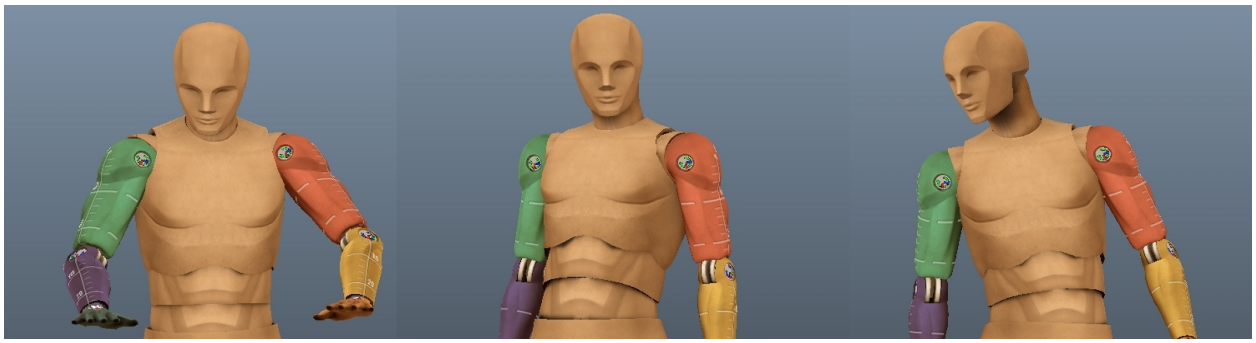
Velocity-based compression of 3D rotation, translation, and scale animations for AAA video games

David Goodhue
Square Enix Co., LTD
Tokyo, Japan



ABSTRACT

In our previous publication [Goodhue 2017], we presented a prototype based on promising new techniques for how the state-of-the-art in animation compression for video game engines might be advanced. Later that year, we completed development on a production-quality version of that technology which has since seen active use in the ongoing production of future AAA titles. Many of our previous hypotheses were put to the test, and the algorithms were generalized to support translation and scale animations keys in addition to rotations.

Having used this new technology for quite some time now, we were able to confirm our expectations regarding the sort of technical problems it presents, as well as how to solve them. We are also able to compare our results to other state-of-the-art techniques for the first time, thus confirming the efficacy of our method.

CCS CONCEPTS

• **Computing methodologies → Animation**.

KEYWORDS

animation, compression, video games

ACM Reference Format:
David Goodhue. 2020. Velocity-based compression of 3D rotation, translation, and scale animations for AAA video games. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '20 Talks), August 17, 2020.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3388767.3407392

SIGGRAPH '20 Talks, August 17, 2020, Virtual Event, USA

1 RELATED WORK

While it seems likely that there may be some interesting proprietary alternatives being used privately by some companies, in terms of what's been published in recent years, Nicholas Frechette's ACL [Frechette 2020], as well as his previous work [Frechette 2017], appear to be the most relevant. Both techniques are very effective at shrinking data by aggressively quantizing keyframe values within tight ranges using bit widths just large enough to retain sufficient accuracy. This is similar to our work, which also utilizes range reduction as well as favoring smaller bit widths when possible.

Our implementation of velocity-based compression is in many respects an extension of those techniques. The values being quantized often represent velocities rather than absolute pose values, requiring multiple sets of range data to handle both scenarios. Furthermore, keyframe data exists sparsely as opposed to maintaining uniform key counts each frame. This requires substantially more complexity in both tools and runtime code, but we aim to demonstrate how it can produce favorable results.

2 TERMINOLOGY

Throughout this document, we shall refer to the algorithm from Nicholas Frechette's GDC presentation [Frechette 2017] as "Simple". Additionally, for the purposes of this document, all references to his ACL [Frechette 2020] are specifically regarding ACL version 1.3, the latest ACL release at the time of this writing.

We also refer to our velocity-based compression as "Predictive", because in some cases it essentially predicts a future pose value, then applies a relative adjustment key in realizing its true destination.

3 GOAL

In past projects, as well as in those we expect to encounter in the future, our primary use case for improved animation compression has been to reduce our animation memory footprint, the bulk of

which tends to be rotational data. Translation and scale keys occupy a significant but much less concerning portion of our total animation data. CPU performance is also a concern, but only in that the decompression cost must remain reasonable, as it tends to be in most game engines.

