ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2018
T. Beeler and N. Thuerey
(Guest Editors)

Volume 37 (2018), Number 8

Collision-Aware and Online Compression of Rigid Body Simulations via Integrated Error Minimization

Timothy Jeruzalski John Kanji Alec Jacobson† David I.W. Levin†

University of Toronto

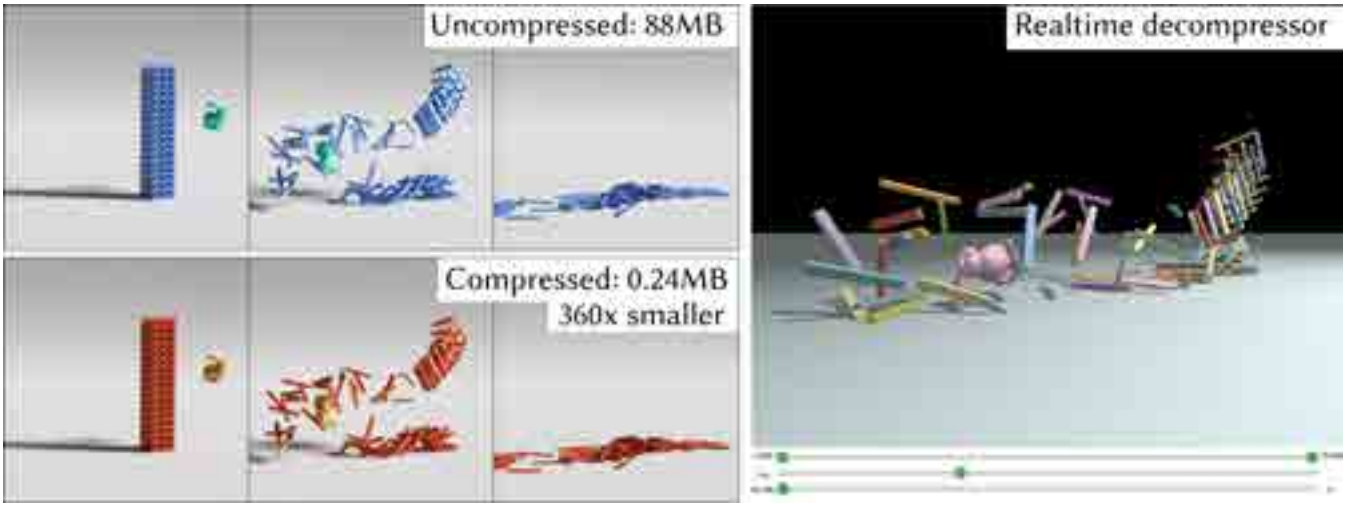

Figure 1: *Our compression for rigid body simulations can reduce data size to a fraction of its original. Our method retains all 3D geometry and performs interpolation in time allowing for the compressed output to be used for a number of tasks such as camera re-positioning, re-lighting and re-timing. Here we show an example of a complex simulation being visualized interactively in a WebGL based viewer.*

Abstract

Methods to compress simulation data are invaluable as they facilitate efficient transmission along the visual effects pipeline, fast and efficient replay of simulations for visualization and enable storage of scientific data. However, all current approaches to compressing simulation data require access to the entire dynamic simulation, leading to large memory requirements and additional computational burden. In this paper we perform compression of contact-dominated, rigid body simulations in an online, error-bounded fashion. This has the advantage of requiring access to only a narrow window of simulation data at a time while still achieving good agreement with the original simulation. Our approach is simulator agnostic allowing us to compress data from a variety of sources. We demonstrate the efficacy of our algorithm by compressing contact-dominated rigid body simulations from a number of sources, achieving compression rates of up to 360 times over raw data size.

CCS Concepts

•***Computing methodologies*** → ***Animation;*** *Simulation tools; Physical simulation;*

1. Introduction

Sweeping advances to physical simulation have dramatically improved the realism of computer animations and the accuracy of engineering analyses. However, the cost of these advancements is a massive accumulation of data. High frame rate visual effects and slow motion sequences can require storing data at 48 frames per second or higher. Sound simulation, or simulation for engineering design can require even higher temporal resolution than that and simulations for cosmology often generate trajectories that are eons in duration. The result is that sharing, browsing or transmitting such

† Joint last authors

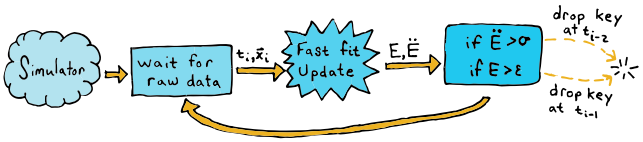

Figure 2: *Frames are output from any off the shelf simulator package and are able to be compressed through our algorithm. This flow applies to both the trajectory and rotation fitting, and is applied separately for each object.*

Figure 3: *Taking a low framerate input animation (left) and downsampling to save space (center) can yield ambiguous spinning directions when interpolating later, whereas our method (right) is able to upsample the given input animation. (see video @ 3:00).*

data becomes almost impossible, often forcing one to regenerate it from scratch each time, wasting hours (or more) of time (see Fig. 1).

Concretely, dynamic simulations are often run with sub-millisecond timesteps for stability and accuracy reasons. This means that a *short* 10-second simulation of 10,000 objects could generate over 100,000 rigid transformations or 4.8 gigabytes of data (assuming six double-precision floats per rigid transformation). We can treat the compression of this data as a sampling problem where our goal is to choose the best possible samples or simulation states from which to reconstruct the original signal. Ideally, we would like our sampling algorithm to be accurate, fast, memory efficient and, in analogy with other popular file and media compression tools, able to seamlessly act on data from any source.

Downsampling simulation data to a lower frame-rate comes with an obvious, though frustrating accuracy-to-compression-rate trade-off. Sub-sampling rigid transformations below the eventual playback frame-rate will result in a choppy animation or requires rigid-transformation interpolation, inheriting its ambiguities and issues (see Fig. 3). These problems are exacerbated when animations are played in slow-motion, a practice common for complex visual effects (see accompanying video). Animation renderings immediately benefit from efficient video-compression algorithms, but reduce the simulation data to an image with a particular choice of viewpoint, lighting, materials, etc. Contact-dominated rigid body simulations do not benefit from standard domain reduction such as principal component analysis (PCA), despite their success for deformable body simulations and fluids. The non-smooth nature of collisions and impacts is essential to their perceptual importance, but also the reason why they are immediately lost by any compression method based on low-pass filtering (e.g., PCA). Meanwhile, domain-agnostic file compression methods do not easily accommodate random-access animation streaming and ignore potential savings lurking in the temporal coherency of rigid-body trajectories.

We present a method for streaming compression of arbitrarily long, n-rigid body simulations that satisfy the criteria above.

