Model reduction for the material point method via learning the deformation map and its spatial-temporal gradients

Peter Yichen Chen\textsuperscript{a,*}, Maurizio Chiaramonte\textsuperscript{b}, Eitan Grinspun\textsuperscript{c,a}, Kevin Carlberg\textsuperscript{b}

\textsuperscript{a}Columbia University, 116th St & Broadway, New York, NY 10027, USA
\textsuperscript{b}Facebook Reality Labs Research, 9845 Willows Road, Redmond, WA 98052, USA
\textsuperscript{c}University of Toronto, 40 St. George Street, Room 4283, Toronto, ON M5S 2E4, Canada

Abstract

This work proposes a model-reduction approach for the material point method on nonlinear manifolds. The technique approximates the \textit{kinematics} by approximating the deformation map in a manner that restricts deformation trajectories to reside on a low-dimensional manifold expressed from the extrinsic view via a parameterization function. By explicitly approximating the deformation map and its spatial-temporal gradients, the deformation gradient and the velocity can be computed simply by differentiating the associated parameterization function. Unlike classical model reduction techniques that build a subspace for a finite number of degrees of freedom, the proposed method approximates the entire deformation map with infinite degrees of freedom. Therefore, the technique supports resolution changes in the reduced simulation, attaining the challenging task of zero-shot super-resolution by generating material points unseen in the training data. The ability to generate material points also allows for adaptive quadrature rules for stress update. A family of projection methods is devised to generate \textit{dynamics}, i.e., at every time step, the methods perform three steps: (1) generate quadratures in the full space from the reduced space, (2) compute position and velocity updates in the full space, and (3) perform a least-squares projection of the updated position and velocity onto the low-dimensional manifold and its tangent space. Computational speedup is achieved via hyper-reduction, i.e., only a subset of the original material points are needed for dynamics update. Large-scale numerical examples with millions of material points illustrate the method’s ability to gain an order-of-magnitude computational-cost saving—indeed \textit{real-time simulations} in some cases—with negligible errors.

\textbf{Keywords:} real-time, physics simulation, deep learning, material point method, model reduction, super-resolution

Highlights

\begin{itemize}
  \item A novel model-reduction technique for MPM is proposed
  \item Kinematics are approximated by learning the deformation map and its gradients
  \item The approximated deformation map supports super-resolution and adaptive quadratures
  \item Dynamics are generated from projections onto the approximated deformation map
  \item An order-of-magnitude speedup from the standard MPM is achieved
\end{itemize}

1. Introduction

Physics simulation plays a pivotal role in modern-day science and engineering with vast applications in physics, chemistry, material science, civil engineering, visual effects, virtual reality, etc. In virtually all
these applications, one finds a dilemma between accuracy and speed. To attain the desirable precision, a fine spatiotemporal resolution has to be adopted, leading to a higher computational cost. Yet, this computational burden would render the simulations inadequate for time-critical applications where real-time and even faster than real-time performance is required.

To address this dilemma, model reduction techniques have been developed to generate a low-dimensional, low-complexity computational model of the original expensive high-dimensional model. These reduced-order models are faster to compute while retaining a high accuracy. They have been successfully employed to solve real-world problems in many fields, such as motor-generator design, batch chromatography, aerodynamics, structural dynamics, computer graphics, and robotics.

This work proposes a novel model reduction method for the Material Point Method (MPM), a widely adopted continuum mechanics simulation framework. To the extent of our knowledge, this is the first time a model reduction technique has been developed for MPM.

MPM was introduced by Sulsky et al. as an extension of the Particle-In-Cell (PIC) method for solid mechanics. It is a hybrid Eulerian–Lagrangian discretization method widely employed in solid, fluid, and multiphase simulations. Due to its dual Eulerian and Lagrangian representations, MPM offers several advantages over the Finite Element Method: large deformation, fracture, as well as automatic contact and collision. Due to its dual representations of the material and the transfer between the two representations, MPM's dual representations may make it incredibly computationally costly. Recent advances in sparse data structure, compiler optimization, and multi-GPU have made phenomenal progress in alleviating the computational cost of MPM, leading to practical applications such as robot control and topology optimization. Yet, real-time, millions of particles MPM simulation on a single desktop remains a daunting challenge. We propose to address this challenge with model reduction techniques.

Model reduction for physics simulations dates back to Sirovich, who applied Principal Component Analysis (PCA) to turbulence simulations and coined the term Proper Orthogonal Decomposition (POD). Model reduction typically amounts to two stages: an offline stage and an online stage. In the offline stage, also known as the training stage, many expensive high-fidelity simulations are performed at different parameter instances within the parameter space from which the user intends to obtain simulation results. From the data of these expensive simulations, one can construct a low-dimensional subspace, also known as the generalized coordinates, that captures the principal modes of the original physical system. Afterward, during the inexpensive online stage, the user no longer needs to compute every single degree of freedom of the high-fidelity model. Instead, one can rapidly evolve the simulation state, i.e., the subspace, on which the model conducts time integration, allowing us to naturally define the initial conditions. ROM also facilitates physical interpretation because it maintains the notion of “system state,” i.e., the subspace, on which the model conducts time integration, allowing us to naturally define physical quantities such as subspace velocity via differentiating the subspace temporally. Recently, there have been rising interests in using graph neural networks to model physical systems and have gained increasing attention. In particular for advection-dominated problems, nonlinear manifolds, often constructed with deep neural networks, have been shown to outperform their linear counterparts significantly. Applicable to both linear and nonlinear manifolds, hyper-reduction techniques have been developed to solve the governing physical equations on a tiny subset of the original computation mesh to ensure computation efficiency.

Compared to non-physics-based, purely data-driven surrogate models, the reduced-order models (ROMs) derived from model reduction respect physics laws, including the enforcement of proper boundary conditions. ROM also facilitates physical interpretation because it maintains the notion of “system state,” i.e., the subspace, on which the model conducts time integration, allowing us to naturally define physical quantities such as subspace velocity via differentiating the subspace temporally. Recently, there have been rising interests in using graph neural networks to model physical systems and have gained increasing attention. In particular for advection-dominated problems, nonlinear manifolds, often constructed with deep neural networks, have been shown to outperform their linear counterparts significantly. Applicable to both linear and nonlinear manifolds, hyper-reduction techniques have been developed to solve the governing physical equations on a tiny subset of the original computation mesh to ensure computation efficiency.

Compared to non-physics-based, purely data-driven surrogate models, the reduced-order models (ROMs) derived from model reduction respect physics laws, including the enforcement of proper boundary conditions. ROM also facilitates physical interpretation because it maintains the notion of “system state,” i.e., the subspace, on which the model conducts time integration, allowing us to naturally define physical quantities such as subspace velocity via differentiating the subspace temporally. Recently, there have been rising interests in using graph neural networks to model physical systems and have gained increasing attention. In particular for advection-dominated problems, nonlinear manifolds, often constructed with deep neural networks, have been shown to outperform their linear counterparts significantly. Applicable to both linear and nonlinear manifolds, hyper-reduction techniques have been developed to solve the governing physical equations on a tiny subset of the original computation mesh to ensure computation efficiency.

Compared to non-physics-based, purely data-driven surrogate models, the reduced-order models (ROMs) derived from model reduction respect physics laws, including the enforcement of proper boundary conditions. ROM also facilitates physical interpretation because it maintains the notion of “system state,” i.e., the subspace, on which the model conducts time integration, allowing us to naturally define physical quantities such as subspace velocity via differentiating the subspace temporally. Recently, there have been rising interests in using graph neural networks to model physical systems and have gained increasing attention. In particular for advection-dominated problems, nonlinear manifolds, often constructed with deep neural networks, have been shown to outperform their linear counterparts significantly. Applicable to both linear and nonlinear manifolds, hyper-reduction techniques have been developed to solve the governing physical equations on a tiny subset of the original computation mesh to ensure computation efficiency.

Compared to non-physics-based, purely data-driven surrogate models, the reduced-order models (ROMs) derived from model reduction respect physics laws, including the enforcement of proper boundary conditions. ROM also facilitates physical interpretation because it maintains the notion of “system state,” i.e., the subspace, on which the model conducts time integration, allowing us to naturally define physical quantities such as subspace velocity via differentiating the subspace temporally. Recently, there have been rising interests in using graph neural networks to model physical systems and have gained increasing attention. In particular for advection-dominated problems, nonlinear manifolds, often constructed with deep neural networks, have been shown to outperform their linear counterparts significantly. Applicable to both linear and nonlinear manifolds, hyper-reduction techniques have been developed to solve the governing physical equations on a tiny subset of the original computation mesh to ensure computation efficiency.
Prior work on model reduction techniques for MPM is scarce if it exists at all. Literature on model reduction methods for other flavors of PIC methods is also limited. Nicolini et al. [77] applied POD to the PIC-based solver of the Maxwell-Vlasov equations. Wiewel et al. [109] used Convolutional Neural Networks (CNNs) to reduce the dimension of the Eulerian grid data of the FLuid Implicit Particle (FLIP) method and used a Long Short Term Memory (LSTM) networks to evolve the subspace.

To develop a model reduction framework for MPM, we first notice that while MPM possesses two representations of the underlying continuum, the Lagrangian representation is the core representation. In contrast, the Eulerian representation merely serves as a “computation scratchpad” for stress calculations. Thus, we focus on reducing the dimensions of the Lagrangian degrees of freedom.

To compute a subspace for the Lagrangian degrees of freedom, we propose a novel nonlinear kinematic approximation (Figure 1b) of the deformation map, the critical quantity in forming continuum mechanics’ governing equations.

![Figure 1: Our approach vs. the classical approach. In classical model reduction techniques, a mapping from the generalized coordinates $\tilde{x}$ is often trained to infer the deformed positions $x^p$ of a finite number $n^p$ of particles concatenated into a column vector. By contrast, our approach builds a manifold-parameterization function that maps $\tilde{x}$ and an arbitrary undeformed position $X$ to its deformed position $x$. Consequently, we can represent an infinite number of particle positions, i.e., the entire deformation map, using the finite-dimensional generalized coordinates $\tilde{x}$.](image)

Traditional model reduction approaches [5] learn a mapping from the generalized coordinates to the positions of a finite set of particles (Figure 1a). Consequently, the network is tied to a specific, pre-defined spatial discretization, preventing the network from querying any point outside the finite set. The network structure has to be modified for different resolutions and discretizations. By contrast, the proposed model reduction approach is mesh-independent by construction since it learns a mapping between the generalized coordinates to an infinite set of material points, i.e., the entire deformation space (Figure 1b). Consequently, the kinematics of arbitrary material points can be recovered from the subspace. This mesh-independence feature enables direct super-resolution of the proposed ROM.

