

# LiCROM: Linear-Subspace Continuous Reduced Order Modeling with Neural Fields

Yue Chang  
University of Toronto  
Canada  
changyue.chang@mail.utoronto.ca

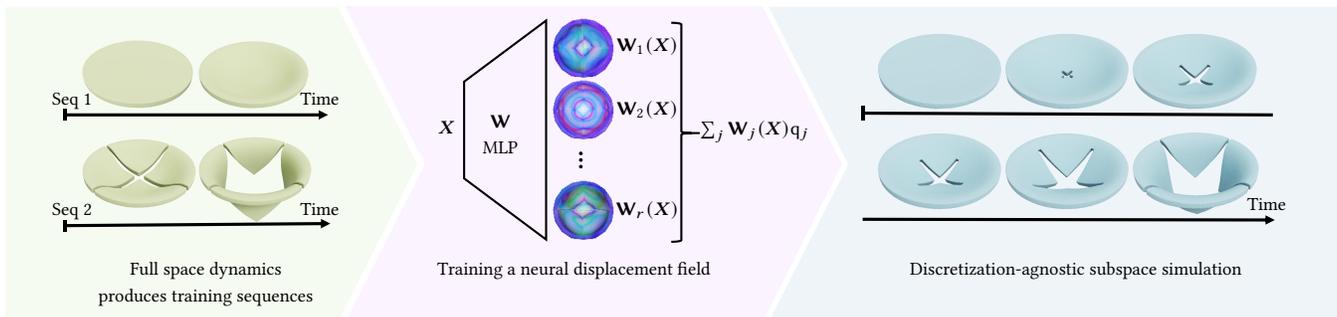
Peter Yichen Chen\*  
MIT CSAIL  
USA  
pyc@csail.mit.edu

Zhecheng Wang  
University of Toronto  
Canada  
zhecheng@cs.toronto.edu

Maurizio M. Chiaramonte  
Meta Reality Labs Research  
USA  
mchiaram@meta.com

Kevin Carlberg  
Meta Reality Labs Research  
USA  
carlberg@meta.com

Eitan Grinspun\*  
University of Toronto  
Canada  
eitan@cs.toronto.edu



**Figure 1:** Starting with training simulations (*left*), we train a linear subspace of neural displacement fields (*center*). Each such field  $W_j$  maps material position  $X$  to displacement  $W_j(X)$ . Because the map is continuous, it effectively forgets the discretizations used in the training simulations. Consequently, the resulting linear subspace  $\sum_j W_j(X)q_j$  is ideally suited for fast simulations of scenarios that benefit from adaptive discretization, such as progressive cutting and topology changes (*right*).

## ABSTRACT

Linear reduced-order modeling (ROM) simplifies complex simulations by approximating the behavior of a system using a simplified kinematic representation. Typically, ROM is trained on input simulations created with a specific spatial discretization, and then serves to accelerate simulations with the same discretization. This discretization-dependence is restrictive.

Becoming independent of a specific discretization would provide flexibility to mix and match mesh resolutions, connectivity, and type (tetrahedral, hexahedral) in training data; to accelerate simulations with novel discretizations unseen during training; and to accelerate adaptive simulations that temporally or parametrically change the discretization.

\*Corresponding authors (e-mail: pyc@csail.mit.edu, eitan@cs.toronto.edu).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0315-7/23/12...\$15.00

<https://doi.org/10.1145/3610548.3618158>

We present a flexible, discretization-independent approach to reduced-order modeling. Like traditional ROM, we represent the configuration as a linear combination of displacement fields. Unlike traditional ROM, our displacement fields are continuous maps from every point on the reference domain to a corresponding displacement vector; these maps are represented as implicit neural fields.

With linear continuous ROM (LiCROM), our training set can include multiple geometries undergoing multiple loading conditions, independent of their discretization. This opens the door to novel applications of reduced order modeling. We can now accelerate simulations that modify the geometry at runtime, for instance via cutting, hole punching, and even swapping the entire mesh. We can also accelerate simulations of geometries unseen during training. We demonstrate one-shot generalization, training on a single geometry and subsequently simulating various unseen geometries.

## CCS CONCEPTS

• Computing methodologies → Physical simulation.

## KEYWORDS

Physical simulation, Reduced-order modeling, Implicit neural representation, Neural Field

**ACM Reference Format:**

Yue Chang, Peter Yichen Chen, Zhecheng Wang, Maurizio M. Chiaramonte, Kevin Carlberg, and Eitan Grinspun. 2023. LiCROM: Linear-Subspace Continuous Reduced Order Modeling with Neural Fields. In *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia*. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3610548.3618158>

**1 INTRODUCTION**

Reduced-order modeling (ROM) using linear subspaces to approximate the solution space can accelerate deformable object simulations by orders of magnitude. The idea is to generate a number of simulated trajectory exemplars, and then identify a low-dimensional basis that approximates the exemplar displacements. We then compute dynamics by evolving only the small number of coefficients of this basis, known as *reduced coordinates* or *latent variables*.

Classical approaches to ROM assume that the input exemplars and output dynamics are all represented by a given spatial discretization, say a mesh of the domain  $\Omega \subset \mathbb{R}^3$ . This reliance on a specific discretization can be restrictive.

Being untethered from a specific discretization is desirable when input exemplars are produced using different meshes (e.g., different connectivity or resolution); simulation outputs are desired for various meshes; we wish to produce simulation output that temporally or parametrically adapts the mesh to suit the deformation (e.g., dynamic remeshing, arbitrary Lagrangian–Eulerian simulation).

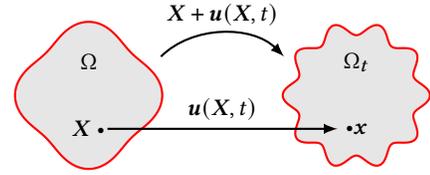
Indeed, variations need not be limited to mesh connectivity and resolution: perhaps we want to vary the mesh *type* (e.g., quad versus tetrahedral meshes) or even the discretization type (e.g., mesh, point sets with generalized moving least squares, radial basis functions, spectral discretizations).

We present such a discretization-agnostic approach to reduced order modeling. Our approach retains the linearity of the subspace of common ROM approaches, but substitutes the discrete representation of each displacement basis field with its continuous analogue.

To make things concrete, consider a simple classical ROM approach tied to a mesh with  $n$  vertices. We denote the time-varying displacement of the mesh from its reference configuration by  $\bar{\mathbf{u}}(t)$  with  $\bar{\mathbf{u}} : \mathcal{T} \rightarrow \mathbb{R}^{3n}$ , where  $\mathcal{T} (\subseteq \mathbb{R})$  denotes the temporal domain. We will place a bar (e.g.,  $\bar{\mathbf{u}}(t)$ ) over those quantities that depend on spatial discretization, i.e., those with an index ranging over  $1 \dots n$ .

In classical ROM, we approximate the time-varying displacement of the mesh as a linear combination  $\bar{\mathbf{u}}(t) \approx \bar{\mathbf{U}}\mathbf{q}(t)$  of some  $r \ll n$  dimensional, time-independent basis  $\bar{\mathbf{U}}$ , where  $\mathbf{q}(t) : \mathcal{T} \rightarrow \mathcal{Q}$  is the reduced or *latent* trajectory in the latent subspace  $\mathcal{Q} \subset \mathbb{R}^r$ , and  $\bar{\mathbf{U}} \in \mathcal{M}_{3n \times r}(\mathbb{R})$  is typically found via Proper Orthogonal Decomposition<sup>1</sup> (POD) over a training set of simulation data (temporal sequences of displacement fields);  $\mathcal{M}_{m \times n}(A)$  denotes the set of  $m \times n$  matrices over the field  $A$ . Each column  $\bar{\mathbf{U}}_k$  is a particular *discrete* displacement field over the  $n$  vertices; the mutually orthogonal columns  $\{\bar{\mathbf{U}}_1 \dots \bar{\mathbf{U}}_r\}$  form the basis for the *discrete* displacement subspace. We will use the sans serif typeface ( $\bar{\mathbf{U}}$ ,  $\mathbf{q}$ ) to denote quantities that depend on the subspace dimension  $r$ .

<sup>1</sup>POD is also known as the Karhunen–Loève transform and is closely related to Principal Component Analysis (PCA).



**Figure 2: Deformation of an elastic body.** The reference domain  $\Omega$  and the deformed domain  $\Omega_t$  at time  $t$  are related by the deformation mapping  $X \mapsto x(X, t) = X + u(X, t)$ : each deformed point  $x(X, t)$  is displaced by  $u(X, t)$  relative to the reference point  $X$ .