1. Our method is accurate, and uses efficient integration to compute the integrated trajectory error, placing keyframes when this error grows too large.
2. Our fast method is memory efficient and parallel across the n objects, allowing new data to be processed as soon as it is produced without the need to store the entire simulation.
3. The simulator itself is treated as a black box, thus our method is agnostic to the underlying integration method, collision handling choices, or simulation software.

We expect as input, a time-series of n rigid transformations. Our output is a time-series of coefficients parametrizing a continuous rigid-body transformation function interpolating between sparse, automatically placed keyframes. Our function minimizes the L_2 error between the compressed reproduction and the input rigid transformations, with careful treatment of rotations. Keyframes are triggered by excessive approximation error or by a "geometry-blind" impact detection method based on processing finite differences of the incoming signal. Compression error is controlled by a small set of simple, intuitive parameters. Our compression rates drastically outperform standard methods such as PCA or file-level compression.

Our compressed output data maintains temporal order and relies only on temporally local data. This avoids large lookup tables (e.g., PCA bases) or requiring that all data is "unzipped" to view the simulation at a particular moment. We demonstrate the effectiveness of our compression method with an interactive simulation viewer. Users may interactively reposition the viewpoint or change rendering configurations (not possible with video output). Our small data size enables interactive scrubbing with memory thrashing. The user may jump to arbitrary far-away moments in simulation time. The continuous nature of our approximation also enables viewing the simulation *in between* the original snapshots, enabling detailed analysis or temporal warping such as fast-forward or slow-motion without aliasing artifacts. We evaluate the performance of our method on a variety of contact dominated rigid-body simulation scenes.

2. Related Work

Our goal is to compress the output of *any* rigid-body simulation software, whether it is a commercial game physics engine (e.g., BULLET [Cou15]) or state-of-the-art research software (e.g., SCISM [SKV*12]). While much recent literature in rigid-body dynamics focuses on improving integration [vZOS08], constraints [MCMJ15] or contact handling [VSK*17], our method is agnostic to such choices.

Particle trajectories Any discrete particle (or center of mass) trajectory can be interpreted as a poly-line in 4D [SXYL16]. For example, de Vries et al. [dVvS10] use the *geometric* Ramer-Douglas-Peucker algorithm [Ram72, DP73] to simplify maritime vessel paths for routing analysis. However, a trajectory is a curve in *space-time* and purely geometric simplification will ignore physically meaningful regularity and events along the temporal axis. Previous works that use poly-line simplification for particle trajectories [PJAP07, RP12] rely on *linear* interpolation in time, but idealized trajectories in a constant force field (e.g., due to near-earth gravity) are *quadratic*. C_1 curve fitting (e.g., with Catmull-Rom splines [Jun15]) will smooth away salient events such as contacts.

Deformable Mesh Animations Most literature on animation compression has focused on deformable shapes, where the input is a sequence of mesh vertex displacements [MLDH15]. Though some methods treat vertex trajectories as independent particles [RP12], most take advantage of the provided regularity and combinatorial structure of the input mesh to compress the data (e.g., [IR03, ASK*12, VMHB14]).A popular strategy is to apply principal component analysis (PCA) to the number-of-mesh-vertices by number-of-animation-frames matrix of vertex displacements [AM00, SSK05]. Karni et al. [KG04] further compress the PCA coefficient-trajectories with a linear model. Character animations have received special attention (see [JDKL14], with a series of methods improving upon the seminal method of James & Twigg [JT05] that approximates a mesh animation by optimizing a set of bone weights and linear blend skinning transformations. Rather than compress the animation of a single complex character, our method compresses the animation of a complex scene of rigid bodies.

Skeletal Joint Angles The bone angle sequences of an animated skeletal rig are similar to the rotation component of our rigid transformation sequences. Previous methods have treated each Euler angle or quaternion component of a bone's rotation as a 2D curve [PJAP07, Jun15] and applied geometric simplification and fitting methods mentioned above. Rigid body rotations are also non-smooth at collisions, and so piecewise smooth function spaces are more appropriate in our case. Interpolating rotations while staying on the $SO(3)$ manifold is more involved than interpolating displacements [Sho85]. Represented as a unit quaternion, special care must be taken for sign flips and maintaining unit norm. (e.g., Jung [Jun15] appears to compress the vector part and recovers the scalar that preserves unit norm). Some previous compression methods opt instead to convert rotations to three particle trajectory signals and recover a best-fit rigid transformation at runtime [LM06, Ari06]. An elusive challenge for compressing skeletal joints is to maintain stationary end effectors in long kinematic chains; Tournier et al. [TWC*09] take special care to avoid this so-called "foot skate" issue. In contrast, our contact and impulse keyframe trigging ensures that resting objects stay put. Recently, Goodhue [Goo17] proposed compressing components of angular velocity along poly-lines, however, this requires sequential access starting from an initial rotation prohibiting random access and scrubbing style interactions (cf. Fig. 1).

Other types of compression Compressing simulation data is an important topic of research across other subareas of physics and animation, such as fluids [HNB*06, WPS14, EWPT17, JSK16]. In astronomy, simulations involving a few objects (e.g., 9 planetary bodies in the solar system) may require durations of eons but integrated at very fine timescales to ensure accuracy. Rein et al. [RL12] describes a method for sub-sampling n-body gravitational simulations, taking special care of floating point error when re-simulating between saved checkpoints. Finally, lossless data compression algorithms (e.g., the LWZ algorithm used to create .ZIP files [Wel84]) and geometric primitive quantization methods (e.g., [KISS15]) are orthogonal to our rigid-motion specific streaming compression. We may apply further quantization to our optimized values or create a compressed zip archive of our output data.

Our work dramatically reduces file sizes for contact-dominated rigid body simulations. This saves disk space but also affords applications that rely on fast access to simulation data. For example, Twigg et al. [TJ07] compress center-of-mass trajectories of many simulations to interactively explore the space of possible simulations. Our algorithm is a drop in improvement to the C_0, piecewise quadratic curve-fitting they propose. Their method measures instantaneous error rather than integrated error and only fits based on the first 10 frames per quadratic segment. Instantaneous error will often trigger only *after* a salient collision, whereas we detect likely contact events automatically (see Fig. 4).

Figure 4: *An input simulation of a cube hitting the ground makes contact during a single animation frame (blue). Our peak detector identifies this salient event and ensures it occurs in our compressed output (orange). Only measuring accumulated trajectory error may erroneously lead to a false, new interpenetration (yellow).*

3. Methods

The input to our method is a sequence of rigid body states. For each frame i, we expect a time stamp t_i, the body's current rotation as unit quaternion components $\mathbf{q}_i \in \mathbb{R}^4$ ($\|\mathbf{q}_i\| = 1$), and the center of mass position $\mathbf{p}_i \in \mathbb{R}^3$. The algorithm is agnostic to the choice of simulator and is able to compress simulations from high-accuracy scientific simulators as well as high speed real-time simulators. Along with the input data, our method is controlled by error parameters $\varepsilon_T, \varepsilon_R$ controlling the bound on trajectory and rotation integrated error, respectively, and σ_T, σ_R controlling the placement of "peak detected" keyframes at likely impulses or contact events. These peak detected keyframes enable our method to treat the input simulation in a black-box fashion as no extra data is required to identify collisions or other events causing discontinuities in an object's velocity.