After the subspace is constructed, we can generate temporal dynamics by projecting a subset of the original material point kinematics onto the manifold-parameterization function. Thanks to the kinematic approximation of the entire deformation space instead of a finite set of material points, we can complete stress evaluation of the material points in an adaptive quadrature manner. Therefore, we bypass the need to track each material point, which serves as quadrature points in the original MPM formulation, thereby obtaining massive dimension reduction.

Section 2 is devoted to recapping the necessary ingredients of MPM before we dive into the proposed model reduction in later sections.

2. Full-order model

This section describes the full-order model by first stating the continuous problem statement in §2.1 and later discretizing it using MPM in §2.2.

![Figure 1: Our approach vs. the classical approach. In classical model reduction techniques, a mapping from the generalized coordinates $\tilde{x}$ is often trained to infer the deformed positions $x^p$ of a finite number $n^p$ of particles concatenated into a column vector. By contrast, our approach builds a manifold-parameterization function that maps $\tilde{x}$ and an arbitrary undeformed position $X$ to its deformed position $x$. Consequently, we can represent an infinite number of particle positions, i.e., the entire deformation map, using the finite-dimensional generalized coordinates $\tilde{x}$.

![Figure 1: Our approach vs. the classical approach. In classical model reduction techniques, a mapping from the generalized coordinates $\tilde{x}$ is often trained to infer the deformed positions $x^p$ of a finite number $n^p$ of particles concatenated into a column vector. By contrast, our approach builds a manifold-parameterization function that maps $\tilde{x}$ and an arbitrary undeformed position $X$ to its deformed position $x$. Consequently, we can represent an infinite number of particle positions, i.e., the entire deformation map, using the finite-dimensional generalized coordinates $\tilde{x}$.

![Figure 1: Our approach vs. the classical approach. In classical model reduction techniques, a mapping from the generalized coordinates $\tilde{x}$ is often trained to infer the deformed positions $x^p$ of a finite number $n^p$ of particles concatenated into a column vector. By contrast, our approach builds a manifold-parameterization function that maps $\tilde{x}$ and an arbitrary undeformed position $X$ to its deformed position $x$. Consequently, we can represent an infinite number of particle positions, i.e., the entire deformation map, using the finite-dimensional generalized coordinates $\tilde{x}$.

![Figure 1: Our approach vs. the classical approach. In classical model reduction techniques, a mapping from the generalized coordinates $\tilde{x}$ is often trained to infer the deformed positions $x^p$ of a finite number $n^p$ of particles concatenated into a column vector. By contrast, our approach builds a manifold-parameterization function that maps $\tilde{x}$ and an arbitrary undeformed position $X$ to its deformed position $x$. Consequently, we can represent an infinite number of particle positions, i.e., the entire deformation map, using the finite-dimensional generalized coordinates $\tilde{x}$.

![Figure 1: Our approach vs. the classical approach. In classical model reduction techniques, a mapping from the generalized coordinates $\tilde{x}$ is often trained to infer the deformed positions $x^p$ of a finite number $n^p$ of particles concatenated into a column vector. By contrast, our approach builds a manifold-parameterization function that maps $\tilde{x}$ and an arbitrary undeformed position $X$ to its deformed position $x$. Consequently, we can represent an infinite number of particle positions, i.e., the entire deformation map, using the finite-dimensional generalized coordinates $\tilde{x}$.

![Figure 1: Our approach vs. the classical approach. In classical model reduction techniques, a mapping from the generalized coordinates $\tilde{x}$ is often trained to infer the deformed positions $x^p$ of a finite number $n^p$ of particles concatenated into a column vector. By contrast, our approach builds a manifold-parameterization function that maps $\tilde{x}$ and an arbitrary undeformed position $X$ to its deformed position $x$. Consequently, we can represent an infinite number of particle positions, i.e., the entire deformation map, using the finite-dimensional generalized coordinates $\tilde{x}$.
2.1. Continuous problem formulation

For a time window \( T := [t_0, t_T] \subseteq \mathbb{R} \) and for an instance of problem parameters \( \mu \in \mathcal{D} \subseteq \mathbb{R}^d \), we aim to compute the deformation of a nonlinear elastic body occupying—in its undeformed state—a domain \( \Omega_{\text{ref}} \subseteq \mathbb{R}^d \), \( d \in \{2, 3\} \). We define the deformation map \( \phi : \Omega_{\text{ref}} \times T \times \mathcal{D} \rightarrow \mathbb{R}^d \), as the mapping from any point on the undeformed domain \( \Omega_{\text{ref}} \) to the corresponding point on the deformed domain \( \Omega(t, \mu) \) at a given time instance \( t \in T \) and parameter instance \( \mu \in \mathcal{D} \) such that

\[
\phi(t_0, t, \mu) : \mathbf{X} \mapsto \mathbf{x}(t, \mu)
\]

\[
: \Omega_{\text{ref}} \rightarrow \Omega(t, \mu) \subseteq \mathbb{R}^d.
\]

In what follows, we omit explicit dependence of variables on problem parameters \( \mu \in \mathcal{D} \) for simplicity of exposition.

To enforce Dirichlet boundary conditions, we restrict the deformation map \( \phi \) to reside in the space of admissible trajectories \( \mathcal{S} \) such that \( \phi \in \mathcal{S} \), where

\[
\mathcal{S} := \{ \psi : \Omega_{\text{ref}} \times T \rightarrow \mathbb{R}^d | \psi(\mathbf{X}, t) = \mathbf{\phi}(\mathbf{X}, t), \forall \mathbf{X} \in \partial D_{\text{ref}}(t), \forall t \in T; \psi(\mathbf{X}, 0) = \mathbf{X} \text{ and } \psi(\mathbf{X}, 0) = \mathbf{V}(\mathbf{X}), \forall \mathbf{X} \in \Omega_{\text{ref}} \}.
\]

Here, \( (\cdot) \) denotes differentiation with respect to time for a fixed position on the reference domain \( \mathbf{X} \in \Omega_{\text{ref}} \), also known as the material time derivative, \( \partial D_{\text{ref}}(t) \subseteq \partial \Omega_{\text{ref}} \) denotes the portion of the reference-domain boundary \( \partial \Omega_{\text{ref}} \) with prescribed Dirichlet boundary conditions at time instance \( t \in T \), \( \mathbf{\phi}(\cdot, t) : \partial D_{\text{ref}}(t) \rightarrow \mathbb{R}^d \) denotes the prescribed boundary displacements at time instance \( t \in T \), and \( \mathbf{V} : \Omega_{\text{ref}} \rightarrow \mathbb{R}^d \) denotes the prescribed initial velocity.

The continuous-problem formulation for the balance of momentum in an Eulerian frame of reference is: find the deformation map \( \phi \in \mathcal{S} \) such that for all \( t \in T \),

\[
\rho \ddot{\mathbf{v}} = \nabla_{\mathbf{X}} \cdot \mathbf{\sigma} + \mathbf{b}, \forall \mathbf{X} \in \Omega(t),
\]

\[
\mathbf{b} = \mathbf{\bar{I}}, \forall \mathbf{X} \in \partial N_{\Omega}(t).
\]

Here, \( \mathbf{v} : (\mathbf{x}, t) \mapsto \dot{\mathbf{\phi}}(\mathbf{\phi}^{-1}(\mathbf{x}, t); t) \) denotes the Eulerian velocity with \( \mathbf{v} : \Omega \times T \rightarrow \mathbb{R}^d; \mathbf{\sigma} \) denotes the Cauchy stress tensor; assuming hyperelasticity, the Cauchy stress tensor is strictly a function of the deformation gradient, i.e., \( \mathbf{\sigma} \equiv \mathbf{\sigma}(\mathbf{F}) \) with \( \mathbf{F} : (\mathbf{X}, t) \mapsto \nabla \mathbf{\phi}(\mathbf{X}, t); \mathbf{b} : \Omega \rightarrow \mathbb{R}^d \) denotes body forces; \( \rho : \Omega \rightarrow \mathbb{R}_+ \) denotes the material density; and \( \mathbf{\bar{I}} : \partial N_{\Omega} \rightarrow \mathbb{R}^d \) denotes the prescribed tractions. Here, \( \partial \Omega_{\text{ref}} \) denotes the boundary of the domain and \( \partial N_{\Omega} \) and \( \partial D_{\text{ref}} \) denote the portion of the boundary \( \partial \Omega \) with prescribed Neumann and Dirichlet boundary conditions, respectively, such that \( \partial \Omega = \partial N_{\Omega} \cup \partial D_{\text{ref}} \) and \( \partial N_{\Omega} \cap \partial D_{\text{ref}} = \emptyset \).

Assuming sufficient regularity, an equivalent weak formulation of the above problem statement is: find the deformation map \( \phi \in \mathcal{S} \) such that

\[
\int_{\Omega} \rho \ddot{\mathbf{v}} \cdot \mathbf{\eta} dV = \int_{\Omega} (\mathbf{b} \cdot \mathbf{\eta} - \mathbf{\sigma} : \nabla \mathbf{\eta}) dV + \int_{\partial \Omega_{\text{ref}}} \mathbf{\bar{I}} \cdot \mathbf{\eta} ds, \forall \mathbf{\eta} \in \mathcal{T}_t \mathcal{S}, \forall t \in T,
\]

where \( \mathcal{T}_t \mathcal{S} \) denotes the space of admissible test functions at time \( t \) defined as

\[
\mathcal{T}_t \mathcal{S} := \{ \mathbf{\zeta} : \Omega \rightarrow \mathbb{R}^d | \mathbf{\zeta}(\mathbf{x}) = 0, \forall \mathbf{x} \in \partial D_{\text{ref}}(t) \}.
\]

The above can be recast in mass-integral form using \( dm = \rho dV \) as

\[
\int_{\Omega} \dot{\mathbf{v}} \cdot \mathbf{\eta} dm = \int_{\Omega} \frac{\mathbf{J}}{\rho_0} (\mathbf{b} \cdot \mathbf{\eta} - \mathbf{\sigma} : \nabla \mathbf{\eta}) dm + \int_{\partial \Omega_{\text{ref}}} \mathbf{\bar{I}} \cdot \mathbf{\eta} ds, \forall \mathbf{\eta} \in \mathcal{T}_t \mathcal{S}, \forall t \in T,
\]

where \( \mathbf{J} := \det(\mathbf{F}) \) and \( \rho_0 := \rho(\mathbf{x}, 0) \).

