Velocity-based Compression of 3D Animated Rotations

David Goodhue, Advanced Technology Division, Square Enix Co., LTD

Goal

Modern video game engines require animation compression which offers fast runtime decompression while simultaneously shrinking data as much as possible. We present techniques by which improved compression can be realized without significantly degrading performance.

Our Method

Three new innovations were developed, each which builds upon the previous as a further improvement. 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 axial and speed components of each angular velocity to be keyed independently of each other. When no key exists, the respective speed or axis remains unchanged from the previous frame's velocity.

Velocity-driven Keyframes

In a typical keyframe animation system, a rotational trajectory between two keys is traversed over time by interpolating between them. Furthermore, to avoid structuring their data in a manner which would require costly searches across wider memory, generally there exists an associated time value for each key. For our method, we considered the difference between two keys as a rotational quantity to be applied over time, which can have its angular component scaled to yield a interpolated value along the desired trajectory. Rather than interpolating, we apply a rotational velocity key with an appropriately scaled angular component to the pose at the time of the previous keyframe. Keyframe times become known to the playback mechanism only upon the frames where the rotational velocity must change from the previous one, thus eliminating the need for time data to be associated with each key.

Encoding Keys as Delta Velocities

It is possible to stream only delta rotations between each velocity key and the next, and apply those to any previous velocity. This normalizes the range of possible key values such that in a typical animation clip, more bones will fall into nearly identical ranges, all of which are centered around zero. Sharing the same ranges for many different bones can mean fewer distinct ranges need to be defined in data and used by the algorithm, which can be good for both data size and CPU performance.

Decoupled Rotational Axis and Speed

Instead of keying a single track for each bone such that each key contains data to change both the axis of rotation as well as the speed component, it is possible to handle them as completely independent data streams which combine to form the complete velocity at any point in time. This way, whenever one component proves to be more linear than the other, less data can be used to represent the two in total.

Challenges

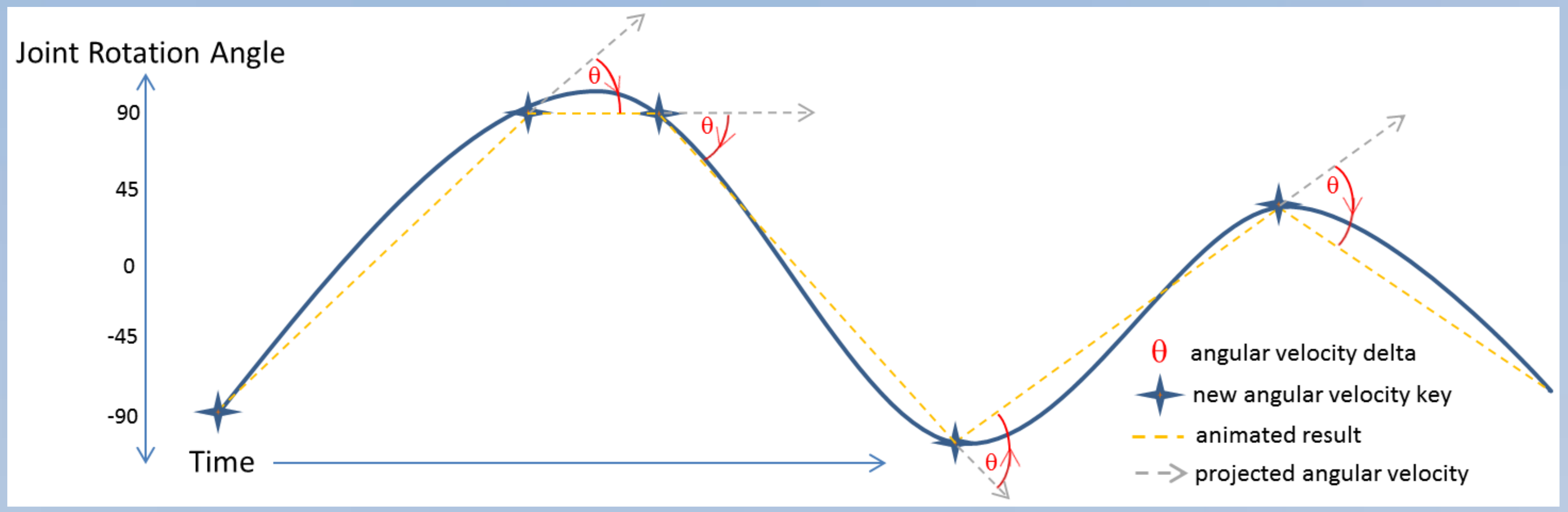
Since velocities are always applied to a previously computed pose value, if it is necessary to begin playback from some arbitrary point in an animation clip, to avoid the high cost of applying all prior keys consecutively from the beginning, fully-keyed poses must be injected into the data near any such start point. Furthermore, it must be possible in data preparation tools to perfectly simulate the runtime result of applying velocities from one key to the next, as any accumulated floating point error must be correctly compensated for in subsequent key values. This could be especially challenging if the runtime platform behaves very differently from the tools platform. It is also not so straightforward to implement decompression such that data precision and CPU performance remain high. Some techniques for achieving that are suggested in our supplemental document.