The output of our method is a sequence of rotation and translation *keyframes*. Each keyframe contains a timestamp recording the beginning of a polynomial fit and the polynomial's coefficients. Each object in the scene is compressed separately allowing for easy parallelization, with no knowledge of the underlying geometry or forces acting upon any of the bodies. The rotations and positions of an individual object are also compressed separately: this avoids pitfalls of storing position information of objects rotating in place or storing rotation information of objects translating without rotating.

3.1. Online Compression

Our compression is *online*. It operates on the current frame from the simulator with a limited memory *window* of state information since the last keyframe dropped (see Fig. 5). Our integrated error allows us to track compression error using efficient running updates and sampling, rather than revisiting large numbers of previous frames for each new input frame. Our online method efficiently compresses very large scenes with many objects, as opposed to other methods which require the whole or significant portions of the simulation to be in memory during the compression [DP73, TJ07].

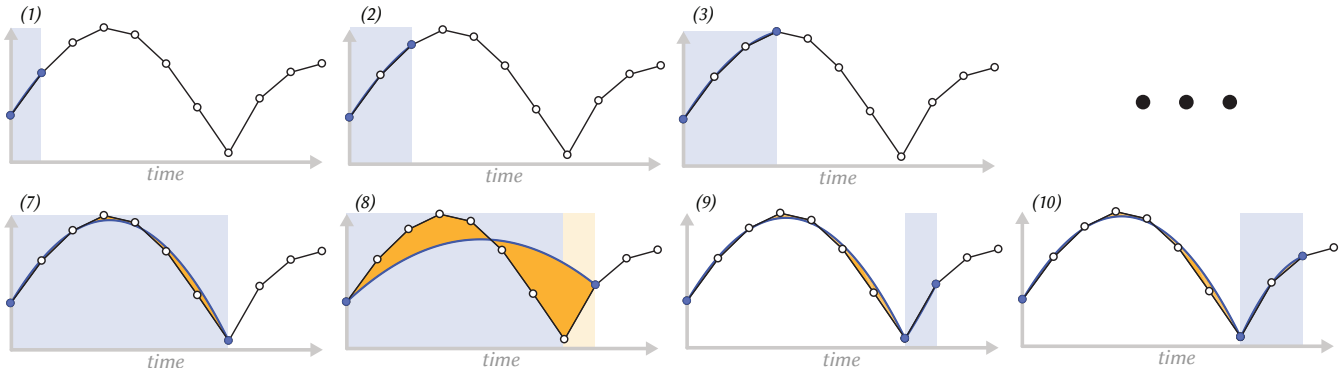

Figure 5: *As more frames are added into the evaluation window, the error in the fit is recomputed as seen in yellow. The frames continue to be added into the evaluation window until the error exceeds a threshold, in which case the system saves a keyframe corresponding to the previous frame. The evaluation window is then reset and new frames are evaluated for the next potential keyframe.*

The online compression pipeline for rotations and positions is the same in high-level structure (see Fig. 2). Upon receiving a new frame of raw state information we re-fit an interpolating polynomial, and evaluate the cumulative error between the reconstructed compressed data and the input simulation data. If the cumulative squared error exceeds user specified bounds, then it automatically records a keyframe in a compressed format in a method similar to Twigg et al. [TJ07] Our method is instead performed on the cumulative error as opposed to a maximum per-frame error bound. In addition to checking if the error bounds are exceeded, peak detection is performed on the cumulative error in order to detect sudden changes in object motion (e.g., from external forces or contact events). The output format of the keyframes is arranged in such a way that it is guaranteed to compress the input data even if the data does not follow the assumed interpolating functions, and provides largest compression ratios for objects undergoing free flight.

3.2. Peak Detector

We focus on *contact-dominated* rigid body simulations. Because an object's trajectory and rotation are not altered much at the *moment* of contact, our standard error thresholds $(\varepsilon_T, \varepsilon_R)$ will only drop a keyframe some frames later when the fit has high error. While technically error-bounded, missing contacts means missing visually (and scientifically) salient events, as can be seen in Fig. 4.

To capture salient events while still treating the simulator as a black box, we propose an additional peak detector for the trajectory and rotation signals with respective tunable thresholds σ_T and σ_R. The peak detector is designed to fire if we missed an event on the last frame: we only know this after the fact. If currently considering frame t_j, then we approximate the second derivative of the error $\ddot{E}$ at the previous frame t_{j-1} using central finite differencing:

$$\ddot{E} = E(t_j) - 2E(t_{j-1}) + E(t_{j-2}). \quad (1)$$

If the second derivative exceeds the peak threshold $\ddot{E} > \sigma$, then a keyframe is added for at t_{j-2}: that is, the frame *before* the spike was deemed to have occurred (see Fig. 6). Intuitively we are measuring the acceleration of the error, itself proportional to the acceleration of the object, which changes dramatically at contact (see Fig. 7).

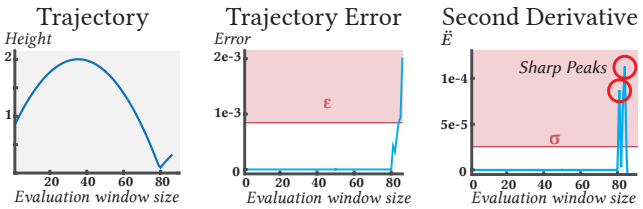

Figure 6: *$\ddot{E}$ is able to detect sudden changes in the trajectory quicker than the E metric, lowering ε increases the number of erroneous keyframes recorded by the system due to the accumulation of errors over longer evaluation windows.*

Figure 7: *Our keyframes (orange) identify collision events.*

Experimentally, it was found that adding another keyframe exactly after the contact frame greatly increased the visual effect on the decompressed animation as it helped separate regions which could be described with ballistic motions. We suspect that this is most necessary for simulator's using penalty forces where a contact "takes place" over more than one time step. Additional differences were noted with events which involved sudden changes in motion such as an object being thrown from a resting position, or an object coming to rest on a plane. Without the peak detector, the object would appear to move during the times when the object was at rest causing poor reconstructions of the decompressed data as seen in Fig. 4. For the sake of generality, we apply this peak detector in all cases.

The peak detector is able to detect the collision events sooner than using just the error signal. This can be seen in Fig. 6, where the spike in the error derivative occurs before the error bounds are exceeded. This small difference in time is visually salient in the reconstructed animations, as it prevents an interpenetration. The peak detector takes precedence over the standard error threshold (see Fig. 2) because it places keyframes one timestep earlier.