2.2. MPM discretization

Equation (7) is the starting point for applying the MPM discretization. The two major ingredients of MPM are:
1. The discretization of the domain by a collection of \( n_p \) material points with masses \( m_p \), \( p = 1, \ldots, n_p \) and reference-configuration positions \( X_p \in \Omega_{\text{ref}}, \ p = 1, \ldots, n_p \). These material points serve as quadrature points and allow for an approximation of the mass domain integrals appearing in Eq. (7) by
\[
\int_{\Omega} (\bullet) \, dm \approx \sum_{p=1}^{n_p} (\bullet) m_p. \tag{8}
\]

2. The approximation of the velocity \( v \), at a fixed time instance, by \( v_h \in S^h_t \) with \( S^h_t = \{ v_h \in S_t | v_h = \sum_{n_b} v_i N_i(x) \} \) and \( S_t := \{ \zeta : \Omega \to \mathbb{R}^d | \zeta(x) = \dot{\phi}(X, t), \forall x \in \partial D \Omega(t) \} \), as well as of the test function \( \eta \) by \( \eta_h \in T^h_t S \) with \( T^h_t S = \{ \eta_h \in T^h_t S | \eta_h = \sum_{n_b} \eta_i N_i(x) \} \). Here, \( N_i : \Omega \to \mathbb{R}, i = 1, \ldots, n_b \) denote basis functions constructed on an Eulerian-domain discretization, i.e., the background grid.

For simplicity of exposition and without loss of generality, we assume \( t = 0 \) in the sequel.

Under mass lumping, the left-hand side of Eq. (9) is approximated as
\[
\sum_{p=1}^{n_p} \sum_{j} \dot{v}_j N_j N_i \big|_{x^p} m^p = \sum_{p=1}^{n_p} \frac{1}{\rho_0} [J (bN_i - \sigma(F) \nabla N_i)] \big|_{x^p} m^p + \int_{\partial D \Omega} \ddot{N}_i, \ i = 1, \ldots, n_b. \tag{9}
\]

For simplicity of exposition and without loss of generality, we assume \( \ddot{I} = 0 \) in the sequel.

Under mass lumping, the left-hand side of Eq. (9) is approximated as
\[
\sum_{p=1}^{n_p} \sum_{j} \dot{v}_j N_j N_i \big|_{x^p} m^p = \sum_{p=1}^{n_p} \sum_{j} (N_j N_i) \big|_{x^p} m^p \dot{v}_j = \sum_{j} M_{ij} \dot{v}_j \approx m_i \ddot{v}_i, \ i = 1, \ldots, n_b, \tag{10}
\]
where \( M_{ij} := (\sum_{p=1}^{n_p} N_j N_i) \big|_{x^p} m^p \) and \( m_i = \sum_{j}^{n_b} M_{ij} \).

Given the spatial discretization (9)–(10), we introduce a time discretization \( t_n, n = 0, \ldots, n_t \) with \( T \equiv [t_0, t_{n_t}] \) and \( \Delta t_n := t_n - t_{n-1}, n = 1, \ldots, n_t, \) where a subscript \( n \) denotes a quantity defined at time step \( n \). We now have enough ingredients to devise an explicit time-integration scheme; Algorithm 1 reports the resulting algorithm. We note that the deformation gradient update is derived from the expression
\[
\dot{F} = \nabla x v F,
\]
which yields a time-discretized update as $F_{n+1} = (1 + \nabla_x v_n \Delta t_n) F_n$.

**Algorithm 1: MPM Algorithm**

**Input:** Deformation gradient $F_n^p$, velocity $v_n^p$, and position $x_n^p$ for each material point $p = 1, \ldots, n^p$ at time instance $t_n$

**Output:** Deformation gradient $F_{n+1}^p$, velocity $v_{n+1}^p$, and position $x_{n+1}^p$, $p = 1, \ldots, n^p$ at time instance $t_{n+1}$

1. Transfer Lagrangian kinematics to the Eulerian grid by performing a ‘particle to grid’ transfer:

   Compute for $i = 1, \ldots, n^b$
   \[
   m_{i,n} = \sum_{p=1}^{n^p} N_i(x_n^p) m_p^p \\
   m_{i,n} v_{i,n} = \sum_{p=1}^{n^p} N_i(x_n^p) m_p^p v_p^p \\
   f_{i,n}^\sigma = -\sum_{p=1}^{n^p} \frac{J(F_n^p)}{\rho_0} \sigma(F_n^p) \nabla_N N_i(x_n^p) m_p^p \\
   f_{i,n}^e = \sum_{p=1}^{n^p} \frac{J(F_n^p)}{\rho_0} b(x_n^p) N_i(x_n^p) m_p^p
   \]

2. Solve Eulerian governing equations by computing for $i = 1, \ldots, n^b$

   \[
   \dot{v}_{i,n+1} = \frac{1}{m_{i,n}} (f_{i,n}^\sigma + f_{i,n}^e) \\
   \Delta v_{i,n+1} = \dot{v}_{i,n+1} \Delta t_n \\
   v_{i,n+1} = v_{i,n} + \Delta v_{i,n+1}
   \]

3. Update the Lagrangian velocity and deformation gradient by performing a ‘grid to particle’ transfer: Compute for $p = 1, \ldots, n^p$

   \[
   v_{n+1}^p = \sum_{i} N_i(x_n^p) v_{i,n+1} \\
   F_{n+1}^p = (1 + \sum_{i} v_{i,n+1} \otimes \nabla_N N_i(x_n^p) \Delta t_n) F_n^p
   \]

4. Update Lagrangian positions for $p = 1, \ldots, n^p$

   \[
   x_{n+1}^p = x_n^p + \Delta t v_{n+1}^p
   \]

Although we assume a PIC-style ‘particle to grid’ transfer, other transfer schemes can also be used \[55, 34\].

3. Reduced-order model

We now introduce a strategy for projection-based model reduction that relies on constructing a nonlinear approximation to the deformation map and a family of projection strategies.

3.1. Kinematics: low-dimensional manifold

In analogy to constructing low-dimensional nonlinear manifolds for finite-dimensional dynamical system state spaces \[52\], one can construct a nonlinear manifold that restricts any element of the reference domain $\Omega_{\text{ref}}$ to evolve on a low-dimensional manifold. We first denote the approximated deformation
Further, Eq. (14) implies that the velocity of the approximate solution can be calculated as differentiating the parameterization function \( g \) to match the prescribed ones during the reduced-order simulation.

\[
\hat{\phi}(X; t, \mu) = g(X; \hat{x}(t, \mu)), \quad \forall X \in \Omega_{\text{ref}}, \; \forall t \in T, \; \mu \in D.
\]

Eq. (14) implies that the deformation gradient of the approximate solution can be calculated from differentiating the parameterization function \( g \) as

\[
\hat{F} : (X, t, \mu) \mapsto \nabla \hat{\phi}(X; t, \mu) \equiv \nabla g(X; \hat{x}(t, \mu))
\]

Further, Eq. (14) implies that the velocity of the approximate solution can be calculated as

\[
\hat{v}(X; t, \mu) = \frac{\partial g}{\partial \mu}(X; \hat{x}(t, \mu))\hat{x}(t, \mu), \quad \forall X \in \Omega_{\text{ref}}, \; \forall t \in T, \; \mu \in D.
\]

where \( \hat{x}(t, \mu) = \frac{\partial \phi}{\partial \mu}(t, \mu) \) denotes the generalized velocity.

Remark (Recover kinematics of any material point). We emphasize that—because this approach approximates the entire deformation map—given the value of the generalized coordinate \( \hat{x}(t, \mu) \), the displacement, the deformation gradient, and the velocity for any element of the reference domain \( X \in \Omega_{\text{ref}} \) can be computed via Eqs. (14), (15), and (16), respectively. Consequently, our approach supports straightforward adaptive quadratures refinement as well as super-resolution.
3.2. Dynamics

We time-step the generalized coordinate dynamics in three steps (Figure 2): gathering necessary full-space quadrature point information (Section 3.2.1), evolution in the full space (Section 3.2.2), and projection onto the reduced space (Section 3.2.3).

Note that only a subset of the material points are necessary to uniquely determine the generalized coordinate updates since $\hat{d} \ll n_p$ where $\hat{d}$ is the dimension of $\hat{x}$. Therefore, we can compute the dynamics on only a subset of the material points, thereby reducing the computation cost.

Algorithm 2 presents the complete reduced-order dynamics algorithm. The goal is to time-step the generalized coordinate from which the full coordinates can be computed as needed, e.g., for rendering purposes.

**Algorithm 2: Reduced-order dynamics**

**Input:** Generalized velocity $\hat{v}_n$, generalized coordinate $\hat{x}_n$.

**Output:** Generalized velocity $\hat{v}_{n+1}$, generalized coordinate $\hat{x}_{n+1}$.

1. Obtain full-space quadrature point kinematics via either Algorithm 3 or Algorithm 4.
2. Evolve the full space dynamics to get $v_{n+1}^{trial}$ and $x_{n+1}^{trial}$ where $p \in \mathcal{M}$ using Algorithm 5.
3. Project $v_{n+1}^{trial}$ and $x_{n+1}^{trial}$ onto the reduced space via either Algorithm 6 or Algorithm 7.

3.2.1. Obtain full-space quadrature point kinematics

Suppose we want to compute the dynamics of a set of sample material points comprising a subset of the original $n_p$ material points. We denote these by a discrete set $\mathcal{M} \subseteq \{1, \ldots, n_p\}$. We defer the discussion on the choice of these material points to Section 3.3.

To evolve the dynamics of these material points, quadrature points are necessary for conducting the ‘particle to grid’ and the ‘grid to particle’ operations as it is in the original MPM algorithm. We propose two techniques for choosing the quadrature points necessary for stress update.
3.2.1.1. Quadrature points via Lagrangian material points.

In the first approach, we adopt an approach most similar to the original MPM algorithm where we use the neighboring material points as quadratures (Figure 3 left). Formally, we consider the neighbors of the sample material points \( N \subseteq \{1, \ldots, n^p\} \), defined as the set of material points that influence the dynamics of the sample material points on the Eulerian grid for all \( t \in T \) and \( \mu \in D \); note that \( \mathcal{M} \cap N = \emptyset \).

Algorithm 3 provides the algorithm identifying these quadrature points as well as obtaining their kinematic information from the approximated deformation map.

**Algorithm 3:** Lagrangian quadrature kinematics (via tracking material points)

**Input:** Generalized velocity \( \hat{v}_n \), generalized coordinate \( \hat{x}_n \).

**Output:** Quadrature point set \( \mathcal{P} \) and quadrature points information \( m^q, v^q, x^q, F^q \) for \( q \in \mathcal{P} \)

1. Compute the position \( x^q_p \) for each sample material point \( p \in \mathcal{M} \) at time instance \( t_n \) by evaluating (14) for \( X = X^p, p \in \mathcal{M} \) and \( t = t_n \).
2. Identify the grid nodes \( I \subseteq \{1, \ldots, n^b\} \) needed to compute dynamics for the sample material points, i.e., \( I = \{i \mid N_i(x^q_p) \neq 0 \text{ for any } p \in \mathcal{M}\} \).
3. Identify the neighbor material points set \( \mathcal{N} \) where for all \( p \in \mathcal{N} \) there exists \( i \in I \) such that \( N_i(g(X^p; \hat{x}_n)) \neq 0 \).
4. Compute the deformation gradient \( F^q_p \) and the velocity \( v^q_p \) for each sample and neighbor material point at time instance \( t_n \) by evaluating (15) and (16) for \( X = X^p, p \in \mathcal{M} \cup \mathcal{N} \), and \( t = t_n \).
5. Set the quadrature point set to be \( \mathcal{P} = \mathcal{M} \cup \mathcal{N} \).