Now here is the crux of the matter: the discrete “architecture” of  $\bar{\mathbf{U}}$  is immutably anchored to the initial discretization. The  $j$ th row of  $\bar{\mathbf{U}}$  is the basis for the  $j$ th degree of freedom, where  $1 \leq j \leq n$ . Indeed, for a mesh discretization, the temporal evolution of the three degrees of freedom associated with  $i$ th vertex is given by

$$\bar{\mathbf{u}}_i(t) = \bar{\mathbf{W}}_i \mathbf{q}(t), \quad (1)$$

where  $\bar{\mathbf{W}}_i \in \mathcal{M}_{1 \times r}(\mathbb{R}^3)$  is a  $1 \times r$  matrix (a row vector) of  $\mathbb{R}^3$ -valued coefficients, i.e., one displacement vector per each of the  $r$  subspace modes. The  $3 \times r$  coefficients of  $\bar{\mathbf{W}}_i$  are drawn from those 3 rows of  $\bar{\mathbf{U}}$  corresponding to vertex  $i$ . (We will use boldface to denote  $\mathbb{R}^3$ -valued entries.)

Stacking the row vectors  $\bar{\mathbf{W}}_i$  of all vertices gives  $\bar{\mathbf{W}} \in \mathcal{M}_{n \times r}(\mathbb{R}^3)$ , an  $n \times r$  matrix with  $\mathbb{R}^3$ -valued entries, mapping  $\bar{\mathbf{u}}(t) = \bar{\mathbf{W}}\mathbf{q}(t)$ . Essentially,  $\bar{\mathbf{W}}$  encodes the time-invariant linear mapping from the latent configuration  $\mathbf{q}(t)$  to the full space displacements  $\bar{\mathbf{u}}(t)$ .

We are nearly ready for our novel step, the transition to the smooth setting. We view  $\bar{\mathbf{W}} = i \mapsto \bar{\mathbf{W}}_i : \{1, \dots, n\} \rightarrow \mathcal{M}_{1 \times r}(\mathbb{R}^3)$  as a map from the vertex index to the row vector of subspace weights. This is a discrete map, and that is what we will now make smooth.

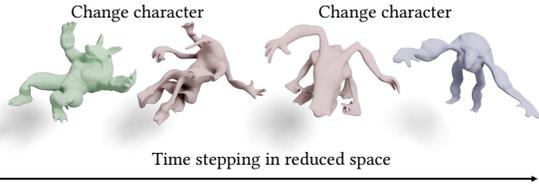
In lieu of the discrete map  $\bar{\mathbf{W}}$ , we propose to instead use a continuous map  $\mathbf{W} = X \mapsto \mathbf{W}(X) : \Omega \rightarrow \mathcal{M}_{1 \times r}(\mathbb{R}^3)$  taking a point  $X \in \Omega$  in the reference domain to its subspace weights, so that

$$\mathbf{u}(X, t) = \mathbf{W}(X)\mathbf{q}(t). \quad (2)$$

Comparing to (1), the discretization-dependent *discrete* index  $i$  is replaced by a discretization-independent *continuous* reference point  $X$  (see Fig. 2). The time-varying spatially-varying displacement field  $\mathbf{u}(X, t)$  is a linear combination of spatially-varying, time-invariant basis of displacement fields, whose time-varying, spatially-invariant weights are given by  $\mathbf{q}(t)$ .

To aid in intuition, we can also compare the columns of  $\bar{\mathbf{U}}$ ,  $\bar{\mathbf{W}}$ , and  $\mathbf{W}$ . In all cases, the  $k$ th column is a representation of a particular displacement—a basis element of the approximating subspace—as a field over the entire domain  $\Omega$ ; the distinction is that  $\mathbf{W}_k$  is a continuous field, whereas the others are discrete column vectors.

Equation 2 is the basis for *linear continuous ROM* (LiCROM). Now the training set can span multiple discretizations of the same geometry, or even multiple geometries. This facilitates and broadens the applicability of reduced order modeling: As we will show, with LiCROM we can compute latent dynamics on geometries unseen during training; simulations that modify the geometry at runtime via cutting, hole punching, or swapping the entire mesh (Fig. 3), without re-initializing the reduced coordinates.



**Figure 3: Interactive manipulation and one-shot generalization.** Training a neural basis on deformations of the Armadillo, our application allows the user to interactively tug at the geometry. Unlike discretization-dependent reduction techniques, we can easily substitute the geometry. We compute the latent dynamics on three meshes not seen during training. While the kinematics are defined by the training set, the physical response is defined by the geometry of the current mesh, as evident in details, such as the wobbling of the stick arms. Frame rate: 30 frames per second. Full space time step cost: 335ms; reduced: 6ms. Hardware: Intel Core i7-10750H.

## 2 RELATED WORK

*Linear reduced-order modeling.* Model-reduction techniques [Benner et al. 2015] have proven to be a powerful tool for enabling high-fidelity models to be run in real-time. They have been successfully applied to problems in many fields, such as fluid dynamics [Bergmann et al. 2005; Carlberg et al. 2017, 2013; Hall et al. 2000; Kim et al. 2019; Kim and Delaney 2013; Lieu et al. 2006; Mainini and Willcox 2015; Treuille et al. 2006; Wiewel et al. 2019; Willcox and Peraire 2002], solid mechanics [An et al. 2008a; Barbič and Zhao 2011; Barbič and James 2005; James et al. 2006; Kim and James 2009; Xu et al. 2015; Yang et al. 2015], secondary motion for rigged animation [Bencheikroun et al. 2023; Xu and Barbič 2016] and robotics [Katzschmann et al. 2019; Tan et al. 2020].

Typically, the reduced space is learned from training exemplars [Barbič and James 2005; Berkooz et al. 1993; Fulton et al. 2019], or identified in a “data-free” manner from energetic first principles [Pentland and Williams 1989; Shabana 2012; Sharp et al. 2023; Yang et al. 2015]. “Online” approaches update the basis at runtime based on the observed trajectory [Kim and James 2009; Mukherjee et al. 2016; Ryckelynck 2005]; a related approach is to interpolate between precomputed bases [Xu and Barbič 2016]. We learn a fixed basis from simulated exemplars.

Most model-reduction methods employ a linear-subspace approximation for the kinematics. Such approximations are accurate for problems displaying a rapidly-decaying Kolmogorov  $n$ -width [Pinkus 2012]. However, nearly all of these operate with a discrete representation; those that do operate with the continuous representation (e.g., reduced-basis methods) are intrinsically tied to an underlying spatial discretization scheme. There have been a few methods that applied nonlinear kinematic approximations, which we will discuss below. Crucially, most of these also operate on a *discrete* representation, with the exception of CROM [Chen et al. 2023a,b], which has been applied to the material point method and to various partial differential equations.

We fill the gap in the literature by developing the first *linear* kinematic approximation that is also independent of any spatial discretization.

*Deep-learning-based reduced-order modeling.* Lee and Carlberg [2018] introduced the first framework utilizing autoencoders to capture nonlinear manifolds. Fulton et al. [2019] extended this idea, combining it with POD for deformable solid dynamics. In a complementary approach, Shen et al. [2021] used nonlinear autoencoders to efficiently execute Hessian-based latent space dynamics by accurately computing high-order neural network derivatives. Furthermore, Romero et al. [2021] introduced contact-induced deformation correction with linear subspace modes. Meanwhile, Luo et al. [2020] focused on displacement correction, aiming to transform linear elastic responses into more complex constitutive ones.

*Discretization-independent representations.* Recently, implicit neural representations have become an exciting area of exploration in many fields, including shape modeling [Chen and Zhang 2019; Park et al. 2019], 3D reconstruction [Mescheder et al. 2019; Mildenhall et al. 2021], image representation and generation [Chen et al. 2021; Shaham et al. 2021; Skorokhodov et al. 2021], and PDE-constrained problems [Chen et al. 2022; Raissi et al. 2019; Yang et al. 2021; Zehnder et al. 2021].

Aigerman et al. [2022] proposed a framework to accurately predict piecewise linear mappings of arbitrary meshes using a neural network. It works with heterogeneous collections of meshes without requiring a shared triangulation. Others aim to learn the latent space representation of continuous vectors: Chen et al. [2023a] proposed a model reduction method for material point method, while Chen et al. [2023b] and Pan et al. [2023] learned a discretization-agnostic latent space for PDEs. To the best of our knowledge, the prototypical factored structure of linear ROM,  $\mathbf{W}(X)q(t)$ , has not been considered in the context of continuous discretization-independent representations for model reduction.

