Momentum-Conserving Graph Neural Networks for Deformable Objects

Jiahong Wang[1] Logan Numerow[2] Stelian Coros[2] Christian Theobalt[1]
Vahid Babaei[1,3,4] Bernhard Thomaszewski[2]
[1]Max Planck Institute for Informatics [2]ETH Zürich [3]University of Bonn
[4]Fraunhofer Institute for Algorithms and Scientific Computing

Abstract

Graph neural networks (GNNs) have emerged as a versatile and efficient option for modeling the dynamic behavior of deformable materials. While GNNs generalize readily to arbitrary shapes, mesh topologies, and material parameters, existing architectures struggle to correctly predict the temporal evolution of key physical quantities such as linear and angular momentum. In this work, we propose MomentumGNN—a novel architecture designed to accurately track momentum by construction. Unlike existing GNNs that output unconstrained nodal accelerations, our model predicts per-edge stretching and bending impulses which guarantee the preservation of linear and angular momentum. We train our network in an unsupervised fashion using a physics-based loss, and we show that our method outperforms baselines in a number of common scenarios where momentum plays a pivotal role.

1. Introduction

Simulating the dynamics of deformable materials is a long-standing research problem in computer graphics and engineering [6, 8, 20, 25, 42]. The applications range from video games and animated movies to soft robotics, surgery training, and packaging design. Although established simulation methods based on, e.g., finite elements are robust, versatile, and accurate, the deep learning breakthrough led to the rise of (neural) surrogate models with the goal of learning and replacing computationally expensive physics-based simulations. By learning the mapping between external forces or constraints and resulting deformations, these surrogate models can offer substantial speedups while maintaining physical plausibility.

Among the various neural architectures, graph neural networks (GNNs) stand out for their ability to model complex physical systems. A landmark work in this area is MeshGraphNets [30], which introduced the use of GNNs for deformable object simulation via an encode-process-decode architecture that takes a simulation mesh as input and predicts nodal accelerations. This framework has since inspired a range of follow-up studies demonstrating compelling results in cloth simulation [21], garment modeling [14, 15], and character animation [46]. Despite their promise, existing graph-based approaches often struggle to conserve fundamental physical quantities, most notably momentum, because of their reliance on implicit, data-driven learning. They typically approximate physical dynamics by observation data without explicitly enforcing physical constraints. As a result, they exhibit non-physical behaviors such as drift and unnatural spin, which is particularly problematic in tasks involving free motion and collisions, where accurate momentum conservation is crucial for realism and reliability.

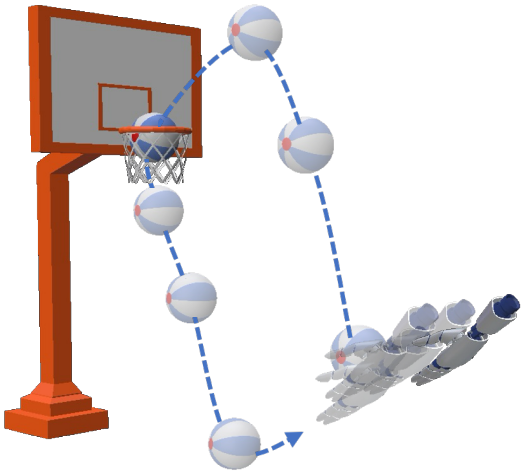

Figure 1. Simulation of a basket-shooting scene, where accurate modeling of momentum plays a crucial role. Our neural simulation method, *MomentumGNN*, accurately reproduces the momentum evolution from a ground truth simulation, leading to a scored shot. (More details in Fig. 9)

To address these limitations, we propose a novel GNN architecture that enforces physically-correct momentum evolution by design. Our main insight is to decompose momentum changes into contributions from external non-conservative forces, and those arising from internal elastic forces that conserve momentum. Our network then predicts

corrective impulses that conserve momentum, effectively refining the state produced by the momentum step. Specifically, we modify MeshGraphNets by replacing per-vertex decoders by *per-edge* decoders that predict momentum-conserving bending and stretching impulses. One key component is a novel layer-by-layer architecture that sequentially updates vertex positions using momentum-conserving impulses at each network layer, enabling greater representational capacity and improved physical fidelity. Briefly, our main contributions are:

- A physics-aware graph neural network that conserves the momentum by working on per-edge impulses.
- A layer-by-layer architecture in which each network layer sequentially updates vertex positions through momentum-conserving impulses.

2. Related Work

Neural Simulation Unlike traditional machine learning problems where inputs and outputs, e.g., images, are uniform in size and regularly structured, deformable object simulations have variable numbers of vertices and are highly complex in connectivity. One potential solution is to infer node-wise corrections to pre-computed linear nodal displacements [22]. Another strategy is model reduction [10, 33]. where neural networks encode mesh to a reduced-size latent embedding, which speeds up numerical simulations. More recent improvements involve the use of higher-order derivatives [40] and self-supervised physics-based energy loss [39, 44].

Neural methods have received particular attention in cloth simulation [9]. In the specific domain of garments, clothes are encoded as displacement offsets from the body [2]. To combat over-smoothing artifacts, various methods are proposed, such as decomposition of low-frequency deformations into high-frequency details [29] and wrinkles with coarse meshes [19]. Other recent methods [3, 4] further exploit the parameterization of body objects and take advantage of the well-established SMPL parameterization and Pose Space Deformation (PSD). SNUG [37] and Neural Cloth Simulation (NCS) [5] extend single-frame prediction to a continuous dynamics by embedding recurrent units into the network.

GNNs Despite impressive results, aforementioned models are limited in generalization ability. They are either subject to a fixed topology or rely on body parameterizations, which restrict them from broader applications in arbitrary shapes and dynamic topology changes. To resolve these drawbacks, mesh-based neural methods using graph neural networks (GNN) have become increasingly popular.

