Velocity-based Compression of 3D Animated Rotations

David Goodhue
Square Enix Co., LTD
6-27-30 Shinjuku, Shinjuku-ku
Tokyo, Japan 160-8430

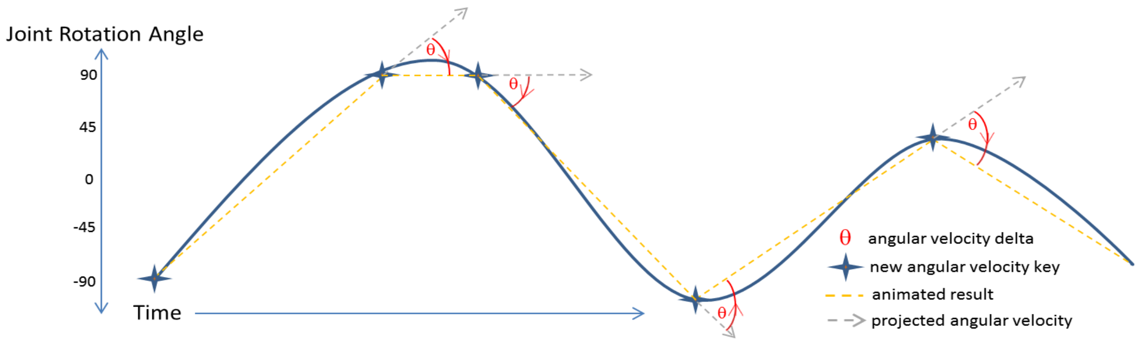

Figure 1: *Two-dimensional example depicting the relationship between angular velocity and rotation keyframes.*

ABSTRACT

Modern video game engines feature animation compression built using algorithms which offer fast runtime decompression. In comparison to other state-of-the-art industry techniques, we present new methods by which better compression ratios can be realized without significantly impacting performance.

We first present a technique for reconstructing a stream of sparsely-keyed rotations from a sequence of angular velocities. Next, we encode those velocities as consecutive deltas, making it possible to use much smaller key sizes. As a final enhancement, we allow the speed component of our angular velocity to in some cases receive influence from velocity keys which do not specify an axis of rotation. Instead, the axis remains unchanged from the previous frame's velocity, yielding smaller data on those frames.

CCS CONCEPTS

• **Computing methodologies → Animation**;

KEYWORDS

animation, compression, video games

ACM Reference format:

David Goodhue. 2017. Velocity-based Compression of 3D Animated Rotations. In *Proceedings of SIGGRAPH 2017 Posters, Los Angeles, CA, USA, August 2017,* 2 pages.
https://doi.org/10.1145/3102163.3102236

SIGGRAPH 2017 Posters, August 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5015-0/17/07…$15.00
https://doi.org/10.1145/3102163.3102236

1 PREVIOUS WORK

Most compression techniques published in academic journals fail to address the CPU performance needs of video game production. Popular techniques in the industry tend to be spline-based [Akhter et al. 2012], or for even better CPU performance as with our method, keyframe-based. That is to say, an encoding wherein the rotation at any point in time results from an interpolation between the two rotation keyframes bounding it on either side timewise within a sequence of time/value key pairs. Unlike some cutting-edge keyframe-based techniques [Frechette 2017], to further reduce unnecessary key data, like many others, we use sparsely distributed keyframes rather than keep constant intervals between adjacent keys. In this report, we are pleased to show significant advances over our previous top proprietary technology, which was also keyframe-based and which offered similar compression ratios to those techniques referenced above along with very fast CPU performance.

2 INTRODUCTION

Velocity-based compression is an extension of conventional keyframe-based compression. For a typical dataset, in contrast, this approach can yield significantly smaller data, eliminating the need for per-track or per-section rotation ranges as well as the need for time/frame data to exist for each keyframe. It is also key-reducible and as such is similarly easy to configure. It preserves the aforementioned benefits of keyframe-based methods and adds more of its own, but not without some drawbacks.

3 TERMINOLOGY

In conventional keyframe-based compression, each key value is approximately equal to the desired animated result at the corresponding time. Using these new approaches, that is not necessarily the case, and keys are generally only used to compute future results. To differentiate from keys, we shall refer to animated results for which specific keys exist as ”keyed results".