## 3 DISCRETIZATION-BLIND SUBSPACE LEARNING

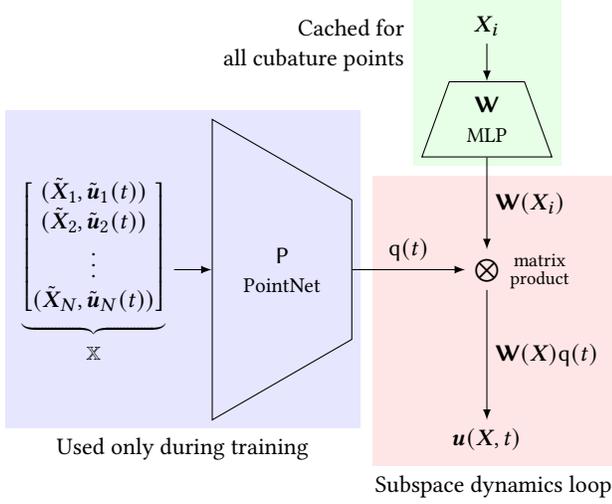
We train LiCROM over an observed trajectory of a deformable object. To simplify notation, assume one trajectory sampled at instances  $\{t^1, \dots, t^m\}$ , although the approach trivially generalizes to sampling multiple trajectories or multiple objects with parallel trajectories.

Let  $\mathbb{X} = \{(\tilde{X}^1, \tilde{u}^1), \dots, (\tilde{X}^m, \tilde{u}^m)\}$  be the training set, where  $(\tilde{X}^j, \tilde{u}^j)$  collects observations of the displacement field at time  $t^j$ . In particular,  $\tilde{u}^j = \{u_1^j, u_2^j, \dots\} \subset \mathbb{R}^3$  consists of a finite number of observations  $u_i^j \equiv \mathbf{u}(X_i^j, t^j)$  of the displacement field at reference positions  $\tilde{X}^j \equiv \{X_1^j, X_2^j, \dots\}$ . We do not assume a consistent structure between point clouds, i.e., the sample positions  $X_i^j$  and  $X_i^{j+1}$  need not be equal, nor the sample counts  $|\tilde{X}^j|$  and  $|\tilde{X}^{j+1}|$ .

We seek a low-dimensional subspace that spans all the observed fields  $(\tilde{X}^j, \tilde{u}^j)$ . In particular, we seek a projection  $P : (\tilde{X}^j, \tilde{u}^j) \mapsto q^j \in Q$ , and a corresponding basis  $\mathbf{W}$  (independent of  $j$ ) such that

$$\mathbf{W}(X_i)P(\tilde{X}^j, \tilde{u}^j) \approx u_i^j, \quad \forall (\tilde{X}^j, \tilde{u}^j) \in \mathbb{X}, \quad \forall X_i \in \tilde{X}^j. \quad (3)$$

We adopt a parametric form for  $P$  and  $\mathbf{W}$ , in particular a PointNet encoder [Qi et al. 2017] and neural implicit field [Mescheder et al.



**Figure 4: Network architecture.** To learn the subspace, the PointNet encoder  $P(X, u)$  and the 5-layer multilayer perceptron  $W(X)$  are optimized over the training set  $\mathbb{X}$ . This yields a compact, factored kinematic space where the displacement field  $u(X, t)$  is a  $q(t)$ -weighted linear combination of  $r$  precomputed time-invariant continuous displacement fields  $\{W_1(X) \dots W_r(X)\}$ . During subspace simulation, the PointNet encoder is no longer required: it has served its purpose, finding an embedding for  $q(t)$ . The evaluation of  $W(X)$  at cubature points  $\{X_i\}$  is performed once when the cubature scheme is established, and cached for reuse. Subsequently, the subspace simulation inner loop requires only the usual matrix-vector product  $W(X_i)q$  that is the cornerstone of linear subspace methods.

2019; Park et al. 2019], respectively, and optimize the parameters to minimize the squared norm residual of (3), as depicted in Fig. 4.

### 3.1 Training Method

In our experiments, we produced the training set using simulations based on tetrahedral mesh discretizations. However, observe that the network does not directly “know” that the input was generated by a mesh, only that a sampled displacement field was generated somehow. The network aims to find a reduced basis that can reconstruct all observed displacement fields, without consideration for loading, boundary conditions, geometry, or discretization.

To produce our training set, we first generate a volumetric tetrahedral mesh for each geometry using TetWild [Hu et al. 2018], and then execute the desired full-space simulation using a CPU-based *taichi* [Hu et al. 2019] implementation that closely follows the default implicit FEM integrator in *warp* [Macklin 2022]. We repeat this process to produce a set of simulation results. Our implementation uses the same cardinality  $\tilde{n} = |(\tilde{X}^j, \tilde{u}^j)|$  for the randomly sampled vertex-based displacements of each animation frame, which simplifies batch processing in PyTorch. We found that training the PointNet encoder  $P$  can be expensive when ( $\tilde{n} > 5000$ ), yet using a large cardinality is helpful for coverage of the domain. Therefore, we further subsample  $\tilde{n} < \tilde{n}$  vertices for the PointNet encoder. We determine the parameters for  $P$  and  $W$  by minimizing the  $L_2$

reconstruction loss

$$\mathcal{L} = \sum_{j=1}^m \sum_{i=1}^{\tilde{n}} \left\| \mathbf{W}(X_i) P \circ S_{\tilde{n}}(\tilde{X}^j, \tilde{u}^j) - \mathbf{u}_i^j \right\|_2, \quad (4)$$

where  $S_{\tilde{n}}$  is the subsampling operator. We used  $\tilde{n} = 2500$  for all examples.

**PointNet architecture.** The PointNet encoder  $P$  is invariant under permutation of input points, a desirable feature for our unordered sets. A standard PointNet is also invariant under input transformations due to its input stage feature-transform net; we removed this stage since latent space variables are not invariant under transformations of *displacements*. The input to the PointNet is an unordered set of points  $(X_i, \mathbf{u}_i) \in \mathbb{R}^3 \times \mathbb{R}^3 \equiv \mathbb{R}^6$  and the output is  $q$ .

**Neural field architecture.** The architecture for the neural field  $W$  is a 5-layer multilayer perceptron (MLP) of width 60 with ELU [Clevert et al. 2016] activation functions. We used this architecture for all presented examples, however, we found that alternatives such as SIREN [Sitzmann et al. 2020] can also generate good results.

**Learning network parameters.** We use PyTorch Lightning to implement the entire training pipeline [Falcon and The PyTorch Lightning team 2019]. We adopted the Adam optimizer [Kingma and Ba 2017] and apply Xavier initialization. We train the model for 3750 epochs with a base learning rate of  $lr = 10^{-3}$ . After the first 1250 epochs, we divide the learning rate by 5, then we further divide it by 10 after another 1250 epochs. We used a batch size 16 for the network’s input, so the batch size is  $16 \cdot \tilde{n}$  for  $W$ .

## 4 DYNAMICS VIA IMPLICIT INTEGRATION

We formulate an implicit timestep in the framework of optimization time integrators [Martin et al. 2011; Pan et al. 2015; Stuart and Humphries 1996], which were recently used for latent space dynamics by Fulton et al. [2019]. The configuration  $q$  at the end of the  $(j + 1)$ th time step minimizes

$$E(q) = \int_{X \in \Omega} \frac{1}{2h^2} \left\| \mathbf{W}(X)q - \mathbf{u}_{\text{pred}} \right\|_g + \Psi(X + \mathbf{W}(X)q) \, d\text{Vol}, \quad (5)$$

where  $h$  is the duration of the time step,  $g$  is the kinetic energy<sup>2</sup> norm, and  $\Psi(x)$  is the elastic energy density, in our implementation stable neohookean [Smith et al. 2018]. The explicit predictor for the  $(j + 1)$ th time step

$$\mathbf{u}_{\text{pred}}^{j+1} = \mathbf{u}^j + h\mathbf{v}^n + h^2 M^{-1} \mathbf{f}_{\text{ext}} \quad (6)$$

requires the full-space velocity given by the finite difference

$$\mathbf{v}^n = \frac{\mathbf{u}^n - \mathbf{u}^{n-1}}{h} = \mathbf{W}(X) \frac{q^n - q^{n-1}}{h} = \mathbf{W}(X) \dot{q}^n, \quad (7)$$

where (by linearity of the subspace)  $\dot{q}^n = (q^n - q^{n-1})/h$ .

