Expediting Precomputation for Reduced Deformable Simulation

Yin Yang* Dingzeyu Li† Weiwei Xu‡ Yuan Tian§ Changxi Zheng†

*University of New Mexico † Columbia University ‡Hangzhou Normal University § U.T. Dallas

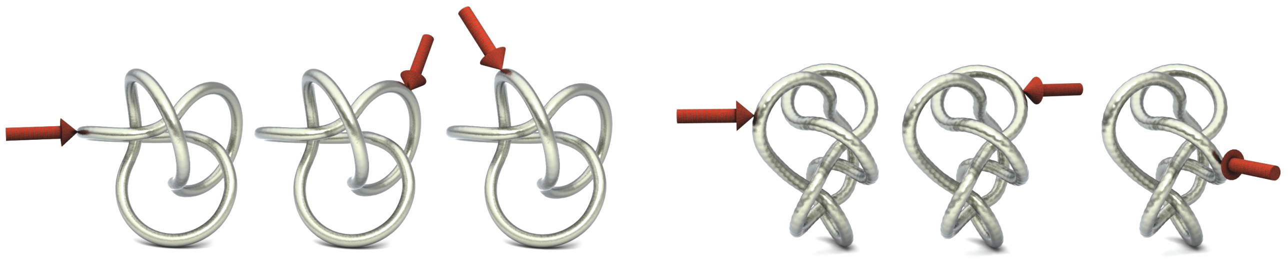

Figure 1: Vibrant wires. *A metal wire model consists of 76,863 elements, simulated using 120 nonlinear modes. The entire precomputation took 48 seconds, which includes the construction of 495 reduced modes (30 linear inertia modes and 465 first-order inertia derivatives), the generation of 680 training poses, and the computation of weights of 342 cubature points using nonlinear St. Venant-Kirchhoff material. Arrows indicate the locations and directions of external forces. Fast precomputation allows us to quickly update the geometry or material, and simulate the wire's deformations, vibrations, and, in this case, the sounds (see the supplemental video).*

Abstract

Model reduction has popularized itself for simulating elastic deformation for graphics applications. While these techniques enjoy orders-of-magnitude speedups at runtime simulation, the efficiency of precomputing reduced subspaces remains largely overlooked. We present a complete system of precomputation pipeline as a faster alternative to the classic linear and nonlinear modal analysis. We identify three bottlenecks in the traditional model reduction precomputation, namely modal matrix construction, cubature training, and training dataset generation, and accelerate each of them. Even with complex deformable models, our method has achieved orders-of-magnitude speedups over the traditional precomputation steps, while retaining comparable runtime simulation quality.

Keywords: Finite element method, Deformable simulation, Precomputation, Model reduction

1 Introduction

Model reduction methods have become popular in many graphics applications, ranging from creating and controlling animations [Barbič and James 2005; An et al. 2008; Barbič et al. 2009], material and shape design [Xu et al. 2015; Wang et al. 2015] to realistic sound synthesis [Zheng and James 2011]. In all these cases, a small number of deformation basis vectors or modes are computed beforehand. The runtime simulation then constrains deformations to a subspace spanned by these modes, tremendously reducing simulated degrees of freedom. While enjoying large runtime performance boost, model reduction methods need to *carefully* construct a set of modes that well express possible deformations during the simulation. This precomputation, however, is usually an expensive step that can take a long time.

The conventional wisdom here is to tax the preprocessing step in exchange for runtime performance. Indeed, if the object geometry and material properties have been decided, it is worthwhile and affordable to precompute once for repeated runtime simulations. However, when the shape or material is frequently altered—for instance, in the case where a user is exploring different animation settings, every geometric and material update dictates a re-computation of the reduced model. Such a long precomputation time would drastically slow down the work flow. In this paper, we focus on accelerating the entire precomputation step in a typical reduced deformable simulation pipeline, a problem that has been largely overlooked.

The standard precomputation of a reduced model undergoes three expensive sub-steps. First, the reduced modes are typically extracted using the modal analysis or principle component analysis (PCA) methods, both of which rely on a generalized eigen analysis or singular value decomposition. For high-resolution finite element meshes, this computation is costly. Secondly, when simulating nonlinear deformations, model reduction methods need to evaluate the object's nonlinear internal forces at runtime. Such runtime evaluations are accelerated using cubature schemes [An et al. 2008; von Tycowicz et al. 2013], but the precompution of cubature points and their weights remains computationally expensive. Lastly, to train the cubature scheme, one also needs to prepare for a set of training poses simulated using a full-space simulation, a typically expensive process. Additionally, to construct the training dataset, there is always a question of what kind of training poses to be included. Insufficient training poses lead to unsatisfactory subspace construction, while excessive poses unnecessarily increase the cost of data preparation.

In light of these challenges, we propose a comprehensive system that precomputes reduced deformable models at a significantly accelerated rate. It features the following technical contributions:

1. We first speed up subspace modal construction. We augment the linear inertia mode technique [Yang et al. 2013] originally designed for substructured objects, and thereby sidestep the expensive eigen decomposition needed in traditional modal construction. We further propose a reduced Gram-Schmidt orthogonalization scheme which quickly regularizes resulting modal vectors, as well as a formula for computing nonlinear modes up to an arbitrary order. Lastly, to extract a compact set of modes, we adopt a random projection algorithm originated in the field of data mining. As a result, our method at this step is 20-40× faster than standard modal-analysis-based methods.
2. We accelerate the precomputation of cubature points and weights. Our algorithm is based on the Newton-Cotes rule,

sampling cubature points uniformly in the solid body and optimizing the cubature weights individually and in parallel across all reduced modes. This cubature training step can be finished within milliseconds.

3. We present a new cubature training scheme to effectively prepare training samples. We extend the idea of training pose sampling [von Tycowicz et al. 2013] and interpret the training step as a Monte Carlo process for finding the expected error of the reduced simulation. From this point of view, we propose a method that incrementally adds training samples and stops when no more samples are needed. This method saves us from generating unnecessary training samples, ultimately reducing the precomputation cost.

Integrating all these technical components, we are able to significantly accelerate the entire precomputation step. Compared to the standard pipeline, we gain orders-of-magnitude speedups in preprocessing while retaining comparable runtime simulation quality. Furthermore, our method is independent from any particular reduced simulation method, so it works in tandem with different types of simulators, including single-body and substructured deformable simulation as well as sound simulation.

2 Related Work

Since the seminal work of [Terzopoulos et al. 1987; Terzopoulos and Fleischer 1988], finite element method (FEM) has become one of the primary tools in computer graphics to simulate elastic deformable objects. The governing dynamic equation and specific elastic material models are based on well-established theory of continuum mechanics [Bonet and Wood 2008]; the FEM solves the equation on a discretized volumetric mesh. We refer the reader to a comprehensive survey [Nealen et al. 2006] of fast deformable models developed in graphics.

FEM, especially when used with high-resolution meshes, is computationally expensive. Therefore, numerous research has been focusing on improving its simulation efficiency. For example, multiresolution [Capell et al. 2002b] and adaptive simulations [Wu et al. 2001; Grinspun et al. 2002] use hierarchical spatial discretization of the simulation domain to accelerate the computation. These types of techniques use high-level bases to represent coarse deformations and refine them for more detailed local deformations if necessary. Other types of mesh discretizations like embedded meshes, coarsened meshes, cages, and skeletons [Capell et al. 2002a; Kharevych et al. 2009; Nesme et al. 2009; García et al. 2013] offer further performance boost or allow deformation control.

One important type of fast finite element methods is well-known as *model reduction methods*. Model reduction is designed to accelerate the simulation speed by reducing the numbers of degrees of freedom (DoFs) of the simulation system. This is achieved by observing that the displacement of a deformable object can be well approximated using a small number of basis vectors. With a set of precomputed basis vectors, a reduced model projects an original high-dimensional system onto a low-dimensional subspace spanned by the bases. While this technique appears straightforward, it is rather challenging to construct a "good" subspace basis because the deformation that an elastic body will undergo is unknown. One of the most widely-used techniques relies on the vibrational analysis or *modal analysis*. The basis vectors are calculated as the vibrational modes at different frequencies. High frequency modes are of higher elastic energies and thus less likely to occur in the runtime deformation. Therefore, they are discarded for system reduction. Modal analysis has been used for interactive deformable simulation since the early work of [Pentland and Williams 1989]. Afterwards, there have been many significant work sharing this core idea, such as character skinning [Kry et al. 2002; Kim and James 2011], deformable animation control and editing [Hauser et al. 2003; Barbič and Popović 2008; Huang et al. 2011; Barbič et al. 2012], as well as subspace collision resolution [James and Pai 2004; Barbič and James 2010; Teng et al. 2014]. It has also been extended to simulate large deformation based on co-rotational elasticity [Müller et al. 2002; Choi and Ko 2005; Hecht et al. 2012], nonlinear deformable materials [Barbič and James 2005; An et al. 2008; von Tycowicz et al. 2013], and multi-domain elasticity [Barbič and Zhao 2011; Kim and James 2011; Yang et al. 2013].

Our work is to complement these work by addressing an orthogonal problem, the problem of constructing the subspace bases rapidly. Consequently, our method can be used with different model reduction methods, as shown in §7.

Besides modal analysis, other methods have also been used for constructing subspace bases. Meyer and Anderson [2007] used pre-selected key points to build the subspace. Kim and James [2009] used earlier frames in an online simulation to create an adaptive and time-varying subspace for subsequent simulation. Their later work [Kim and James 2011] uses PCA to prune a pose set to obtain basis vectors. Meanwhile, other recent work constructs the subspace by enriching local deformable modes [Harmon and Zorin 2013; Hahn et al. 2014].

3 Background and Overview

Reduced Model Simulation. Consider a finite element mesh with n nodes. Model reduction approximates its deformation using a linear combination of a set of modal vectors U, or $\boldsymbol{u} = \mathsf{U}\boldsymbol{q}$, resulting in a reduced Euler-Lagrange equation:

$$\mathsf{M}_q \ddot{\boldsymbol{q}} + \boldsymbol{f}_{dis}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{f}_{ext}, \tag{1}$$

where $\mathsf{M}_q = \mathsf{U}^\top \mathsf{M} \mathsf{U}$, $\boldsymbol{f}_{dis}$ and $\boldsymbol{f}$ are the reduced mass matrix, dissipative force, and internal force, respectively. The reduced internal force $\boldsymbol{f}$ is a function of the modal displacement $\boldsymbol{q}$, and its Jacobian matrix, also referred as the reduced stiffness matrix, is $\mathsf{K}_q(\boldsymbol{q})$. $\boldsymbol{f}_{dis}$ is often computed as $(\zeta \mathsf{M}_q + \xi \mathsf{K}_q(\boldsymbol{q}))\dot{\boldsymbol{q}}$ (known as Rayleigh damping). $\boldsymbol{q} \in \mathbb{R}^r$ is a low-dimensional vector (i.e., $r \ll 3n$). Thus, solving the nonlinear equation (1) is significantly faster than solving the its unreduced counterpart.