4 VELOCITY-DRIVEN KEYFRAMES

As can be seen in Fig. 1, a single constant angular velocity exists which can transform between every pair of subsequent keyed results in an animated rotation track. While preserving its axis of rotation, scaling the angular component within the velocity by the time lapse between the two results yields a rotational difference which, when applied to the previous keyed result, will yield the expected value for the next result. Key values under this scheme are the velocities which exist from each keyed result to the next. Intermediate results at any other time are computed in an identical manner; which is to say, by taking the velocity key from the time of the previous keyed result, scaling the velocity angle by time elapsed since then to get a delta rotation, and applying that rotation to the previous keyed result.

Since no result can be computed without the presence of a previous result, an initial set of results, or seed frame, must exist at time==0.0 as input to this algorithm. Data must be traversed sequentially from that point forth to yield results for later times, so for practical purposes in video games where high-performance is required, additional seed frames may need to be inserted into specific chunks for animations which wish to begin playback at later times. While that is an unfortunate drawback, there are benefits to doing this as well.

Making the techniques in the following sections possible is probably the largest benefit of velocity-driven keyframes, however another of significance is that, in contrast to conventional keyframe compression, it eliminates the need for a time value to be associated with each key. Those systems interpolate between previous and future keys, so a key time needs to be read from data whenever a new key is encountered. With our method however, there is no future time involved, just the velocity and the previous key. Velocity keys correspond to the frame upon which they are encountered in the data stream, with no additional time data required.

5 VELOCITY KEYS ENCODED AS DELTAS

Rather than pack velocity values as our keys as described above, it is also possible to just store the delta rotation between one velocity key and the next. While it is still a reasonable solution to have rotation animation driven by velocity keys directly, the advantage to using deltas instead is that, for typical character animation data, they will almost all fit into an extremely small range centered around zero. The most advanced forms of conventional keyframe-based compression achieve rotational range reduction by encoding data to tag individual sections of individual channels with various ranges, including both the range center and the range size. However, by using velocity deltas and a range center of zero, just a single range size for all channels and all sections even will still yield excellent results. The downside is that some small number of keys may not fit within the range, so a means for loading those as packed velocities rather than deltas, and with no range restriction, is needed as well.

6 AXIS-FREE VELOCITY KEYS

In section 4 we discussed decomposing velocities into axis and angle components, then scaling each angle by elapsed time to attain the corresponding rotational difference to be applied to the previous keyframe's pose value. While there are many different and interesting ways mathematically and programmatically to implement that, the best among them all seem to deal with the angle, or in other words the rotational speed, at some point as an individual float value. Therefore it follows that we can place velocity keys into our data stream which modify the speed while leaving the axis of rotation as it was from the previous velocity. Of course, this should only be done on frames where the desired axis is close enough to that of the current velocity to the extent that a speed correction alone will keep our results within our error tolerance. If packing the new speed as a delta from the previous within some restricted range, a single byte is often sufficient. However, just like our deltas from the previous section, in the rare case that a key must exceed the restricted range, an alternative larger and unrestricted key value must be available. This is especially powerful for compressing animation where bones frequently rotate around a largely stable axis such as rotors or wheels. We experienced massive data shrinkage on such clips, with many keys going from 32-bits or higher under conventional methods down to just 8-bits using velocity speed deltas.

7 CONCLUSIONS

The techniques presented expose many new possibilities for advancements in keyframe-based animation compression. The implementation however is non-trivial and much less straightforward than conventional techniques. Additionally, clever architecture is needed for it to decode competitively fast. For large teams with enough resources to fully refine these technologies, it seems worth the additional work, with any downsides being entirely manageable.

Despite using only a prototype key reduction algorithm at the moment which does not compute an ideally minimal set of keys, the compressed data tends to be about 65% smaller than it would be using our previous top compression technology, or in some cases even far smaller than that. Those numbers will improve once we create a more sophisticated key reduction algorithm which specifically targets this data format. We speculate that if one were to combine maximally aggressive quantization [Frechette 2017] with an optimal set of sparse keyframe data, that result would likely prove more competitive than our previous top technology here, but we have not built or even managed to identify such an implementation to compare to as of yet.

8 FUTURE WORK

We plan to implement similar velocity-based techniques for translation and scale keys as well. Furthermore, it could have uses in graphics or elsewhere, such as for compressing a traversal of consecutive surface normals.

REFERENCES

Ijaz Akhter, Tomas Simon, Sohaib Khan, Iain Matthews, and Yaser Sheikh. 2012. Bilinear Spatiotemporal Basis Models. *ACM Trans. Graph.* 31, 2, Article 17 (April 2012), 12 pages. https://doi.org/10.1145/2159516.2159523

Nicholas Frechette. 2017. Animation Compression: Advanced Quantization. (March 2017). Retrieved from http://nfrechette.github.io/2017/03/12/anim_compression_advanced_quantization/.