We approximate the domain integral (5) via cubature

$$E(q) \approx \sum_i \frac{w_i}{2h^2} \left\| \mathbf{W}(X_i)q - \mathbf{u}_{\text{pred}} \right\|_g + w_i \Psi(X_i + \mathbf{W}(X_i)q), \quad (8)$$

<sup>2</sup>The kinetic energy norm  $\|\mathbf{v}(X)\|_g = \int_{\Omega} \frac{1}{2} \rho(X) \mathbf{v}(X)^2 \, d\text{Vol}$ , where  $\rho(X)$  is the mass density.

where  $w_i$  is the weighting of the  $i$ th cubature point  $X_i$ . Our implementation performs the cubature and energy density computation using a mesh, motivated by the readily available methods for volumetric deformables [An et al. 2008b], although the mathematics are not tied to mesh-based cubature.

Regardless, the cubature mesh is not and need not be tied to the representation of the training data. Furthermore, the cubature mesh need not be the same across time steps, since the state is carried across time steps by the latent configuration  $q$ . The cubature should be chosen to adequately control the approximation (8) and to enforce the essential boundary conditions.

This freedom makes scenarios that have connectivity changes (e.g., fracture, cutting), and topology changes (e.g., punching out a hole, growth of voids) refreshingly trivial: we simply choose an appropriate cubature scheme for the next time step. For instance, if a hole is instantaneously punched out, we simply refrain from integrating over the excised domain, by switching to a cubature mesh that reflects the revised topology and revised boundary conditions.

An alternative to switching the mesh would be to skip cubature points that lie in the void. The key point is that there is a lot of freedom in the approach—even across time steps—to integrating of the domain integral (8), because the representation of the configuration,  $q$ , is separated from the representation of cubature.

## 5 MINIMIZATION VIA CUBATURE

*Minimization.* We minimize  $E(q)$  using gradient descent [Macklin 2022]. We initialize the increment at every cubature point with the explicit time stepping prediction  $\Delta u_i = h v^j + h^2 M^{-1} f_{\text{ext}}$ . At every descent iteration, we compute the increments at all cubature points, and then find the best-fit increment to the latent configuration. The descent increment at the  $i$ th cubature point is

$$\Delta u_i = \alpha \left( \frac{M}{h^2} (\mathbf{W}(X_i)q - q_{\text{pred}}) + \frac{\partial \Psi(X_i + \mathbf{W}(X_i)q)}{\partial \mathbf{W}(X_i)q} \right). \quad (9)$$

After evaluating the full space increment at every cubature point, which we project to find the best fit subspace increment by minimizing the quadratic

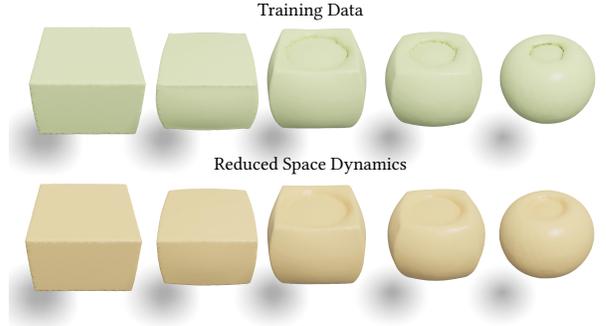
$$\Delta q = \arg \min_{\Delta q} \sum_i w_i \|\mathbf{W}(X_i)\Delta q - \Delta u_i\|^2, \quad (10)$$

which amounts to solving a symmetric positive definite linear system. The matrix depends only on the position and weight of the cubature points, and whilst these are invariant, a single Cholesky factorization allows for repeated projections via backsubstitution.

When the cubature set changes, we reassemble the system matrix. Since  $\mathbf{W}(X_i)$  is a function of  $X_i$ , we cache it at each cubature point, eliminating the network inference  $\mathbf{W}(X_i)$  except at newly introduced cubature points.

*Cubature Sampling.* Previous cubature sampling [An et al. 2008b; von Tycowicz et al. 2013] provides promising results. One generates a set of training poses for the cubature optimization preprocess. This preprocess identifies desirable cubature points and associated nonnegative weights to achieve accurate energy approximation over the training poses.

But what about integrating subspace dynamics on novel meshes unseen during training? In this case, the aforementioned approach



**Figure 5: One neural basis spans the deformations of multiple shapes. During precomputation, we train one neural basis,  $(\mathbf{W}, q)$ , with a single training set encompassing the full-space simulated deformations of five shapes spanning cube to sphere (blue). Using a continuous displacement field basis makes training on multiple shapes straightforward. During the online subspace dynamics, we simulate the same shapes with the same loading conditions (yellow), observing good agreement, including for the top surface details.**

is not directly applicable. We implemented a naïve cubature scheme, which we found satisfactory for the examples that we tested. We

- (1) select  $m$  vertices randomly from the tetrahedral mesh, and
- (2) additionally, select all the vertices incident to the  $m$  vertices.

These steps yield the equiweighted cubature points  $\{X_i\}$ .

In all presented results that do not involve remeshing, we precompute the cubature scheme. For the remeshing examples, the cubature points in principle would change (locally) when the mesh is changed (locally). In our simplified demonstration of remeshing, where we know the sequence of meshes in advance, we precompute the cubature points and  $\mathbf{W}(X_i)$  for all meshes.

## 6 RESULTS

We conduct experiments to evaluate the unique features of LiCROM. We ask whether one neural basis,  $(\mathbf{W}, q)$ , can

- (1) be trained over diverse inputs generated by different meshes?
- (2) reproduce deformations on geometries seen during training?
- (3) and on novel geometries unseen during training?
- (4) facilitate mesh connectivity and topology changes?

### 6.1 Unique capabilities of a continuous ROM

*Training with different shapes.* We train one neural subspace  $(\mathbf{W}, q)$  using a training set comprised of different shapes deformed under similar load, and ask whether the subspace dynamics reconstruct the different behaviors of the shapes included in the training set. We generate five shapes spanning cube to sphere, with equal bounding cubes,  $[\pm 0.5, \pm 0.5, \pm 0.5]$ . We prescribe equal compressive displacement: for every vertex with underformed position near the top ( $y < 0.45$ ) or bottom ( $y > -0.45$ ) we prescribe an equal downward ( $-2m/s$ ) or stationary ( $0m/s$ ) velocity. A single training set includes the full-space dynamic deformations for these five meshes. We simulate the same five shapes in the reduced model (see Fig. 5), noting agreement with the training data.

**Table 1: Performance statistics. We list the average simulation time cost (in seconds) for full-space simulations and reduced-space simulations. We also listed the number of sampled vertex and tetrahedrons. A latent space dimension  $r = 20$  was used for all examples. The Young’s modulus is  $5 \times 10^5$  for the dragon and bunny example, and is  $2.5 \times 10^6$  for other examples. We adopted a Poisson ratio of 0.25 for all examples. Hardware: Intel Core i7-10750H.**

Example	vertex count	tetrahedron count	sampled vertex count	sampled tet count ( $\bar{n}$ )	full space step cost (ms)	reduced space step cost (ms)	speedup
Training with different shapes	20K	103K	1.3K	3K	142	8	17
Kinematic tearing	20K	91k-94K	3.7K	5.6K	323	11	29
Hole punching	20K	95K-100K	3.8K	6.3K	288	13	22
Falling Animals	40K	200k-210K	1.3K	2.1K	350	8	43
Interactive application	100K	516K	1.4K	2.2K	335	6	56
Dragon	80K	429K	3.3K	5K	307	9	34
Bunny	20K	101K	1.6K	2.4K	267	9	29

**Table 2: Statistics on precomputation. We include the data volume and time required for data generation and training for each example. During the data collection phase, we capture snapshots from various loading conditions, recording vertex displacements at specific time step. We also listed the total cost of all sampling operations, including selecting cubature points and caching  $\mathbf{W}(X)$ .**

Example	vertex count	training snapshots count	number of loadings	data generation cost (min)	training cost (h)	cubature selection (ms)	evaluating $\mathbf{W}(X)$ (ms)
Training with different shapes	20K	1650	5	3.4	4.9	15.9	32.7
Kinematic tearing	20K	1300	2	24.0	4.1	90.6	210.2
Hole punching	20K	5600	8	3.7	16.0	70.0	12.3
Falling Animals	40K	3600	3	32.0	10.6	23.1	6.4
Interactive application	100K	1200	20	96.1	7.7	188.0	11.1
Dragon	80K	1275	1	13.3	5.1	40.6	14.1
Bunny	20K	4800	8	12.9	13.4	26.2	10.4

We repeat this experiment, this time with subspace dynamics on novel shapes unseen during training (see Fig. 7). We observe good agreement for the overall deformation, albeit sometimes with missing surface details, when these were not seen during training.