3.2.1.2. Quadrature points via Eulerian finite elements.

While the previous approach can incur an operation count independent of the original number of material points \( n^p \) and Eulerian basis functions \( n^b \), it requires computing (and tracking) the set of neighboring material points. In the worst case, tracking would result in computation on the order of \( O(n^p) \).

This section presents an alternative that does not require the tracking of every material point. Recall in MPM, each material point (Equation (8)) serves two purposes: (1) a Lagrangian tracker of the shape of the material; (2) a quadrature for stress update Equation (9). While the sample material points \( p \in \mathcal{M} \) alone can serve as Lagrangian trackers, any stress update would involve the neighboring material points as quadratures. To avoid the costly tracking of these neighboring material points, we generate Eulerian quadratures instead (Figure 3 right). These quadratures discretize the space instead of the material. \( l \) quadrature points are generated per dimension and per background grid cell. Therefore, for each quadrature point, we have its current position \( x^q \) and its volume \( V^q = \frac{1}{l} V^c \). \( V^c \) is the volume of the grid cell. In practice, we choose \( l = 2 \).

To compute the Eulerian quadrature point’s undeformed position, we can invert the approximated deformation by solving for \( X^q \), such that \( g(X^q; \hat{x}) = x^q \). Other kinematic quantities can then be computed using Equation (15) and Equation (16). The mass \( m^q \) of each quadrature point can be computed as \( m^q = \frac{\rho_0}{J^n} V^q \).
With these Eulerian quadrature points, instead of Equation (9), we can discretize the weak form (7) as

\[ \sum_{q=1}^{n^q} \left( \sum_{j} \mathbf{\dot{v}}_j N_j(x_q) \right) m^q = \sum_{q=1}^{n^q} \frac{1}{\rho_0} \left| J(x_q) \right| \mathbf{f}_q m^q + \int_{\partial\Omega} \mathbf{t} \mathbf{N}_i, \quad i = 1, \ldots, n^b. \]  

(20)

where \( n^q \) is the number of the Eulerian quadrature points.

Algorithm 4 presents the associated algorithm. In contrast to the first approach, this approach does not require tracking any neighboring material points. Instead, leveraging the invertibility of the deformation map, we can generate quadrature points necessary for updating the dynamics on the related grid nodes \( I \) only. Consequently, the approach can incur an operation count independent of the original number of material points \( n^p \) and Eulerian basis functions \( n^b \) without any additional tracking.

Algorithm 4: Eulerian quadrature kinematics

\begin{itemize}
\item **Input:** Generalized velocity \( \mathbf{\dot{v}}_n \), generalized coordinate \( \mathbf{\dot{x}}_n \).
\item **Output:** Quadrature point set \( \mathcal{P} \) and quadrature points information \( m^q, v^q, x^q, F^q \) for \( q \in \mathcal{P} \).
\item 1 Compute the position \( x^q_p \) for each sample material point \( p \in \mathcal{M} \) at time instance \( t_n \) by evaluating (14) for \( X = x^q_p, p \in \mathcal{M} \) and \( t = t_n \).
\item 2 Identify the grid nodes \( I = \{ i | N_i(x^q_p) \neq 0 \text{ for any } p \in \mathcal{M} \} \) needed to compute dynamics for the sample material points, i.e., \( I_g = \{ \phi | \phi(x^q_p) \neq 0 \text{ for any } p \in \mathcal{M} \} \).
\item 3 Define a quadrature rule comprising quadrature points and their volumes \( x^q \in \Omega, v^q \in \mathbb{R}^+, q \in \mathcal{O} = \{ 1, \ldots, n^q \} \), used to assemble the governing equations at the sample nodes \( I \).
\item Compute the undeformed positions of the quadrature points \( X^q \) by solving \( g(X^q ; \mathbf{\dot{x}}_n) = x^q \).
\item 4 Compute the deformation gradient \( F^q \) and the velocity \( v^q \) for each quadrature point at time instance \( t_n \) by evaluating (15) and (16) for \( X = X^q \) and \( t = t_n \).
\item 5 Compute the mass \( m^q \) for each quadrature point, \( m^q = \frac{\rho_0}{\mathcal{V}} \mathcal{V}^q \) where \( \mathcal{V}^q = \text{det}(F^q) \) is the determinant of the deformation gradient.
\item 6 Set the quadrature point set to be \( \mathcal{P} = \mathcal{O} \).
\end{itemize}

**Remark.** In addition to reducing the computational cost, the ability to generate arbitrary quadratures also enables adaptive refinement, which can be instrumental when the Jacobian \( J \) of the deformation gradient gets very large, and the fixed material point approach runs into under-integration problems.

3.2.2. Evolution on the full space

This section aims to compute the trial positions for the material points belonging to the subset \( p \in \mathcal{M} \) at \( t^{n+1} \). We deem these positions trial because they do not necessarily live on the low-dimensional manifold. Algorithm 5 presents the MPM-style stress update that works for both the Lagrangian quadratures and the Eulerian quadratures. Note that, compared with the original MPM algorithm (Algorithm 1), the stress update presented here no longer needs to evolve the deformation gradient explicitly. The deformation gradient is readily available from the spatial gradient of the approximated deformation.
**Algorithm 5: MPM-style full space dynamics**

**Input:** Quadrature points information $m^q$, $v^q$, $x^q$, $F^q_n$ for $q \in P$

**Output:** Full velocities $v^p_{n+1}^{trial}$ and full coordinates $x^p_{n+1}^{trial}$ for $p \in M$.

1. Perform the ‘particle to grid’ transfer by computing for $i \in I$

   
   \[
   m_{i,n} = \sum_{q \in P} N_i(x^q_n)m^q
   \]

   \[
   m_{i,n}v_{i,n} = \sum_{q \in P} N_i(x^q_n)m^qv^q_n
   \]

   \[
   f^\sigma_{i,n} = -\sum_{q \in P} \frac{J(F^q_n)}{\rho_0} \sigma(F^q_n)\nabla_x N_i(x^q_n)m^q
   \]

   \[
   f^e_{i,n} = \sum_{q \in P} \frac{J(F^q_n)}{\rho_0} b(x^q_n)N_i(x^q_n)m^q.
   \]

2. Perform the update step by computing for $i \in I$

   \[
   \dot{v}_i = \frac{1}{m_i}(f^\sigma_{i,n} + f^e_{i,n})
   \]

   \[
   \Delta v_i = v_i \Delta t
   \]

   \[
   v_{i,n+1} = v_{i,n} + \Delta v_i.
   \]

3. Perform the ‘grid to particle’ transfer by computing for $p \in M$

   \[
   v^p_{n+1}^{trial} = \sum_{i \in I} N_i(x^p_n)v_{i,n+1}
   \]

4. Update Lagrangian positions for $p \in M$

   \[
   x^p_{n+1}^{trial} = x^p_n + \Delta t v^p_{n+1}^{trial}
   \]

### 3.2.3. Project onto the reduced space

This section proposes two approaches to project the newly computed material point information onto the low-dimensional manifold and thereby effectively update the generalized coordinate.

Algorithm 6 presents the complete nonlinear projection algorithm where a least-square type of projection in the $l_2$-norm is conducted both on velocities and positions.

**Algorithm 6: Project the full space velocities and positions onto the reduced space**

**Input:** Full velocities $v^p_{n+1}^{trial}$ and full coordinates $x^p_{n+1}^{trial}$ for $p \in M$.

**Output:** Generalized velocity $\tilde{v}_{n+1}$ and generalized coordinate $\tilde{x}_{n+1}$.

1. $\tilde{v}_{n+1}$ and $\tilde{x}_{n+1}$, which should satisfy the minimization problem

   \[
   \tilde{v}_{n+1} \in \arg \min_{\tilde{v} \in \mathbb{R}^d} \sum_{p \in M} \| \frac{\partial g}{\partial x}(X^p; \tilde{x})\tilde{v} - v^p_{n+1}^{trial} \|^2_2. \tag{21}
   \]

   \[
   \tilde{x}_{n+1} \in \arg \min_{\tilde{x} \in \mathbb{R}^d} \sum_{p \in M} \| g(X^p; \tilde{x}) - x^p_{n+1}^{trial} \|^2_2. \tag{22}
   \]

   Since the discretization of the Jacobian term, $\frac{\partial g}{\partial x}$, is explicit, Equation (21) is a linear least square problem which can be resolved in the closed-form. In contrast, Equation (22) is a nonlinear least-square problem. We solve it using the Gauss-Newton method [78] with backtracking line search. We use a simple linear interpolation of the previous generalized coordinates as the initial guess, $\tilde{x}^{guess} = 2\tilde{x}_n - \tilde{x}(t_{n-1})$, and the solver typically converges in 2-3 iterations.
To reduce computational costs, we can linearize the nonlinear solve as follows. See detailed derivation in Appendix A.

Algorithm 7: Project the full space velocities and positions onto the reduced space

**Input:** Full coordinates $v_{n+1}^{\text{trial}}$ for $p \in \mathcal{M}$ and generalized coordinate $\hat{x}_n$.

**Output:** Generalized velocity $\hat{v}_{n+1}$ and generalized coordinate $\hat{x}_{n+1}$.

1. $\hat{x}_{n+1} = \hat{x}_n + \Delta t \hat{v}_{n+1}$, where $\hat{v}_{n+1}$ satisfies the minimization problem

$$\hat{v}_{n+1} \in \arg \min_{\hat{v} \in \mathbb{R}^d} \sum_{p \in \mathcal{M}} \left\| \frac{\partial g}{\partial \hat{x}}(X^p; \hat{x}_n) \hat{v} - v_{n+1}^{\text{trial}} \right\|_2^2.$$  

3.3. Hyper-reduction

The key to obtaining a practical speedup is choosing the sample material point set $\mathcal{M}$, whose dynamics are computed at every time step and used to update the generalized coordinate. If $\mathcal{M}$ is chosen to be all the original material points, i.e., no hyper-reduction, the reduced method offers no speedup. Since the dimension of the generalized coordinate is significantly smaller than the number of original material points, $\hat{d} << n^p$, if we choose $|\mathcal{M}| = n^p$, we will be solving an overdetermined system during projection (Algorithm 6 and 7). Therefore, we can, in fact, choose $|\mathcal{M}| << n^p$ while still obtaining a reasonable estimate of the optimal $\hat{x}$ and $\hat{v}$.