Precomputation Overview. Figure 2 outlines the proposed precomputation step. Given an input mesh, we first construct a set of linear deformation modes, known as linear inertia modes (§4.2). To further capture nonlinear shape deformations during the runtime simulation, we also compute asymptotic inertia derivatives to generate higher-order nonlinear modes. We then condense all the linear and nonlinear modes into the final mode matrix U using a random projection method (§4.3). Next, we uniformly sample a set of cubature points on the object's mesh. Their cubature weights are calculated independently for every reduced mode (§5). The cubature is trained iteratively: in each iteration, we add a few training poses; we stop the iteration when an error metric converges (§6). At that point, a reduced deformable model, including a set of deformation modes and related cubature weights, are ready for subsequent runtime simulations.

4 Fast Construction of Reduced Modes

We start by describing our fast algorithm for precomputing reduced modes. This algorithm is built on the *Krylov subspace*

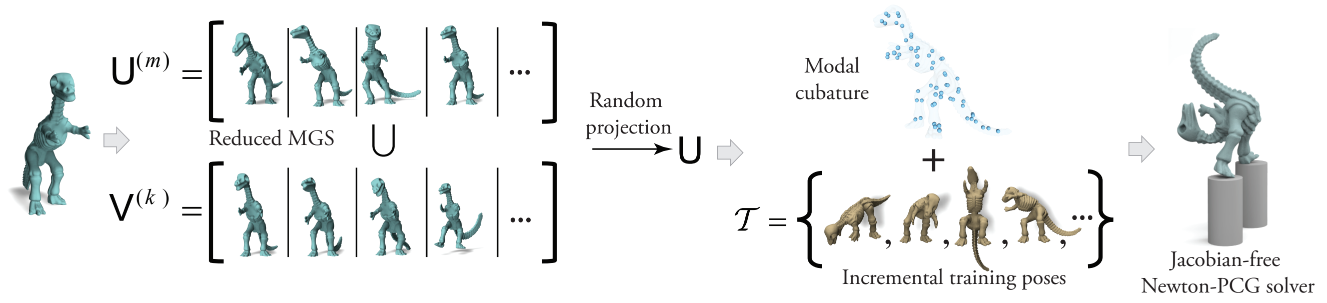

Figure 2: Precomputation overview. *For a given input 3D model, we compute a set of linear ($\mathsf{U}^{(m)}$) and nonlinear ($\mathsf{V}^{(k)}$) inertia modes using augmented Krylov iterations, and then create the final mode matrix U using random projection. For cubature training, we generate training poses incrementally. At runtime, the reduced model is simulated using a proposed Jacobian-free Newton-PCG solver.*

method, a well-known technique for model reduction in structural analysis. We propose a method to accelerate the mode construction (§4.2) and further extend the Krylov iteration to handle higher-order and nonlinear deformations (§4.3).

4.1 Background on Linear Inertia Mode

Traditionally, reduced modes are constructed using linear modal analysis, which involves a eigenvalue decomposition, a computation that is generally expensive and has limited room for an acceleration. Instead, we use the Krylov subspace method [Craig and Hale 1988; Craig 2000]. In graphics, this method has been used for computing substructured modes by [Yang et al. 2013]. The modes, known as *linear inertia modes*, are computed recursively (up to an order of m):

$$\mathsf{U}^{(m)} = \mathsf{A}^{m-1}\mathsf{U}^{(0)}, \quad \text{where } \mathsf{A} = \mathsf{K}^{-1}\mathsf{M}, \tag{2}$$

where K and M are respectively the rest shape stiffness and mass matrices, and $\mathsf{U}^{(0)}$ is for mode initialization, typically chosen as the object's six rigid-body modes (i.e., $\mathsf{U}^{(0)} = \mathsf{U}^r$). Consequently, each order of linear modes construction can be in interpreted as generating a new set of inertia deformation. Essentially, Eq. (2) constructs a Krylov subspace of order m, denoted as $\mathcal{K}_{(m)} \triangleq \mathsf{span}(\mathsf{U}^{(1)}) \cup \cdots \cup \mathsf{span}(\mathsf{U}^{(m)})$, where $\mathsf{span}(\mathsf{B})$ stands for the column space of a matrix B. We also note that repeated premultiplication of matrix A makes higher-order modes closer to its dominant eigen vectors. As a result, high-order linear inertia modes represent subtle localized deformations, similar to the standard high-frequency modal vibrational modes.

Unconstrained Inertia Mode. Eq. (2) introduced in [Yang et al. 2013] was to construct reduced modes of *substructure components* with well-defined boundary conditions. However, when an object is unconstrained, K is singular, and thus Eq. (2) is unusable. We notice that a deformable object's motion is a superposition of its rigid-body motion $\boldsymbol{u}^r \in \mathsf{span}(\mathsf{U}^r)$ and a pure deformation $\boldsymbol{u}^d$. The reduced modes approximating $\boldsymbol{u}^d$ should therefore be orthogonal to U^r. This orthogonality leads to a constrained linear system for computing unconstrained inertia modes (Figure 3),

$$\begin{bmatrix} \mathsf{K} & \mathsf{U}^r \\ \mathsf{U}^{r\top} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathsf{U}^{(m)} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathsf{M}\mathsf{U}^{(m-1)} \\ \mathbf{0} \end{bmatrix}, \tag{3}$$

where $\boldsymbol{\lambda}$ is a Lagrange multiplier. We note that this formula of constructing unconstrained modes is new.

Numerical Challenges. In essence, constructing the Krylov subspace in Eq. (2) amounts to partially performing a linear modal analysis, which solves an eigen problem, $\mathsf{KU} = \mathsf{MUS}$ (or equivalently $\mathsf{AU} = \mathsf{US}^{-1}$). Indeed, $\mathcal{K}_{(m)}$ is a good approximation of the subspace spanned by leading eigenvectors [Saad 1981] and has been widely used in classic eigen solvers such as Arnoldi and Lanczos methods [Saad 2011]. However, Krylov iterations undermine the linear independence among modal vectors: after a few iterations, the mode matrix quickly becomes ill-conditioned. A common recipe to address this problem applies regularization such as a mass Modified Gram-Schmidt (mass-MGS) process [Golub and van Van Loan 1996] after each iteration.

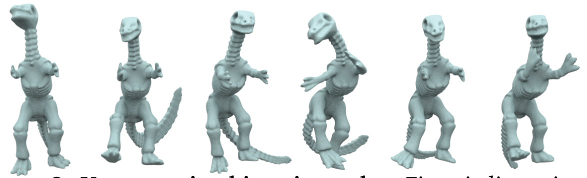

Figure 3: Unconstrained inertia modes. *First six linear inertia modes of an unconstrained model (see the supplemental video).*

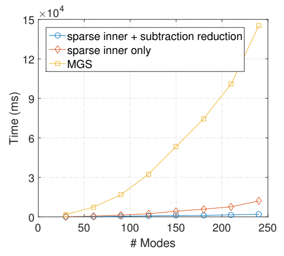

Figure 4: Performance gain *of rMGS. Time used in regular MGS, rMGS with only sparse inner product and rMGS with both sparse inner product and subtraction reduction (α_τ = 0.001) on the dinosaur example. Compared to the standard MGS, our rMGS is* ∼ **75**× *faster.*

The standard mass-MGS utilizes a mass inner product between two modes $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ (i.e., $\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathsf{M}} \triangleq \boldsymbol{u}_i^\top \mathsf{M} \boldsymbol{u}_j$), to prune the i-th mode $\boldsymbol{u}_i$ using previously regularized ones $\boldsymbol{u}_j$, $j < i$:

$$\boldsymbol{u}_i \leftarrow \boldsymbol{u}_i - \sum_{j=1}^{i-1} \frac{\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathsf{M}}}{\langle \boldsymbol{u}_j, \boldsymbol{u}_j \rangle_{\mathsf{M}}} \boldsymbol{u}_j. \tag{4}$$

This regularization is applied to every single mode $\boldsymbol{u}_i$, $i = 1..n$, resulting in a time complexity of $O(nr^2)$. For a high-resolution mesh with a moderate number of reduced modes, this greatly increases the precomputation cost.

4.2 Reduced Mass-MGS

We propose a *reduced* mass-MGS (rMGS) to regularize modal vectors. We first accelerate the mass inner product, which effectively reduces the $O(n)$ factor in the $O(nr^2)$ complexity. We then reduce the cost of repeated subtraction in Eq. (4), lowering the $O(r^2)$ factor. Figure 4 illustrates the efficacy of our method.

Sparse Inner Product. The mass inner product of two modes $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ is in the continuous setting a numerical discretization of a volume integral,

$$\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathsf{M}} \approx \int_{\Omega} \rho(\boldsymbol{x}) \boldsymbol{u}_i(\boldsymbol{x}) \boldsymbol{u}_j(\boldsymbol{x}) \mathrm{d}V, \tag{5}$$

where $\boldsymbol{u}_i(\boldsymbol{x})$ is the continuous displacement counterpart of the displacement vector $\boldsymbol{u}_i$. This integral can be numerically evaluated using the Newton-Cotes rule [Atkinson 1989], which sums up the integrand values at a set of sample points $\mathcal{S}$ over Ω,

$$\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathsf{M}} \approx \langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathcal{S}} \triangleq \sum_{\boldsymbol{p} \in \mathcal{S}} w_p [\boldsymbol{u}_i(\boldsymbol{p}) \cdot \boldsymbol{u}_j(\boldsymbol{p})], \tag{6}$$

where $\boldsymbol{p}$ indicates a sample point; w_p is its nonnegative weight for numerical integration. In the rest of this section, we use $\langle \cdot, \cdot \rangle_{\mathcal{S}}$ to denote this sparse inner product.

The Newton-Cotes rule requires sample points be evenly placed over the object volume Ω. We therefore create an axis-aligned bounding box of the mesh and subdivide it along three axes into cubic boxes. If a box B intersects with the input mesh, we add the finite element node nearest to the center of B into $\mathcal{S}$ and compute its weight as the total mass inside B, $w_p = \int_{B \cap \Omega} \rho(\boldsymbol{x}) \mathrm{d}V$. Appendix A provides an error analysis of this scheme. In our implementation, we find that setting $|\mathcal{S}| \propto \log(n)$ provides a good balance between efficiency and accuracy (Figure 5).

Subtraction Reduction. Next, we reduce the cost of $O(r^2)$ subtraction in the mass-MGS. Our key observation is that among all pairs of linear inertia modes, a considerable portion is already almost orthogonal even without applying mass-MGS. In other words, $\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathsf{M}}$ is small for many pairs of $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ (see statistics in Figure 7).