Graph Networks (GN) [34] were initially proposed for rigid systems in skeleton robotics. They were further extended to an encode-process-decode architecture for general fluid particle systems [36]. Arguably, the most important work in mesh-based methods is MeshGraphNets [30], as it inaugurates mesh-based methods in deformable objects. It borrows the encode-process-decode ideas and upgrades GN blocks with residual connections. It has shown stunning results in modeling the cloth dynamics in the wind and simple collisions. Later works in this area are derived mostly from MeshGraphNets, where they add random edge connections [12] or construct hierarchical U-net-like graph architectures [12, 14] to facilitate long-distance propagation, followed by further improvements in multi-layer collision handling in garments [15]. MeshGraphNetRP [21] furthermore adds RNN-based state encoding to reproduce cloth oscillations. Deep Emulator [46] applies mesh-based simulation to improve cage-based deformations of animated characters, where they use GNNs to predict the secondary dynamics of cages surrounding the character. Compared with existing mesh-based methods, our work focuses on a different objective, i.e., momentum conservation.

Physics-Inspired Networks Initial explorations into leveraging neural networks to accelerate physical simulation can be traced back to the 1990s [17]. More recently, the emergence and widespread adoption of physics-informed neural networks (PINN) [32] in solving partial differential equations (PDEs) have reignited significant interest in this domain. The resurgence has been particularly impactful in computational fluid dynamics (CFD), enabling advances such as surface pressure fields estimation [18], solutions to the Reynolds-Averaged Navier–Stokes equations [43], the acceleration of traditional turbulence solvers [27], and interactive aerodynamics design tools [1]. Concurrently, there is another line of research in approximating energy-like quantities such as Hamiltonian [13] and Lagrangian [23] by neural networks. They are further improved with graph neural networks [35] and decompositions of dissipative and conservative dynamics [41]. Unlike prior methods that emphasize PDEs and energy conservation, we focus on momentum conservation. Although recent studies consider momentum-conserving dynamics for fluids [31] and multi-body particles [38], these approaches do not readily extend to solid and cloth systems governed by elastic potentials. We propose specialized stretching and bending impulses to handle these settings.

3. Background

Our work builds on MeshGraphNets, a mesh-based graph neural network (GNN) architecture trained to predict the dynamics of deformable materials. It implements an Encode-Process-Decode design: starting from the current configuration (e.g., vertex positions and velocities), per-vertex input features are passed through an MLP encoder, a GNN processor, and an MLP decoder to produce per-vertex

accelerations. These accelerations are then integrated in time to obtain the vertex positions at the end of the time step. We explain each component in the following paragraphs.

Input graph The input graph $G_{\text{in}} = (V, E)$ is constructed from the initial simulation mesh M^0. A graph node $\mathbf{x}_i$ is created for every mesh vertex, and a bidirectional graph edge $\mathbf{e} = (\mathbf{e}_{ij}, \mathbf{e}_{ji})$ is added for every mesh edge. Furthermore, each node is endowed with a vertex feature $\mathbf{r}_i$ and every graph edge is given two edges features, $\mathbf{s}_{ij}$ and $\mathbf{s}_{ji}$, corresponding to the two uni-directional edges, respectively.

Encoder The encoder includes a node MLP and an edge MLP, which take the input graph and lift input features such as vertex velocity and edge length to a higher-dimensional latent space.

Processor The processor is composed of n_L message passing layers of graph network blocks [34]. Each layer learns a separate set of parameters and is applied sequentially to propagate information in local neighborhoods. The message passing begins by computing messages $\mathbf{m}_{ij}$ for every edge $\mathbf{e}_{ij}$ by an edge MLP f^{edge},

$$\mathbf{m}_{ij} \leftarrow f^{\text{edge}}(\mathbf{s}_{ij}, \mathbf{r}_i, \mathbf{r}_j) \,, \tag{1}$$

where $\mathbf{r}_i$ and $\mathbf{r}_j$ are the vertex features of the two nodes belonging to edge $\mathbf{e}_{ij}$. With all messages computed in this way, node and edge features are then updated as,

$$\mathbf{s}_{ij} \leftarrow \mathbf{s}_{ij} + \mathbf{m}_{ij} \,, \tag{2}$$

$$\mathbf{r}_i \leftarrow \mathbf{r}_i + f^{\text{vertex}}(\mathbf{r}_i, \sum_{j \in \mathcal{N}_i} \mathbf{m}_{ij}) \,, \tag{3}$$

where the vertex MLP f^{vertex} aggregates incoming edge messages with vertex features. Each layer in the processor block corresponds to a single message passing step. It is worth noting that, for a given message passing step, all vertex and edge features are processed with the same vertex and edge MLP, respectively. However, every layer has its own vertex and edge MLP.

Decoder After a given number of message passing steps, the vertex features from the last layer are passed into a decoder MLP which returns per-vertex accelerations $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_{|V|})^T$. These accelerations are then integrated in time to obtain updated positions,

$$\mathbf{x}^{i+1} = \mathbf{x}^i + \Delta t \mathbf{v}^i + \Delta t^2 \mathbf{a} \,, \tag{4}$$

from which velocities are updated using finite differences,

$$\mathbf{v}^{i+1} = \frac{1}{\Delta t}(\mathbf{x}^{i+1} - \mathbf{x}^i) \,. \tag{5}$$

Training The MeshGraphNets architecture outlined above can be trained in a supervised way using curated simulation data [30]. As shown by Grigorev et al. [14], MeshGraphNets can also be trained in a self-supervised way when using a loss function that directly encodes the governing physics. Using the optimization-based formulation of implicit Euler [26] as a basis, the per-time-step potential is defined as

$$E_{\text{IE}}(\mathbf{x}) = \frac{1}{2}\Delta \mathbf{v}^T \mathbf{M} \Delta \mathbf{v} + E_{\text{int}}(\mathbf{x}) + \mathbf{f}_{\text{ext}}^T \mathbf{x} \,, \tag{6}$$

where $\Delta \mathbf{v} = \frac{1}{\Delta t}(\mathbf{x} - \mathbf{x}_i) - \mathbf{v}_i$, $\mathbf{M}$ is the mass matrix, E_{int} is the internal elastic energy, and $\mathbf{f}_{\text{ext}}$ denotes external (i.e., non-conservative) forces. Each minimizer of this potential,

$$\mathbf{x}^{i+1} = \arg\min_{\mathbf{x}} \; E_{\text{IE}}(\mathbf{x}) \,, \tag{7}$$

is a solution to the implicit Euler equations. Total loss is defined as the sum of per-step losses over all samples in the corresponding training set.