3.3. Trajectory Fitting

For trajectories, we use piecewise quadratic splines as inspired by the analytical solution for ballistic motion (see, e.g., [TJ07]):

$$\tilde{\mathbf{p}}_i(t) = \alpha_i(t-t_i)^2 + \beta_i(t-t_i) + \gamma_i, \tag{2}$$

where the coefficients $\alpha_i, \beta_i, \gamma_i \in \mathbb{R}^3$ correspond to the 3D acceleration, initial velocity and initial position of the body over a time interval beginning at t_i. This quadratic spline fitting is done in a memory efficient manner, only requiring the new frames from the data to be parsed as given from the simulator. We find a full least squares solution from all of the frames seen previously, and make no assumptions about the magnitude nor direction on which gravity acts. The following memory efficient method also applies to higher order fitting in cases where quadratic splines are insufficient to describe the motion effectively.

We optimize the coefficients of our quadratic fit to minimize the integrated squared error between the interpolated and input simulation trajectories, from the time of the previous keyframe t_i and the current simulation frame t_j, subject to equality constraints forcing interpolation of the trajectory at the previous keyframe $\mathbf{p}_i$ and at the current frame $\mathbf{p}_f$:

$$\alpha_i, \beta_i, \gamma_i = \underset{\alpha,\beta,\gamma}{\operatorname{argmin}} \underbrace{\frac{1}{2}\int_{t_i}^{t_j} \left\| \alpha(t-t_i)^2 + \beta(t-t_i) + \gamma - \mathbf{p}(t) \right\|^2 dt}_{E_T},$$

$$\text{subject to: } \gamma = \mathbf{p}_i \text{ and } \alpha(t_j-t_i)^2 + \beta(t_j-t_i) + \gamma = \mathbf{p}_j$$

where $\mathbf{p}(t)$ provides the input trajectory at any time t. Typically, simulators output trajectories as values at discrete time steps. Without loss of generality we assume the time step size is uniform, and discretize our error as a summation:

$$E_T \approx \frac{1}{2}\sum_{f=i}^{j} \|\alpha(t_f-t_i)^2 + \beta(t_f-t_i) + \gamma - \mathbf{p}_f\|^2 \tag{3}$$

.

The trajectory error is quadratic in the unknown coefficients (α, β, γ) and linear in the equality constraints, so the global optimum is the solution to the linear system of equations:

$$\begin{pmatrix} \mathbf{A}^\mathsf{T}\mathbf{A} & \mathbf{C}^\mathsf{T} \\ \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \lambda_i \\ \lambda_j \end{pmatrix} = \begin{pmatrix} \mathbf{A}^\mathsf{T}\mathbf{P} \\ \mathbf{p}_i \\ \mathbf{p}_j \end{pmatrix}, \tag{4}$$

where $\mathbf{A}, \mathbf{P} \in \mathbb{R}^{(j-i)\times 3}$ are matrices whose fth rows contain $[(t_f-t_i)^2, (t_f-t_i), 1]$ and $\mathbf{p}_f$ resp., $\mathbf{C} \in \mathbb{R}^{2\times 3}$ and λ_i, λ_j are the interpolation constraint matrix an Lagrange multipliers, resp. The resultant 5×5 system is solved directly.

We compute the scalar energy E_T induced by the optimal coefficients $\alpha_i, \beta_i, \gamma_i$, using the same matrices:

$$E_T = \frac{1}{2}\|\mathbf{A}\begin{pmatrix} \alpha_i \\ \beta_i \\ \gamma_i \end{pmatrix} - \mathbf{P}\|_F^2 \tag{5}$$

$$= \frac{1}{2} tr\left(\begin{pmatrix} \alpha_i \\ \beta_i \\ \gamma_i \end{pmatrix}^\mathsf{T} \mathbf{A}^\mathsf{T}\mathbf{A} \begin{pmatrix} \alpha_i \\ \beta_i \\ \gamma_i \end{pmatrix} + \mathbf{P}^\mathsf{T}\mathbf{P} - 2\begin{pmatrix} \alpha_i \\ \beta_i \\ \gamma_i \end{pmatrix} \mathbf{A}^\mathsf{T}\mathbf{P} \right).$$

If the energy at time t_j exceeds a user-specified error bound $E_T > \varepsilon_T$, then a new trajectory keyframe is added at t_{j-1} and the coefficients β_i, γ_i for the fit *between* t_i and t_{j-1} are saved. Recording the previous frame ensures that the data recorded can be interpolated to recreate the simulation output with a lower error than the maximum bound specified by the user.

We take advantage of our interpolation when choosing which data to save. Suppose we stored successive keyframes at $t_i < t_j < t_k$. Then the quadratic fits p_i and p_j describe the trajectories in the intervals $[t_i, t_j]$ and $[t_j, t_k]$ respectively. For interpolating trajectories, the end of one fit exactly equals the beginning of the next, so:

$$p_i(t_j) = p_{j-1}(t_j), \tag{6}$$

$$\gamma_j = \alpha_i(t_j-t_i)^2 + \beta_i(t_j-t_i) + \gamma_i, \tag{7}$$

$$\alpha_i = \frac{\gamma_j - \gamma_i - \beta_i(t_j-t_i)}{(t_j-t_i)^2}. \tag{8}$$

This progression reveals that we do not need to store α_i since it is always recoverable from the other information at this keyframe β_i, γ_i, t_i and information at the immediate next keyframe γ_j, t_j.

3.3.1. Cumulative Least Squares Optimization

Naively, if we did not drop a keyframe at t_j then we would need to append another row to the matrices $\mathbf{A}$ and $\mathbf{P}$, to compute the optimal coefficients and their energy for time t_{j+1}. As the window between keyframes grow to n frames , the memory requirements grows linearly $O(n)$, but worse the cost of constructing the matrices $\mathbf{A}$ and $\mathbf{P}$ grows quadratically $O(n^2)$. Since we conduct this computation for each frame as we grow the window, the total computation is cubic $O(n^3)$. This is an unacceptable punishment for finding a low-error and large fit (Though one endured by previous work): if the window is large, our method will obtain great savings with regards to runtime and memory usage.

Fortunately, neither solving the linear system in Equation (4) nor computing the energy in Equation (5) depend on explicitly constructing $\mathbf{A}$ or $\mathbf{P}$, we merely need to know $\mathbf{A}^\mathsf{T}\mathbf{A}$, $\mathbf{A}^\mathsf{T}\mathbf{P}$, and $\mathbf{P}^\mathsf{T}\mathbf{P}$.