This observation inspires us to cull unnecessary subtractions in (4) using a *sparse cosine* metric,

$$\alpha = \frac{\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathcal{S}}}{\sqrt{\langle \boldsymbol{u}_i, \boldsymbol{u}_i \rangle_{\mathcal{S}} \langle \boldsymbol{u}_j, \boldsymbol{u}_j \rangle_{\mathcal{S}}}},$$

which approximates the cosine value of the angle between $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ using the sparse inner product (6). This metric is fast to compute owing to the sparse inner product, and is normalized by the (approximated) lengths of $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ so that it is insensitive to the length of reduced displacement vector. If $|\alpha|$ is smaller than a threshold α_τ, the corresponding subtraction in Eq. (4) is skipped. We present an pseudo-code of this subtraction reduction in Appendix B, where some critical implementation details are also highlighted. Meanwhile, a visualization of the mass-orthogonality of the resulting mode matrix processed with our method is shown in Figure 6.

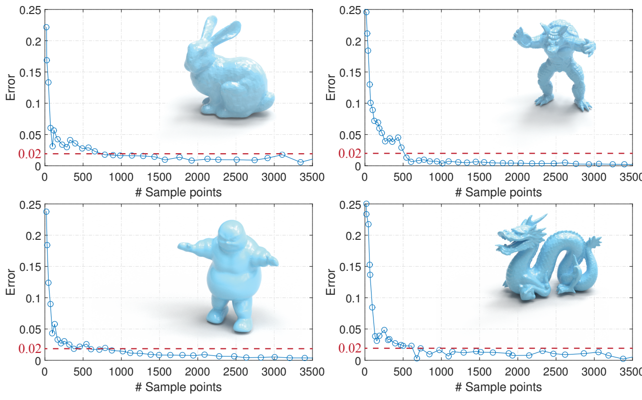

Figure 5: Sparse inner product. *The approximation error of sparse inner products w.r.t. the full-size mass inner products, i.e.,* $\left| \frac{\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathcal{S}} - \langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathsf{M}}}{\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{\mathsf{M}}} \right|$. $\boldsymbol{u}_i$ *and* $\boldsymbol{u}_j$ *are from 240 linear inertia modes, and the error is the average of all 57,360 sparse inner products.*

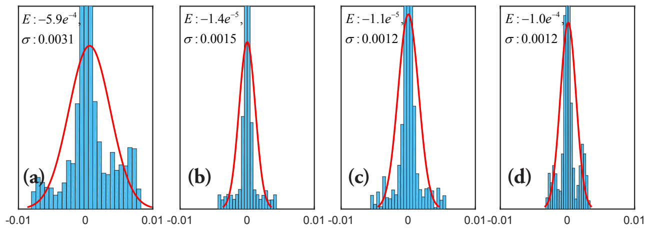

Figure 7: Statistics of modal pair orthogonality. *The histogram statistics and the fitted normal distributions (red curve) for 28,680 direction cosines computed using all pairs of 240 inertia modes. (a) Bunny, (b) Armadillo, (c) Stay-Puft and (d) Dragon.*

In sum, the speedup offered by our modal construction algorithm is twofold: first, the Krylov iteration is just part of the computation in standard eigen solvers such as Lanczos methods, wherein, in addition to constructing the Krylov subspace, extra computation of Ritz vectors/values are needed to approximate eigen-vectors/values at each iteration. Since the Ritz vector computation only orthogonalizes basis vectors but not change the spanned subspace, it is not needed for our purpose of constructing subspace basis. Second, our algorithm further accelerates the Krylov iterations by reducing the mass-MGS cost, which is a major performance bottleneck of this step.

4.3 Nonlinear Inertia Derivatives

Linear modes are insufficient to capture nonlinear deformations. To address this limitation, Barbič and James [2005] introduced a method of computing *modal derivatives* to expand the subspace. While their method is based on linear modal analysis, we show that our Krylov-type modes can also be extended to capture nonlinear deformations (§4.3.1). We call those nonlinear modes the *inertia derivatives*, for which we derive an asymptotic formula up to an arbitrary order. Additionally, as a faster alternative to the classic PCA for condensing the modal basis, we adopt the random projection method from the field of data mining (§4.3.2).

4.3.1 Generation of Nonlinear Inertia Derivatives

When nonlinear deformations are considered, the stiffness matrix K depends on the current displacement $\boldsymbol{u}$. Consider a small perturbation of $\mathsf{K} = \mathsf{K}(\boldsymbol{0})$, that is, $\mathsf{K}(\Delta \boldsymbol{u}) = \mathsf{K} + \Delta\mathsf{K}$. We expand the inverse of $\mathsf{K}(\Delta \boldsymbol{u})$ using a Taylor series [Miller 1981]:

$$\mathsf{K}(\Delta \boldsymbol{u})^{-1} = (\mathsf{K} + \Delta\mathsf{K})^{-1} = \mathsf{K}^{-1} - \mathsf{K}^{-1}\Delta\mathsf{K}\mathsf{K}^{-1} + O(\| \Delta \boldsymbol{u} \|^2). \tag{7}$$

Applying Eq. (7) to the computation of inertia modes yields an asymptotic expansion of nonlinear modes:

$$\mathsf{K}^{-1}\mathsf{M}\mathsf{U}^{(m-1)} - \mathsf{K}^{-1}\Delta\mathsf{K}\mathsf{K}^{-1}\mathsf{M}\mathsf{U}^{(m-1)} + \cdots = \mathsf{U}^{(m)} - \mathsf{K}^{-1}\Delta\mathsf{K}\mathsf{U}^{(m)} + \cdots.$$

To compute the first-order nonlinear modes, we approximate $\Delta\mathsf{K}$ using the directional derivative of K along a direction $\boldsymbol{u}$,

$$\mathsf{K}^{-1}\Delta\mathsf{K}\mathsf{U}^{(m)} = \mathsf{K}^{-1}(\mathsf{H} : \boldsymbol{u})\mathsf{U}^{(m)},$$

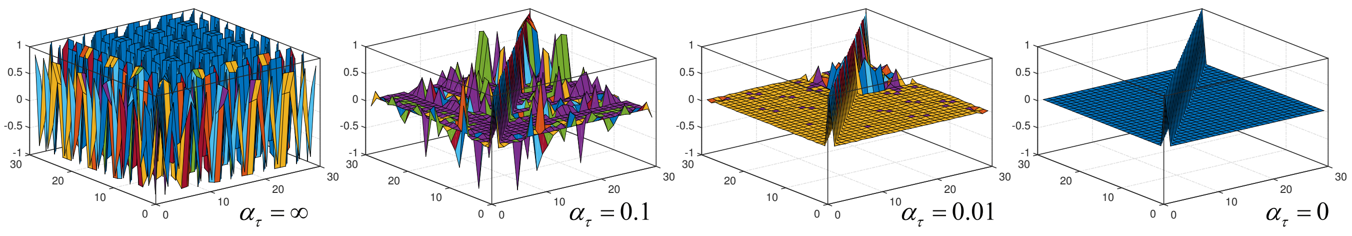

Figure 6: Mass-orthogonality using rMGS. *We plot $M_q = U^\top MU$ as a height field for the Stay-Puft model with 30 modes, when different culling thresholds α_τ are used. The rightmost plot shows a perfect mass-orthogonal mode matrix, wherein M_q is an identity matrix; the leftmost plot shows M_q calculated with an un-regularized mode matrix ($\alpha_\tau = \infty$). We use $\alpha_\tau = 0.001$ in all of our examples.*

where $H = \nabla K$ is the stiffness Hessian, a third-order tensor.

When $\boldsymbol{u}$ is chosen as individual linear modes, we obtain the formula of computing the first-order nonlinear inertia derivatives,

$$\boldsymbol{v}_{ij}^{(1)} = K^{-1}(H : \boldsymbol{u}_i)\boldsymbol{u}_j, \tag{8}$$

where both $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ are linear inertia modes, and the superscript of $\boldsymbol{v}_{ij}^{(1)}$ indicates a first-order nonlinear mode. We note that Eq. (8) is identical to the modal derivative formula [Barbič and James 2005] suggesting that our Krylov-subspace-based modes are indeed a good approximation of the standard eigen modes.

More importantly, such derivation facilitates the computation of arbitrarily high-order nonlinear modes. Higher-order expansion of Eq. (7) shows that $K(\Delta\boldsymbol{u})^{-1} = K^{-1} - K^{-1}\Delta KK^{-1} + K^{-1}\Delta KK^{-1}\Delta KK^{-1} + \cdots$, from which we can compute the k-th order inertia derivatives recursively,

$$V^{(k)} = K^{-1}(H : \boldsymbol{u})V^{(k-1)}, \tag{9}$$

With high-order nonlinear inertia derivatives, one can carefully choose the initial Krylov vectors (i.e., $U^{(0)}$) to construct a fine-tuned yet compact subspace, and nonlinear deformations even at some extreme cases can be captured (Figure 8).

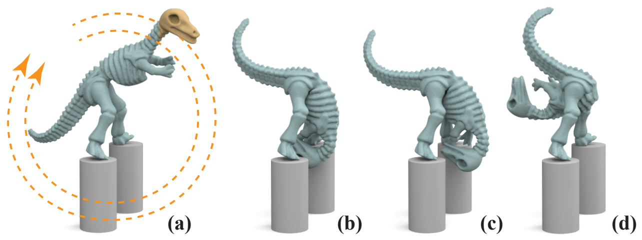

Figure 8: Bending comparison. *The highlighted head part of the dinosaur undergoes a circular external forces. (a) 2 linear inertia modes are used, where $U^{(0)}$ contains only vertical and horizontal translations. (b) 2 linear inertia modes + 3 first-order inertia derivatives. (c) 20 first-order modal derivatives (generated from 8 linear modal modes and 36 first-order modal derivatives with mass-PCA). (d) 20 high-order inertia modes (2 linear inertia modes + 3 first-order inertia derivatives + 15 second-order inertia derivatives).*

4.3.2 Random Projection

As one chooses to use increasingly higher-order modes, the total number of modes increases exponentially (i.e., the column size of $V^{(k)}$ increases exponentially with respect to k). Thus, the reduced simulation slows down. Most existing work [Barbič and James 2005; Kim and James 2011; von Tycowicz et al. 2013; Harmon and Zorin 2013] use PCA to select the most prominent modes out of the constructed ones. PCA has a time complexity of $O(\min(r^6, n^3))$ [Johnstone and Lu 2009][1]. In traditional modal-analysis-based precomputation, this is hardly a bottleneck, as the eigen-decomposition step is usually more expensive. However, our subspace construction method has no use of eigen-decomposition, leaving PCA indeed a performance bottleneck.

