LibEE: A multithreaded dependency graph for character animation

Martin Watt
DreamWorks Animation

Mark Hampton
Intel

Abstract

In the quest for ever greater realism, character rigs continue to grow in complexity (for example higher resolution surfacing, clothing and hair simulations.) This leads to increasing performance demands as animators wish to be able to interact with such complex rigs in real time (24fps.) Since single threaded CPU performance is no longer increasing at the same rate, it has become necessary to look for alternative means to increase performance, in particular multithreading.

We have developed a new dependency graph evaluation engine for our next generation in-house animation tool which is designed from the ground up to be very heavily multithreaded, and is thus able to deliver extremely high performance to animators.

A heavily multithreaded graph requires not just that individual expensive nodes in the graph are internally threaded, but that multiple nodes in the graph can evaluate concurrently. This raises numerous concerns, for example how to ensure the cores are used most effectively, mechanisms to handle non-threadsafe code, and tools to help riggers create character graphs that are optimized for parallel evaluation.

1 Requirements

We explored a variety of approaches to parallel graph evaluation. Since we wanted to allow nested threading as there would be both node and graph level parallelism, we did not want to manage threads explicitly, preferring a task-based system that would allow us to define units of work that could be mapped to cores by an underlying engine. Since many nodes are extremely lightweight, we also needed needed a low overhead system. We explored writing our own engine, but finally settled on adopting Intel's Threading Building Blocks (TBB) which offered high performance and was well proven.

We built an abstraction for graph-based evaluation on top of TBB. This layer was specifically focused on character animation workloads, so design decisions were optimized for this case. In particular, since animators tend to work with the same relatively small sets of controls, we would pre-cache the lists of nodes that required evaluation for any particular edit operation, thus minimizing expensive walking of the graph. We could also cache optimal evaluation strategies for the particular set of nodes that required evaluation, further streamlining the evaluation process. Since the graph topology is fixed, we could avoid maintaining an editing infrastructure that would add overhead to a standard graph implementation.

2 Challenges

The animation tool provides an API that allows both developers and artists to write their own nodes. Since these nodes would ideally run concurrently with other nodes in the graph, thread-safety becomes a concern. Not all developers are experts in writing threadsafe code, and during initial development threadsafety may not be a top priority, so we provide a mechanism that allows the user to declare the level of threadsafety of their node. The author can declare their node as non-threadsafe, meaning the engine will ensure no other nodes are in flight when this node is executed.

Although the engine is new, the reality is that there is a lot of existing studio code that may be invoked from nodes in the graph. Since not all of this code is threadsafe, we invested significant effort in both making critical parts of our codebase threadsafe and building testing frameworks to try to keep in that way.

For riggers, building a graph that allows the engine to extract maximum scalability from the workload is a new and difficult challenge. We have implemented tools that allow the riggers to visualize data flow and node evaluation in the graph, so they can see at a glance which components of their character are running in parallel and where there are chokepoints where graph serialization occurs, or where unexpected dependencies exist between parts of the rig. Over time we are learning how to build characters in this new environment to be as scalable as possible.

3 Results

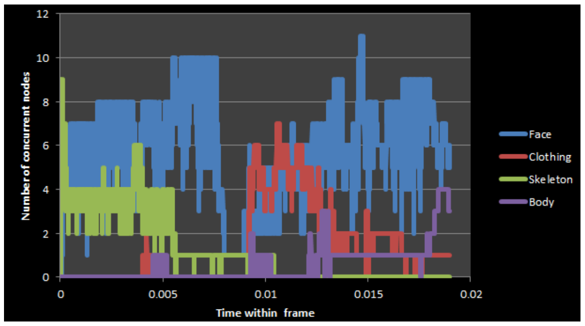

Figure 1: *Multicore evaluation of part of a single frame of animation for a hero character on a 12 core machine. Vertical axis shows concurrent nodes in flight for various parts of the character.*

We have been able to achieve well over an order of magnitude speedup over our existing in-house animation tool, and are also seeing performance significantly higher than we are able to achieve in third party commercial tools. This has allowed us to dramatically increase the level of complexity and realism in our upcoming productions. The fluidity of the workflow for animators gives immediate real-time feedback even for very heavy production level characters, and the incorporation of simulations such as hair allows for better visualization of the impact of such simulations downstream, reducing the number of turnaronds when shots get sent back to animation for fixup.

References

REINDERS, J 2007. Intel Threading Building Blocks. O'Reilly