Discussion Regardless of the training strategy, existing MeshGraphNets architectures output per-vertex accelerations. Since these accelerations have no *a priori* restrictions, they can induce arbitrary momentum changes. Our experiments show that spurious momentum changes are a general problem with existing MeshGraphNets architectures, particularly noticeable for ballistic motions. We conjecture that the reason for this nonphysical behavior is the fact that learning the relationship between vertex accelerations and global orientation of the system would require a dense sampling of rotational transformations and thus lead to excessive amounts of data. We propose a method that avoids this problem by restricting accelerations to the momentum-conserving subset by construction.

4. Method

The goal of our work is to develop a GNN architecture that accurately predicts the temporal evolution of linear and angular momentum. We start by separating discrete time steps into momentum-changing and momentum-conserving substeps (Sec. 4.1). We then introduce a basis for momentum-conserving impulses (Sec. 4.2) and explain how to implement this basis within the MeshGraphNets architecture (Sec. 4.3).

4.1. Momentum Decomposition

If a mechanical system is subject to *external*, non-conservative forces due to, e.g., gravity or contact, the total momentum of the system will not be conserved. Nevertheless, the *internal* forces arising, e.g., from an elastic potential will not induce any change in momentum. Using

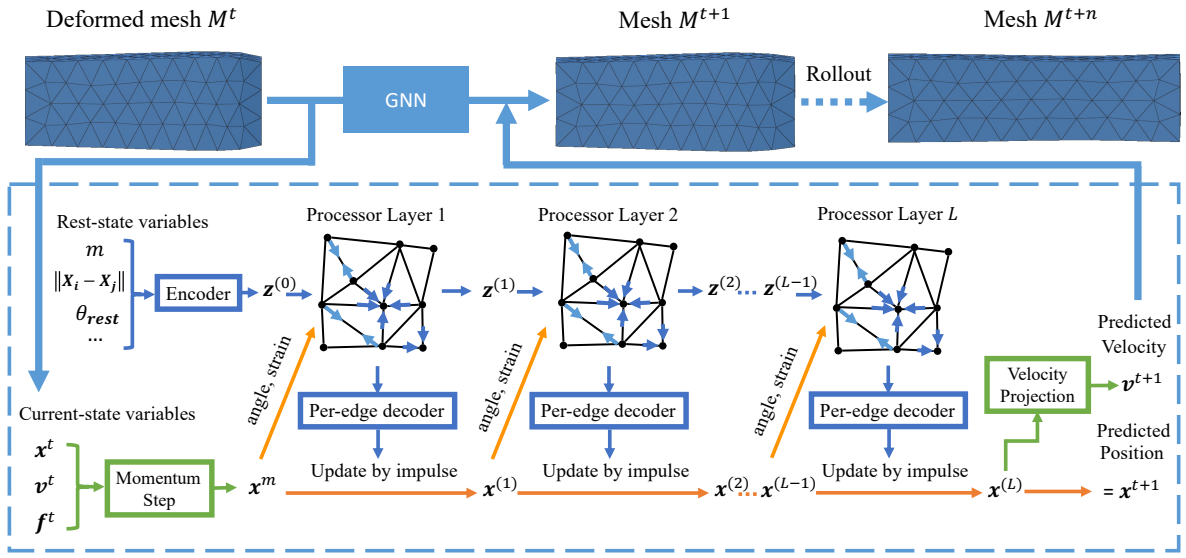

Figure 2. We visualize MomentumGNN and denote latent edge and node features jointly by $\mathbf{z} = \{\mathbf{s}_{ij}, \mathbf{r}_i\}$. Starting from a *momentum step* that integrates contributions from inertia and non-conservative external forces, our network predicts momentum-conserving stretching and bending impulses in each layer and sequentially updates position $\mathbf{x}^{(l)}$ to obtain the final prediction.

this observation, we decompose the integration of the system's equations of motions into two sub-steps. We first integrate the system forward in time using only inertia and non-conservative external forces. We then correct this *momentum step* by adding momentum-conserving impulses predicted by a neural network.

To implement this strategy, we recast the variational formulation of implicit Euler (6) into a two-step update. Following [7, 11], we define the momentum step as

$$\mathbf{x}^m = \mathbf{x}^i + \Delta t \mathbf{v}^i + \Delta t^2 \mathbf{M}^{-1} \mathbf{f}_{\text{ext}} \,. \tag{8}$$

End-of-step position are then obtained as the minimizer of the modified implicit Euler potential,

$$\min_{\mathbf{x}} \frac{1}{2\Delta t^2} ||\mathbf{x} - \mathbf{x}^m||^2_{\mathbf{M}} + E_{\text{int}}(\mathbf{x}) \,. \tag{9}$$

While this two-step version is equivalent to Eq. (6), it provides the basis for a clean separation between momentum-changing accelerations due to external forces and momentum-conserving accelerations due to internal forces. Specifically, we train our GNN to predict momentum-conserving accelerations $\mathbf{a}_{\text{mc}}$ such that the corresponding updated positions,

$$\mathbf{x}^{i+1} = \mathbf{x}^m + \Delta t^2 \mathbf{a}_{\text{mc}} \,, \tag{10}$$

minimize the modified potential (9) accumulated over all samples in the training set. While the end-of-step positions $\mathbf{x}^{i+1}$ are generally not a true (i.e., unconstrained) minimizer of (9), they constitute the momentum-conserving (i.e., constrained) solution that is closest to the implicit Euler step. With this basis laid out, we next describe how to build momentum-conservation into our GNN architecture.