As a faster alternative of PCA, we propose to use *Random Projection* (RP), a method that has been used in data mining [Bingham and Mannila 2001; Halko et al. 2011]. This method is based on a key observation by Hecht-Nielsen [1994]: in a high-dimensional vector space, it is very likely that two random vectors are almost orthogonal to each other. This suggests that we can simply use a random thin matrix to condense the constructed modes.

Concretely, we first follow the same strategy used in existing literatures [Barbič and James 2005; von Tycowicz et al. 2013] to weight the modes according to their generalized Rayleigh quotients [Parlett 1987]: for every constructed mode $\boldsymbol{u}_i$, $i = 1...m$, we normalize it using $\boldsymbol{u}_i \leftarrow \boldsymbol{u}_i(\boldsymbol{u}_i^\top K\boldsymbol{u}_i)/(\boldsymbol{u}_i^\top M\boldsymbol{u}_i)$. We then concatenate all the $\boldsymbol{u}_i$ into a superset matrix $\widetilde{U}$, and compute the final modal matrix U for runtime simulation using $U = \widetilde{U}R$, where R is a $m \times r$ matrix to condense the number of modes from m to r. The entry of R is randomly generated as in [Achlioptas 2001]:

$$R_{i,j} = \sqrt{3} \cdot \begin{cases} +1 & \text{with probability of } \frac{1}{6} \\ 0 & \text{with probability of } \frac{2}{3} \\ -1 & \text{with probability of } \frac{1}{6} \end{cases}. \tag{10}$$

As noted in [Bingham and Mannila 2001], when $m \gg r$, the column vectors of R are always nearly orthogonal to each other. Thus, $\widetilde{U}R$ can be considered as a *pseudo projection* of $\text{span}(\widetilde{U})$ to a smaller space, $\text{span}(R)$. Since R is sparse, this matrix multiplication is much faster than running PCA. However, unlike PCA, random projection can not guarantee to choose the most salient subspace indicated by eigenvalues nor yield an optimal subspace in that sense.

5 Fast Precomputation of Modal Cubature

For nonlinear deformable simulation, model reduction needs to compute at runtime the internal elastic force and its Jacobian. To improve the computation efficiency, An et al. [2008] introduced a fast method based on the numerical cubature scheme. This method selects in the precomputation step a set of cubature elements $\mathcal{E}$ on the mesh, and computes a weight w_e for each element

[1]Here, we assume that first-order nonlinear inertia derivatives are used. Therefore, the number of constructed modes is $O(r^2)$.

$e \in \mathcal{E}$. At runtime, the reduced internal force $\boldsymbol{f}(\boldsymbol{q})$ and its Jacobian $\partial \boldsymbol{f}/\partial \boldsymbol{q}$ are respectively computed as weighted summations over all the cubature elements:

$$\boldsymbol{f}(\boldsymbol{q}) \approx \sum_{e\in\mathcal{E}} w_e \boldsymbol{g}_e(\boldsymbol{q}) \text{ and } \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} \approx \sum_{e\in\mathcal{E}} w_e \frac{\partial \boldsymbol{g}_e(\boldsymbol{q})}{\partial \boldsymbol{q}},$$

where $\boldsymbol{g}_e(\boldsymbol{q})$ is the reduced internal force induced by a modal displacement $\boldsymbol{q}$ at a cubature element e. While this scheme is fast at runtime, the precomputation remains expensive: it iteratively adds cubature elements and updates their weights; each iteration solves an expensive nonnegative least-squares (NNLS) problem.

Rationale. We seek to also accelerate the cubature precomputation. One straightforward attempt is again to exploit the Newton-Cotes rule: using evenly spaced cubature points to avoid repeated NNLS solves. However, to maintain sufficient approximation accuracy using the Newton-Cotes rule, there need to be densely sampled cubature points, which in turn burden the NNLS in precomputation and the runtime evaluation. We propose a simple solution to address this dilemma: instead of using a single weight at each cubature point, we prepare multiple weights, each for an individual reduced coordinate. Our experiments (see Figure 15 and the video) show that such a simple extension largely accelerates cubature training while retaining a comparable accuracy as the standard training scheme [An et al. 2008]. In addition, this extension also lays out an important foundation to enable a faster computation of training data, which will be elaborated in the next section.

Method. Concretely, for every component f^j of the reduced internal force $\boldsymbol{f}$, we precompute a set of cubature weights w_e^j, and approximate the reduced internal force as

$$\boldsymbol{f}(\boldsymbol{q}) \approx \sum_{j=1}^{r} \left(\sum_{e\in\mathcal{E}} w_e^j g_e^j(\boldsymbol{q}) \right) \boldsymbol{e}^j,$$

where $\boldsymbol{e}^j$ is the canonical unit basis vector of $\mathbb{R}^r$ (e.g., $\boldsymbol{e}^1 = [1, 0, \cdots]^\top$); g_e^j is the j-th component of $\boldsymbol{g}_e$, the internal force at a cubature element e. We stack w_e^j for all $e \in \mathcal{E}$ into a vector $\boldsymbol{w}^j$ and precompute it by solving a NNLS problem, $\mathsf{A}^j \boldsymbol{w}^j = \boldsymbol{b}^j$, where A^j and $\boldsymbol{b}^j$ are constructed based on a training set $\mathcal{T}$ with T samples. Specifically,

$$\mathsf{A}^j = \begin{bmatrix} \frac{g_1^{j,1}}{\|\boldsymbol{f}^1\|} & \cdots & \frac{g_{|\mathcal{E}|}^{j,1}}{\|\boldsymbol{f}^1\|} \\ \vdots & \ddots & \vdots \\ \frac{g_1^{j,T}}{\|\boldsymbol{f}^T\|} & \cdots & \frac{g_{|\mathcal{E}|}^{j,T}}{\|\boldsymbol{f}^T\|} \end{bmatrix}, \quad \boldsymbol{b}^j = \begin{bmatrix} \frac{f^{j,1}}{\|\boldsymbol{f}^1\|} \\ \vdots \\ \frac{f^{j,T}}{\|\boldsymbol{f}^T\|} \end{bmatrix},$$

where $f^{j,i}$ denotes the j-th component of a reduced internal force $\boldsymbol{f}^i$ in the i-th training example.

In the rest of this paper, to distinguish from the standard cubature training, to which we refer as Optimized Cubature (OC), we refer to our cubature scheme as Modal Cubature (MC). Using MC, the reduced internal force Jacobian can be written as

$$\mathsf{K}_q = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} \approx \sum_{j=1}^{r} \sum_{e\in\mathcal{E}} w_e^j \left(\boldsymbol{e}^j \otimes \frac{\partial g_e^j}{\partial \boldsymbol{q}} \right). \tag{11}$$

We also sample the positions of cubature points evenly using axis-aligned voxels, as used in the sparse inner product step (§4.2). While we need to solve a NNLS problem for every single component of the reduced coordinate, the size of each NNLS problem

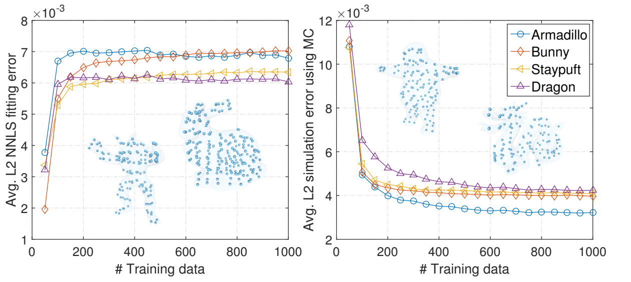

Figure 9: Observations in cubature training. *Left: the NNLS fitting error during cubature weights calculation. Right: the MC simulation error w.r.t 50K full-space random simulations. Four models (Armadillo, bunny, Stay-Puft and dragon) are tested under the same cubature sampling resolution.*

is much smaller, and all the solves can be performed in parallel, yielding 1000-5000× speedups (see the video and detailed benchmarks in Table 3).

Extension. It is noteworthy that the Jacobian matrix K_q resulted from Eq. (11) is not necessarily symmetric. One can approximately symmetrize it as $\mathsf{K}_q \leftarrow \frac{1}{2}(\mathsf{K}_q + \mathsf{K}_q^\top)$. On the other hand, we present a new Newton-PCG solver, which requires no runtime evaluation of the Jacobian matrix, and thus completely sidesteps the asymmetry problem. On the down side, the proposed solver requires additional implementation efforts and a change of the runtime integrator. We describe this runtime solver in Appendix C, as an extension of our proposed pipeline.

6 Incremental Generation of Training Data

Most reduced simulation methods compute the cubature weight from a training data $\mathcal{T}$, which is often taken for granted. When there does not exist a prior training set, one reasonable strategy is to blindly sample a deformation subspace. For instance, von Tycowicz et al. [2013] randomly sampled reduced displacements following a normal distribution. The sampled displacements are used in a full-space simulation to evaluate corresponding internal forces and assemble the training dataset. Since our goal is to expedite the entire precomputation pipeline, we wish to carefully generate the training samples to avoid full-space simulation as much as possible. To the best of our knowledge, this problem has been largely unexplored.

Our modal cubature scheme (§5) allows us to reduce the cost of training sample generation by incrementally expanding the training dataset. This differs from the traditional optimized cubature scheme [An et al. 2008; von Tycowicz et al. 2013], wherein the training data is given, and their goal is to find the best set of cubature points. In contrast, with a fixed set of cubature points, we seek for a compact training dataset.

6.1 Observations and Rationale

Our algorithm of training data generation is inspired by two key observations:

First, it is known that the accuracy of approximating a force integral using the Newton-Cotes rules is bounded by the sampling interval (see Appendix A). This implies that the increase of training samples has a diminishing return on the accuracy improvement as shown in Figure 9 (right), where we incrementally add randomly generated training samples and evaluate the Modal

Algorithm 1 Incremental training data generation.

```
1: $\Delta e \leftarrow \infty$
2: $N_I \leftarrow \frac{3}{2}|\mathcal{E}|$;                 ▷ the size of the initial training set
3: for each mode vector $\mathbf{u}_i$, $i = 1 \dots r$ do
4:     $Q_i \leftarrow \frac{\boldsymbol{u}_i^\top \mathsf{K} \boldsymbol{u}_i}{\boldsymbol{u}_i^\top \mathsf{M} \boldsymbol{u}_i}$; $E_i \leftarrow 0$; $\sigma_i \leftarrow \frac{1}{\sqrt{Q_i}}$
5: end for
6: while $\Delta e > e_\tau$ do
7:     for $t = 1 : N_I$ do
8:         generate $q_i^t$ following $\mathcal{N}(E_i, \sigma_i)$ for all $i = 1 \dots r$
9:         compute internal force $\boldsymbol{f}(\boldsymbol{q})$
10:        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\boldsymbol{q}, \boldsymbol{f}\}$
11:    end for
12:    NNLS fitting over $\mathcal{T}$
13:    update $\Delta e$
14:    update $\mathcal{T}_e$          ▷ stores poses with top 20% fitting error
15:    $\mathbf{q} \leftarrow$avg$(\mathcal{T}_e)$                ▷ recenter samples
16:    for $i = 1 : r$ do
17:        $E_i \leftarrow q_i$; $\sigma_i \leftarrow \begin{cases} \frac{1}{\sqrt{Q_i - E_i}}, & E_i \geq 0 \\ \frac{1}{\sqrt{Q_i + E_i}}, & E_i < 0 \end{cases}$
18:    end for
19:    $N_I \leftarrow \frac{1}{3}|\mathcal{E}|$
20: end while
```

Cubature simulation error[2]. Initially, when the training data is insufficient, the error is large. However, as we expand the training dataset, the error is eventually bounded from below.

