Procedural Muscle Simulation and Ziva

Dhruv Govil, Paxton Gerrish*
Sony Pictures Imageworks

Abstract

In this talk and paper, we discuss the use of Ziva muscle simulation in our visual effects pipeline, including a full body digidouble for Enchantress in *Suicide Squad*, Slimer in *Ghostbusters* and sharks for *Megaladon*. We will go over our use of Ziva and the accompanying pipeline integration using a procedural simulation framework.

Ziva is a new simulation package offering higher fidelity simulation with faster simulation speeds. It helped us get more physically believable deformations without adding simulation time. We used it for everything from bone deformations to simulating muscles firing, fat squashing, skin sliding and even clothing riding on top of the body.

With Ziva, we also developed a pipeline for procedurally building simulations, with configurable stages to automate the simulation from an animation input through to renderable geometry. This procedural workflow greatly reduced the time required for simulation iterations, and reduced the required number of Character artists for a project.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: muscle simulatiion, pipeline, character effects

1 Building Creature Rigs

While general deformer based rigs have been a staple of character work in the industry, there has been increasing demand to achieve more anatomically correct results.

Our approach with Ziva takes this idea further, building entire physically modelled skeleton and muscle systems that can simulate off one another to give incredibly nuanced and detailed results, while also achieving very fast simulation speeds.

We take a procedural approach to building and simulation the character elements. We start with the skeleton, continuing on to the muscle fibers, fat, sliding skin and clothing. The final result is of higher quality and nuance than could be achieved with older systems and workflows.

This presentation aims to go over the creation of the physical rig layers, and show how it can create a very realistic result in production.

*e-mail:dhruvagovil@gmail.com, pgerrish@imageworks.com

2 Procedural Simulation Framework

Our procedural simulation framework consists of a Character TD defining Stages in a dependency graph. A Stage is a step in the simulation, essentially representing the simulation of a physical layer, though it could consist of other things like publishing of data.

Stages allow for manual breakpoints for a TD to interject, and the procedural caching nature allows for stages to reuse exiting simulation data without rerunning the whole graph. The ability to interrupt the graph at any point allowed for fine tuning of a setup geared to more general performances.

This procedural graph allows for automatically simulating new animation as well as sharing setups between characters. This effectively allows productions to reduce the crewing requirements for Character artists from up to 10 or more (depending on the project), to 1 or 2 artists on a show.

A single lead artist could set up the stages and the graph, and from there would only require manual intervention on extreme cases.

3 Overview of Graph Architecture

3.1 Stages

Stages are similar to nodes in a node graph but present clustered behaviour and data abstractions for an artist. What defines a stage is up to the artist, but usually it defines a layer of the simulation: Bones, Muscle, Skin, Fat etc...

Each stage has inherited methods from a base stage that take care of importing any dependencies, as well as setup and baking of the stage. The artist is responsible for giving further details to the setup of each stage such as connection of elements within the rig or setting of simulation parameters at runtime.

Stages also handle abstraction from our studio API's and artists can call very convenient wrappers. This abstraction also allows artists to define a stage without ever referring directly to a specific asset or character. Instead the stage provides placeholders that get evaluated at runtime. This allows very quick development of stages as well as reuse between characters.

Stages handle their dependency chains and scene management for the artist, and methods get broken up into actual nodes in the node graph. This allows the artist to work on a stage as a single unit, while allowing various sub-stages like setups, baking, rendering etc to invisibly act as individual nodes in our node graph.

Thus the artist never needs to care about the parallel nature of the graph. They simply work in the stage as a single unit, which greatly reduces the mental map required of the stage.

Each stage writes out a set of configurable cache types, and additionally can be reinstantiated from a given set of caches. This is useful for when something needs to be tweaked without requiring a full simulation and is also key to splitting up our simulation into layers, as dependent stages can import a cache of our current stage.

3.2 Graph Configuration

The Graph is currently configured using a standard XML format at Imageworks. This allows for fine grained control of graph node parameters, but also allows for quickly overriding graph behaviours per character.

This can additionally be visualized as a conventional node graph, however editing of the graph from a node based UI is still a work in progress.

The XML file allows for easily describing the dependencies of stages, as well as configuring parameters. These graphs can then be shared between projects and characters since our framework abstracts much of the character and project specific nature of the setups.

Each stage is split up invisibly to the artist as several nodes in the graph. For example the Bone stage will have a setup, bake and render node implicitly defined in it. However if the following stage, Muscle, doesn't require the bake to be complete, it can start as soon as the setup node completes rather than waiting.

Subsequent stages typically use the baked geometry cache from their dependencies but this is also configurable.

This allows for highly parallel behaviour in our simulations, without much forethought required from the point of view of the artist.

Graphs are typically executed fully every time a performance changes, but in the case of an artist tweaking values on an already simulated performance, the graph can instantiate itself from existing stage data that is cached after the execution of each node. This again greatly reduces the time to adjust a simulation of a performance. Prior to this, it would often require manual set up or a complete resimulation which can be very heavy.

3.3 Pipeline Integration

The system integrates into our existing pipeline in a few ways.

Firstly, the system is robust enough that it can be automatically launched from every animation publish when a performance changes. This already reduces the amount of manual effort required by the Creature TD and the latency between a publish and when an artist starts.

The system also automatically generates renders of the results so that they can be reviewed. If there are changes that need to be made, the Creature TD can start at any point along the graph while reusing the cached states of any dependencies. Thus they don't have to perform a full simulation of the character if they are modifying a single part.

Finally, the system integrates into our asset pipeline and will automatically be picked up by lighters if so desired. Once the system has been appropriately configured and tuned early on the show, later simulations often require little to no interaction by the artist.

4 Final Thoughts

4.1 What works well

The system has over all worked very well on each show and has had a lot of benefits:

- Greatly reduces the time required to iterate on simulations as artists can tweak parameters while reusing existing caches
- Artists can spend less time per shot as their simulation parameters stabilize through the project. The system can automate most simulations for them.
- Artists need a smaller mental map of the simulation since it can be compartmentalized into Stages
- Reduction in character specific setups means that we can share setups much easier between characters on multiple projects.
- Lower latency between department handoffs i.e Animation to Creature FX and then to Lighting as the system can automatically run between them.
- Time savings causes significant reductions in resource usage including machine time on the render farm.
- The time savings per artist also means that artists spend less time on each simulation, thereby reducing the number of artists needed per project. This can often be from 10 people down to one or two people.

4.2 What needs improving

Currently there are a few things that we think need improving for future projects:

- While simulation setups can be shared between characters, we need to work on sharing the actual simulation geometry like skeletons and muscle shapes. This is currently possible but needs refinement.
- A better graph based UI would be a big step forward for set up and configuration of the simulation graphs. Currently we can visualize the graph but any configuration needs to be done in an XML file.
- Our simulation system is mainly used for muscles and soft tissues, but there's no reason it cannot be consolidated with our simulation systems for cloth and other effects. In the future we will be looking into consolidating our systems

Overall, this new procedural system has been very succesful in production and will see lots of use going forward at Imageworks.

5 Acknowledgements

Lonnie Kraatz for setting up the Ziva workflow and prototype pipeline.

References

JACOBS, J., AND OTHERS. Ziva dynamics. `http://zivadynamics.com/`.