Building highly parallel character rigs

Guido Zimmermann* DreamWorks Animation
Kevin Ochs† DreamWorks Animation
Robert Helms‡ DreamWorks Animation

Abstract

DreamWorks Animation introduced a new parallel graph system, LibEE [Watt et al. 2012] as the engine for our next generation in-house animation tool. It became clear that we needed to make changes in how we set up our character rigs for production.

The new graph engine has two types of multithreading: first individual nodes are internally multithreaded, second the graph itself can run nodes and groups of nodes in parallel. The second type in particular turns out to give the greatest performance gains for the evaluation of our characters. It is also the part that is determined by the construction of the rig itself. To take full advantage of this new system we needed to restructure our characters by enabling different parts of the character to evaluate in parallel as much as possible.

This talk focuses on how we build our character rigs to improve graph performance, including changes to workflows and strategies required by our transition from serial to parallel graph structures. Because our animation software engine is the first in the industry to have a parallelized graph, many of these changes are novel, and some were unexpected.

Motivation

The ultimate goal is for our characters to run in real-time at full fidelity inside the animation package. This is now possible to achieve because the new parallel graph engine offers the potential for much faster evaluation speeds. Creating character rigs that utilize the full potential of the parallel graph is a key factor to achieve our goal.

Parallelizing the character setup

A typical character rig consists of a motion system that acts on a skeleton which then is the base for a skin deformation system which typically consists of multiple layers. We have additional systems like face, hair, clothing and accessories. Under our old evaluation engine all these systems ran in a serial chain. Parallelizing the character graph requires the combination of several rigging strategies:

Critical path optimization The critical path is the chain of nodes that takes the longest time to evaluate, and it determines the overall evaluation time of the character per frame. By trying to bring nodes outside of this path we are able to reduce the overall evaluation time. For example if we find an expensive node on the critical path we try to optimize that node internally or we try to change its dependency in order to have it evaluate off the critical path.

Dependency ordering A node will be ready to evaluate once all its inputs are available. By being smarter about the inputs we can make it evaluate earlier. For example we found an expensive deformer at the start of body deformations that needed to finish evaluating before all other layers could start. By restructuring the systems that this deformer requires to start we were able to have it and the following nodes evaluate earlier. On a dragon for example it is usually not necessary for wing deformers to wait for the tail motion system to finish.

Figure 1: *Serial graph, converted to parallel graph*

Serial chains in the graph In previous character setups we often placed nodes in a serial chain even though they all required the same inputs in order to run successfully. By restructuring these nodes we were able to utilize graph parallelism to achieve significant reductions in critical path times. This technique was used to split up and parallelize portions of the face system.

Bottlenecks of graph evaluation There are many examples in character rigs where parts of the rig are combined and then reused further down the graph. While this is a logical step from a workflow perspective, keeping them in their separate chains can greatly improve the performance of those sections of the graph. We found a significant bottleneck in the output of the motion system since we tried to combine everything into one model before passing along downstream. By immediately passing finished parts of the motion system we were able to get some deformations start earlier. It was also beneficial to avoid combining affectors for certain deformers as that allows us more flexibility in how multiple inputs are handled. This allows us to better leverage internally multithreaded nodes, allowing deformation to begin earlier and complete more quickly.

Independent systems At a higher level we can identify and separate systems like the motion system, clothing etc. While certain dependencies exist, for example the motion system of an arm needs to evaluate before the arm can deform, we found ways to have some systems run in parallel. This meant re-thinking how we connect these systems to their upstream dependencies, and changing existing operators. As an example, a new operator was created to manage the connection of head to body. This allows us to have the face run fully in parallel with the body.

Experience

The change has proved both challenging and rewarding for riggers, who are now able to extract high performance from their rigs from careful design of those rigs to expose as much parallelism as possible for the engine to exploit. This dramatically changes the complexity and realism that can be built into the rigs, leading directly to higher quality animated characters. It is clear that the multi-core era requires changes, not just from R&D, but also from those building production assets in order to extract the full performance of current and future hardware which will only increase in parallelism.

References

WATT, M., CUTLER, L. D., POWELL, A., DUNCAN, B., HUTCHINSON, M., AND OCHS, K. 2012. Libee: A multithreaded dependency graph for character animation. In *Proceedings of the Digital Production Symposium*, ACM, New York, NY, USA, DigiPro '12, 59–66.