While more advanced methods for choosing $\mathcal{M}$ exist, we adopt a straightforward stochastic sampling scheme for its simplicity. Before starting the simulation, the size of the sample material point set is chosen to be $|\mathcal{M}|$. $\mathcal{M}$ is constructed by randomly sampling $|\mathcal{M}|$ points from the $n^p$ material points. We also re-sample at every time step to ensure good coverage of the entire domain. If kinematic boundaries exist, special attention is given to them by ensuring that some of the material points from these boundaries are included in $\mathcal{M}$. In practice, we sample at least five material points from each kinematic boundary. Figure 4 displays an example of the sample set $\mathcal{M}$.

![Figure 4: The sample material points $\mathcal{M}$ is shown on the right. Note that both the top and the bottom kinematic boundaries are sampled.](image-url)
4. Manifold-parameterization construction

\[
\tilde{x}(t_n, \mu) = \begin{pmatrix} x^1 \\ \vdots \\ x^n_p \end{pmatrix}
\]

Encoder function \(e\)

Manifold parameterization function \(g\)

\[
g(\tilde{x}, \tilde{t}_n, \mu) \approx x^p
\]

\[
\frac{\partial g}{\partial X}(\tilde{x}, \tilde{t}_n, \mu) \approx F^p
\]

\[
\frac{\partial g}{\partial \tilde{x}} X^p, \tilde{t}_n, \mu) \approx v^p
\]

Figure 5: The manifold parameterization function \(g\) is constructed via a neural network. With a continuously differentiable activation function, we also obtain the approximated deformation gradient and the approximated velocity. An encoder network \(e\) is used for generating \(\hat{x}\) from the simulation snapshot.

While in principle, the manifold-parameterization function \(g : \Omega_{\text{ref}} \times \mathbb{R}^d \rightarrow \mathbb{R}^d\) can be constructed in various ways, we will employ a fully-connected deep-learning architecture (i.e., a multilayer perceptron) for this purpose (Figure 5). We adopt ELU as the activation function for its continuous differentiability. The input to \(g\) is the undeformed position of a material point \(X^p\) and \(\tilde{x}\), which is shared among all material points at time \(t_n\) and has a dimension of \(\tilde{d}\). The output of \(g\) is the approximated deformed position of the material point. Thanks to the network’s continuous differentiability, we also obtain the approximated deformation gradient and approximated velocity. Our goal is to train the network such that these approximated quantities match the actual quantities, i.e., \(x^p\), \(F^p\), and \(v^p\). The detailed network structure of \(g\) is listed in Table 1.
Encoder network $e$

| Layer | Kernel size | Stride size |
|-------|-------------|-------------|
| 1     | 6           | 2           |
| $\cdots$ | 6           | 2           |
| $n_{conv}$ | 6           | 2           |

Fully-connected layers

| Layer | Input dimension | Output dimension |
|-------|-----------------|------------------|
| $n_{conv} + 1$ | $d_{conv}$ | 32 |
| $n_{conv} + 2$ | 32 | $\hat{d}$ |

Manifold-parameterization function $g$

| Layer | Input dimension | Output dimension |
|-------|-----------------|------------------|
| 1     | $d + \hat{d}$  | 30               |
| 2     | 30              | 30               |
| 3     | 30              | 30               |
| 4     | 30              | 30               |
| 5     | 30              | $\hat{d}$       |

Table 1: For the manifold-parameterization function $g$, we adopt a lightweight network by using 5 fully connected hidden layers, each with a size of 30, where $d \in \{2, 3\}$ and $\hat{d}$ is the dimension of the generalized coordinate. For the encoder function $e$, since the concatenated input vector can be arbitrarily large depending on the size of the simulations, we avoid extensive usages of fully connected layers. Instead, several 1D convolution layers with a kernel size of 6 and a stride size of 2 are used to reduce the dimension of the input vector down to $d_{conv}$, which is as low as possible but no smaller than 32. After that, a fully connected layer transforms the previous layer into a vector of size 32 and then another fully connected layer transforms the previous layer into a vector of the size $\hat{d}$, the dimension of the generalized coordinate.

4.1. Encoder

To train the manifold-parameterization function $g$, we also need to define the value of $\hat{x}$ at each time step $t_n$ for each parameter instance $\mu$. We do so implicitly by introducing an encoder network $e$. $\bar{\mathbf{x}}(t_n, \mu)$ is the input to $e$, which is defined by concatenating the deformed positions of all the material points. Such an input encourages the injectivity of $g$ with respect to $\hat{x}$ since there exists a unique $\hat{x}$ that corresponds to a simulation state, as defined by all the positions of the material points. This input is particularly suitable for history-independent problems, e.g., elasticity. For history-dependent problems, history-dependent variables can also be concatenated to the input to the encoder function to define a simulation state uniquely. In addition, the encoder structure also encourages a spatially and temporally coherent representation of the simulation state where nearby $\hat{x}$s correspond to nearby simulation states \[10\]. The detailed network structure of $e$ is listed in Table 1.

Remark. One can also define $\hat{x}$ without using the encoder network. For example, $\hat{x}$ can be defined explicitly by either manual choice or directly exposing $\hat{x}$ to the training scheme as a training variable along with the network weights. In practice, we found these approaches challenging to scale to large-scale problems as the number of $\hat{x}$s scales linearly with the number of simulation snapshots used for training. Furthermore, it is inconvenient to enforce injectivity as well as spatial and temporal coherence with these techniques.
4.2. Loss function

Let $\theta_\theta^*$ and $\theta_e^*$ be the weights of network $g$ and $e$. $\theta_\theta^*$ and $\theta_e^*$ are computed as the (approximate) solution to the minimization problem

$$
\min_{\theta_\theta, \theta_e} \sum_{n=0, \cdots, n^t, \mu \in D_{\text{train}}} (\| g_{\theta_\theta}(X^p; \hat{x}(t_n, \mu)) - \phi(X^p; t_n, \mu) \|^2_2 + \lambda_F \| \nabla g_{\theta_\theta}(X^p; \hat{x}(t_n, \mu)) - \nabla \phi(X^p; t_n, \mu) \|^2_F + \lambda_v \| \frac{\partial g_{\theta_\theta}}{\partial \hat{x}}(X^p; \hat{x}(t_n, \mu)) \frac{\hat{x}(t_{n+1}, \mu) - \hat{x}(t_n, \mu)}{\Delta t} - \phi(X^p; t_n, \mu) \|^2_2)
$$

(24)

where $\hat{x}(t_n, \mu) = e_\theta(\bar{x}(t_n, \mu))$, $n^t$ is the number of time steps, $n^p$ is the number of the material points, $D_{\text{train}} \subseteq D$ is the parameter instances for training, at which the full-order model has been solved and solutions are available, and $\lambda_F, \lambda_v \in \mathbb{R}_+$ denote penalty parameters for the deformation gradient and velocity, respectively.

Deformation gradient penalty $\lambda_F$ is necessary for the network to accurately describe the spatial gradients (Equation (15)), which is essential for both Lagrangian quadratures (Algorithm 3) and Eulerian quadratures (Algorithm 4). Velocity penalty $\lambda_v$ is necessary so that the full space velocity is well represented through the tangent space of the manifold (Equation (16)). Additional penalties could be introduced for higher-order derivatives.

Note that the $\frac{\hat{x}(t_{n+1}, \mu) - \hat{x}(t_n, \mu)}{\Delta t}$ term is a finite difference approximation of the generalized velocity $\dot{\hat{v}}$. Such an approximation effectively guarantees the validity of the linearized projection (Appendix A).

The rest of the training details are listed in Appendix B.

5. Numerical experiments

We demonstrate the robustness of the proposed reduced-order approach on several large-deformation nonlinear elasticity problems. The particular constitutive law we adopt is the fixed corotated hyperelastic energy by Stomakhin et al. [28]. In principle, any hyperelastic model would work with the proposed approach with no modification necessary. The baseline, full-order model is directly computed through the open-source explicit MPM implementation by Wang et al. [105]. Both the full-order model and the reduced-order runs 12 threads on a 2.30GHz Intel Xeon E5-2686 v4 CPU. In addition, the neural network portion of the reduced-order pipeline is implemented using the LibTorch library [29] and runs on a single NVIDIA Tesla V100 GPU.

All errors reported are the accumulative position errors of the test simulations for validation purposes that are not exposed during training. This error is defined as the L2 relative error:

$$
\left( \frac{\sum_{n=0, \cdots, n^t, \mu \in D_{\text{test}}} \| g_{\theta_\theta}(X^p; \hat{x}(t_n, \mu)) - \phi(X^p; t_n, \mu) \|^2_2}{\sum_{n=0, \cdots, n^t, \mu \in D_{\text{test}}} \| \phi(X^p; t_n, \mu) \|^2_2} \right)^{\frac{1}{2}}
$$

where we sum the relative errors over all material points, over all time steps, and over all simulation instances of the testing dataset. This error is zero when the reduced-order model, i.e., the approximated deformation map $g_{\theta_\theta}$, agrees exactly with the full-order model, i.e., the deformation map $\phi$.

### Table 2: Material properties and discretization parameters

| Experiment | Geometry | Young’s modulus | Poisson’s ratio | # of particles | Grid cell width | Particles per cell per dimension | Time step | Time steps | % of training | % of testing |
|------------|----------|-----------------|----------------|----------------|----------------|-------------------------------|----------|------------|-------------|-------------|
| Section 8.1| Cylinder | 12500 Pa        | 0.3            | 1368           | 0.04 cm        | 2                             | $\frac{1}{16}$ s | 30         | 24          | 6           |
| Section 8.2| Cylinder | 12500 Pa        | 0.3            | 1757           | 0.04 cm        | 2                             | $\frac{1}{16}$ s | 30         | 29          | 7           |
| Section 8.3| Cylinder | 12500 Pa        | 0.3            | 1368           | 0.04 cm        | 2                             | $\frac{1}{16}$ s | 432        | 12          | 24          |
| Section 8.4| Tower    | 80000 Pa        | 0.2            | 1176           | 0.48 cm        | 3                             | $\frac{1}{16}$ s | 80         | 16          | 4           |

Material points are initially positioned through the Poisson disk sampling approach [15] by sampling a fixed number of particles per grid cell per dimension [59]. The time step size is computed from the stability analysis based on the speed of the elastic wave and the element characteristic length scale [39].
5.1. Gravity

We conduct the numerical experiment on an elastic cylinder with a radius of 1 cm and a height of 4 cm. Its material and numerical properties are listed in Table 2. The elastic cylinder is attached to a vertical wall on one side and deforms under the influence of gravity (Figure 6). We generate training and testing data by sampling 30 evenly spaced numbers over \( g \in [1, 10] \) m/s\(^2\), where \( g \) is the magnitude of the gravitational force. For each \( g \), we run a simulation of 30 time steps. Therefore, a total of 930 simulation snapshots are generated, including the initial conditions. We then randomly split the dataset of 30 simulations into an offline training dataset of 24 simulations and an online testing dataset of 6 simulations.