For each, new frame of simulation data $\mathbf{p}_{j+1}$, we can update these matrices with the relevant 3×3 *outer* products. Introducing, the row vector $\mathbf{t} = [(t_{j+1}-t_i)^2, (t_{j+1}-t_i), 1]$, then:

$$\mathbf{A}^\mathsf{T}\mathbf{A} \mathrel{+}= \mathbf{t}^\mathsf{T}\mathbf{t}, \tag{9}$$

$$\mathbf{A}^\mathsf{T}\mathbf{P} \mathrel{+}= \mathbf{t}^\mathsf{T}\mathbf{p}_{j+1}, \tag{10}$$

$$\mathbf{P}^\mathsf{T}\mathbf{P} \mathrel{+}= \mathbf{p}_{j+1}^\mathsf{T}\mathbf{p}_{j+1}. \tag{11}$$

The system matrices $\mathbf{A}^\mathsf{T}\mathbf{A}$, $\mathbf{A}^\mathsf{T}\mathbf{P}$, $\mathbf{P}^\mathsf{T}\mathbf{P} \in \mathbb{R}^{3\times 3}$ are constant size, and so the memory footprint for adding additional frames to the fit remain unchanged. As a result, the total computation for a n-frame window is linear $O(n)$ and the memory required is constant $O(1)$.

3.4. Rotation Fitting

Exact solutions to the rotation of free rigid bodies exist, and provide a method to reconstruct the motions of free bodies efficiently [vZS07]. However, these methods depend upon having the inertia

tensor for each object in the scene. As our method has no information from the simulation, instead of performing an exact solve a piecewise spline approximation is performed.

We explored using first, second, or third order polynomials to fit the rotation part of the rigid body simulation. Higher order polynomials theoretically provide a more accurate fit but their coefficients grow in number, increasing storage size. For the gyroscopic example shown in the video, using second-order polynomial rotation fitting allowed for an average 30% larger keyframe window before exceeding the error thresholds, but doubled the storage size of the compressed output keyframe. For more typical scenes, the cost-benefit ratio is even worse. Based on this observation we instead interpolate rotations linearly by integrating a constant angular velocity. In contrast to spherical linear interpolation (a.k.a. "slerp") [Sho85, Jun15], we allow rotations by more than 2π between keyframes.

Our linear rotation fit interpolates the previous keyframe quaternion q_i at time t_i and the current quaternion q_j at time t_j. By modifying the standard spherical linear interpolation formulation, we can expand it to include multiple rotations around an axis. Using this the reconstructed rotation at a time t in between is given by:

$$q_i(t) = (q_j q_i^{-1})^{(r_i \frac{2\pi}{\theta}+1)\frac{t-t_i}{t_j-t_i}} q_i, \tag{12}$$

where $r_i \in \mathbb{N}$ is an *integer* encoding the signed number of full revolutions the object experiences between time t_i and t_j. When the difference in time $(t_j - t_i)$ is small, this number is typically zero $r_i = 0$. For fast spinning objects or long fits, this number may increase $|r_i| > 0$.

We measure the error of our fit as the integrated squared distances between the interpolated and input simulation rotations as quaternions. As for the trajectories in Section 3.3, we approximate this integral as a discrete sum over the input frames:

$$E_R = \int_{t_i}^{t_j} \min_{s=\pm 1} \left\| (q_j q_i^{-1})^{(r_i \frac{2\pi}{\theta}+1)\frac{t-t_i}{t_j-t_i}} q_i - s q(t) \right\|^2 dt \tag{13}$$

$$\approx \sum_{f=i}^{j} \min_{s=\pm 1} \left\| (q_j q_i^{-1})^{(r_i \frac{2\pi}{\theta}+1)\frac{t_f-t_i}{t_j-t_i}} q_i - s q_f \right\|^2, \tag{14}$$

where s accounts for the unit quaternions' double cover of the rotation group (i.e., $q = -q$) and θ is the known rotation angle between q_i and q_j. For the degenerate case where $q_i = q_j$, we use the axis of rotation from the previous non-degenerate frame (for low error fits, this typically only happens due to non-general position aliasing and we just need to make it to the next frame).

Due to our interpolation constraints, the only unknown r_i. The formula inside the norm is nonlinear with respect to r_i, but the integer constraint implies a brute force solution. The minimal E_R must be achieved for some integer $r_i \in [-(j-i), \ldots, 0, \ldots, (j-i)]$. In practice, r_i is usually very small, so we use a truncated search over values sorted by absolute values $r_i \in [0, 1, -1, 2, -2]$.

Just as in the trajectory fitting, if the minimal error exceeds a user-specified threshold $E_R > \varepsilon_R$, then a keyframe is dropped at the *previous frame* t_{j-1} and the vector part of the quaternion $\pm q_{j-1}$ and the best integer r_i for the window t_i to t_{j-1} are saved.

When recording quaternion keyframe $q = xi + yj + zk + w$, we record the vector part x, y, z or $-x, -y, -z$ depending if w is positive or negative. This way, we can utilize the fact that rotation quaternions have unit norm $\|q\| = 1$ and quaternions double cover rotations $q = -q$ to reconstruct a valid non-negative scalar part on the fly: $w = \sqrt{1 - x^2 - y^2 - z^2}$.

For a window of n frames, each energy evaluation requires $O(n)$ computation, meaning total computation for this window will be quadratic $O(n^2)$. The non-linearity of rotations prevents us from applying the same cumulative least-squares trick in Section 3.3.1. In our experiments, we observe that a constant size Monte-Carlo estimation of the integral in Equation (13) is just as effective as the full summation. To compute our estimation of E_R we sample rotations randomly in the time window scaling appropriately and avoiding the endpoints where interpolation forces small error. This sampling drastically reduces the storage cost for this computation, as the whole set of data does not need to be stored from the previous recorded keyframe.

3.5. Decompression Scheme

As supplemental material, we include an example of efficient online decompression. This system is able to play large simulations with ease using a simple interface (Fig. 8).

With access to only two keyframes per object, the system can efficiently interpolate between them using the piecewise spline basis functions. This drastically reduces the memory cost for replaying the simulation. As the compression system is able to transform the discrete samples from the simulation into a piecewise continuous representation, the simulation can be replayed at any temporal resolution without the need for resimulation. An example of this can be seen with the tomahawk example as seen in Fig. 3, and in the video.