Concerning loss of accuracy in decompressed poses, provided that those viewing the game aren't aware of any degradation and that accuracy tuning is not consuming the time of our design team, it's not a problem. Typically, having a relatively loose accuracy default for gameplay animations, then another which is more strict for high resolution cinematic cut scenes, works out well. Additionally, the ability to override these per character or per animation clip is important. Of those two default settings, by far our largest concern is the more aggressive compression level needed for ordinary gameplay. This is because cut scenes tend to stream in most of their data as they are playing, dumping previous chunks as they progress, making total size much less of an issue. As for the more relaxed gameplay default, it's not that we are tolerant of our characters looking worse than they should, it's because is it so minor that it goes unnoticed. Unless a camera is zooming up close as is typically only seen in our games during cut scenes, such inaccuracy will remain a secret.

In summary, our main goals were to shrink rotation data significantly, shrink translations and scales as well within reason, and avoid algorithms which will yield poor CPU performance,

4 IMPLEMENTATION

Unlike what was used in our first prototypes, the engine into which we were integrating this technology stores all bone rotations in exponential map format. In other words, each rotation is stored as the natural logarithm of its quaternion, or "quatlog" for short. This requires that animation decompression produce local-space rotation values which are quatlogs rather than quaternions as the case would be in most engines. There is no reliable way to accurately interpolate between quatlogs without a costly conversion to quaternion followed by a costly slerp. This is not a major problem for our compression algorithm, however, since it can just simulate this inaccurate interpolation of quatlogs offline, and anytime the result isn't accurate enough, make sure that a keyframe exists instead. This likely means that we are removing somewhat fewer keys than a typical quaternion-based engine which uses slerp approximations between keys might do, but in comparison to our quaternion-based prototype, the difference seems minor.

We found other benefits to using quatlogs as well, especially the fact that we wished to sometimes have rotation keys which change only the angular speed while preserving the axis of rotation from the previous velocity. This is a costly operation requiring sin or cos if performed accurately with quaternions, but with quatlogs, it is very cheap. Furthermore, our runtime decompression performance is improved because applying a velocity in quatlog space is cheap, costing nothing more than an addition of 3 floating point numbers. In a typical game programming scenario, that same operation would be more expensive, because one would generally need to check that the resultant quatlog has not extended beyond the unit sphere, and wrap its magnitude if it has. However, since animation playback can be perfectly simulated offline in tools, we never insert velocities which need to be wrapped except for in the extremely rare but useful special case of applying our highest precision key type.

5 RESULTS

Our accompanying materials show the results of an ambitious test comparing 4000+ production-quality animation clips using our velocity-based algorithm against both ACL and Simple. We are able to compare to Simple despite it not being publicly available thanks to it being part of our proprietary holdings. ACL and Simple are similar technologies which, not surprisingly, yield similar results to each other. By applying our preferred default error threshold and virtual bone length settings for gameplay animation to all three, we often see data sizes at around 50 percent of what the other techniques require at equivalent error thresholds.

It's not a perfect comparison, as all three algorithms fail to keep their max error anywhere near the specified error threshold on a small subset of the total clips. In all such cases, those are either bugs or a lack of the features needed to handle such exotic data. For example, special handling for instant teleportation across obscenely long distances. While it's unfortunate that time constraints prevented a more polished test at this time, the limited number of such errors is not an indication of the validity of the compression techniques involved.

We have also measured in-game CPU performance against a fully optimized, production-quality implementation of Simple, finding that decompression costs are consistently reduced by more than half when using our default settings. However, decompression costs in either case are far below a level which would cause concern for our productions. As for ACL, it would be fruitless to attempt such a comparison at the moment without first spending significant time to create a similarly thorough, engine-specific integration. While it would allow for testing, naively copying the pose over from one system to the other each time it's computed would add enough additional cost that any comparisons would hold little relevance. Likewise, ignoring the cost of such copying would be equally invalid. ACL appears to be carefully crafted with performance in mind all the way from the math library up, so it is likely a strong contender.

6 CONCLUSIONS

The hypotheses and conclusions proposed in our previous publication have been further confirmed. The implementation of this sort of system is quite complex and involved, but for teams with the resources to do it, significant memory savings and stellar CPU performance awaits. However, nowadays, with fantastic open source alternatives such as ACL being freely available, it won't make sense for most teams to pursue something like this unless they desperately wish to go beyond such easy and reliable options.

REFERENCES

Nicholas Frechette. 2017. Simple and Powerful Animation Compression. (March 2017). https://www.gdcvault.com/play/1024009/Simple-and-Powerful-Animation GDC 2017.

Nicholas Frechette. 2020. *Animation Compression Library.* https://github.com/nfrechette/acl

David Goodhue. 2017. Velocity-Based Compression of 3D Animated Rotations. In *ACM SIGGRAPH 2017 Posters* (Los Angeles, California) *(SIGGRAPH '17).* Association for Computing Machinery, New York, NY, USA, Article 14, 2 pages. https://doi.org/10.1145/3102163.3102236