Our second observation looks at the change of the normalized NNLS fitting error (Figure 9 (left)). The error is computed as

$$e = \frac{1}{T} \sum_{j=1}^{r} \frac{||\mathsf{A}^j \boldsymbol{w}^j - \boldsymbol{b}^j||}{||\boldsymbol{b}^j||}. \tag{12}$$

Initially, when T is small, the NNLS problem is under-constrained, and thus the fitting error is low. As more samples are added in the cubature training, the fitting error grows, but eventually becomes bounded from above. One interpretation of this observation is from the Monte Carlo integration point of view [Baraff and Witkin 1992]. The error defined in Eq. (12) computes the averaged fitting error across all training samples. As the number of samples increases, it is equivalent to evaluating, using a Monte Carlo process, the *expected* fitting error in the deformation subspace. Since the number of cubature points is fixed, the expected fitting error is bounded.

6.2 Incremental Training Samples

The above interpretation from a Monte Carlo point of view suggests that the error metric defined in Eq. (12) can be a natural indicator for when to stop adding training samples. One simple algorithm is as follows: we incrementally add samples into the training dataset. Every time when generating a training sample, we follow the way in [von Tycowicz et al. 2013] to sample a reduced modal displacement whose component q_i follows a Gaussian distribution of $q_i \sim \mathcal{N}(0, \sigma_i)$, where $\sigma_i = 1/\sqrt{Q_i}$, and Q_i is the generalized Rayleigh quotient of the modal vector $\boldsymbol{u}_i$. Here Q_i estimates the effective eigenvalue of $\boldsymbol{u}_i$, so the low-frequency mode will produce samples with larger variance. Corresponding to the reduced displacement sample, a full-space displacement vector is used in a full-space simulation to compute the internal force, which, together with the reduced displacement sample, forms a training sample. After adding a few samples, we update the fitting error Eq. (12), and evaluate its change Δe. We stop generating new training samples, if Δe is smaller than a threshold e_τ ($e_\tau = 0.005$ in all of our examples).

[2]The MC simulation error is calculated as the average deviation from 50K random full-space simulation.

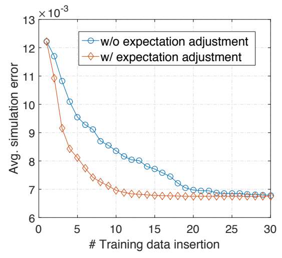

Figure 10: Effectiveness of expectation adjustment. *We plot the MC simulation error with respect to 50K randomly generated full-space results using the incremental training. The Armadillo model is tested with $N_I = 35$. The error converges faster with the expectation adjustment (i.e., Algorithm 1).*

We further improve our sampling algorithm by adding more poses in under-sampled regions in the reduced deformation space. As detailed in Algorithm 1, we start by generating $\frac{3}{2}|\mathcal{E}|$ samples using the Gaussian distribution described above (line 7-11 of Algorithm 1), so the resulting NNLS problem is over-constrained. We then iteratively add more samples. In each iteration, we add $\frac{1}{3}|\mathcal{E}|$ samples. At the end of each iteration, we adjust E_i and σ_i for subsequent samples (line 14-18 of Algorithm 1) based on the training poses that have large fitting errors. To this end, we maintain a set of training poses $\mathcal{T}_e$ that have the top 20% fitting error among currently assembled training poses. For the next sample generation, we set E_i as the averaged modal pose of $\mathcal{T}_e$, and adjust σ_i such that $\mathcal{T}_e$ are in the radius of the Gaussian distribution (line 17 of Algorithm 1). The iteration process stops if the error change Δe is below a threshold. The efficacy of this improved algorithm is shown in Figure 10.

In every iteration, we generate $\frac{1}{3}|\mathcal{E}|$ training samples in parallel. With our incremental training generation, we are able to stop computing training samples when they become unnecessary, so the training process is accelerated. Even for large-scale models, the entire precomputation, including the generation of training data, can be completed within tens of seconds. Of course, if $\mathcal{T}$ is given *a priori* in certain cases, (or not required, for instance one may choose to use geometrical warping to produce nonlinear deformation [Choi and Ko 2005; Huang et al. 2011] as in the example shown in Figure 18), the precomputation can be even faster (finish within seconds/milliseconds).

7 Validation and Applications

We have tested the proposed precomputation pipeline and evaluated its performance, scalability, quality, and versatility. We implement our method on a machine with an Intel i7-5960 3.0GHz 8-core CPU and 32GB RAM. We use `pThread` to parallelize the computation for modal cubature training and training data generation. Most proposed numerical algorithms (e.g., sparse inner product, pMGS, nonnegative least square and Jacobian-free Newton-PCG solver) are implemented with the help of the Intel `MKL` libraries. We also refer the reader to the accompanying executable (`.exe` file for 64-bit Windows systems). Table 1 reports detailed statistics of the 3D models we tested and their precomputation time benchmarks.

7.1 Validation

Random Projection. As a validation of our random projection scheme introduced in §4.3.2, we plot in Figure 11 the average

Model	#Ele.	#DoF	$\lvert\mathcal{T}\rvert$	Time
Bunny	50,000	32,103	217	**4.2s**
Armadillo	65,496	52,443	225	**4.8s**
Stay-Puft	77,337	68,427	228	**6.6s**
Dragon	100,000	79,308	198	**7.2s**
Dinosaur	193,700	491,370	150	**9.7s**

Table 1: Precomputation benchmarks. *The size of the subspace is 30 (i.e., $r = 30$), which are constructed from 12 linear modes and 78 first-order inertia derivatives. #Ele.: number of elements; #DoF: number DoFs; $\lvert\mathcal{T}\rvert$: the size of training set; Time: total precomputation time including the generation of training poses.*

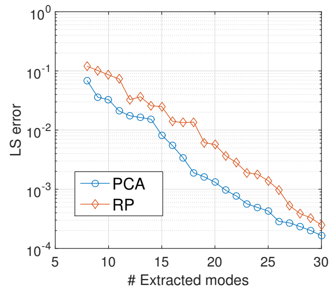

Figure 11: Quality comparison between PCA and RP. *We plot the average least-squares error of how well the extracted mode matrix* U *can express the original 324 candidates (24 linear + 300 nonlinear modes). Note that the least-squares error axis is logarithmic.*

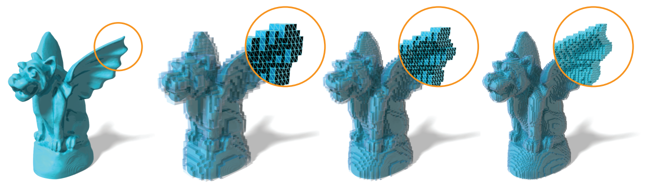

Figure 12: Multi-resolution meshing by voxelization. *The gargoyle model is voxelized at different resolutions to generate tetrahedral meshes from 50K to 1M elements.*

least-squares error when using U, extracted with either PCA or RP, to fit the entire mode pool $\tilde{\mathrm{U}}$. When r is a very small number, U computed with RP yields a higher error due to its randomness in picking low-dimensional bases. Fortunately, their difference becomes quite subtle when the subspace size is moderately big (e.g., $r = 30$). This result also matches the performance analysis of RP in existing literatures [Dasgupta 2000; Bingham and Mannila 2001; Halko et al. 2011].

Scalability Test. We test the scalability of our modal construction algorithm using the gargoyle model. To ease the control of number of elements, we voxelize the model, so we can generate tetrahedral meshes whose sizes range from 50K to 1M (Figure 12). The computation time for generating linear inertia modes (our method) and linear eigen modes (previous method) are reported in Table 2. The linear inertia modes are computed using the `PARDISO` solver shipped with `MKL`. The modal analysis-based modes are computed using `Matlab` 2015a's built-in `eigs` function with multi-threading enabled[3]. This test shows that with the help of rMGS, the construction of inertia modes is 20-30× faster than linear modal analysis on average. Under such circumstance, mode refinement with PCA becomes a performance bottleneck. As plotted in Figure 13, the computation time for PCA quickly scales up with increased mesh resolution, while random projection is much faster and more scalable.

[3]`eigs` calls `MKL` and `ARPACK++` at background, which gives a good performance on our hardware platform (Intel CPU).

# Ele.	$r = 30$		$r = 60$		$r = 120$		$r = 240$	
50K	9.8s	**0.7s**	14.4s	**1.0s**	28.7s	**2.1s**	68.7s	**4.2s**
100K	12.4s	**1.2s**	20.1s	**2.0s**	50.9s	**3.7s**	132.8s	**3.7s**
200K	41.2s	**2.8s**	59.7s	**4.9s**	131.7s	**9.2s**	290.7s	**19.1s**
400K	96.2s	**5.8s**	140.6s	**10.2s**	284.8s	**17.7s**	579.3s	**33.6s**
600K	164.6s	**8.6s**	249.0s	**15.5s**	575.0s	**29.7s**	1496.8s	**65.5s**
800K	235.7s	**11.8s**	341.1s	**18.3s**	883.2s	**30.0s**	1894.3s	**56.9s**
1M	260.0s	**16.2s**	568.1s	**27.3s**	1042.2s	**44.9s**	2238.8s	**64.9s**

Table 2: Scalability tests. *Performance benchmark of modal construction using linear modal analysis (left columns) and rMGS augmented Krylov iteration (right columns, bold).*

Simulation Quality. Figure 14 gives a quantitative comparison between two deformable animations produced with standard modal derivatives and our method. The time series of magnitudes of the displacement at the dragon's horn is also plotted. Here we use St. Venant-Kirchhoff material. This figure (and also the video) shows that while a set of approximation and acceleration methods have been employed, the proposed precomputation pipeline produces results that are visually comparable to the standard modal analysis/derivative-based precomputation approach. Yet, our approach enjoys a much faster precomputation time.