5.1.1. The effect of gradient penalties

![Figure 7: The effect of gradient penalties during training. The temporal penalty term \( \lambda_v \) improves the accuracy of the linear projection schemes (c and d) but not the nonlinear projection schemes (a and b). The spatial penalty term \( \lambda_F \) improves both projection schemes. The Lagrangian (a and c) and the Eulerian (b and d) quadrature approaches yield similar results.](image)

In Figure 7, we study the influence of offline training parameters on the accuracy of the online reduced-order simulation. The \( \lambda_F \) term that trains the spatial gradient varies between zero and nonzero; The \( \lambda_v \) term that trains the temporal gradient also varies in the same manner. The dimension of the generalized coordinate is fixed to be 7, though similar trends are observed with other generalized coordinate dimensions. After training, we conduct reduced-order simulations using the four different combinations proposed in Section 3.2 and Algorithm 2. To isolate the source of error, we do not conduct hyper-reduction here.

Training with a positive \( \lambda_v \) significantly improves the accuracy of the linear projection technique of Algorithm 7 (Figure 7c and d). In contrast, the nonlinear projection scheme of Algorithm 6 (Figure 7c and d) is less sensitive to the choice of \( \lambda_v \). Such observations hold for both the Lagrangian quadratures
and the Eulerian quadratures. This numerical result aligns well with the theoretical analysis of linearization (Section 4). Furthermore, training with a positive $\lambda_F$ augments all four algorithm combinations (Figure 7 a, b, c, and d) due to a better deformation gradient approximation.

### 5.1.2. The effect of the generalized coordinate dimension

![Diagram showing the effect of the generalized coordinate dimension on position error.](image)

Figure 8: Increasing the dimension of the generalized coordinate improves the accuracy due to larger learning capacities.

Figure 8 demonstrates the effect of the generalized coordinate dimension $\hat{d}$, which is a key hyperparameter of the network structure (Section 4). We train networks with different generalized coordinate dimensions while fixing $\lambda_v$ to be 0.01 and $\lambda_F$ to be 100. Afterward, the trained networks are tested for reduced-order simulations. In order to isolate the source of error, hyper-reduction is not applied. Figure 8 shows that the four different algorithm combinations from Algorithm 2 demonstrate the same trend: increasing the generalized coordinate dimension improves the simulation accuracy because the network has a larger learning capacity.
5.1.3. The effect of hyper-reduction

Figure 9: The effectiveness of hyper-reduction is evidenced by the fact that using 10 projection points yields less than 1% error, and using just 100 points yields the same accuracy as no hyper-reduction, where all 1,368 material points are used for projection.

Figure 9 shows the projection point number’s influence on the accuracy of the reduced-order simulations. After the offline training with a setup of $\lambda_c = 0.01$, $\lambda_F = 100$, $\hat{d} = 4$, we conduct online, reduced-order simulations with various numbers of points for projection. Notably, with just 10 points for projection, all four quadrature and projection combinations yield an error of less than 1%. In addition, using just 100 points delivers the same level of accuracy as no hyper-reduction, i.e., all 1,368 points’ dynamics are computed (Section 3.2.2) and used for projection onto the generalized coordinates (Section 3.2.3).

5.1.4. Hyperparameter summary

To summarize all the offline and online hyperparameter options, we plot all the choices together (Figure 10). Section 5.1.1, Section 5.1.2, and Section 5.1.3 each presents a “slice” of the hyperparameter study in Figure 10 in order to articulate the effect of a particular parameter.
Figure 10: Hyperparameter summary. Training with positive spatial and temporal gradient penalties yields the optimal result. The size of the generalized coordinate size should be larger than one in order to attain the best accuracy. At least 10 sample points should be used to guarantee projection accuracy.

As shown in Figure 10, independent of quadrature and projection combinations, training with $\hat{d} = 1$ always yields a worse result, e.g., (100, 0.01, 1) vs. (100, 0.01, 4), (100, 0.01, 1) vs. (100, 0.01, 7). Therefore, the default training strategy should use $\hat{d} > 1$. Unlike the nonlinear projection scheme (Figure 10a and b), the linear projection scheme (Figure 10c and d) also consistently yields better result when training with a positive temporal gradient penalty $\lambda_v > 0$, e.g., (0, 0.0, 4) vs. (0, 0.01, 4), (0, 0.0, 7) vs. (0, 0.01, 7). Therefore, the default training strategy should use a positive temporal gradient penalty, especially when using linear projection. The spatial gradient penalty also improves simulation accuracy, as discussed in Section 5.1.1. Therefore, the default training strategy should also include the spatial gradient penalty term, though its significance is lesser than the other terms (Figure 10). With the training strategy mentioned earlier, all four reduced-order schemes achieve less than 1% error with just 10-100 sample points for projection. To obtain meaningful results, projection with just one point should be avoided as it can cause an error over 100%.

Since this section focuses on small-scale experiments that serve to support an extensive parameter study; as such, the opportunity for wall-time speedup of the reduced-order method over the full-order method is diminished; experiments in Section 5.4 will report wall-time speedups for higher-dimension problems.
5.2. Torsion and tension

We apply tension and torsion to an elastic, rectangular cuboid (Figure 11). The dimensions of the cuboid are 1 cm, 1 cm, and 4 cm. Its material and discretization properties are listed in Table 2. We generate simulation data by varying the translational velocity \( v \in [0, 0.6] \) m/s and the rotational velocity \( \omega \in [0, 2] \) rad/s. A total number of 36 simulations are generated via full factorial sampling of the translation and the rotational velocities with 6 evenly spaced samples in each dimension. Each simulation consists of 30 time steps. Therefore, a total of 1,116 simulation snapshots, including the initial conditions, are generated. Afterward, we randomly assign 29 simulations for training and 7 for testing.

| Scheme                      | \# of points for projection (50) | \# of points for projection (all) |
|-----------------------------|----------------------------------|----------------------------------|
| Lagrangian quad + nonlinear proj | 0.28%                           | 0.22%                           |
| Lagrangian quad + linear proj      | 0.39%                           | 0.29%                           |
| Eulerian quad + nonlinear proj    | 0.33%                           | 0.24%                           |
| Eulerian quad + linear proj       | 0.34%                           | 0.27%                           |

Table 3: Torsion and tension: errors of the reduced-order simulations on the testing dataset.

An approximated deformation map network is trained with \( \lambda_F = 100, \lambda_v = 0.01, \hat{d} = 7 \) (c.f., Section 5.1.4). We then conduct reduced-order simulations using this network. Table 3 reports the testing errors of the reduced-order simulations using different quadrature and projection combinations with and without hyper-reduction, demonstrating the effectiveness of the reduced-order simulation in modeling tension and torsion.
### 5.2.1. Zero-shot super-resolution

![Image](image_url)

Figure 12: The approximated deformation map is trained with low-resolution simulations (b). We can then run high-resolution, reduced-order simulations (c) that agree well with high-resolution, full-order simulations (d). The low-resolution simulation has 1,757 material points (Table 2). The high-resolution simulation increases the spatial resolution by 2 times in each dimension and has 13,900 material points in its initial setup. The Poisson disk sampling approach (Table 2) employed for generating initial material points has a random nature. None of the high-resolution material points shares identical initial positions with the low-resolution material points. In (a), we construct a super-resolution baseline by using the “tracer particle technique” [96, 34], where we advect the high-resolution material points using the velocity field computed from the low-resolution simulation. In comparison, our reduced high-resolution simulation has a higher accuracy than the baseline, both visually and quantitatively, measured by the position error from the high-resolution simulation ground truth.

| Scheme | Position error |
|--------|----------------|
| Lagrangian quad + nonlinear proj | 0.49% |
| Lagrangian quad + linear proj | 0.48% |
| Eulerian quad + nonlinear proj | 0.47% |
| Eulerian quad + linear proj | 0.46% |
| Baseline | 2.04% |

Table 4: Zero-shot super-resolution: errors of the reduced-order simulations on the high-resolution, torsion and tension testing dataset. To isolate the source of error, no hyper-reduction is conducted.

An advantage of training the deformation map instead of a finite number of material points is that we can easily adjust the resolution of the reduced-order simulation. We can infer the dynamics of an infinite number of material points so long as they belong to the reference domain. Consequently, even though the deformation map is trained on low-resolution simulations (Figure 12b), we can run high-resolution reduced-order simulations (Figure 12c) by using a finer MPM grid. Since the high-resolution simulation is never exposed to the training process, zero-shot super-resolution is achieved [67]. We construct a super-resolution baseline by advecting high-resolution material points on the velocity field from the low-resolution simulation (a). Both simulations are compared with the high-resolution ground truth (Figure 12d). Table 4 lists the errors of the super-resolution simulations of all the reduced-order schemes and the baseline, across the entire testing dataset. All reduced-order methods outperform the super-resolution baseline.
5.3. Poke-and-recover

Figure 13: The material is poked at the top by different forces, resulting in different deformed states. The material then recovers to its initial state because of elasticity.

Poking is a frequent use case of the elasticity simulation where a force is applied in a particular direction at a small portion of the material and is released after a short period. The material then recovers to its undeformed state due to elasticity.

The elastic cylinder from Section 5.1 is poked at the top (Figure 13). The poking force is characterized by the spherical coordinate, where \( f(\rho, \theta, \phi) = (r \sin \phi \cos \theta, r \sin \phi \sin \theta, r \cos \phi) \). The corresponding poking location, at which the force is applied, is \((-r \cos \theta, -r \sin \theta, h)\), where \( r \) and \( h \) are the radius and the height of the cylinder, respectively. \( r \) is chosen such that the poking point moves at a constant speed of 4.8 cm/s. \( \phi \) is fixed to be \( \frac{11}{12} \pi \). The poking force is applied for 0.25 s before it is released. After the force is released, the cylinder recovers to its initial state due to elasticity. We generate simulation data by varying \( \theta \in [0, 2\pi) \), the direction at which the poking force is applied. In particular, \( \theta \) is evenly sampled between \( \theta \in [0, 2\pi) \) with an interval of \( \frac{\pi}{18} \) and a total of 36 simulations are generated. We use 12 of these 36 simulations for training. The \( \theta \) of these 12 simulations is evenly spaced with an interval of \( \frac{\pi}{7} \). The rest 24 simulations are used for testing. The goal of this training and testing split is gauging online reduced-order simulations’ ability to poke at an arbitrary \( \theta \). Each simulation consists of 432 time steps. Therefore, a total of 15,588 simulation snapshots, including the initial conditions, are used for training and testing.

| Scheme                        | # of points for projection (50) | # of points for projection (all) |
|-------------------------------|---------------------------------|----------------------------------|
| Lagrangian quad + nonlinear proj | 1.47%                          | 1.07%                            |
| Lagrangian quad + linear proj  | 1.47%                          | 1.04%                            |
| Eulerian quad + nonlinear proj | 1.67%                          | 1.41%                            |
| Eulerian quad + linear proj    | 1.55%                          | 1.37%                            |

Table 5: Poke-and-recover: errors of the reduced-order simulations on the testing dataset.