*Kinematic tearing.* Because the cubature mesh does not carry state, it need not be tied to the meshing used in previous time steps nor the training phase. Combined with the ability to train on meshes with different connectivity, these traits make subspace modeling of tearing and fracture easier (see Fig. 6). During precomputation, we train one neural basis using a single training set comprised of two full-space simulations: (1) a clamped plate sagging under gravity; (2) the same plate, with a Y-shaped cut, sagging under gravity.

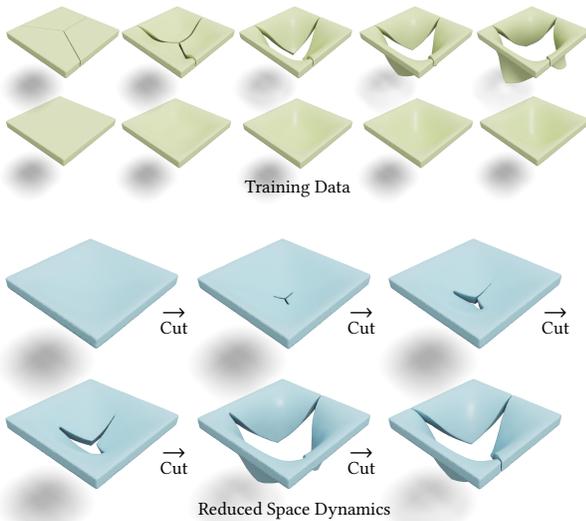
In the online phase, we model the tearing of the plate using subspace dynamics. Over time, we progressively redefine the cubature mesh to grow a Y-shaped cut (see Fig. 6). The cuts introduced in the cubature mesh have the desired effect on the force computation, but they do not require a transfer of state variables from the previous mesh. Recall that the training set includes only the intact and fully cut geometry; the deformations for the partial cut arise (as in all linear subspace approaches) from a weighted sum of the precomputed displacement fields.

A natural question then is “how well does the continuous neural displacement field capture a discontinuous deformation?” This is particularly poignant as our implementation employs smooth ELU activation functions. We visualize the basis displacement field  $\mathbf{W}(X)$  (see Fig. 8), observing the discontinuity.

Since the neural basis has no “knowledge” of the geometry, the cubature bears full responsibility for providing geometric knowledge, and therefore producing distinct dynamics for distinct geometries. Undersampling produces artifacts (see Fig. 9). Using 3713 random samples (compared to 20k vertices in the original data) is sufficient to obtain a 29× speedup over the full space simulation.

*Hole punching.* In addition to simulating fractures, our method is capable of simulating the process of punching the cube and generating voids in real-time. In the example shown in Figure 10, we run simulation on 5 meshes with a fixed bottom under gravity. After training, we can simulate the process the cube being “damaged” (i.e. holes being cut out) by runtime remeshing. Note that after each remesh, the deformed position of the rest of the cube is consistent with the frame before the remeshing, except for the newly generated empty part.

*Rolling animals.* Our method is able to simulate the collision and friction between the animals and the static inclined plane. For the example shown in Figure 11, when generating training data, we simulated an elastic animal under static gravity  $g = -9.8m/s$ . In each frame, we check if any vertex intersects with an infinite plane with normal  $[0, \sqrt{2}/2, \sqrt{2}/2]$ . If an intersection happens, we apply a penalty force along the normal of the plane to handle collision and set the velocity orthogonal to the plane normal to zero (infinite friction force). Results show that our latent space dynamics can reconstruct the colliding and rolling interaction between different animals and the plane.



**Figure 6: Kinematically-prescribed Y-shaped tear.** During pre-computation, we train one neural basis on the full-space simulations of both an intact plate and a Y-cut plate sagging under gravity. During the online subspace dynamics, the plate is cut progressively (on a prescribed schedule) by redefining the connectivity of the cubature mesh. This novel partial-cut connectivity of the cubature mesh is unseen during training. The deformations for the partial cut arise naturally from the available neural basis displacement fields.

*Animal interpolation.* After training on the three animals in Fig. 12, we interpolate among these three meshes via Wasserstein distances [Solomon et al. 2015]. Thanks to the discretization-agnostic nature of our method, we can readily deploy the previously trained model with all these meshes. Fig. 12 demonstrates the corresponding latent space dynamics for each mesh.

## 6.2 Interactive application

We trained a neural basis on deformations induced by tugging at the armadillo (see Fig. 3). The full-space and reduced simulations require 335ms and 6ms per time step, respectively, on an Intel Core i7-10750. The 56 $\times$  speedup enables interactive manipulation at 30 frames per second.

The user can also load in previously unseen geometric models that can be swapped for the armadillo, mid-simulation, without resetting the kinematic configuration or momentum. Note that the physical response is evaluated on the current geometry. Although the kinematic training was conducted solely on the armadillo, the physical response reflects the geometry, as evident, e.g., in the higher frequency oscillations of the thinner arms. This demonstrates the one-shot generalization potential of LiCROM. To the best of our knowledge, this is the first interactive-rate demonstration of model reduction that includes online substitutions of geometric model, including previously unseen geometric models. Indeed, by training on a single geometry, our approach generalizes to other geometries, effectively achieving *one-shot generalization*.

## 6.3 Comparison with nonlinear CROM

Our method shares a similar motivation with Continuous Reduced-Order Modeling (CROM) [Chen et al. 2023a,b]. Both seek discretization independence. In CROM, a nonlinear decoder  $(q, X) \mapsto u$  maps the reduced configuration and reference position to the corresponding deformed position. Compared to a linear basis, a nonlinear approach may be more complex to implement, analyze and compute, or more carefully chosen training data to avoid overfitting. We observed artifacts when applying CROM to deformable simulation, which spurred our investigation into a linear subspace (see Fig. 14).

LiCROM offers an important advantage over CROM in the projection (10), which, due to the linearity of the basis, becomes a simple minimization of a quadratic, i.e., the solution of a linear system which, by prefactorization, can be reused along with cubature points. By contrast, the nonlinearity of the CROM basis [Chen et al. 2023b] does not allow for such a trivial projection.

We leverage this fast projection (which amounts to just back-substitution on the prefactored matrix) to implement implicit time stepping, which requires *repeated* projections each time step. In the nonlinear CROM, each such projection would require multiple expensive network Jacobian evaluations.

## 7 DISCUSSION

We have presented the first discretization-independent linear model reduction method, in the sense that the subspace basis does not explicitly store, refer, or rely on particulars of the discretizations employed to generate the training set, integrate the forces, or output the resulting animation.

This discretization-independence is achieved by defining the subspace basis vectors as *continuous* displacement fields over the reference domain, which we implement using neural implicit fields.

Consequently, we are able to demonstrate that a single subspace model can be trained from differing discretizations or even differing geometries. The learned basis can accelerate simulation by about 20–50 $\times$  whilst supporting phenomena not typically seen in subspace methods, such as phenomena that typically require remeshing (e.g., cutting), changes to topology (e.g., hole punching), or novel geometry unseen during training.

*Limitations.* These novel features are accompanied by novel limitations. First, the trained subspace is of course limited by the observed data. For a neural implicit field, this usual limitation is accompanied by a novel aspect: the field will not hesitate to “hallucinate” an extrapolated result in portions of the reference domain  $\Omega$  that had few or no data observations. As a corollary, if we train a displacement basis on a thin geometry, this basis may not be suitable for a thick geometry, where some cubature points will sample a potentially unsuitable extrapolation of the displacement field. It would be interesting to incorporate regularizers for such extrapolation [Liu et al. 2022]. Fig.15 (a) and (b) demonstrate two modes of generalization failures of our approach: vastly different loading conditions and geometric sizes.

Indeed, since the training of the subspace has no explicit knowledge about the geometry, the trained subspace may fail to reconstruct certain surface details when tested on novel geometry that is not included in the training data, as shown in Fig. 7(b). It would be

interesting to ameliorate this limitation by introducing an explicit “geometry code” when training and later using the network.

Second, the combined training of a neural implicit field and PointNet is expensive compared to POD, requiring several hours. This is the cost we trade for the benefit of PointNet’s permutation invariance. Interestingly, if this permutation invariance were discarded in lieu of a simpler, permutation-dependent decoder, some aspects of discretization-independence would remain. In particular, while the resulting embedding would no longer be independent of input discretization, the resulting displacement field basis would *still* be continuous and therefore *not* impose any discretization on the cubature scheme nor the subspace dynamics output. Future work may involve accelerating training while retaining permutation-invariance.