Modal Cubature. We compare the performance of the modal cubature with the standard optimized cubature on a training set $\mathcal{T}$, $T = 1000$. Here we use the lazy cubature optimization strategy [An et al. 2008], wherein the new seed element is picked from a subset of 1,000 elements. Both cubature schemes are tested on the Armadillo model. We choose multiple NNLS fitting error levels. For each error level, we examine the number of cubature elements needed to reach the error level, and compare the cubature training costs of both schemes. Table 3 lists the benchmarks. For an error of about 1%, MC is about **1500**× faster than OC. An improved greedy cubature training strategy was also reported in [von Tycowicz et al. 2013], which is roughly 4× faster than the standard OC scheme, but is still significantly slower than our method. Figure 15 (a) presents snapshots of the animation produced respectively with MC and OC.

Training Pose Generation. We test our incremental training pose generation using the Armadillo with an St. Venant-Kirchhoff material (Figure 15). Our tests show that we only need a few hundred training poses to produce deformable animations of similar quality to the ones generated using a larger training set (Figure 15 (b) and also the video). Because of the per-mode weight storage in MC training, MC consumes r times more memory than OC does. One drawback of MC is associated with the asymmetry of the resulting Jacobian matrix, which can result in a larger numerical error than OC when used to compute the force gradient. Fortunately, the proposed Jacobian-free PCG solver in Appendix C addresses this issue. In all our examples, our inner PCG solver is able to converge within 3 to 5 iterations (when setting the convergence threshold to be 0.001).

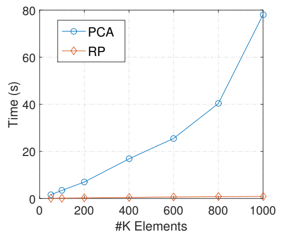

Figure 13: Scalable test of PCA and RP. *The same gargoyle models as in Figure 12 are tested in this example. We record the time used to construct a 30-dimension modal matrix out of 189 mode candidates (18 linear inertia modes + 171 nonlinear inertia derivatives) using both PCA and RP.*

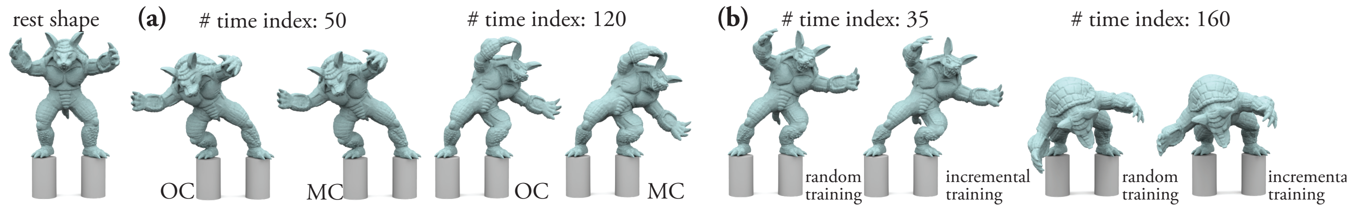

Figure 15: Comparative animations with different training strategies. *(a) Animation snapshots using OC training and MC training with ($T = 1000$), and (b) animation snapshots using incremental training ($T = 225$) and regular random training ($T = 5000$).*

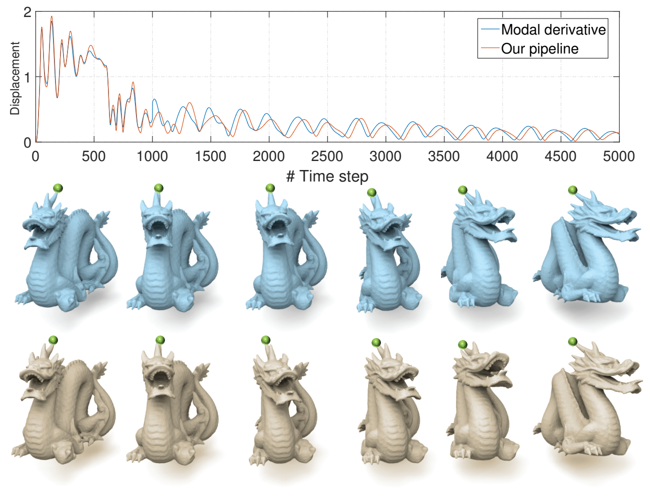

Figure 14: Quantitative comparison. *The first row of snapshots are from an animation produced using classic modal analysis/derivative. The second row are snapshots produced using our precomputation method. The magnitude of displacements at dragon's horn (highlighted as green spheres) is also plotted. Our method is 20× faster than traditional methods for modal construction.*

NNLS fitting err.	60%	25%	10%	3%	1%	0.5%	0.1%
#Cubature ele. (OC)	2	7	21	50	110	141	374
#Cubature ele. (MC)	**1**	**8**	**17**	**28**	**59**	**65**	**113**
Training time (OC)	9.2s	29.7s	74.5s	128.9s	206.0s	273.7s	2.2h
Training time (MC)	**0.02s**	**0.02s**	**0.03s**	**0.05s**	**0.13s**	**0.15s**	**0.36s**

Table 3: Comparative benchmarks between OC and MC. *We report the number of cubature elements and the computation time for given error levels using both lazy optimized cubature and modal cubature training strategies. The Armadillo is tested with 30 nonlinear inertia modes. 1000 training examples are used, and the listed timing information does not include the time for the training data generation.*

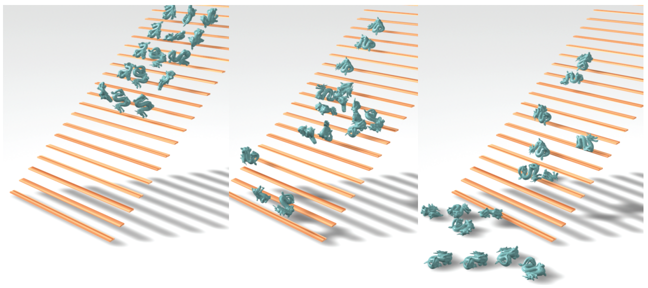

Figure 16: Free-floating deformable simulation. *15 dragon models fall onto the staircase. Each model has 18 nonlinear unconstrained modes. The precomputation time including the generation of training poses took 4.2 seconds.*

Free-moving deformable bodies can be well accommodated within our framework. Figure 16 shows the results of simulating 15 dragons falling. Each dragon model has 18 unconstrained nonlinear inertia modes. The reduced deformation is coupled with its rigid body motion, and the dynamics is integrated using the generalized Newton-Euler equation [Shabana 2005]. Because all the dragons have the same rest shape geometry, the precomputation takes only 4.2 seconds.

7.2 Applications

Application I: Multi-domain Simulation with High-order Nonlinear Modes. Our precomputation pipeline can work in tandem with different types of simulation methods and hyperplastic materials. Beside the typical single-domain solid deformable simulation, here we illustrate the application of our precomputation in a substructured simulation. Following the state-of-the-art domain decomposition methods [Barbič and Zhao 2011; Kim and James 2011; Yang et al. 2013], precomputation is localized in small domains. Figure 17 shows animation snapshots of a maple tree model, simulated with a tetrahedra mesh of 1102K elements grouped into 513 domains, and a Mooney-Rivlin material model. In this example, domains have different geometries from each other, which means the local precomputation at domains cannot be reused. The second order inertia modes corresponding to translational motions are computed at leaf domains. The precomputation of all the domains took less than a minute.

Application II: Simulation-in-the-loop Character Animation. The proposed precomputation pipeline allows a fast preview of physics-based animation on top of the classic skeleton-driven character animations. In this application, the user is able to tweak material parameters of the character. Owing to the proposed fast precomputation, we are able to update the reduced model and rerun the simulation quickly. Figure 18 is the snapshots of applying the deformable skinning to the animation of a walking Stay-Puft. The deformation of Stay-Puft is simulated using 30 unconstrained linear inertia modes with geometric warping [Choi and Ko 2005], triggered by the inertia forces from the rigid-body motion as formulated in [Kim and James 2011]. In this example, only the linear material model is used, so the training data and cubature optimization are not required, and the precomputation took only 0.8 second.

Application III: Nonlinear Sound Synthesis. Lastly, fast deformable precomputation also allows a quick forelook for vibrational sound synthesis. As shown in Figure 1, the finite element mesh of a metal wire model consists of 76,863 elements. We simulate its nonlinear surface vibrations using 120 nonlinear inertia modes and the St. Venant-Kirchhoff material model. Then the sound is synthesized using the method [O'Brien et al. 2001]. The

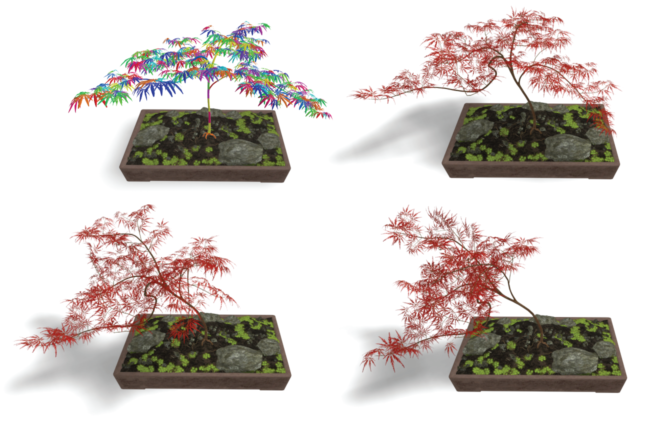

Figure 17: Domain decomposition with high-order nonlinear modes. *We show the simulation results of the maple tree model consisting of 1102K elements and 513 domains. Mooney-Rivlin material model is adopted, and second-order nonlinear inertia derivatives are used at the leaf domains where $r_{leaf} = 35$. The precomputation time is less than 1 min.*

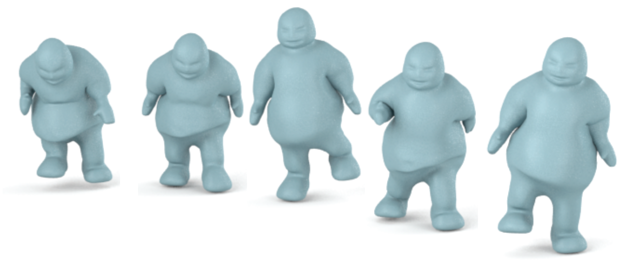

Figure 18: Simulation-in-the-loop character animation. *We apply physics-based character skinning using 30 unconstrained linear inertia modes with geometric warping. The resulting animation is almost instantly available with a 0.8 second precomputation.*

precomputation time took 48 seconds with 680 training poses and 342 cubature elements. The fast precomputation enables us to quickly update the reduced model whenever the user changes its geometry or material and re-simulate the surface vibration for physics-based sound synthesis.

8 Conclusion and Limitation

In this paper, we present a complete system to accelerate the precomputation of nonlinear elastic deformable models. We optimize three performance bottlenecks in the standard precomputation pipeline: the mode construction, the cubature training, and the generation of the training poses. Eventually, the deformable precomputation is made orders-of-magnitude faster.