4.2. A Basis for Momentum-Conserving Impulses

To restrict our GNN to predict momentum-conserving accelerations by construction, we observe that the net change in linear momentum $\mathbf{p}$ due to a given set of vertex accelerations $\mathbf{a}_i$ is

$$\frac{d\mathbf{p}}{dt} = \sum_i \frac{d\mathbf{p}_i}{dt} = \sum_i m_i \frac{d\mathbf{v}_i}{dt} = \sum_i m_i \mathbf{a}_i \,. \tag{11}$$

The last term in the above expression suggests that one way of achieving momentum conservation is to impose constraints requiring the mass-weighted sum of accelerations to vanish. Since enforcing hard constraints during training is difficult and can slow down learning significantly, we therefore seek to find a basis for the space of admissible impulses that guarantees momentum conservation by construction.

Momentum-Conserving Basis If we design impulses to only affect quantities invariant to translations and rotations, then such impulses cannot change linear or angular momentum. Let q be an intrinsic quantity with Taylor expansion

$$q(\mathbf{x} + \Delta\mathbf{x}) = q(\mathbf{x}) + \nabla q(\mathbf{x}) \cdot \Delta\mathbf{x} + \ldots \tag{12}$$

Invariance implies that the first-order term vanishes for any $\Delta\mathbf{x}$ corresponding to rigid transformation, i.e., $\nabla q(\mathbf{x}) \cdot \Delta\mathbf{x} = 0$. The gradient of each of the six momentum components (three translational, three rotational), computed with respect to vertex positions, corresponds exactly to a rigid transformation direction (along xyz and around xyz). Therefore, ∇q is orthogonal to each momentum gradient, and impulses along ∇q preserve both linear and angular momentum.

Gradients of invariant quantities thus provide a basis for momentum-conserving impulses, but what quantities should be used? As a natural choice in our context, finite element discretizations and discrete differential geometry operators offer a large range of deformation measures that are invariant to rigid rotations and translations, such as areas and volumes. As the most basic option, we build our basis on edge lengths. To this end, we endow each edge with a single scalar w_{ij} that describes the magnitude of an impulse along the gradient of edge length,

$$\frac{\partial l(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_i} = \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|} \quad \text{and} \quad \frac{\partial l(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_j} = -\frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}\,, \tag{13}$$

which, for each vertex, is simply the normalized edge vector. Edge impulses are then translated into pairs of per-vertex impulses as

$$\Delta\mathbf{p}_i = w_{ij}\frac{\partial l(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_i} \quad \text{and} \quad \Delta\mathbf{p}_j = -\Delta\mathbf{p}_i\,. \tag{14}$$

We collect corresponding basis vectors for each edge in a matrix $\mathbf{B}$ and express momentum-conserving impulses as $\Delta\mathbf{p} = \mathbf{B}\mathbf{w}$, where $\mathbf{w} \in \mathbb{R}^{|E|}$ is a vector of per-edge impulse magnitudes. It is clear by construction that all impulse generated in this way are momentum-conserving. In the supplemental material, we show that this basis is likewise complete in the sense that any momentum-preserving impulse can be generated in this way.

The above formulation directly extends to the 3D case of volumetric objects represented using tetrahedral meshes. However, the case of triangle meshes embedded in 3D space—corresponding to thin shell materials such as cloth and paper—requires special attention.

Extensions to Thin Shells While 2D elastic sheets resist local changes in length, 3D thin shell materials must also respond to local changes in curvature, i.e., bending deformations. Our per-edge basis can generate any momentum-conserving in-plane responses, but it cannot create out-of-plane motion and must therefore be extended. While different options are available in the literature, we follow Grinspun et al. [16] and use the dihedral angle formed by two edge-adjacent triangles. As indicated in the inset figure, the dihedral angle is determined by the four vertex positions of its two adjacent triangles. To obtain momentum-conserving impulses that only change the dihedral angle (to first order), we endow each edge with an additional impulse magnitude variable w^b_{ij} and define the four per-vertex impulses $\Delta\mathbf{p}_i = w^b_{ij}\frac{\partial\theta}{\partial\mathbf{x}_i}$. Closed form expressions for the corresponding derivatives can be found in [45]. Since the dihedral angle is invariant to rigid transformations, it follows that these impulses conserve linear and angular momentum.

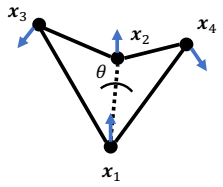

Figure 3. Dihedral angle θ defined by two edge-adjacent triangles. Per-vertex gradients $\frac{\partial\theta}{\partial\mathbf{x}_i}$ are shown in blue.

4.3. MomentumGNN

Implementing our momentum-conserving impulses requires only slight modifications to MeshGraphNets that replace per-vertex decoders with per-edge decoders.

Per-Edge Decoders Specifically, we define an edge decoder MLP f^{stretch} that takes as input max-pooled edge features and predicts an impulse magnitude,

$$w^{\text{stretch}}_{ij} = f^{\text{stretch}}(\max(\mathbf{s}_{ij}, \mathbf{s}_{ji}))\,, \tag{15}$$

from which we then compute per-vertex impulses as

$$\Delta\mathbf{p}^{\text{stretch}}_{ij} = w^{\text{stretch}}_{ij}\frac{\partial l_{ij}}{\partial \mathbf{x}_i}\,. \tag{16}$$

In an analogous way, we define an edge decoder MLP f^{bend} that predicts bending impulse magnitudes,

$$w^{\text{bend}}_{ij} = f^{\text{bend}}(\max(\mathbf{e}_{ij}, \mathbf{e}_{ji}))\,, \tag{17}$$

from which we compute per-vertex impulses,

$$\Delta\mathbf{p}^{\text{bend}}_{ij} = w^{\text{bend}}_{ij}\frac{\partial\theta}{\partial\mathbf{x}_i}\,, \tag{18}$$

along the gradient of the dihedral angle θ_{ij} for edge $\mathbf{e}_{ij}$. For each node, we sum the per-vertex contributions of all incident edges and transform the resulting impulses into nodal accelerations as

$$\mathbf{a}_i = \frac{1}{m_i\Delta t}\sum_j \left(\Delta\mathbf{p}^{\text{stretch}}_{ij} + \Delta\mathbf{p}^{\text{bend}}_{ij}\right)\,. \tag{19}$$

Position Updates Having replaced the per-vertex decoders with our momentum-conserving per-edge decoders, we can train the resulting GNN in the usual way. While the position updates predicted by this network conserve momentum, the space of possible updates is limited by the fact that all impulse directions are computed using the geometry of the momentum step $\mathbf{x}^m$. To illustrate this limitation, we consider the example of a 2D mass-spring chain as shown in Fig. 4. An external force acting on vertex $\mathbf{x}_1$ leads to the momentum step predicting an updated position $\mathbf{x}^m_1$ whereas the positions for the other nodes remain unchanged. When computing per-edge impulses based on this geometry (*orange*), neither $\mathbf{x}_3$ nor $\mathbf{x}_4$ can be displaced vertically—but such vertical displacement is required to reach the implicit Euler solution indicated in green in Fig. 4.