Our model also shares the shortcomings and benefits of linear-subspace model reduction methods: the dimension of the subspace typically exceeds that of nonlinear approaches, regardless of whether the displacement-field is encoded as a discrete [Fulton et al. 2019] or continuous [Chen et al. 2023a,b] field. However—with the exception of methods developed in the computational math community [Lee and Carlberg 2018]—the state of the art in nonlinear approaches (especially in graphics) still seems to rely on linear subspaces for regularization [Fulton et al. 2019; Shen et al. 2021]; perhaps these same kinds of regularizations can be applied in the continuous domain, e.g., by regularizing CROM with LiCROM.

Unlike other linear-subspace ROMs, ours is not trained using POD, nor does the training objective explicitly ask for orthogonality. Orthogonality optimizes the conditioning of the basis, and is desirable for reducing error during projection; we did not observe any challenges with projection. We intend to evaluate the angle between basis vectors and report this in the near future.

*Future work.* Our preliminary implementation leaves open many immediate steps. We employed a random cubature sampling approach with equal weights, solely for its simplicity and immediacy. Recall that the Y-shaped tear required 3.7k random samples. It seems reasonable to expect that a data-aware sampling approach in the spirit of An et al. [2008b] could reduce the number of cubature points. Since our examples include geometries unseen during training, the sampling strategy would have to be adapted to the data at runtime.

Following *warp*, we used gradient descent to minimize the energy, however, alternatives abound. For instance, our implementation is immediately amenable to incorporating an (L-)BFGS solver, which approximates Newton’s method without using a Hessian. Indeed, due to the linearity of the subspace, computing the reduced energy Hessian, as required for an exact Newton’s method, is straightforward via  $\text{Hess}_q \Psi(\mathbf{X} + \mathbf{u}(q)) = \mathbf{W}(\mathbf{X})^T \text{Hess}_u \Psi(\mathbf{X} + \mathbf{W}(\mathbf{X})q) \mathbf{W}(\mathbf{X})$ , which can be assembled via cubature at  $\{X_i\}$ . Note that the exact Hessian evaluation does not require differentiating through the neural network, which would be the case for a nonlinear subspace [Chen et al. 2023b; Fulton et al. 2019].

Although we began with *warp* on the GPU, we ultimately implemented our online subspace dynamics solely on the CPU with *taichi*. In our intended application domains (virtual reality, games) there is significant contention over GPU acceleration, which is primarily reserved for rendering. Achieving interactive rates on a CPU, albeit

more limiting, was an important criterion. However, for other use cases, a fast GPU implementation remains desirable, and we intend to re-implement this method on the GPU.

*Open source.* Our implementation of LiCROM will be released.

## ACKNOWLEDGMENTS

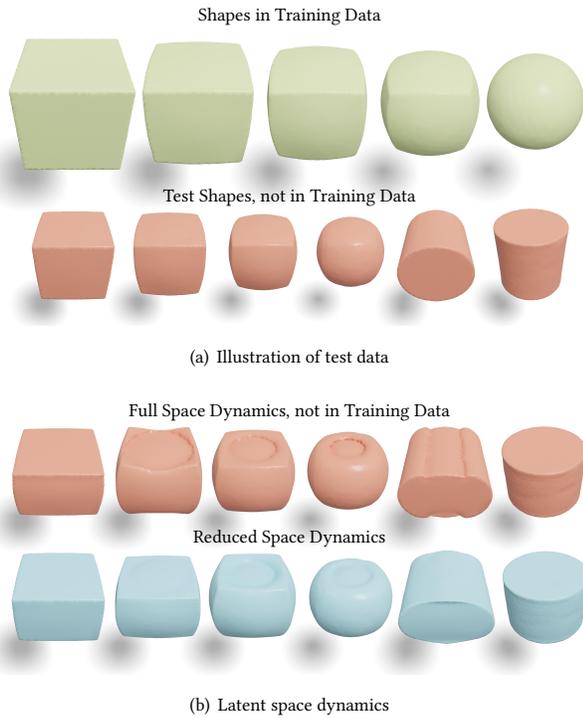
We would like to thank Otman Benckroun, Jonathan Panuelos, Kateryna Starovoit, and Mengfei Liu for their feedback on Fig 1. We would also like to thank our lab system administrator, John Hancock, and our financial officer, Xuan Dam, for their invaluable administrative support in making this research possible. This project is funded in part by Meta and the Natural Sciences and Engineering Research Council of Canada (Discovery RGPIN-2021-03733). We thank the developers and community behind PyTorch, the Taichi programming language, and NVIDIA Warp for empowering this research. The meshes in Fig 3 are derived from entries 133568, 133078 and 170179 of the Thingi10k dataset [Zhou and Jacobson 2016].

## REFERENCES

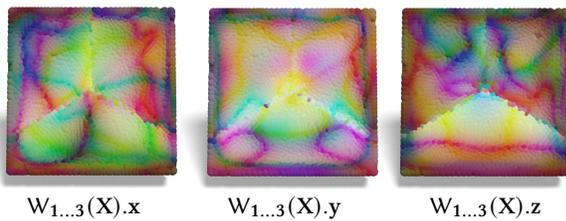
- Noam Aigerman, Kunal Gupta, Vladimir G. Kim, Siddhartha Chaudhuri, Jun Saito, and Thibault Groueix. 2022. Neural Jacobian Fields: Learning Intrinsic Mappings of Arbitrary Meshes. *ACM Transactions on Graphics (TOG)* 41, 4, Article 109 (jul 2022), 17 pages. <https://doi.org/10.1145/3528223.3530141>
- Steven S An, Theodore Kim, and Doug L James. 2008a. Optimizing cubature for efficient integration of subspace deformations. *ACM transactions on graphics (TOG)* 27, 5 (2008), 1–10.
- Steven S. An, Theodore Kim, and Doug L. James. 2008b. Optimizing Cubature for Efficient Integration of Subspace Deformations. *ACM Transactions on Graphics (TOG)* 27, 5, Article 165 (dec 2008), 10 pages. <https://doi.org/10.1145/1409060.1409118>
- Jernej Barbič and Yili Zhao. 2011. Real-time large-deformation substructuring. *ACM transactions on graphics (TOG)* 30, 4 (2011), 1–8.
- Jernej Barbič and Doug L. James. 2005. Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models. *ACM Transactions on Graphics (TOG)* 24, 3 (jul 2005), 982–990. <https://doi.org/10.1145/1073204.1073300>
- Otman Benckroun, Jiayi Eris Zhang, Siddhartha Chaudhuri, Eitan Grinspun, Yi Zhou, and Alec Jacobson. 2023. Fast Complementary Dynamics via Skinning Eigenmodes. *ACM Transactions on Graphics (TOG)* 42, 4, Article 106 (jul 2023), 21 pages. <https://doi.org/10.1145/3592404>
- Peter Benner, Serkan Gugercin, and Karen Willcox. 2015. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM review* 57, 4 (2015), 483–531.
- Michel Bergmann, Laurent Cordier, and Jean-Pierre Brancher. 2005. Optimal rotary control of the cylinder wake using proper orthogonal decomposition reduced-order model. *Physics of fluids* 17, 9 (2005), 097101.
- Gal Berkooz, Philip Holmes, and John L Lumley. 1993. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual review of fluid mechanics* 25, 1 (1993), 539–575.
- Kevin Carlberg, Matthew Barone, and Harbir Antil. 2017. Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction. *J. Comput. Phys.* 330 (2017), 693–734.
- Kevin Carlberg, Charbel Farhat, Julien Cortial, and David Amsallem. 2013. The GNAT method for nonlinear model reduction: effective implementation and application to computational fluid dynamics and turbulent flows. *J. Comput. Phys.* 242 (2013), 623–647.
- Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. 2022. Implicit Neural Spatial Representations for Time-dependent PDEs. *arXiv preprint arXiv:2210.00124* (2022).
- Peter Yichen Chen, Maurizio M. Chiaramonte, Eitan Grinspun, and Kevin Carlberg. 2023a. Model reduction for the material point method via an implicit neural representation of the deformation map. *J. Comput. Phys.* 478 (2023), 111908. <https://doi.org/10.1016/j.jcp.2023.111908>
- Peter Yichen Chen, Jinxu Xiang, Dong Heon Cho, Yue Chang, G A Pershing, Henrique Teles Maia, Maurizio M Chiaramonte, Kevin Thomas Carlberg, and Eitan Grinspun. 2023b. CROM: Continuous Reduced-Order Modeling of PDEs Using Implicit Neural Representations. In *The Eleventh International Conference on Learning Representations*. <https://openreview.net/forum?id=FUORz1tG8Og>