Results

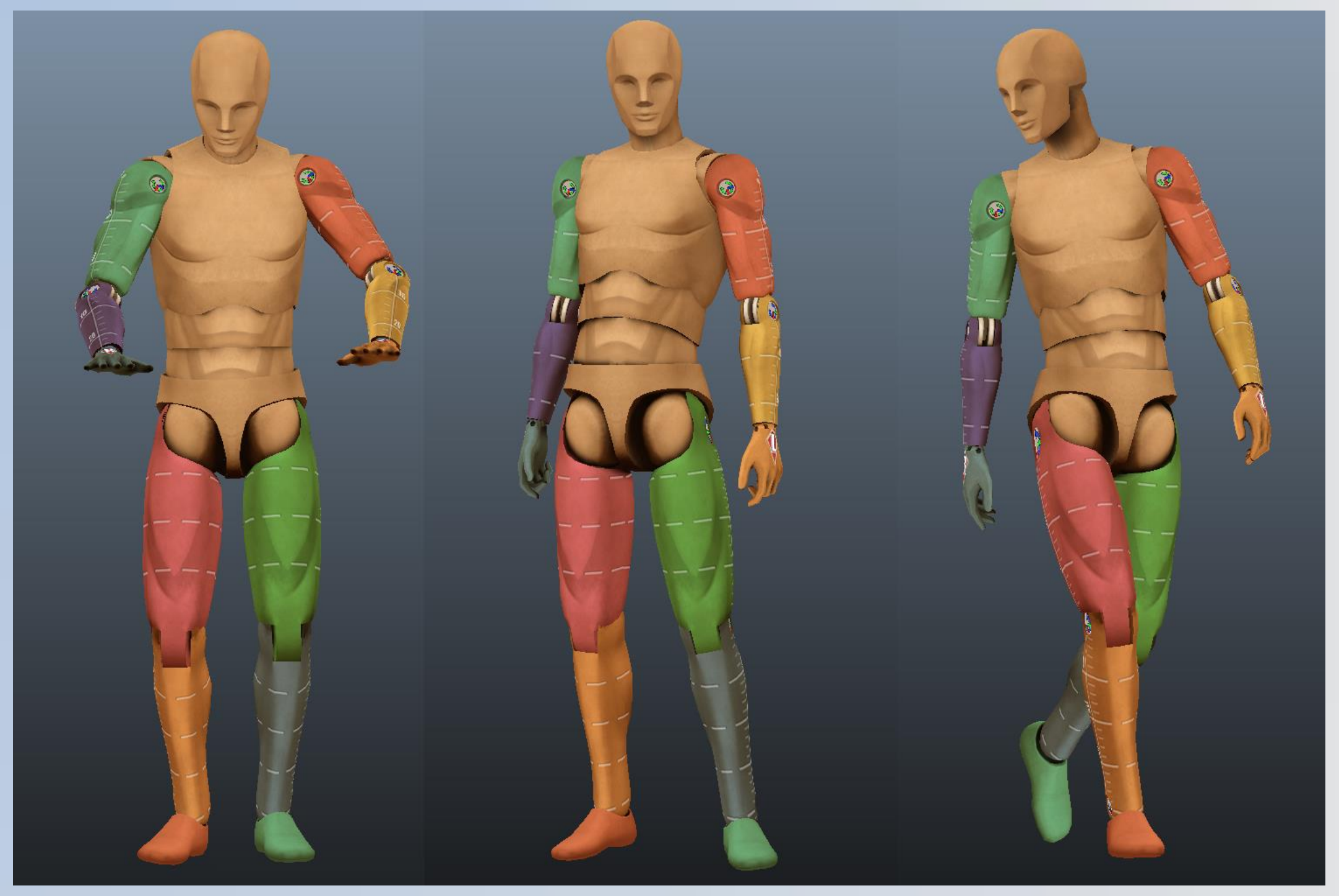
Within Square Enix we already have proprietary animation compression technologies which outperform various popular commercial engines and other published approaches such as spline-based compression [Ahkter, et al. 2012] using known keyframe-based techniques. In our prototype, our new methods are able to surpass our previous top technology and create animation data which is about 65% of its previous compressed size.