This work remains a few limitations, and there are many opportunities for future work. Our modal cubature scheme requires r times more memory than the standard cubature scheme. The resulting subspace basis has less elegant properties that the modal-analysis-based basis has: the basis vectors are not perfectly mass-orthogonal to each other; the random projection can not choose the most salient subspace that PCA is able to produce; and the asymmetry of the Jacobian matrix requires extra efforts in the runtime simulation. While the Jacibian-free Newton-PCG solver provides a fix, in the future we would like to produce a symmetric matrix in the first place. In addition, When a large number of modes are needed, the current bottleneck of the precomputation become the matrix-vector production in the Krylov iteration. We will explore to accelerate this operation.

It is also interesting to investigate the levels-of-detail simulation for large-scale complex scenes based on the fast construction of subspace modes, where the modal matrix can be dynamically adapted according to the moving camera. Finally, expediting the construction of other reduced models such as reduced fluid simulation is an interesting potential direction.

Acknowledgements

We thank anonymous reviewers for their feedback. This research was supported in part by the National Science Foundation (CRII-1464306, CAREER-1453101), National Science Foundation China (No. 61272392, No. 61322204), UNM RAC & OVPR research grants, and generous gifts from Intel and DJI. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funding agencies or others.

References

ACHLIOPTAS, D. 2001. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, 274–281.

AN, S. S., KIM, T., AND JAMES, D. L. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. Graph. 27*, 5 (Dec.), 165:1–165:10.

ATKINSON, K. 1989. *An Introduction to Numerical Analysis*, 2 edition ed. Wiley, New York, Jan.

BARAFF, D., AND WITKIN, A. 1992. Dynamic simulation of non-penetrating flexible bodies. *SIGGRAPH Comput. Graph. 26*, 2 (July), 303–308.

BARBIČ, J., AND JAMES, D. L. 2005. Real-time subspace integration for st. venant-kirchhoff deformable models. In *ACM Trans. Graph.*, vol. 24, ACM, 982–990.

BARBIČ, J., AND JAMES, D. L. 2010. Subspace self-collision culling. In *ACM Trans. Graph.*, vol. 29, ACM, 81.

BARBIČ, J., AND POPOVIĆ, J. 2008. Real-time control of physically based simulations using gentle forces. In *ACM Trans. Graph.*, vol. 27, ACM, 163.

BARBIČ, J., AND ZHAO, Y. 2011. Real-time large-deformation substructuring. In *ACM Trans. Graph.*, vol. 30, ACM, 91.

BARBIČ, J., SIN, F., AND GRINSPUN, E. 2012. Interactive editing of deformable simulations. *ACM Trans. Graph. 31*, 4, 70.

BARBIČ, J., DA SILVA, M., AND POPOVIĆ, J. 2009. Deformable object animation using reduced optimal control. *ACM Trans. Graph. 28*, 3, 53:1–53:9.

BERRUT, J.-P., AND TREFETHEN, L. N. 2004. Barycentric lagrange interpolation. *SIAM Review 46*, 3, 501–517.

BINGHAM, E., AND MANNILA, H. 2001. Random projection in dimensionality reduction: Applications to image and text data. KDD '01, 245–250.

BONET, D. J., AND WOOD, D. R. D. 2008. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press.

BROWN, P., AND SAAD, Y. 1990. Hybrid krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing 11*, 3, 450–481.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. SIGGRAPH '02, 586–593.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '02, 41–47.

CHAN, T., AND JACKSON, K. 1984. Nonlinearly preconditioned krylov subspace methods for discrete newton algorithms. *SIAM Journal on Scientific and Statistical Computing 5*, 3, 533–542.

CHOI, M. G., AND KO, H.-S. 2005. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics 11*, 1 (Jan.), 91–101.

CRAIG, R. R., AND HALE, A. L. 1988. Block-Krylov component synthesis method for structural model reduction. *Journal of Guidance, Control, and Dynamics 11*, 6, 562–570. 00055.

CRAIG, R. J. 2000. Coupling of substructures for dynamic analyses - An overview. In *41st Structures, Structural Dynamics, and Materials Conference and Exhibit*, Structures, Structural Dynamics, and Materials and Co-located Conferences. American Institute of Aeronautics and Astronautics, Apr.

DASGUPTA, S. 2000. Experiments with random projection. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, 143–151.

GARCÍA, F. G., PARADINAS, T., COLL, N., AND PATOW, G. 2013. *cages:: A multilevel, multi-cage-based system for mesh deformation. *ACM Trans. Graph. 32*, 3 (July), 24:1–24:13.

GOLUB, G. H., AND VAN VAN LOAN, C. F. 1996. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*. Johns Hopkins University Press.

GRINSPUN, E., KRYSL, P., AND SCHRÖDER, P. 2002. Charms: A simple framework for adaptive simulation. SIGGRAPH '02, 281–290.

HAHN, F., THOMASZEWSKI, B., COROS, S., SUMNER, R. W., COLE, F., MEYER, M., DEROSE, T., AND GROSS, M. 2014. Subspace clothing simulation using adaptive bases. *ACM Trans. Graph. 33*, 4 (July), 105:1–105:9.

HALKO, N., MARTINSSON, P. G., AND TROPP, J. A. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev. 53*, 2 (May), 217–288.

HARMON, D., AND ZORIN, D. 2013. Subspace integration with local deformations. *ACM Trans. Graph. 32*, 4 (July), 107:1–107:10.

HAUSER, K. K., SHEN, C., AND O'BRIEN, J. F. 2003. Interactive deformation using modal analysis with constraints. In *Graphics Interface*, CIPS, Canadian Human-Computer Commnication Society, 247–256.

HECHT, F., LEE, Y. J., SHEWCHUK, J. R., AND O'BRIEN, J. F. 2012. Updated sparse cholesky factors for corotational elastodynamics. *ACM Trans. Graph. 31*, 5 (Sept.), 123:1–123:13.

HUANG, J., TONG, Y., ZHOU, K., BAO, H., AND DESBRUN, M. 2011. Interactive shape interpolation through controllable dynamic deformation. *Visualization and Computer Graphics, IEEE Transactions on 17*, 7 (July), 983–992.

HUGHES, T. J. R. 2000. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis (Dover Civil and Mechanical Engineering)*. Dover Publications.

JAMES, D. L., AND PAI, D. K. 2004. Bd-tree: Output-sensitive collision detection for reduced deformable models. SIGGRAPH '04, 393–398.

JOHNSTONE, I. M., AND LU, A. Y. 2009. Sparse Principal Components Analysis. *arXiv:0901.4392 [math, stat]* (Jan.).

KHAREVYCH, L., MULLEN, P., OWHADI, H., AND DESBRUN, M. 2009. Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. Graph. 28*, 3 (July), 51:1–51:8.

KIM, T., AND JAMES, D. L. 2009. Skipping steps in deformable simulation with online model reduction. *ACM Trans. Graph. 28*, 5 (Dec.), 123:1–123:9.

KIM, T., AND JAMES, D. L. 2011. Physics-based character skinning using multi-domain subspace deformations. SCA '11, 63–72.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: Real time large deformation character skinning in hardware. SCA '02, 153–159.

MEYER, M., AND ANDERSON, J. 2007. Key point subspace acceleration and soft caching. *ACM Trans. Graph. 26*, 3, 74.

MILLER, K. S. 1981. On the inverse of the sum of matrices. *Mathematics Magazine 54*, 2, pp. 67–72.

MÜLLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. SCA '02, 49–54.

NEALEN, A., MÃČÂIJLLER, M., KEISER, R., BOXERMAN, E., AND CARLSON, M. 2006. Physically based deformable models in computer graphics. *Computer Graphics Forum 25*, 4, 809–836.

NESME, M., KRY, P. G., JEŘÁBKOVÁ, L., AND FAURE, F. 2009. Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph. 28*, 3, 52.

O'BRIEN, J. F., COOK, P. R., AND ESSL, G. 2001. Synthesizing sounds from physically based motion. In *Proceedings of ACM SIGGRAPH 2001*, 529–536.

PARLETT, B. N. 1987. *The Symmetric Eigenvalue Problem (Classics in Applied Mathematics)*. Society for Industrial and Applied Mathematics.

PENTLAND, A., AND WILLIAMS, J. 1989. Good vibrations: Modal dynamics for graphics and animation. SIGGRAPH '89, 215–222.

PERNICE, M., AND WALKER, H. 1998. Nitsol: A newton iterative solver for nonlinear systems. *SIAM Journal on Scientific Computing 19*, 1, 302–318.

R., H.-N. 1994. Context vectors: general purpose approximate meaning representations self-organized from raw data. *Computational Intelligence: Imitating Life, IEEE Press*, 43–56.

SAAD, Y. 1981. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation 37*, 155.

SAAD, Y. 2011. *Numerical Methods for Large Eigenvalue Problems, Revised Edition (Classics in Applied Mathematics)*. SIAM - Society for Industrial & Applied Mathematics.

SHABANA, A. A. 2005. *Dynamics of Multibody Systems*, third ed. Cambridge University Press. Cambridge Books Online.

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., CMU.

TENG, Y., OTADUY, M. A., AND KIM, T. 2014. Simulating articulated subspace self-contact. *ACM Trans. Graph. 33*, 4.

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Deformable models. *The Visual Computer 4*, 6, 306–331.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. *SIGGRAPH Comput. Graph. 21*, 4 (Aug.), 205–214.

VON TYCOWICZ, C., SCHULZ, C., SEIDEL, H.-P., AND HILDEBRANDT, K. 2013. An efficient construction of reduced deformable objects. *ACM Trans. Graph. 32*, 6 (Nov.), 213:1–213:10.

WANG, Y., JACOBSON, A., BARBIČ, J., AND KAVAN, L. 2015. Linear subspace design for real-time shape deformation. *ACM Trans. Graph. 34*, 4 (July), 57:1–57:11.

WU, X., DOWNES, M. S., GOKTEKIN, T., AND TENDICK, F. 2001. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. *Computer Graphics Forum 20*, 3, 349–358.

XU, H., LI, Y., CHEN, Y., AND BARBIVČ, J. 2015. Interactive material design using model reduction. *ACM Trans. Graph. 34*, 2, 18.

YANG, Y., XU, W., GUO, X., ZHOU, K., AND GUO, B. 2013. Boundary-aware multidomain subspace deformation. *IEEE Transactions on Visualization and Computer Graphics 19*, 10, 1633–1645.

ZHENG, C., AND JAMES, D. L. 2011. Toward high-quality modal contact sound. *ACM Trans. Graph. 30*, 4, 38.