- Yimbo Chen, Sifei Liu, and Xiaolong Wang. 2021. Learning Continuous Image Representation With Local Implicit Image Function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8628–8638.
- Zhiqin Chen and Hao Zhang. 2019. Learning Implicit Fields for Generative Shape Modeling. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5932–5941. <https://doi.org/10.1109/CVPR.2019.00609>
- Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). arXiv:1511.07289 [cs.LG]
- William Falcon and The PyTorch Lightning team. 2019. *PyTorch Lightning*. <https://doi.org/10.5281/zenodo.3828935>
- Lawson Fulton, Vismay Modi, David Duvenaud, David I. W. Levin, and Alec Jacobson. 2019. Latent-space Dynamics for Reduced Deformable Simulation. *Computer Graphics Forum* (2019).
- Kenneth C Hall, Jeffrey P Thomas, and Earl H Dowell. 2000. Proper orthogonal decomposition technique for transonic unsteady aerodynamic flows. *AIAA Journal* 38, 10 (2000), 1853–1862.
- Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 201.
- Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Transactions on Graphics (TOG)* 37, 4, Article 60 (July 2018), 14 pages. <https://doi.org/10.1145/3197517.3201353>
- Doug L James, Jernej Barbič, and Dinesh K Pai. 2006. Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 987–995.
- Robert K Katzschmann, Maxime Thieffry, Olivier Goury, Alexandre Kruszewski, Thierry-Marie Guerra, Christian Duriez, and Daniela Rus. 2019. Dynamically closed-loop controlled soft robotic arm using a reduced order finite element model with state observer. In *2019 2nd IEEE international conference on soft robotics (RoboSoft)*. IEEE, 717–724.
- Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. 2019. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 59–70.
- Theodore Kim and John Delaney. 2013. Subspace fluid re-simulation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–9.
- Theodore Kim and Doug L. James. 2009. Skipping Steps in Deformable Simulation with Online Model Reduction. *ACM Transactions on Graphics (TOG)* 28, 5 (dec 2009), 1–9. <https://doi.org/10.1145/1618452.1618469>
- Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]
- Kookjin Lee and Kevin Carlberg. 2018. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *arXiv preprint arXiv:1812.08373* (2018).
- Thuan Lieu, Charbel Farhat, and Michel Lesoinne. 2006. Reduced-order fluid/structure modeling of a complete aircraft configuration. *Computer methods in applied mechanics and engineering* 195, 41–43 (2006), 5730–5742.
- Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. 2022. Learning Smooth Neural Functions via Lipschitz Regularization. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) (SIGGRAPH '22). Association for Computing Machinery, New York, NY, USA, Article 31, 13 pages. <https://doi.org/10.1145/3528233.3530713>
- Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Xiang Chen, Kun Zhou, and Yin Yang. 2020. NNWarp: Neural Network-Based Nonlinear Deformation. *IEEE Transactions on Visualization and Computer Graphics* 26, 4 (2020), 1745–1759. <https://doi.org/10.1109/TVCG.2018.2881451>
- Miles Macklin. 2022. Warp: A High-performance Python Framework for GPU Simulation and Graphics. <https://github.com/nvidia/warp>. NVIDIA GPU Technology Conference (GTC).
- Laura Mainini and Karen Willcox. 2015. Surrogate modeling approach to support real-time structural assessment and decision making. *AIAA Journal* 53, 6 (2015), 1612–1626.
- Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-Based Elastic Materials. *ACM Transactions on Graphics (TOG)* 30, 4, Article 72 (jul 2011), 8 pages. <https://doi.org/10.1145/2010324.1964967>
- Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4455–4465. <https://doi.org/10.1109/CVPR.2019.00459>
- Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *Commun. ACM* 65, 1 (dec 2021), 99–106. <https://doi.org/10.1145/3503250>
- Rajaditya Mukherjee, Xiaofeng Wu, and Huamin Wang. 2016. Incremental Deformation Subspace Reconstruction. *Comput. Graph. Forum* 35, 7 (oct 2016), 169–178.
- Shaowu Pan, Steven L. Brunton, and J. Nathan Kutz. 2023. Neural Implicit Flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data. *Journal of Machine Learning Research* 24, 41 (2023), 1–60. <http://jmlr.org/papers/v24/22-0365.html>
- Zherong Pan, Hujun Bao, and Jin Huang. 2015. Subspace Dynamic Simulation Using Rotation-Strain Coordinates. *ACM Transactions on Graphics (TOG)* 34, 6, Article 242 (nov 2015), 12 pages. <https://doi.org/10.1145/2816795.2818090>
- Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Alex Pentland and John Williams. 1989. Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. 215–222.
- Allan Pinkus. 2012. *N-widths in Approximation Theory*. Vol. 7. Springer Science & Business Media.
- Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.
- Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707. <https://doi.org/10.1016/j.jcp.2018.10.045>
- Cristian Romero, Dan Casas, Jesús Pérez, and Miguel Otaduy. 2021. Learning Contact Corrections for Handle-Based Subspace Dynamics. *ACM Transactions on Graphics (TOG)* 40, 4, Article 131 (jul 2021), 12 pages. <https://doi.org/10.1145/3450626.3459875>
- David Ryckelynck. 2005. A priori hyperreduction method: an adaptive approach. *J. Comput. Phys.* 202, 1 (2005), 346–366. <https://doi.org/10.1016/j.jcp.2004.07.015>
- Ahmed Shabana. 2012. *Vibration of Discrete and Continuous Systems*. Springer New York. <https://books.google.ca/books?id=VfTxwBwAAQBAJ>
- Tamar Rott Shaham, Michaël Gharbi, Richard Zhang, Eli Shechtman, and Tomer Michaeli. 2021. Spatially-Adaptive Pixelwise Networks for Fast Image Translation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 14877–14886. <https://doi.org/10.1109/CVPR46437.2021.01464>
- Nicholas Sharp, Cristian Romero, Alec Jacobson, Etienne Vouga, Paul G Kry, David IW Levin, and Justin Solomon. 2023. Data-Free Learning of Reduced-Order Kinematics. *arXiv preprint arXiv:2305.03846* (2023).
- Siyuan Shen, Yin Yang, Tianjia Shao, He Wang, Chenfanlu Jiang, Lei Lan, and Kun Zhou. 2021. High-Order Differentiable Autoencoder for Nonlinear Model Reduction. *ACM Transactions on Graphics (TOG)* 40, 4, Article 68 (jul 2021), 15 pages. <https://doi.org/10.1145/3450626.3459754>
- Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) (NIPS'20). Curran Associates Inc., Red Hook, NY, USA, Article 626, 12 pages.
- Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. 2021. Adversarial Generation of Continuous Images. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10748–10759. <https://doi.org/10.1109/CVPR46437.2021.01061>
- Breannan Smith, Fernando De Goes, and Theodore Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Transactions on Graphics (TOG)* 37, 2, Article 12 (mar 2018), 15 pages. <https://doi.org/10.1145/3180491>
- Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. 2015. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–11.
- Andrew M. Stuart and Antony R. Humphries. 1996. *Dynamical Systems And Numerical Analysis*.
- Qingyang Tan, Zherong Pan, Lin Gao, and Dinesh Manocha. 2020. Realtime simulation of thin-shell deformable materials using CNN-based mesh embedding. *IEEE Robotics and Automation Letters* 5, 2 (2020), 2325–2332.
- Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model reduction for real-time fluids. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 826–834.
- Christoph von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. 2013. An Efficient Construction of Reduced Deformable Objects. *ACM Transactions on Graphics (TOG)* 32, 6, Article 213 (nov 2013), 10 pages. <https://doi.org/10.1145/2508363.2508392>
- Steffen Wiewel, Moritz Becher, and Nils Thuerey. 2019. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 71–82.
- Karen Willcox and Jaime Peraire. 2002. Balanced model reduction via the proper orthogonal decomposition. *AIAA Journal* 40, 11 (2002), 2323–2330.
- Hongyi Xu and Jernej Barbič. 2016. Pose-Space Subspace Dynamics. *ACM Transactions on Graphics (TOG)* 35, 4, Article 35 (jul 2016), 14 pages. <https://doi.org/10.1145/2897824.2925916>
- Hongyi Xu, Yijing Li, Yong Chen, and Jernej Barbič. 2015. Interactive material design using model reduction. *ACM Transactions on Graphics (TOG)* 34, 2 (2015), 1–14.
- Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. 2021. Geometry Processing with Neural Fields. *Advances in Neural Information Processing Systems* 34 (2021).