While smaller step sizes will alleviate this problem, its root cause is the fact that, while each message passing layer adjusts all impulse magnitudes, the direction of these impulses is kept constant across all steps. A natural way of increasing the space of momentum-conserving impulses is therefore to update the geometry with the current impulses after each message passing step. Since each such per-layer update conserves momentum, any sequence of per-layer updates will likewise conserve momentum. We implement this layered position update strategy as shown in Figure 2. The resulting *MomentumGNN* architecture decodes and updates positions for each layer (indicated in orange). The positions output by the last layer $\mathbf{x}^{(L)}$ are used as the final prediction $\mathbf{x}^{i+1}$. In order to feed these position updates back into the message passing process, we explicitly recompute dihedral angles θ_{ij}^l and edge strains ε_{ij}^l from the intermediate positions $\mathbf{x}^{(l)}$ and modify the edge MLP from Eq. (1) as

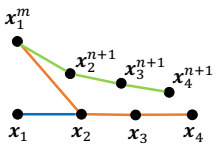

Figure 4. A 2D mass-spring chain.

$$\mathbf{m}_{ij}^{l+1} \leftarrow f^{\text{edge}}(\mathbf{s}_{ij}, \mathbf{r}_i, \mathbf{r}_j, \theta_{ij}^l, \varepsilon_{ij}^l) \,. \tag{20}$$

Velocity Projection Our network predicts momentum-conserving position updates, but it does not directly return updated velocities. While one could use the finite difference approximation of (5) to compute velocities $\mathbf{v}^{\text{FD}}$, those velocities would generally not conserve angular momentum when evaluated on the updated positions. We therefore compute velocities by finding the smallest correction to $\mathbf{v}_i^{\text{FD}}$ that preserves momentum. We cast this problem as a convex quadratic program,

$$\min_{\Delta\mathbf{v}} \frac{1}{2}\|\Delta\mathbf{v}\|^2 \quad \text{s.t.} \quad \sum_i m_i(\mathbf{v}_i^{\text{FD}} + \Delta\mathbf{v}_i) = \mathbf{p}^t \tag{21}$$

$$\sum_i m_i \mathbf{x}_i \times (\mathbf{v}_i^{\text{FD}} + \Delta\mathbf{v}_i) = \mathbf{L}^t \,. \tag{22}$$

The Lagrangian of this optimization problem is

$$L = \frac{1}{2}\|\Delta\mathbf{v}\|^2 + \boldsymbol{\lambda}^T \mathbf{C}(\mathbf{v}^{\text{FD}} + \Delta\mathbf{v}) \,, \tag{23}$$

where $\mathbf{C}$ summarizes all 6 constraints. The first-order optimality conditions require the gradient of the Lagrangian with respect to $\mathbf{v}$ and $\boldsymbol{\lambda}$ to vanish, which leads to the linear system

$$\begin{bmatrix} \mathbf{M} & \nabla\mathbf{C}^T \\ \nabla\mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{v} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -\mathbf{C}(\mathbf{v}^{\text{FD}}) \end{bmatrix} \,. \tag{24}$$

For efficiency, we first compute Lagrange multipliers $\boldsymbol{\lambda}$ by solving the 6×6 linear system

$$\nabla\mathbf{C}\mathbf{M}^{-1}\nabla\mathbf{C}^T\boldsymbol{\lambda} = \mathbf{C}(\mathbf{v}^{\text{FD}}) \,, \tag{25}$$

Table 1. Experimental setup.

Experiment	# of GNN layers	# of vertices	# of cells	Timestep
Falling cloth (torus)	10	2119	4068	11
Swinging and falling cloth	20	514	946	21
Shooting baskets	20	993	4078	17
Bouncing tennis ball	20	396	1480	12
Armadillo	20	10281	55091	82

and then compute momentum-corrected velocities as

$$\mathbf{v}^{t+1} = \mathbf{v}^{\text{FD}} - \mathbf{M}^{-1}\nabla\mathbf{C}^T\boldsymbol{\lambda} \,. \tag{26}$$

5. Results

5.1. Training scheme

We implement our method in PyTorch [28] and perform training and testing on a single RTX 4090 GPU. Each GNN layer is implemented by a 2-layer MLP with 128 latent dimensions, layer normalization, and Gaussian Error Linear Units (GELU) as activation functions. We use discrete shells and constant strain triangles with a Saint Venant-Kirchhoff material for cloth and a Neo-Hookean material for volumetric solids. We summarize our experiment setup in Table 1.

We generate training data procedurally using a Python script that collects sequences of simulation snapshots of random cuboids and sheets recovering from initial deformations. We randomly sample tetrahedral cuboids and sheets of different sizes using the Gmsh package. We then randomly deform cuboids by affine transformations and sheets along Bezier curves. Cuboids and sheets are assigned random initial velocities.