We first conduct offline training with \( \lambda_F = 100, \lambda_v = 0.01, \hat{d} = 6 \) (c.f., Section 5.1.4) and then runs online reduced-order simulations. Table 5 reports the testing errors of the reduced-order simulations using different quadrature and projection combinations with and without hyper-reduction, demonstrating the effectiveness of the reduced-order simulation in modeling the poke-and-recover problem.

5.3.1. Reduced-space trajectory
Figure 14 plots the trajectory of the generalized coordinate $\hat{x}$ of a reduced poke-and-recover simulation. Since $\hat{x}$ is high-dimensional in general, we visualize $\hat{x}$ by projecting it onto a 2D plane spanned by its first two principal components.

Under the influence of the poking force, the material takes up a deformed state in the full space $\mathbf{x}$; after the force is removed, the material then returns to its undeformed state. It is also desirable to maintain such a “return” property in the reduced space $\hat{x}$, where $\hat{x}$ returns to its initial value. In general, one state in the full space can correspond to multiple generalized coordinates, i.e., the approximate deformation map is not necessarily injective with respect to $\hat{x}$. Using the encoder training scheme in Section 4.1 that encourages injectivity, we can maintain the “return” property in the reduced space. As shown in Figure 14, the generalized coordinate $\hat{x}$ returns to the origin, which maps to the undeformed state in the full space.

5.3.2. Continual manipulation

One advantage of maintaining the “return” property is that even though the training data consists of simulations poking only once, we can actually run reduced-order simulations that poke repetitively. After each poke-and-recover sequence, the generalized coordinate $\hat{x}$ returns to its starting value, ready to be poked again in an arbitrary direction.

Therefore, we run a reduced-order simulation consisting of ten consecutive poke-and-recover sequences, where the poking direction is chosen each time randomly. This simulation has 4320 time steps or 30 s in total. Table 6 reports its error in comparison with the full-order simulation. All quadrature and projection combinations produce good agreements with and without hyper-reduction. Note that no full-order simulation of 30 s is included in the training data. All training simulations are 3 s.

| Scheme                  | # of points for projection (50) | # of points for projection (all) |
|-------------------------|---------------------------------|----------------------------------|
| Lagrangian quad + nonlinear proj | 1.72%                          | 1.45%                            |
| Lagrangian quad + linear proj | 1.82%                          | 1.41%                            |
| Eulerian quad + nonlinear proj | 1.88%                          | 1.81%                            |
| Eulerian quad + linear proj | 1.96%                          | 1.74%                            |

Table 6: Continual poking and recovering: errors of the reduced-order simulations.
5.4. Application

To demonstrate the efficiency of the reduced-order simulation in comparison to the full-order simulation, we conduct large-scale experiments with a complex tower geometry (Figure 15). The material and numerical properties of the object are listed in Table 2.

Both the top and the bottom of the object experiences Dirichlet boundary conditions. The top is kinematically moved under a fixed velocity while the bottom stays still. The fixed velocity is parameterized by a spherical coordinate, \( \mathbf{v} = (r, \theta, \phi) = (r \cos \phi \cos \theta, r \cos \phi \sin \theta, r \sin \phi) \), where \( r \in [0.6, 0.8) \), \( \theta \in [0, \frac{\pi}{2}) \), and \( \phi \in [-\frac{\pi}{3}, -\frac{\pi}{12}) \). We investigate the reduced-order approach’s accuracy and efficiency in modeling this parametrized setting. Training data is generated by running 16 simulations with different \((r, \theta, \phi)\) triplets sampled using the Latin hypercube method [97]. Another 4 simulations are sampled using the same approach for testing purposes.

Figure 15: An object with complicated geometry undergoes elastic deformation (visualized with mesh, c.f. Figure 4 for raw material point data). Each row records a different configuration in the testing dataset (i.e., unseen during training). Each column corresponds to a different time during the simulation. The leftmost column is the beginning of the simulation, while the rightmost column is the end of the simulation. In each snapshot, the white tower on the left is the full-order simulation while the yellow tower on the right is the reduced-order simulation. The full-order model and the reduced-order model match overall in all configurations and at all times. However, the reduced model lacks secondary wrinkles that are present in the full-order model. More complex network architecture can be explored in order to capture these secondary features.
Figure 16: Position error vs. wall time. Each data point corresponds to a particular reduced-order setup (quadrature choices, projection types, and the number of points for projection). Wall time is the average computational cost for every physical second of simulation. A real-time simulation requires the wall clock time to be 1 (green line). The red line indicates the wall clock time of the original simulation. Setups using Eulerian quadratures and linear projection with fewer than 50 projection points (blue cross and orange cross) reach the real-time criteria and are over 20 times faster than the full-order model while maintaining decent accuracy.

An approximated deformation map network is trained with $\lambda_F = 100, \lambda_v = 0.01, \hat{d} = 6$ (c.f., Section 5.1.4). Afterward, we conduct systematic tests over the different reduced-order approaches and different numbers of material points for projection (Figure 16). All setups using the Eulerian quadratures (pentagon and cross) offer a significant speedup over the full-order model while maintaining a decent accuracy. By contrast, due to their need to track every material point, Lagrangian quadrature approaches (circle and square) do not offer a reduction in computation complexity and do not offer speedup over the full-order model. While linear projection methods (square and cross) generally have a slightly higher error than nonlinear projection methods (circle and pentagon), they are also computationally faster. Across all methods, the fewer projection points, the faster the simulation and the higher the error. In particular, the reduced-order simulations employing Eulerian quadratures and linear projection with fewer than 50 projection points (blue cross and orange cross) attain real-time performances. They are over 20 times faster than the baseline full-order model. Figure 16 displays all four testing simulations using the Eulerian quadratures and linear projection with 50 projection points (orange cross). Visually, the reduced-order simulations agree well with the full-order simulations while missing some secondary wrinkle deformations. Further research can be conducted on increasing the complexity of the network to capture these secondary deformations.

Remark. Instead of computing the dynamics of over 3 million material points, we only need to compute the dynamics of no more than 50 points (over 600,000 times reduction). However, the speedup number we observe is reduced to 20X. This discrepancy can be understood by the nonlocal nature of MPM where a neighborhood of quadrature points are required for computing the dynamics of even just one material point. Consequently, in order to update the dynamics of 50 points, over 20,000 quadrature points are involved in the particle to grid transfer.

Remark. We adopt a random sampling approach for choosing hyper-reduction projection points (Section 3.3). While such a method is easy to implement, it does not guarantee optimality in errors and computation costs. Future work should be conducted to select the optimal set of hyper-reduction points to minimize the position error and the computation cost.

6. Conclusion and future work

In this work, we present the first attempt at model reduction of MPM. We train a deep neural network to approximate the “deformation map” kinematics in continuum mechanics. We then utilize this trained
network to drive the MPM simulation via optimal-projection-based dynamics, ensuring the simulated trajectory remains on the low-dimensional manifold associated with the deformation-map approximation. We demonstrate that this approach can drastically reduce the dimension of the MPM hyperelasticity and offers an order-of-magnitude wall-time speedup.

In the future, we would like to extend our work to support other phenomena, such as plasticity, fracture, contact, and collision. This work focuses on the spatial model reduction of MPM. Future work should also consider a reduction in the temporal domain in order to take a larger time step size. Since the stress evaluation is completed on the CPU while the neural network evaluation is completed on the GPU, expensive CPU-GPU transfer is conducted at each time step size. Future work might investigate a full GPU implementation to avoid the costly transfer. Even though the proposed model reduction framework is designed for MPM, the proposed manifold parameterization function is, in fact, discretization independent. Therefore, we would like to go beyond MPM and explore its ability in model reduction of other continuum mechanics discretizations, such as the finite element method (FEM) and Smoothed-Particle Hydrodynamics (SPH). For the same reason, we would also like to explore the manifold parameterization function’s ability to learn from simulation data with adaptive refinement.