References

AKHTER, I., SIMON, T., KHAN, S., MATTHEWS, I., AND SHEIKH, Y. 2012. Bilinear spatiotemporal basis models. *ACM Trans. Graph.* 31, 2, Article 17 (April 2012), 12 pages. DOI:http://doi.acm.org/10.1145/2159516.2159523

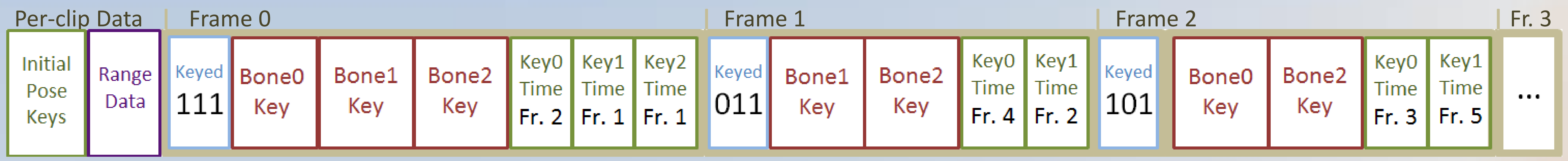


A two-dimensional example depicting the relationship between angular velocity and rotation keyframes.



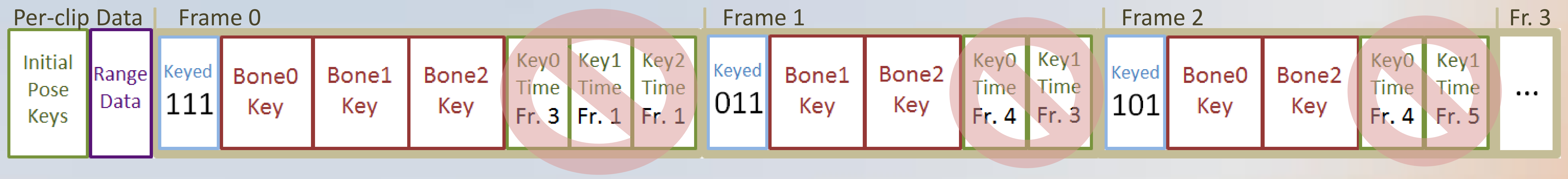
Data Comparison

Typical Keyframe Animation System – Example Data Layout – Three Bone Clip



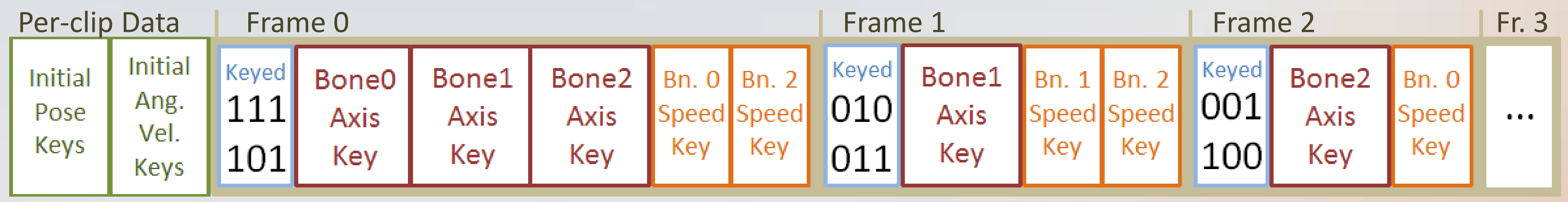
- Each new key in the data stream has an associated **Key Time**. This time is needed to compute the correct weight to use when interpolating it with the previous key, while also avoiding searching ahead in the clip data to know when the next key might occur.

Using Velocity-driven Keys



- **Key times** become unnecessary. Keyed velocities are simply applied to the evaluated pose value at the time where each previous key existed.
- As many bones often maintain very similar velocities, **Range Data** can now contain fewer distinct ranges. This can lead to more efficient quantization, smaller range data, and faster CPU performance.

Using Velocity Deltas in Conjunction with Fully-decoupled Axis and Speed Keys



- **Initial Data** includes velocities as well as a pose, since most keys are deltas.
- Additional **Keyed Bits** indicate the presence or absence of both axis and speed keys which can exist independently of each other.
- For a given frame, anytime a single **Axis Key** or **Speed Key** can exist where otherwise there would have been equally precise data keyed for both, data can naturally be made smaller than it otherwise would be since there is less total information to encode.
- Since most keys can be applied as deltas quantized within a small bounding range, **Range Data** may be best handled as a bit within each key to indicate whether it is a delta or not. In the case where it is not a delta, it is generally not so wasteful to use a larger, more precise key since such keys are so rare. Alternatively, there are many other efficient schemes to choose from for controlling quantization ranges and individual key sizes, all of which benefit from the small ranges centered around zero which result from using velocity deltas.