Following recent work [5, 14, 37, 39], we train our GNN in a self-supervised way using a physics-based loss function that sums up the per-step potential from (9). We use the Adam optimizer with a learning rate 1×10^{-5} and add two types of noise to the training data. The first training noise is applied to all vertices and sampled from a normal distribution $\mathcal{N}(0, \sigma)$ ($\sigma = 0.001$ for cloth and $\sigma = 6 \times 10^{-4}$ for solid). The second training noise is applied to a random local neighborhood within a radius of $1/4$ (cloth) or $1/2$ (solid) of the longest edge of the sheet or cuboid. We sample the same direction for all vertices and independently sample magnitudes from $\mathcal{U}(0, 0.05)$. For a fair comparison, when training self-supervised MeshGraphNets baselines, we additionally sample random rotations for both rest and deformed configurations, as they are not equivariant to rotations.

We collect additional data for the pinned cloth. We randomly sample gravity-like external forces of uniform direction and a magnitude from a uniform distribution $\mathcal{U}(0, 20)$. We collect the snapshots by releasing the pinned cloth from its rest state. We finetune pre-trained cloth checkpoints by jointly training on both pinned and unpinned data.

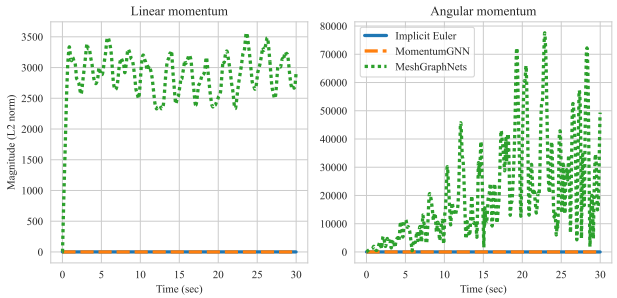

Figure 5. MomentumGNN conserves linear and angular momentum of the Armadillo, while MeshGraphNets quickly leads to an explosion.

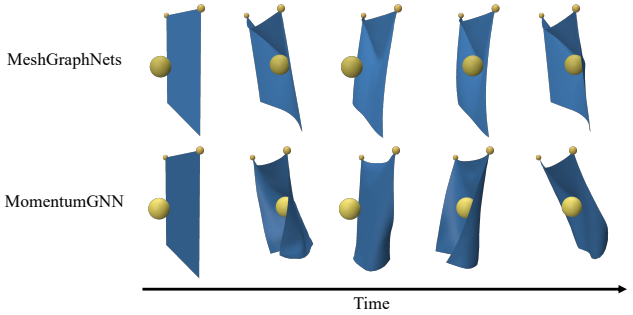

Figure 6. We simulate a hanging cloth interacting with a moving ball.

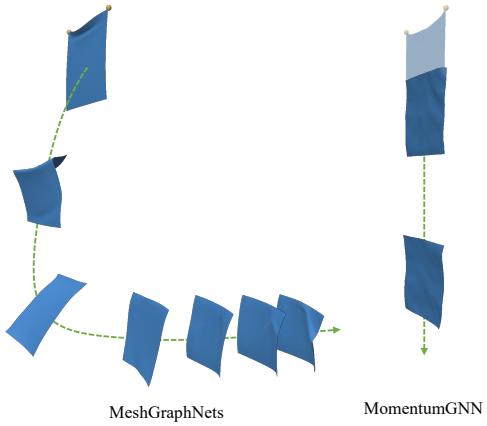

Figure 7. We release the pinned-vertex constraints for the hanging cloth and simulate its falling motion.

5.2. Swinging and Falling Cloth

We begin our evaluation by comparing MomentumGNN (our method) with MeshGraphNets on cloth simulation. We train MeshGraphNets using L2 supervision, following their distributed codebase and dataset. As shown in Figure 6, both models achieve comparable quality in simulating a swinging cloth. However, MomentumGNN produces more dynamic and realistic motion, attributed to its explicit conservation of momentum. The limitations of MeshGraphNets become apparent under slight scene modifications. In Figure 7, we simulate a falling cloth by removing the pins. MeshGraphNets does not generalize to this new scenario. Rather than falling vertically under gravity, the cloth drifts horizontally and exhibits unnatural momentum. This discrepancy is more evident in the supplementary video. In contrast, MomentumGNN accurately captures the expected physical behavior. It accurately predicts the evolution of momentum—in particular zero linear momentum in lateral directions—enabling the cloth to fall straight downward while naturally forming wrinkles during descent.

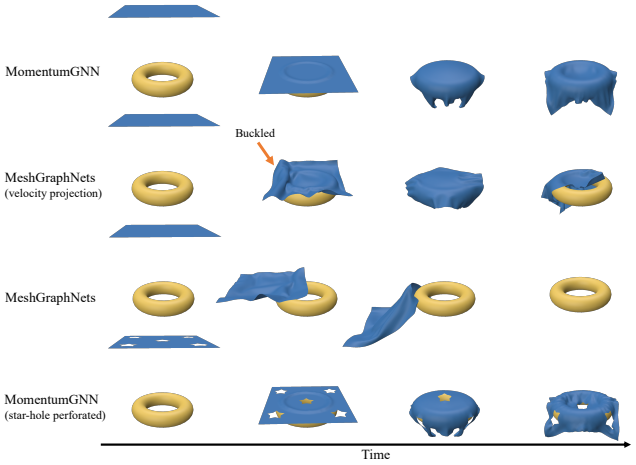

Figure 8. We drop a cloth onto a torus and simulate the cloth dynamics. We also showcase an extension to star holes (genus 5).

5.3. Falling Cloth on Torus

To further validate this observation, we retrain MeshGraphNets on the same dataset as our model using self-supervision. When dropping a cloth onto a torus (Figure 8), MeshGraphNets again exhibits drift and does not drape on the torus. We further introduce an enhanced baseline by replacing MeshGraphNets' finite-difference velocity with our velocity projection formulation. This modification mitigates drift but introduces buckling artifacts, indicating that velocity projection alone is insufficient, and highlighting the importance of coherent momentum modeling throughout the architecture. In contrast, MomentumGNN produces stable, physically-plausible trajectories, accurately capturing both free-fall dynamics and visually plausible folding after contact, and generalizes to perforated star shapes.

