Hierarchy Models: Building Blocks for Procedural Rigging

Michael Hutchinson
m.s.hutchinson@gmail.com

Sandy Kao
DreamWorks Animation
sandy.kao@dreamworks.com

Kevin Ochs
DreamWorks Animation
kevin.ochs@dreamworks.com

Gilbert Davoud
gdavoud@gmail.com

Alex Powell
alexander.powell@gmail.com

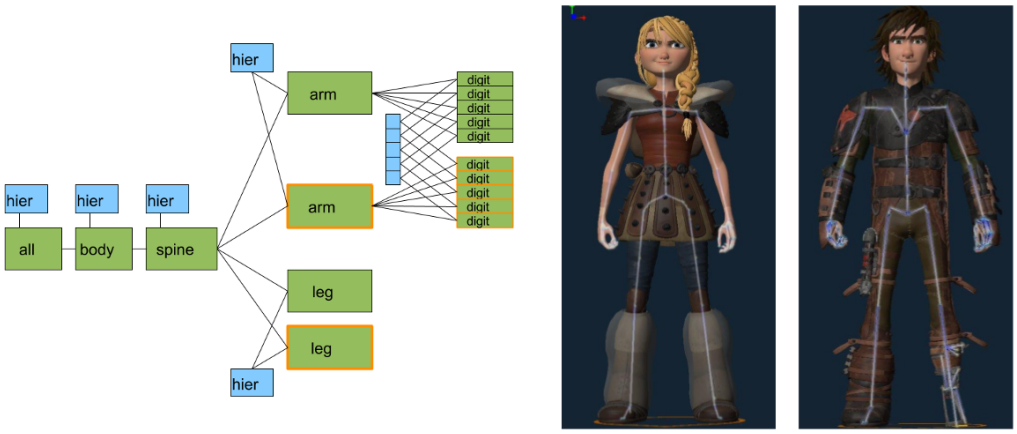

Figure 1: Hiccup and Astrid from *How to Train Your Dragon 2* using Hierachy Models

ABSTRACT

Hierarchy Models provide an encapsulation mechanism for joint hierarchies that yield an essential building block for procedural rigging. With Hierarchy Models, joints travel through dependency graphs (DGs) as an atomic entity. Operation nodes in the DG can modify all aspects of input hierarchy and even perform topological modifications like adding or removing joints. The Hierarchy Model reduces complexity in character rigs, improves separation between data and behavior, provides a clean interface, and simplifies understanding and debugging rigs. It offers geometric evaluation optimizations, and promotes parallelism in the DG structure.

KEYWORDS

Character Rigging, Procedural Rigging, Dependency Graph Optimization

ACM Reference Format:
Michael Hutchinson, Sandy Kao, Kevin Ochs, Gilbert Davoud, and Alex Powell. 2019. Hierarchy Models: Building Blocks for Procedural Rigging. In *Proceedings of SIGGRAPH '19 Talks.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3306307.3328160

SIGGRAPH '19 Talks, July 28 - August 01, 2019, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6317-4/19/07.
https://doi.org/10.1145/3306307.3328160

1 INTRODUCTION

In typical Digital Content Creation tools (DCCs), joints are represented as named, first-class entities that are organized into hierarchies with directed acyclic graphs (DAGs). Each joint has one set of values representing a single state. Complex behaviors are defined by layering transforms, adding constraints, and driving values through dependency graph nodes.

This works well for simple cases, but can become unwieldy as complexity increases. In production rigs it is not uncommon to see joints nested below a dozen intermediate transforms, each with a specific but esoteric purpose. Joints may be duplicated across multiple hierarchies, reverse hierarchies, or broken hierarchies, all tied together with complex constraints and connections. State and execution order become increasingly difficult to design, debug, and understand as execution weaves in and out of the DAG and dependency graph.

As the industry shifts towards greater levels of procedural-based solutions ([Nieto et al. 2018]), joint hierarchies are not able to take advantage of mass manipulation operations that we see in geometry. The Houdini procedural effects and modeling is a paradigm that has not been applied to joints and motion systems natively in a package, except at DreamWorks. The hierarchy model represents a solution that can be more in-lined with operations to geometry and use the same approach and methodology. It could also take advantage of SIMD graphics cards harware which exploits data level parallelism.

2 PROCEDURAL BUILDING BLOCKS

Rather than joints being represented by individual first-class objects, joints are components of the Hierarchy Model, much like a vertex on a poly model. Besides the base level transformation data, each joint component have additional attributes, including, but not limited to, parent, rotation order, scale propagation, flipping behavior, and color. The hierarchy can then flow through the dependency graph just like a geometric model with the output of each of these nodes representing a distinct state of the hierarchy that can be viewed and inspected independently.