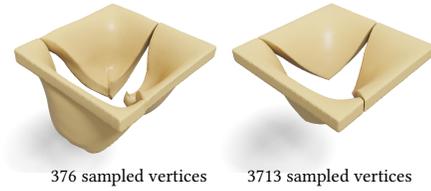
- Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. 2015. Expediting precomputation for reduced deformable simulation. *ACM Trans. Graph* 34, 6 (2015).
- Jonas Zehnder, Yue Li, Stelian Coros, and Bernhard Thomaszewski. 2021. Ntopo: Mesh-free topology optimization using implicit neural representations. *Advances in Neural Information Processing Systems* 34 (2021), 10368–10381.
- Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).



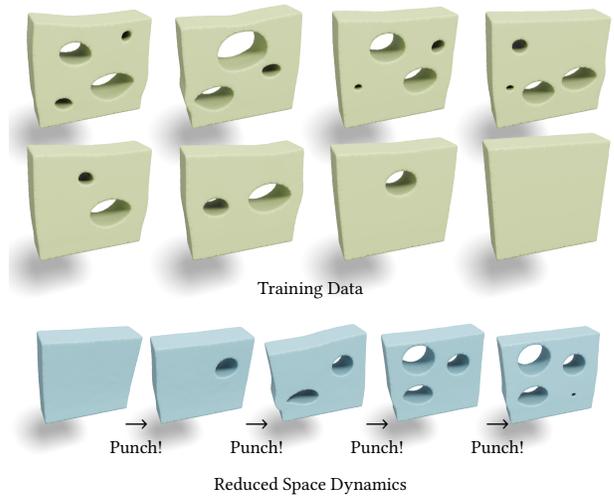
**Figure 7: One neural basis can span plausible deformations for new shapes.** During precomputation, we train one neural basis,  $(W, q)$ , with a single training set encompassing the full-space simulated deformations of five shapes spanning cube to sphere (blue). The training omits the test shapes (coral). During the online subspace dynamics, we simulate the test shapes with the same loading conditions, observing good general agreement, albeit with some missing surface details unseen during training.



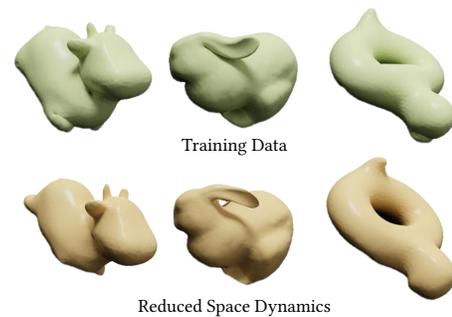
**Figure 8: Displacement basis visualization.** We visualize the first three dimensions of the continuous reduced basis  $W(X)$  over the reference domain  $\Omega$  for the tearing scenario (see Fig. 6). Red, green, and blue correspond to the displacements of  $W_1(X)$ ,  $W_2(X)$  and  $W_3(X)$ , respectively. Each  $W_i(X)$  is vector-valued: we visualize the  $x$ ,  $y$ , and  $z$  components of displacement in the left, middle, and right, respectively. It is evident that the basis includes displacement discontinuities, particularly along the  $xz$  plane of the plate.



**Figure 9: Comparison of cubature point density for the tearing scenario** (see Fig. 6), comparing 376 (left) versus 3713 (right) sampled vertices.



**Figure 10: Punching Cube.** We train this example using 5 meshes with different void. After we train a network with the 5 sequence, we are able to simulate the process of punching the cube and generate voids at runtime. In latent space dynamics, we apply remeshing whenever we "punch" a new void.



**Figure 11: Rollin' along.** We learn a subspace for the deformation induced by rolling of the bunny, the duck "bob", and the cow "spot" down an inclined plane. We then apply this subspace to simulate the deformations of myriad interpolated animal shapes.

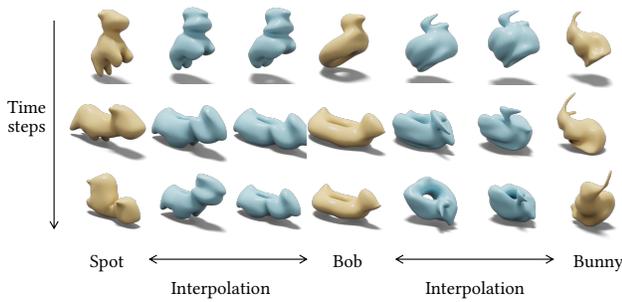


Figure 12: *Animation interpolation.* After training on the dynamics of a few meshes, our method can simulate the dynamics of the interpolated meshes.

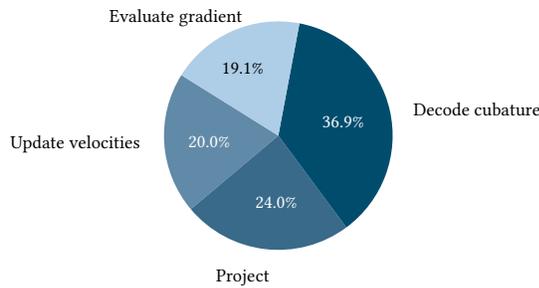


Figure 13: *Breakdown of computational cost for interactive manipulation* (see Fig. 3). In operation order: Over all cubature points, (a) update velocities by evaluating  $W(X)\dot{q}$ ; (b) decode cubature by evaluating  $W(X)q$ ; (c) evaluate the gradient (9). Finally, (d) project to the reduced space by least squares (10), solving the prefactored linear system using backsubstitution.

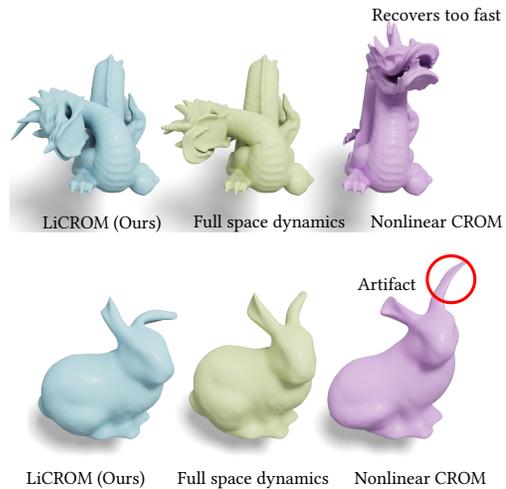


Figure 14: *Comparison with CROM.* From left to right: our method, ground truth full-space simulation, CROM. In the dragon’s example, we applied a force on the dragon’s head that is not included in training data. We found that CROM suffers from overfitting and looks less similar to ground truth compared with our method. In the bunny’s example, there are visible artifacts on the bunny’s ear using CROM, while our method generates a reasonable result.

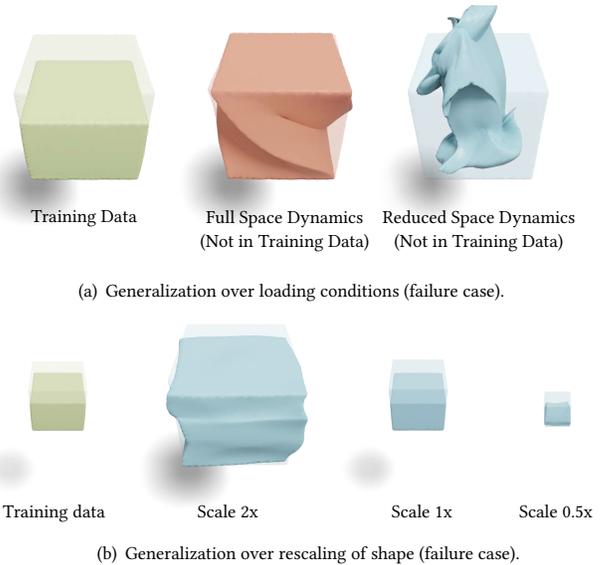


Figure 15: *Failures of generalization:* We tested the displacement field from Fig. 5 (training with different shapes). (a) We consider generalization over loading conditions. Training on a cube compressed by a vertical load and testing on a cube subject to rotational loads reveals poor agreement between full and reduced space dynamics. (b) We also consider generalization over spatial scale. Training on a unit cube with vertical load, and testing on cubes where both spatial extent and loading are scaled accordingly, we observe unexpected surface features, particularly when scaling up.