5.4. Shooting Baskets

We evaluate MomentumGNN on elastic solid simulation using a basket-shooting scenario, where a robotic hand throws a ball toward a hoop (Figure 9). As shown in the figure and the supplementary video, MomentumGNN faithfully captures the ball's natural trajectory, successfully completing the shot. In contrast, MeshGraphNets fails to reproduce the correct motion, with the ball veering off course due to spurious momentum injection. Augmenting MeshGraphNets with velocity projection partially corrects the trajectory, directing the ball toward the basket, but the shot ultimately strikes the backboard and misses.

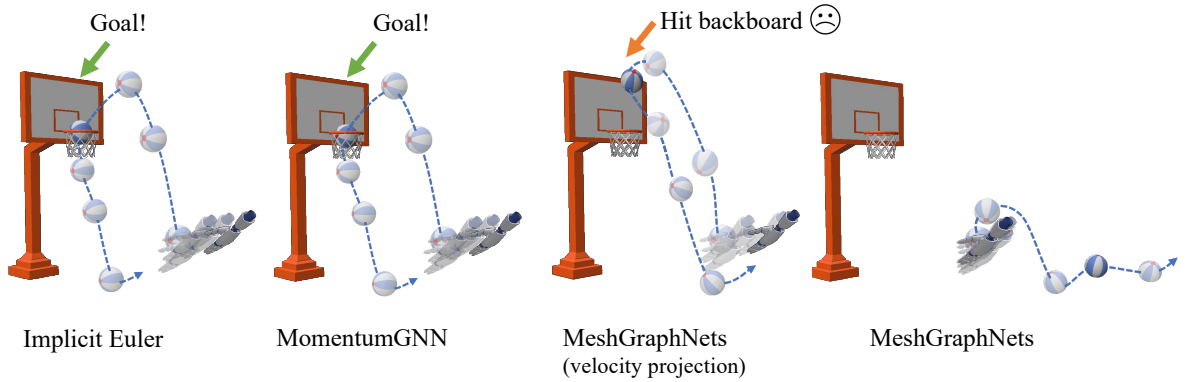

Figure 9. We simulate a basket-shooting scene, where a robotic hand throws a beach ball to the basket. Both MomentumGNN and Implicit Euler successfully complete the throw, whereas MeshGraphNets fails. The improvement with the velocity projection directs the ball toward the basket but does not score.

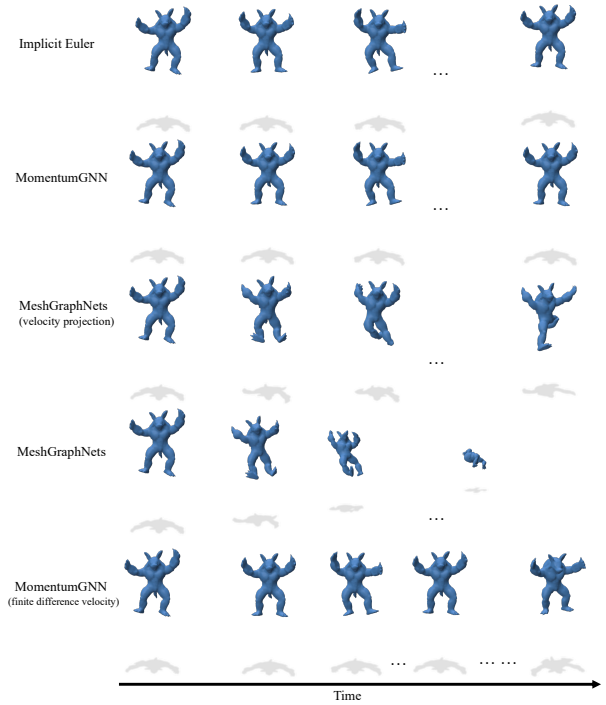

Figure 10. We simulate an Armadillo released from a nonlinear deformed pose with zero initial velocity and no external forces.

5.5. Armadillo

Despite being trained exclusively on simple cuboids, MomentumGNN generalizes effectively to complex shapes, such as Armadillo (see Figure 10). The Armadillo is initially deformed into a nonlinear pose with zero initial velocity and then allowed to evolve freely without external forces. MomentumGNN accurately captures the natural swinging motion of the arms and legs. In contrast, MeshGraphNets exhibits drift, rotating the Armadillo arbitrarily out of the scene, violating momentum conservation. Although augmenting MeshGraphNets with velocity projection helps maintain the Armadillo within the scene, it still is incapable of producing physically correct dynamics. These differences are best seen in the supplementary video.

We also provide a quantitative evaluation by computing the linear and angular momentum of the Armadillo, plotted in Figure 5. Since the Armadillo begins at rest with no external forces applied, both momenta should remain zero throughout the simulation. As illustrated in Figure 5, MeshGraphNets exhibits rapid growth in momentum, diverging from the expected behavior. In contrast, MomentumGNN faithfully conserves momentum with only a modest computational overhead compared to MeshGraphNets (12 fps vs. 14 fps), while offering a substantial performance improvement over implicit Euler (2 fps).

Finally, we conduct a brief ablation study by replacing our momentum-consistent velocity with the finite difference velocity during simulation. As shown in Figure 10, MomentumGNN still effectively captures the swinging motion of the arms and legs, but, as expected, produces gradual deviations in orientation.

6. Conclusion

We presented a graph-based neural architecture for simulating deformable objects that explicitly conserves linear and angular momentum. Our method predicts momentum-conserving accelerations by modeling stretching and bending impulses via per-edge decoders. To enhance the representational capacity of the network, we introduced a layer-by-layer update scheme that allows richer hierarchical processing of deformation dynamics. Trained on canonical geometries such as sheets and cuboids, our network generalizes effectively to a wide array of challenging scenarios involving cloth and soft solids. Experimental results showcased our model's ability to capture rich deformation patterns while preserving the momentum of the system.

Limitations and future work. We did not explore hierarchical message passing, which we consider a complementary and promising direction. Moreover, although our networks can, in principle, support variations in material parameters, we have not conducted any experiments in this direction; integrating learned constitutive laws [24] from observational data represents an exciting avenue to discover novel material behaviors within a unified framework.

