETAILS

In these nodes, the cooking mechanism follows the same basic pattern: find the relevant node, cook it and return a result. The complexity comes from ensuring that our node maintains the correct dependencies and gets cooked at the right time.

The Solo View node dynamically adds a dependency on the selected child node, as well as any parameters (on its parent node) with the solo_path tag. The Cache node adds dependencies on input nodes, its internal output nodes and associated parameters.

4.1 Analytics Integration

One additional advantage of wrapping around the cooks of other nodes is that we can instrument these cooks for analytics purposes. In this case, Cache and Solo View SOPs enclose the calls to cook with our distributed tracing library (based on OpenTracing and Jaeger). The Solo View node captures the entire groom evaluation in a single span, and the Cache node creates a span for the individual groom operation. Because the groom operation cooks are contained within the Solo View cook, we're able to associate the individual operations with the larger groom, giving us a complete breakdown of what a groom rig is doing. This gives us groom rig performance information at a studio level.

5 PRODUCTION ROLL-OUT

We are currently rolling these nodes out into groom rigs for our active productions. The Cache Network is set up in nodes managed by R&D and Production Technology, and can be enabled/disabled through an environment variable. The Solo View Network is added by artists at the rig level, so they can choose whether or not to enable it when they build the rig.

Once these nodes are in widespread use, we plan to make more use of the tracing functionality to understand where further performance improvements can be made. We also see these nodes as general purpose "interactivity tools," and we hope to apply them to other areas beyond grooming, such as environment building and cloth tools.

REFERENCES

SideFX 2022. *Houdini HDK Data IDs.* Retrieved May 9, 2022 from https://www.sidefx.com/docs/hdk/_h_d_k__geometry__intro.html#HDK_Geometry_Intro_Data_IDs

xxHash 2022. *xxHash - Extremely fast hash algorithm.* Retrieved May 9, 2022 from https://github.com/Cyan4973/xxHash