Like geometric models, the Hierarchy Model can be modified procedurally in many different ways. Topological modifications can be made to add, remove, or re-parent joints. Or, more commonly, joints can be translated, rotated, or scaled, or have their transforms set directly. Constraints can be written that modify one or more joints in complex ways all represented by a single node with minimal inputs and outputs.

3 USAGE

As an example, consider the construction of a simple limb rig with the following behavior: independent IK and FK chains with a blend in local-space, and an additional layer of FK offsets on top for subtle secondary motion.

Using first-class joints, each of these states must be represented using duplicated (but slightly different) explicit hierarchies. Local space blends and FK offsets are difficult to set up, and rely on joint properties like rotation order staying in sync between the different versions of the hierarchies. The DAG and dependency graph become difficult to trace and lack meaningful semantic clues about intended behavior. (Figure 2)

However, using Hierarchy Models, the dependency graph simply looks like a flowchart used to explain the desired behavior (Figure 3). The construction of the graph flows naturally from an intuitive understanding of the desired behavior, and provides good semantic insights into intended behavior when later debugging or modifying.

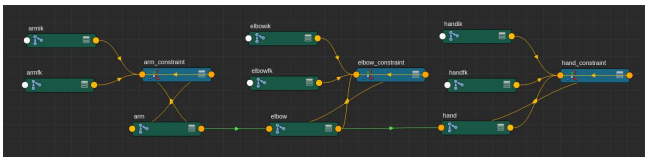

Figure 2: Typical DAG based joints in fk and ik setup

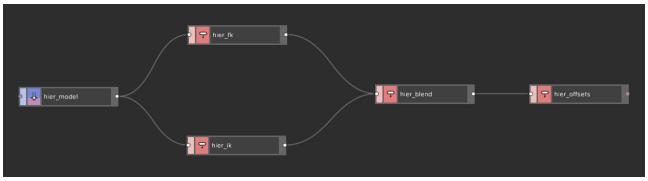

Figure 3: Hierarchy Model joints in fk and ik setup

Another example is the reverse hierarchy. In the typical construction, a literal reverse hierarchy would be constructed to blend to the normal hierarchy. This usually requires 1-to-1 mapping of duplicated math nodes to process the transformation and a series of constraints to accomplish the behavior as well as ensuring attributes of the hierarchy such as joint rotation order are matched and do not get out of sync.

With the Hierarchy Model, we have an operation type that can specify an arbitrary joint as pivot. This can then modify all the joints within the hierarchy, including itself. Transient propagation state can also be set along the hierarchy per operation. For instance, you could make a chain of joints curl or spin in place based on the propagation attributes.

A benefit of Hierarchy Models is the ability to scale. When joints are treated as individual nodes, a computationally expensive operation, such as a constraining joints to a surface, will rely on the efficiency of the DG to parallelize the evaluation. But with Hierarchy Models, since the joints are essentially points, they can be deformed by geometry like any other geometry. All of the optimization of the deformation, can take advantage of any parallel-processing available in the deformation libraries.

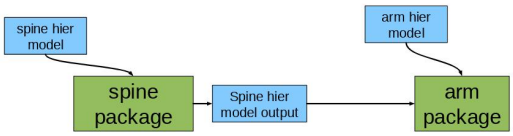

Figure 4: Hier Models flowing through character sections

The encapsulation of Hierarchy Models allows for the simplification of the character graph. Figure 4 shows an example of how a character's dependencies can be represented. Downstream systems can point to a single output Hierarchy Model instead of a list of joints. It also illustrates how the user can replace the input joint data quickly with one model. The separation of input data to the character motion systems allows the interface with other data types with less rewiring of the graph. For instance, crowd data can control the entire character using FBX motion capture and the arm can be intercepted to use Hierarchy Models by changing one connection.

4 CONCLUSION

The DreamWorks Hierarchy Model provides a solution for joint data that is self-contained and can be optimized using known geometry techniques. It allows the users to make complex motion systems with less operations and makes the flow of operations more sequential in a motion system. The resulting graph of Hierarchy Model operations represents the logic of the system as bundled steps instead of individual operations per transform. The data model can be expanded on through promising new interchange formats like USD.

REFERENCES

Jesus Nieto, Charlie Banks, and Ryan Chan. 2018. Abstracting rigging concepts for a future proof framework design. In *ACM DIGIPRO 2018*. ACM. DOI: http://dx.doi.org/10.1145/3233085.3233088