The included online decompressor includes a playback and scrubbing interface, similar to those found in professional video editing

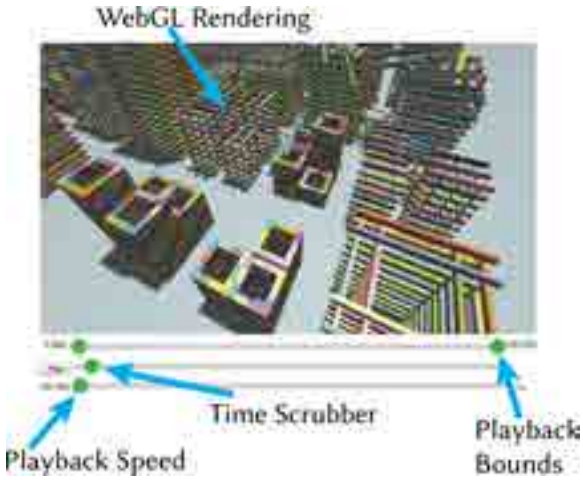

Figure 8: *Our WebGL decompressor: After loading the corresponding files, it allows for easy real-time visualization of the compressed datasets. The camera can be repositioned as desired by interacting with the rendering window, and the simulation can be explored with the sliders (see the video for full sequence).*

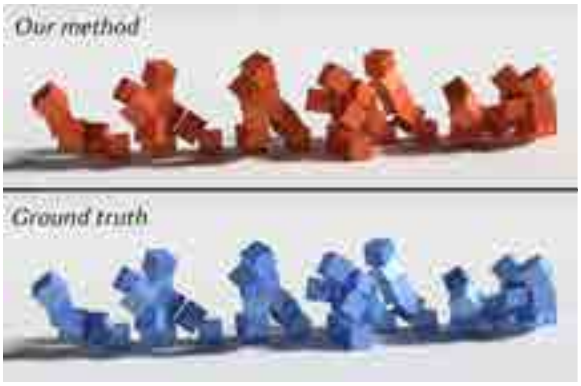

Figure 9: *The compression of falling boxes with the input simulation performed in Houdini's rigid body solver*

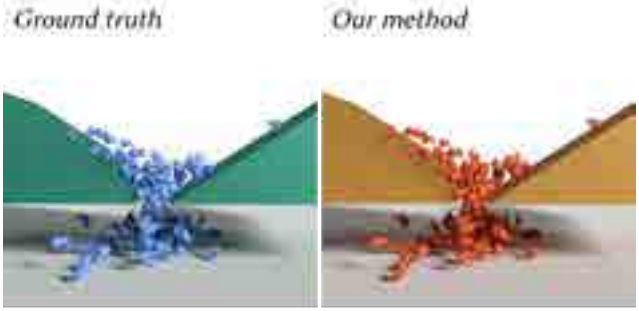

Figure 10: *The compression of falling bunnies simulated in SCIsim with the reference simulation on the left and our compressed version on the right, achieving a 71 times compression ratio over the input simulation at 24 frames per second (see video @ 4:15)*

software as seen in Fig. 8. It also allows the user to set the desired playback speed, and loop through interesting portions of the animations. With this decompressor, it is able to replay large simulations with many objects which would not be possible with the standard method. The city example when uncompressed exceeds 30 GB (at 240 fps), far too large to be viewed normally but is able to be viewed easily when decompressed in real time.

4. Results and Discussion

We tested our compression algorithm using a 32-core Xeon E5-2637 v3 running at 3.5 GHz, with 64 GB of RAM. Table 1 shows the performance of our algorithm (both in terms of wall clock time and compression rate) across a number of examples of varying complexity. Of most importance is that our method achieves higher compression rates than simply storing data at low frame rates (see video for examples). We accomplish this while still retaining essentially infinite temporal precision allowing for interesting editing operations such as adding slow motion to movie scenes. We also demonstrate that our method works well for a number of popular simulation packages (Houdini [Sid17] in Fig. 3.5, SCIsim [SKV*12] in Fig. 3.5 and Bullet [Cou15] in all other Figs.) achieving between 5x and 360x compression rates. Again, our method is simulator

Figure 11: *As error bounds increase, data can be compressed further at the cost of a poorer reconstruction.*

agnostic so these results were obtained with no changes to our algorithm or implementation. Finally, we show that our compression times are negligible compared to the required simulation time (our slowest example takes only 10% of the simulation time). Because our streaming compression is pipelined with the simulation and can be parallelized, the actual waiting time imposed on the user would be much less in practice.

The changes in the compression ratios between examples stem from the nature of the method being able to compress ballistic motion, but suffers upon increased number of contact events. Even in the examples with exceedingly high number of contact events such as the drum example, the method is still able to compress the simulation. Through the implementation of the output data format, the method is guaranteed to never exceed the input simulation size.

4.1. Parameter Selection

fps	ε_T	ε_R	σ_T	σ_R
24	5e-2	5e-2	5e-4	5e-2
120	5e-2	5e-2	1e-4	1e-2
1000	5e-2	5e-2	5e-5	5e-3

The selection of the error bounds and peak detector values ε, σ respectively change the quality of the compression ratios and the reconstructions as can be seen in Fig. 6 and Fig. 11. Raising the error bounds to higher thresholds allows for smaller compressed filesizes, with a lower limit arising due to the number of collision events. Raising the bounds higher introduces more error in the reconstruction and smoothes out the effect of contacts. The optimal values for the peak detector's σ depend on the timestep of the input simulation, due to the derivative calculated in input frame space. Through experimentation it was determined that the optimal values for the parameters are as shown.

4.2. Accuracy

All results obtained from the decompression of compressed files are visually identical to the input simulation data (see the accompanying video), and are able to reconstruct the data at any temporal resolution requested. The reconstruction is able to recreate the simulation, even to the state of reproducing simulation artifacts in the input

simulation	# bodies	length	runtime	fps	raw size	our runtime	our size	ratio	# peaks	# error fits
Drop	1	1.4s	1.4s	1000	83 KB	0.17s	528 B	160×	2	0
Bounce	1	6s	6s	1000	357 KB	0.39s	2.3 KB	155×	40	0
Throw	1	4s	4s	1000	250 KB	0.50s	5.8 KB	43×	130	0
Spin	1	3.5s	3.5s	120	50 KB	0.08s	416 B	120×	0	1
Tomahawk	1	4s	4s	24	6.3 KB	0.44s	248 B	25×	0	1
Gyro Spin	1	23s	23s	120	330 KB	0.17s	12 KB	28×	0	121
Cylinder Roll	1	5s	30s	24	2 KB	0.05s	1.2 KB	1.6×	8	13
Orbits	5	10s	10s	24	68 KB	0.14s	6.3 KB	11×	78	18
Houdini boxes	55	10s	10s	24	530 KB	0.3s	104 KB	5×	1222	641
Tower	107	20s	155s	1000	88 MB	5.5s	244 KB	360×	1378	3473
SCIsim bunnies	200	10s	1800s	24	43 MB	0.9s	602 KB	71×	6764	4958
Large City	6019	300s	*17 hours*	120	17 GB	2650s	28 MB	607×	260542	281244
Granular Flow	6571	30s	*10 hours*	120	5.5 GB	596s	1.6 GB	3.5×	2e7	3e6

Table 1: *We report both total compressed size and time to compress, along with the number of key frames dropped due to error-bounds exceptions and due to peak detection. Although downsampling the input animation is a way to reduce the total storage cost, it can be seen that the compressed representation is still smaller for nearly all of the examples.*

Figure 12: *When compressed to the same output file size as our algorithm, PCA (3 components) and Douglas Peucker ($\varepsilon = 1e-3$) show significant artifacts in the decompressed simulation*

simulation as seen at the end of the "bounce" example. Running the compression algorithm on lower sample rate input simulations yields smaller compressed file sizes than high-resolution ones and takes less computation time. However, this provides visually poorer reconstruction around the contact events when upsampled. Contacts are not often well resolved during large timestep simulations and so we inherit this fundamental issue from the simulation itself.