A Error Analysis of the Sparse Inner Product

Eq. (6) is essentially an open-type Newton-Cotes formula[4], wherein the *midpoint rule* with a constant-value interpolating function is used. In an illustrative 1D case shown in Figure 19, the error of the numerical integration can be analytically expressed based on *Lagrange Interpolation Theorem* [Berrut and Trefethen 2004]:

$$\begin{aligned} e &= \int_a^b f(x)\mathrm{d}x - Hf(\frac{H}{2}) \\ &= \frac{H^3}{24}f^{(2)}(\upsilon) + \frac{H^5}{1920}f^{(4)}(\upsilon) + ..., \end{aligned} \quad (13)$$

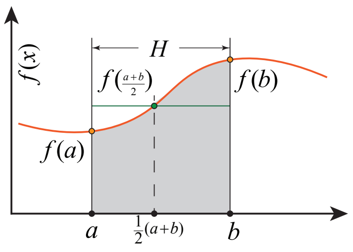

Figure 19: 1D Newton-Cotes integration. *An illustrative example of 1D open-type Newton-Cotes integration with midpoint rule.*

where $H = b - a$ is the size of the integration interval. $f^{(k)}$ denotes the k-th order derivative function of $f(x)$. υ is some value between $[a, b]$. It is noteworthy that Eq. (13) gives the approximation error of adopting sparse inner product w.r.t Eq. (5), the inner product between two vector-valued functions, while our real approximating target is the full-size mass inner products. Therefore, the numerical error induced by adopting the sparse inner product is bounded by $O((H - H_e)^3)$, where H_e is the maximum size of the element on the mesh.

[4]Eq. (6) corresponds to the open-type formula because we do not use values of target function at end points of an interval. Instead, the value at the midpoint is used.

Algorithm 2 Reduced mass MGS

```
1:  for each mode vector $u_i \in \mathsf{U}$, $i = 1...r$ do
2:      $u_i \leftarrow \frac{u_i}{\sqrt{\langle u_i, u_i \rangle}}$
3:  end for
4:  $v_0 \leftarrow u_0$; $\mathsf{U}^* \leftarrow [v_0]$                ▷ $\mathsf{U}^*$ hosts the regularized modes
5:  for $i = 2 : r$ do
6:      $v_i \leftarrow u_i$; $l \leftarrow 1.0$
7:      for $j = 1 : i-1$ do
8:          $\alpha \leftarrow \frac{\langle v_i, v_j \rangle_S}{l}$                ▷ $\alpha$ is the sparse cosine
9:          if $|\alpha| < \alpha_\tau$ then
10:             $v_i \leftarrow v_i - \alpha l v_j$
11:         end if
12:         if $1 - \alpha^2 < 0.01$ then
13:             $v_i \leftarrow \frac{v_i}{\sqrt{\langle v_i, v_i \rangle}}$; $l \leftarrow 1.0$          ▷ re normalize $v_i$
14:         else
15:             $l \leftarrow \sqrt{1 - \alpha^2} l$          ▷ incremental norm evaluation
16:         end if
17:     end for
18:     $\mathsf{U}^* \leftarrow [\mathsf{U}^* | \frac{v_i}{l}]$
19: end for
```

B Implementation Details of rMGS

The pseudo-code outlining the proposed rMGS is given in Algorithm 2. It can be seen that rMGS needs to update the sparse-norm of $\mathbf{v}_i$ (i.e., variable l in the pseudo-code) immediately, after a projection-subtraction is executed in order to evaluate α for the next loop. This subroutine sits in the innermost loop of rMGS and can be sped up by updating its sparse norm incrementally:

$$\begin{aligned} \|v_i - \langle v_i, v_j \rangle v_j\|_S &= \sqrt{(\langle v_i - \langle v_i, v_j \rangle_S v_j), (v_i - \langle v_i, v_j \rangle_S v_j)} \\ &= \sqrt{\langle v_i, v_i \rangle_S + \langle v_i, v_j \rangle_S^2 - 2\langle v_i, v_j \rangle_S^2} \\ &= \sqrt{1 - \alpha^2} l. \end{aligned}$$

This equation (line 15 of Algorithm 2) however, could accumulate roundoff error and lead to negative square rooting when α goes large and l gets smaller. To maintain the numerical stability, the values of $1 - \alpha^2$ and l are regularly checked (line 12 of Algorithm 2). If necessary, we fresh evaluate l directly using sparse inner product and re sparse-normalize $\mathbf{v}_i$ (line 13 of Algorithm 2).

C Jacobian-free Newton-PCG Solver

The Newton's method is a common choice for online subspace integration. At each time step, Newton's method seeks for an incremental displacement $\triangle \boldsymbol{q}$ as the residual minimizer iteratively. It typically requires the explicit formulation of the current tangent stiffness matrix, which is an $O(|\mathcal{E}|r^2)$ procedure. Besides, an accurate force gradient may not be available with MC scheme, recalling that Eq. (11) does not even preserve its the symmetry. To tackle this limitation associated with MC training, we do not use any direct solvers (e.g., LU, Cholesky) to calculate $\triangle \boldsymbol{q}$ within a Newton iteration. Instead, a preconditioned conjugate gradient (PCG) solver is adopted, which only needs the evaluation of the matrix-vector product. We approximate these matrix-vector products numerically instead of resorting to the analytical evaluation of the force Jacobian. Suppose that the implicit Newmark time integration [Hughes 2000] is used. Each Newton iteration needs to solve a r-dimension linear system of $\mathsf{A}\triangle\boldsymbol{q} = -\boldsymbol{e}$, where

$$\begin{aligned} \mathsf{A} &= (\alpha_1 + \zeta\alpha_4)\mathsf{M}_q + (1+\xi\alpha_4)\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\right|_{\boldsymbol{q}_i}, \\ \boldsymbol{e} &= \mathsf{M}_q((\alpha_1+\zeta\alpha_4)\delta\boldsymbol{q}_i + (\zeta\alpha_5-\alpha_2)\dot{\boldsymbol{q}}_i + (\zeta\alpha_6-\alpha_3)\ddot{\boldsymbol{q}}_i) \\ &\quad + \xi\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\right|_{\boldsymbol{q}_i}(\alpha_4\delta\boldsymbol{q}_i + \alpha_5\dot{\boldsymbol{q}}_i + \alpha_6\ddot{\boldsymbol{q}}_i) + \boldsymbol{f}(\boldsymbol{q}_i) - \boldsymbol{f}_{ext}. \end{aligned}$$

Here $\delta\boldsymbol{q}_i = \boldsymbol{q}_{i+1} - \boldsymbol{q}_i$ is the displacement deviation at current time step. $\dot{\boldsymbol{q}}_i$ and $\ddot{\boldsymbol{q}}_i$ are the known reduced velocity and acceleration at the previous step. ζ and ξ are damping coefficients. $\boldsymbol{f}_{ext}$ is the reduced external force. $\alpha_1, \alpha_2, ...\alpha_6$ are constant coefficients computed as: $\alpha_1 = \frac{1}{\beta h^2}$, $\alpha_2 = \frac{1}{\beta h}$, $\alpha_3 = \frac{1-2\beta}{2\beta}$, $\alpha_4 = \frac{\gamma}{\beta h}$, $\alpha_5 = \frac{\beta-\gamma}{\gamma}$, $\alpha_6 = \frac{(2\beta-\gamma)h}{2\beta}$, where $\beta = \frac{1}{2}$, $\gamma = 1$ are two parameters of the Newmark integrator. h is the size of each time step.

Matrix-vector product between the system matrix A and a certain vector $\boldsymbol{p}$ in the PCG solver can be written as the summation of two items according to the formulation of A:

$$\begin{aligned} \mathsf{A}\boldsymbol{p} &= (\alpha_1 + \zeta\alpha_4)\mathsf{M}_q\boldsymbol{p} + (1+\xi\alpha_4)\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\right|_{\boldsymbol{q}_i}\boldsymbol{p} \\ &= (\alpha_1 + \zeta\alpha_4)\mathsf{M}_q\boldsymbol{p} + (1+\xi\alpha_4)\cdot\mathcal{D}\boldsymbol{f}(\boldsymbol{q}_i)[\boldsymbol{p}]. \end{aligned} \tag{14}$$

The first term on the r.h.s can be directly evaluated as M_q is a constant matrix. The second term is essentially the scaled directional derivative of the reduced internal force, where the notation of $\mathcal{D}\Pi(\boldsymbol{x})[\boldsymbol{u}]$ stands for the directional derivative of a function Π at $\boldsymbol{x}$ in the direction of $\boldsymbol{u}$. Understanding this important fact allows us to use the numerical directional derivative [Chan and Jackson 1984; Brown and Saad 1990] to approximate the matrix-vector product associated with the reduced tangent stiffness matrix:

$$\mathsf{K}_q(\boldsymbol{q}_i)\boldsymbol{p} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\right|_{\boldsymbol{q}_i}\boldsymbol{p} = \mathcal{D}\boldsymbol{f}(\boldsymbol{q}_i)[\boldsymbol{p}] \approx \frac{\boldsymbol{f}(\boldsymbol{q}_i + \epsilon\boldsymbol{p}) - \boldsymbol{f}(\boldsymbol{q}_i)}{\epsilon}. \tag{15}$$

The choice of ϵ in Eq. (15) is not trivial: if ϵ is too large, the derivative is poorly approximated and if it is too small the result of the finite difference is contaminated by floating-point roundoff error. We follow the choice used in NITSOL package [Pernice and Walker 1998]:

$$\epsilon = \frac{\sqrt{1 + \| \boldsymbol{q}_i \|_2}}{\| \boldsymbol{p} \|_2}\epsilon_{machine}, \tag{16}$$

where $\epsilon_{machine}$ is the *machine epsilon*. It is typically set as 10^{-6} for 64-bit double precision and is regard as the a most suitable small number of perform an idea finite difference. The coefficient of $\frac{\sqrt{1+\|\boldsymbol{q}_i\|_2}}{\|\boldsymbol{p}\|_2}$ makes sure that the final adopted ϵ is not impaired by an over-scaled $\boldsymbol{p}$.

Preconditioning. The preconditioner plays a critical role for the PCG solver. Unfortunately, there does not yet exist a well-established theory finding the best preconditioner for every case [Shewchuk 1994]. Most preconditioning methods such as Jacobi, Gauss-Seidel, or SOR preconditioning require the information of the system matrix A, which is not available in our case as the tangent stiffness matrix is unknown. Alternatively, we design the preconditioner P as the largest invariant portion of A:

$$\mathsf{P} = (\alpha_1 + \zeta\alpha_4)\mathsf{M}_\mathsf{q} + (1+\xi\alpha_4)\mathsf{K}_\mathsf{q}(0). \tag{17}$$

We find that using the preconditioner defined in Eq. 17 is able to double the convergence rate. The initial guess of the PCG is set as $\delta\boldsymbol{q}_i$ at the very beginning of each time step and as a zero vector for the rest Newton iterations following the logic that the current $\triangle\boldsymbol{q}$ should be similar to the previous one at the first the Newton iteration, while it should quickly converge to a zero vector as Newton iteration moves forward.