[1] Abgrall, R., Crisovan, R., 2018. Model reduction using l1-norm minimization as an application to nonlinear hyperbolic problems. International Journal for Numerical Methods in Fluids 87 (12), 628–651.
[2] Amsallem, D., Cortial, J., Carlberg, K., Farhat, C., 2009. A method for interpolating on manifolds structural dynamics reduced-order models. International Journal for Numerical Methods in Engineering 80 (9), 1241–1258.
[3] Amsallem, D., Zahr, M. J., Farhat, C., 2012. Nonlinear model order reduction based on local reduced-order bases. International Journal for Numerical Methods in Engineering 92 (10), 891–916.
[4] An, S. S., Kim, T., James, D. L., 2008. Optimizing cubicute for efficient integration of subspace deformations. ACM transactions on graphics (TOG) (27) (5), 1–10.
[5] Barbič, J., James, D. L., 2005. Real-time subspace integration for st. venant-kirchhoff deformable models. ACM transactions on graphics (TOG) 24 (3), 982–990.
[6] Barbič, J., Zhao, Y., 2011. Real-time large-deformation substructuring. ACM transactions on graphics (TOG) 30 (4), 1–8.
[7] Bardenhagen, S., Brackbill, J., Sulskey, D., 1998. Shear deformation in granular materials. Tech. rep., Los Alamos National Lab., NM (United States).
[8] Barone, M. F., Kalashnikova, I., Segalman, D. J., Thornquist, H. K., 2009. Stable galerkin reduced order models for linearized compressible flow. Journal of Computational Physics 228 (6), 1932–1946.
[9] Baur, U., Beattie, C., Benner, P., Gugercin, S., 2011. Interpolatory projection methods for parameterized model reduction. SIAM Journal on Scientific Computing 33 (5), 2489–2518.
[10] Bengio, Y., Courville, A., Vincent, P., 2013. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence 35 (8), 1798–1828.
[11] Benner, P., Feng, L., Li, S., Zhang, Y., 2015. Reduced-order modeling and rom-based optimization of batch chromatography. In: Numerical Mathematics and Advanced Applications ENUMATH 2013. Springer, pp. 427–435.
[12] Benner, P., Gugercin, S., Willcox, K., 2015. A survey of projection-based model reduction methods for parametric dynamical systems. SIAM review 57 (4), 483–531.
[13] Bergmann, M., Bruneau, C.-H., Iollo, A., 2009. Enablers for robust pod models. Journal of Computational Physics 228 (2), 516–538.
[14] Bergmann, M., Cordier, L., Brancher, J.-P., 2005. Optimal rotary control of the cylinder wake using proper orthogonal decomposition reduced-order model. Physics of fluids 17 (9), 097101.
[15] Bridson, R., 2007. Fast poisson disk sampling in arbitrary dimensions. SIGGRAPH sketches 10 (1).
[16] Bruns, A., Benner, P., 2015. Parametric model order reduction of thermal models using the bilinear interpolatory rational krylov algorithm. Mathematical and Computer Modelling of Dynamical Systems 21 (2), 103–129.
[17] Carlberg, K., 2015. Adaptive h-refinement for reduced-order models. International Journal for Numerical Methods in Engineering 102 (5), 1192–1210.
[18] Carlberg, K., Bou-Mosleh, C., Farhat, C., 2011. Efficient non-linear model reduction via a least-squares petrov–galerkin projection and compressive tensor approximations. International Journal for numerical methods in engineering 86 (2), 155–181.
[19] Carlberg, K., Farhat, C., 2011. A low-cost, goal-oriented ‘compact proper orthogonal decomposition’basis for model reduction of static systems. International Journal for Numerical Methods in Engineering 86 (3), 381–402.
[20] Carlberg, K., Farhat, C., Cortial, J., Amsallem, D., 2013. The gnat method for nonlinear model reduction: effective implementation and application to computational fluid dynamics and turbulent flows. Journal of Computational Physics 242, 623–647.
[21] Carlberg, K., Tuminaro, R., Boggs, P., 2015. Preserving lagrangian structure in nonlinear model reduction with application to structural dynamics. SIAM Journal on Scientific Computing 37 (2), B133–B184.
[22] Chen, P. Y., Chantharayukhonthorn, M., Yue, Y., Grinspun, E., Kamrin, K., 2021. Hybrid discrete-continuum modeling of shear localization in granular media. Journal of the Mechanics and Physics of Solids 153, 104404.
[23] Craig Jr, R. R., Bampton, M. C., 1968. Coupling of substructures for dynamic analyses. AIAA journal 6 (7), 1313–1319.
[24] Daphalapurkar, N. P., Lu, H., Coker, D., Komanduri, R., 2007. Simulation of dynamic crack growth using the generalized interpolation material point (gimp) method. International Journal of Fracture 143 (1), 79–102.
[25] Daviet, G., Bertails-Descoubes, F., 2016. A semi-implicit material point method for the continuum simulation of granular materials. ACM Transactions on Graphics (TOG) 35 (4), 1–13.
[26] Drohmann, M., Haasdonk, B., Ohlberger, M., 2012. Reduced basis approximation for nonlinear parametrized evolu-
tation equations based on empirical operator interpolation. SIAM Journal on Scientific Computing 34 (2), A937–A969.

[27] Erickson, N. B., Muehlebach, M., Mahoney, M. W., 2019. Physics-informed autoencoders for lyapunov-stable fluid flow prediction. arXiv preprint arXiv:1905.10866.

[28] Everson, R., Sirovich, L., 1995. Karhunen–loeve procedure for gappy data. JOSA A 12 (8), 1657–1664.

[29] Fang, F., Pain, C., Navon, I., Elsheikh, A., Du, J., Xiao, D., 2013. Non-linear Petrov–Galerkin methods for reduced order hyperbolic equations and discontinuous finite element methods. Journal of Computational Physics 234, 540–559.

URL https://www.sciencedirect.com/science/article/pii/S0021999112006006

[30] Fang, Y., Hu, Y., Hu, S.-M., Jiang, C., 2018. A temporally adaptive material point method with regional time stepping. In: Computer graphics forum. Vol. 37. Wiley Online Library, pp. 195–204.

[31] Fang, Y., Li, M., Gao, M., Jiang, C., 2019. Silly rubber: an implicit material point method for simulating non-equilibrated viscoelastic and elastoplastic solids. ACM Transactions on Graphics (TOG) 38 (4), 1–13.

[32] Fang, Y., Qu, Z., Li, M., Zhang, X., Zhu, Y., Anjaneya, M., Jiang, C., 2020. Lypmpm: an interface quadrature material point method for non-sticky strongly two-way coupled nonlinear solids and fluids. ACM Transactions on Graphics (TOG) 39 (4), 51–1.

[33] Fei, Y., Guo, Q., Wu, R., Huang, L., Gao, M., 2021. Revisiting integration in the material point method: a scheme for easier separation and less dissipation. ACM Transactions on Graphics (TOG) 40 (4), 1–16.

[34] Fu, C., Guo, Q., Gast, T., Jiang, C., 2019. A polynomial particle-in-cell method. ACM Transactions on Graphics (TOG) 36 (6), 1–12.

Fulton, L., Modi, V., Duvenaud, D., Levin, D. I., Jacobson, A., 2019. Latent-space dynamics for reduced deformable simulation. In: Computer graphics forum. Vol. 38. Wiley Online Library, pp. 379–391.

Galbally, D., Fidkowski, K., Willcox, K., Ghattas, O., 2010. Non-linear model reduction for uncertainty quantification in large-scale inverse problems. International Journal for numerical methods in engineering 81 (12), 1581–1608.

Gao, M., 2018. Sparse Faged Grid and its Applications to Adaptivity and Material Point Method in Physics Based Simulations. The University of Wisconsin-Madison.

Gao, M., Wang, X., Wu, K., Pradhan, A., Sifakis, E., Yukel, C., Jiang, C., 2018. Gpu optimization of material point methods. ACM Transactions on Graphics (TOG) 37 (6), 1–12.

Gast, T. F., Schroeder, C., Stomakhin, A., Jiang, C., Teran, J. M., 2015. Optimization integrator for large time steps. IEEE transactions on visualization and computer graphics 21 (10), 1103–1115.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, pp. 249–256.

Gu, C., 2011. Model order reduction of nonlinear dynamical systems. University of California, Berkeley.

[36] Gugercin, S., Antoulas, A. C., Beattie, C., 2008. H_{2} model reduction for large-scale linear dynamical systems. SIAM journal on matrix analysis and applications 30 (2), 609–638.

Hall, K. C., Thomas, J. P., Dowell, E. H., 2000. Proper orthogonal decomposition technique for transonic unsteady aerodynamic flows. AIAA journal 38 (10), 1853–1862.

Han, X., Gast, T. F., Guo, Q., Wang, S., Jiang, C., Teran, J., 2019. A hybrid material point method for frictional contact with diverse materials. Proceedings of the ACM on Computer Graphics and Interactive Techniques 2 (2), 1–24.

Hartman, D., Mestha, L. K., 2017. A deep learning framework for model reduction of dynamical systems. In: 2017 IEEE Conference on Control Technology and Applications (CCTA). pp. 1917–1922.

Holmes, P., Lumley, J. L., Berkooz, G., Rowley, C. W., 2012. Turbulence, coherent structures, dynamical systems and symmetry. Cambridge university press.

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., Durand, F., 2019. Diffthaichi: Differentiable programming for physical simulation. arXiv preprint arXiv:1910.00935.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhan, A., Jiang, C., 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Transactions on Graphics (TOG) 37 (4), 1–14.

Hu, Y., Li, T.-M., Anderson, L., Ragan-Kelley, J., Durand, F., 2019. Taichi: a language for high-performance computation on spatially sparse data structures. ACM Transactions on Graphics (TOG) 38 (6), 1–16.

Hu, Y., Liu, J., Spielberg, A., Tenenbaum, J. B., Freeman, W. T., Wu, J., Rus, D., Matusik, W., 2019. Chainqueen: A real-time differentiable physical simulator for soft robotics. In: 2019 International conference on robotics and automation (ICRA). IEEE, pp. 6265–6271.

[39] James, D. L., Barbič, J., Pai, D. K., 2006. Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. ACM Transactions on Graphics (TOG) 25 (3), 987–995.

[40] Jiang, C., Gast, T., Teran, J., 2017. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. ACM Transactions on Graphics (TOG) 36 (4), 1–14.

Jiang, C., Schroeder, C., Selle, A., Teran, J., Stomakhin, A., 2015. The affine particle-in-cell method. ACM Transactions on Graphics (TOG) 34 (1), 1–10.

Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., Selle, A., 2016. The material point method for simulating continuum materials. In: ACM SIGGRAPH 2016 Courses. pp. 1–52.

[42] Jiang, Y., Li, M., Jiang, C., Alonso-Marroquin, F., 2020. A hybrid material-point spheropolygon-element method for solid and granular material interaction. International Journal for Numerical Methods in Engineering 121 (14), 3021–3047.

Khazumi, K., 2016. Nonlinear model reduction by deep autoencoder of noise response data. In: 2016 IEEE 55th Conference on Decision and Control (CDC). pp. 5750–5755.
A. Projection linearization

Plugging $\hat{x}_{n+1} = \hat{x}_n + \Delta t_n \hat{v}_n + \tilde{v}_n$ into Equation (22), we have

$$\hat{v}_{n+1} \in \arg \min_{\tilde{v} \in \mathbb{R}^d} \sum_{p \in M} \| g(X^p; \hat{x}_n + \Delta t_n \hat{v}) - x_{n+1}^{trial} \|^2_2. \quad (A.1)$$
Using Taylor’s theorem, we have

\[
g(X; \dot{x}_n + \Delta t_n \dot{v}) - x_{n+1}^{trial} \approx g(X; \dot{x}_n) + \Delta t_n \frac{\partial g}{\partial x}(X; \dot{x}_n) \dot{v} - x_{n+1}^{trial}
\]

\[
= g(X; \dot{x}_n) + \Delta t_n \frac{\partial g}{\partial x}(X; \dot{x}_n) \dot{v} - (x_n^p + \Delta t_n v_{n+1}^{trial})
\]

\[
= \Delta t_n \frac{\partial g}{\partial x}(X; \dot{x}_n) \dot{v} - \Delta t_n v_{n+1}^{trial}
\]

Under this linearization, Equation (A.1) becomes Equation (23). Consequently, the effectiveness of linear projection depends on the accuracy of such a linearization.

**B. Training details**

We implemented the network in PyTorch [79] and trained the network with the ADAM optimizer with an adaptive learning rate, decreasing from $1e^{-3}$ to $1e^{-6}$. We initialize the neural network’s weights using the Xavier initialization [40]. Standard feature-scaling is conducted for the input and the output of the network to ease the training process. Min-max normalization is performed for the reference positions, while standardization is conducted for the network’s output to have zero mean and unit variance.