We also performed visual comparison with standard algorithms which are often applied for animation compression. Our comparisons to Principal Component Analysis (PCA) and Douglas Peucker line fitting (see Fig. 12) make it clear that our compressed output is of much higher visual fidelity and more closely tracks the input simulation. Both previous methods exhibit distracting artifacts with the rotational errors exhibited by PCA being particularly egregious. More impressive is that we achieve these superior results without needing to analyze the entire set of simulation data (unlike the methods we compare against).

It is worth noting that our results can be compressed further with other independent compression tools such as bzip2. Here we remove the ability to operate in a streaming fashion in exchange for an approximately 1.3× additional compression as seen in Table 2. In these experiments bzip2 is applied to the raw binary frame data straight from the simulator, and then to the compressed binary data from our method. Although the bzip2 is able to reduce input simulation filesize, it is not able to achieve results similar to our method. Our output can be further compressed with bzip2.

The high compression rates yielded by our approach open up new avenues for disseminating and interacting with rigid body simulation data. Our supplemental video shows an example of a web-based viewer which allows for full 3D viewing and nonlinear scrubbing through large simulation datasets. Our compressed versions of the simulations not only loads but permits interactive viewing, scrubbing and re-timing. Prior to our algorithm, sharing and interacting with such data in this manner was difficult if not impossible. There are still limitations to our WebGL decompressor, where our largest scene (6019 bodies, 5.5 GB uncompressed, 1.6 GB compressed) cannot be interacted with due to memory constraints.

4.3. Limitations & Conclusion

In this work we have presented an online streaming compression algorithm for contact-dominated rigid body simulations. Our method is simulator agnostic, operates inline with the simulation itself and can achieve compression rates of up to 360 times (while taking only a fraction of the simulation time). We have demonstrated our method

Example	Input	bzip2	ratio	Ours	Our ratio	Ours + bzip2	bzip2 ratio	combined ratio
Bounce	175 KB	76 KB	2.3×	2.3 KB	76×	1.6 KB	1.4×	109×
Tower	21 MB	16 MB	1.3×	244 KB	86×	184 KB	1.3×	114×
SCIsim bunnies	43 MB	31 MB	1.4×	602 KB	72×	461 KB	1.3×	93×

Table 2: *Comparison of file sizes before and after compression with bzip2. Input resolutions for the uncompressed files are at 240 fps. The bzip2 algorithm is applied to the raw simulation data and also to the compressed output from our method. The compression achieved by our method is orders of magnitude better at compressing the input simulation. The specific bzip2 command used was* `bzip2 -z -9 simulation.dat`

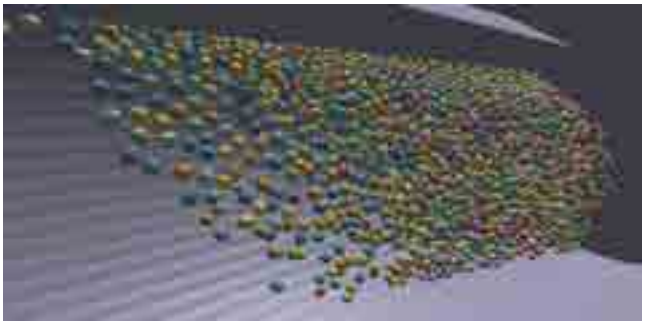

Figure 13: *Granular bodies are spun in a rotating drum at a constant velocity. This example is able to compress the input simulation 3.5× smaller, unfortunately still too large for our online decompressor to handle at interactive rates.*

on a wide range of simulated scenes of varying complexity, taken from three different rigid body simulation packages. We are excited about the prospects that this work opens up. Beyond the ability to simply store data or transmit it more quickly to a render farm, our compression scheme opens up new ways of interacting with simulation data. From exploring 3D web-based worlds to directly using three dimensional simulation layers in video editing tools to easily sharing scientific data in remote locations with potentially poor connectivity. Below we outline limitations of our work along with exciting avenues for future work.

Our compression method is inherently a lossy algorithm, and is unable to exactly recreate the input simulation. The maximum error is bounded through the construction of the compression scheme, and so the reconstruction will be close to the input.

The compression ratios from the algorithm are highly dependent upon the input simulation. Scenes with very high number of collision events will store many keyframes and will be significantly larger than scenes with fewer collisions. This drawback is seen with our granular flow example Fig. 13, where the compression ratio is low. Our method is guaranteed to never increase the amount of data over the input simulation and so the method can be applied to all simulations without fear of exploding filesizes. Error bounds are parameters, which should ideally be tuned for each simulation. Smaller, faster objects may need smaller bounds.

While our compressor acts across time, we also explored compressing across space via object clustering. Investigations into clustering were performed using PCA and overcomplete dictionary dimensionality reduction.

Clustering of trajectories is very well studied, but usually in the context of many objects undergoing a similar route (e.g., shipping convoys). In contrast, rigid body objects may travel along highly *uncorrelated* trajectories, especially during explosions [SXYL16].

The clustering approaches investigated were deemed insufficient due to the constraint that the decompressed clustered animations must be exact in order to achieve the error bounds set within our method. These methods clustered similar (but not identical) transformations, yielding in poor reconstruction. Storing an "offset" to the cluster centers yields a storage requirement equivalent to saving the absolute position of each object, and thus increases the overall storage cost of the method. Clustering was performed in velocity space, but has not yielded sufficient gains in the compression ratios with the low error bounds set for the method.

It would be interesting to explore enriching our compression function space. For instance, when rigid bodies come to rest upon a surface they are affected by friction. This leads to a form of exponential falloff in the velocity of the object which is not well represented with a quadratic. This is only a problem with simulations which have a long time with frictional contact between objects, and in the worst case our method will add additional keyframes to minimize the error. Including more physically aware basis functions into our method could lead to further improvements for some scenes.

The same could be said about our rotational function space. The physics for a rotating object cannot be exactly represented with our linear interpolation method. While exact solution to the free-flight Newton-Euler equations exist [vZS07], they require computing the moments of inertia of the object, which is goes against our desire to be geometry and simulation agnostic. However, exploring approximations which observe these properties could be beneficial.

Finally, we would like to explore using our keyframed representation for editing. Editing rigid body simulations requires changing the parameters and resimulating either portions or the entire scene. By isolating the collision events and modifying them, it could be possible to change the behavior of the simulation without needing to resimulate entirely. Although this would not create physically accurate simulations, it could help creators have more control over the output of these algorithms.

Acknowledgements

This work is funded in part by NSERC Discovery Grants (RGPIN-2017-05235 & RGPAS-2017-507938, RGPIN-2017-05524, RGPAS-2017-507909), Connaught Funds (NR2016-17), the Canada Research Chairs Program, and a gift by Adobe Systems Inc.

References

[AM00] ALEXA M., MÜLLER W.: Representing animations by principal components. *Comput. Graph. Forum* (2000). 3

[Ari06] ARIKAN O.: Compression of motion capture databases. *ACM Trans. Graph.* (2006). 3

[ASK*12] AKHTER I., SIMON T., KHAN S., MATTHEWS I., SHEIKH Y.: Bilinear spatiotemporal basis models. *ACM Trans. Graph.* (2012). 3

[Cou15] COUMANS E.: Bullet physics simulation. In *ACM SIGGRAPH Courses* (2015). 2, 7

[DP73] DOUGLAS D. H., PEUCKER T. K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2 (1973), 112–122. 2, 3

[dVvS10] DE VRIES G., VAN SOMEREN M.: *Clustering Vessel Trajectories with Alignment Kernels under Trajectory Compression*. 2010. 2

[EWPT17] EBERHARDT S., WEISSMANN S., PINKALL U., THUEREY N.: Hierarchical Vorticity Skeletons. *Proceedings of the Symposium on Computer Animation (SCA '12) to appear* (Jun. 2017), 11. 3

[Goo17] GOODHUE D.: Velocity-based compression of 3d animated rotations. In *ACM SIGGRAPH Posters* (2017). 3

[HNB*06] HOUSTON B., NIELSEN M. B., BATTY C., NILSSON O., MUSETH K.: Hierarchical rle level set: A compact and versatile deformable surface representation. *ACM Trans. Graph. 25*, 1 (Jan. 2006), 151–175. 3

[IR03] IBARRIA L., ROSSIGNAC J.: Dynapack: Space-time compression of the 3d animations of triangle meshes with fixed connectivity. In *Procöf SCA* (2003). 3

[JDKL14] JACOBSON A., DENG Z., KAVAN L., LEWIS J.: Skinning: Real-time shape deformation. In *ACM SIGGRAPH Courses* (2014). 3

[JSK16] JONES A. D., SEN P., KIM T.: Compressing fluid subspaces. In *SCA* (Aire-la-Ville, Switzerland, Switzerland, 2016), Eurographics Association, pp. 77–84. 3

[JT05] JAMES D. L., TWIGG C. D.: Skinning mesh animations. *ACM Trans. Graph. 24*, 3 (July 2005), 399–407. 3

[Jun15] JUNG J.: Compressing skeletal animation data. https://engineering.riotgames.com/news/compressing-skeletal-animation-data, 2015. Accessed: 2018-01-23. 2, 3, 6

[KG04] KARNI Z., GOTSMAN C.: Compression of soft-body animation sequences. *Computers & Graphics* (2004). 3

[KISS15] KEINERT B., INNMANN M., SÄNGER M., STAMMINGER M.: Spherical fibonacci mapping. *ACM Trans. Graph.* (2015). 3

[LM06] LIU G., MCMILLAN L.: Segment-based human motion compression. In *Proc. of SCA* (2006). 3

[MCMJ15] MÜLLER M., CHENTANEZ N., MACKLIN M., JESCHKE S.: Long range constraints for rigid body simulations. In *Proc. SCA* (2015). 2

[MLDH15] MAGLO A., LAVOUÉ G., DUPONT F., HUDELOT C.: 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Comput. Surv.* (2015). 3

[PJAP07] PREDA M., JOVANOVA B., ARSOV I., PRÊTEUX F.: Optimized mpeg-4 animation encoder for motion capture data. In *Proc. of Web3D* (2007). 2, 3

[Ram72] RAMER U.: An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing 1*, 3 (1972), 244–256. 2

[RL12] REIN H., LIU S.-F.: REBOUND: an open-source multi-purpose n-body code for collisional dynamics. *Astronomy & Astrophysics* (2012). 3

[RP12] ROSEN P., POPESCU V.: Simplification of node position data ;for interactive visualization of dynamic data sets. *IEEE Trans. Vis. Comput. Graph.* (2012). 2, 3

[Sho85] SHOEMAKE K.: Animating rotation with quaternion curves. *SIGGRAPH* (1985). 3, 6

[Sid17] SIDEFX: *SideFX Houdini*, 2017. 7

[SKV*12] SMITH B., KAUFMAN D. M., VOUGA E., TAMSTORF R., GRINSPUN E.: Reflections on simultaneous impact. *ACM Trans. Graph. 31*, 4 (July 2012), 106:1–106:12. 2, 7

[SSK05] SATTLER M., SARLETTE R., KLEIN R.: Simple and efficient compression of animation sequences. In *SCA* (New York, NY, USA, 2005), ACM, pp. 209–217. 3

[SXYL16] SUN P., XIA S., YUAN G., LI D.: An overview of moving object trajectory compression algorithms. *Mathematical Problems in Engineering* (2016). 2, 9

[TJ07] TWIGG C. D., JAMES D. L.: Many-worlds browsing for control of multibody dynamics. *ACM Trans. Graph. 26*, 3 (July 2007). 3, 4, 5

[TWC*09] TOURNIER M., WU X., COURTY N., ARNAUD E., REVÉRET L.: Motion compression using principal geodesics analysis. *Comput. Graph. Forum* (2009). 3

[VMHB14] VASA L., MARRAS S., HORMANN K., BRUNNETT G.: Compressing dynamic meshes with geometric laplacians. *Computer Graphics Forum 33*, 2 (2014), 145–154. 3

[VSK*17] VOUGA E., SMITH B., KAUFMAN D. M., TAMSTORF R., GRINSPUN E.: All's well that ends well: Guaranteed resolution of simultaneous rigid body impact. *ACM Trans. Graph.* (2017). 2

[vZOS08] VAN ZON R., OMELYAN I. P., SCHOFIELD J.: Efficient algorithms for rigid body integration using optimized splitting methods and exact free rotational motion. *The Journal of chemical physics* (2008). 2

[vZS07] VAN ZON R., SCHOFIELD J.: Numerical implementation of the exact dynamics of free rigid bodies. *Journal of Computational Physics 225*, 1 (2007), 145 – 164. 5, 9

[Wel84] WELCH T. A.: A technique for high-performance data compression. *Computer* (1984). 3

[WPS14] WEISSMANN S., PINKALL U., SCHRODER P.: Smoke rings from smoke. *ACM Trans. Graph. 33*, 4 (July 2014), 140:1–140:8. 3