References

[1] Kelsey Allen, Tatiana Lopez-Guevara, Kimberly L Stachenfeld, Alvaro Sanchez Gonzalez, Peter Battaglia, Jessica B Hamrick, and Tobias Pfaff. Inverse design for fluid-structure interactions using graph network simulators. In *Advances in Neural Information Processing Systems*, pages 13759–13774. Curran Associates, Inc., 2022. 2

[2] Hugo Bertiche, Meysam Madadi, and Sergio Escalera. Cloth3d: clothed 3d humans. In *European Conference on Computer Vision*, pages 344–359. Springer, 2020. 2

[3] Hugo Bertiche, Meysam Madadi, and Sergio Escalera. Pbns: physically based neural simulation for unsupervised garment pose space deformation. *ACM Trans. Graph.*, 40(6), 2021. 2

[4] Hugo Bertiche, Meysam Madadi, Emilio Tylson, and Sergio Escalera. Deepsd: Automatic deep skinning and pose space deformation for 3d garment animation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5471–5480, 2021. 2

[5] Hugo Bertiche, Meysam Madadi, and Sergio Escalera. Neural cloth simulation. *ACM Trans. Graph.*, 41(6), 2022. 2, 6

[6] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4), 2014. 1

[7] George E. Brown, Matthew Overby, Zahra Forootaninia, and Rahul Narain. Accurate dissipative forces in optimization integrators. *ACM Trans. Graph.*, 37(6), 2018. 4

[8] Barbara Cutler, Julie Dorsey, Leonard McMillan, Matthias Müller, and Robert Jagnow. A procedural approach to authoring solid models. *ACM Trans. Graph.*, 21(3):302–311, 2002. 1

[9] Luca De Luigi, Ren Li, Benoit Guillard, Mathieu Salzmann, and Pascal Fua. DrapeNet: Garment Generation and Self-Supervised Draping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. 2

[10] Lawson Fulton, Vismay Modi, David Duvenaud, David I. W. Levin, and Alec Jacobson. Latent-space dynamics for reduced deformable simulation. *Computer Graphics Forum*, 2019. 2

[11] T. F. Gast and C. Schroeder. Optimization integrator for large time steps. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 31–40, Goslar, DEU, 2015. Eurographics Association. 4

[12] Rini Jasmine Gladstone, Helia Rahmani, Vishvas Suryakumar, Hadi Meidani, Marta D'Elia, and Ahmad Zareei. Mesh-based gnn surrogates for time-independent pdes. *Scientific reports*, 14(1):3394, 2024. 2

[13] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. 2

[14] Artur Grigorev, Bernhard Thomaszewski, Michael J Black, and Otmar Hilliges. Hood: Hierarchical graphs for generalized modelling of clothing dynamics. *Computer Vision and Pattern Recognition (CVPR)*, 2023. 1, 2, 3, 6

[15] Artur Grigorev, Giorgio Becherini, Michael Black, Otmar Hilliges, and Bernhard Thomaszewski. Contourcraft: Learning to resolve intersections in neural multi-garment simulations. In *ACM SIGGRAPH 2024 Conference Papers*, New York, NY, USA, 2024. Association for Computing Machinery. 1, 2

[16] Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 62–67, Goslar, DEU, 2003. Eurographics Association. 5

[17] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 9–20, 1998. 2

[18] Derrick Hines and Philipp Bekemeyer. Graph neural networks for the prediction of aircraft surface pressure distributions. *Aerospace Science and Technology*, 137:108268, 2023. 2

[19] Zorah Lahner, Daniel Cremers, and Tony Tung. Deepwrinkles: Accurate and realistic clothing modeling. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2

[20] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4), 2020. 1

[21] Emmanuel Ian Libao, Myeongjin Lee, Sumin Kim, and Sung-Hee Lee. Meshgraphnetrp: Improving generalization of gnn-based cloth simulation. In *Proceedings of the 16th ACM SIGGRAPH Conference on Motion, Interaction and Games*, New York, NY, USA, 2023. Association for Computing Machinery. 1, 2

[22] Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Xiang Chen, Kun Zhou, and Yin Yang. Nnwarp: Neural network-based nonlinear deformation. *IEEE Transactions on Visualization and Computer Graphics*, 26(4):1745–1759, 2020. 2

[23] Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations (ICLR)*, 2019. 2

[24] Pingchuan Ma, Peter Yichen Chen, Bolei Deng, Joshua B Tenenbaum, Tao Du, Chuang Gan, and Wojciech Matusik. Learning neural constitutive laws from motion observations for generalizable pde dynamics. In *International Conference on Machine Learning*, pages 23279–23300. PMLR, 2023. 8

[25] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, page 49–54, New York, NY, USA, 2016. Association for Computing Machinery. 1

[26] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. In *ACM*

SIGGRAPH 2011 Papers, New York, NY, USA, 2011. Association for Computing Machinery. 3

[27] Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowliswharan. Cfdnet: a deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM International Conference on Supercomputing*, New York, NY, USA, 2020. Association for Computing Machinery. 2

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, pages 8024–8035, 2019. 6

[29] Chaitanya Patel, Zhouyingcheng Liao, and Gerard Pons-Moll. Tailornet: Predicting clothing in 3d as a function of human pose, shape and garment style. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[30] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations (ICLR)*, 2021. 1, 2, 3

[31] Lukas Prantl, Benjamin Ummenhofer, Vladlen Koltun, and Nils Thuerey. Guaranteed conservation of momentum for learning particle-based fluid dynamics. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2022. Curran Associates Inc. 2

[32] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019. 2

[33] Cristian Romero, Dan Casas, Jesús Pérez, and Miguel Otaduy. Learning contact corrections for handle-based subspace dynamics. *ACM Trans. Graph.*, 40(4), 2021. 2

[34] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4470–4479. PMLR, 2018. 2, 3

[35] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. In *Proceedings of the NeurIPS 2019 Workshop on Machine Learning and the Physical Sciences*, 2019. 2

[36] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020. 2

[37] Igor Santesteban, Miguel A Otaduy, and Dan Casas. Snug: Self-supervised neural dynamic garments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8140–8150, 2022. 2, 6

[38] Vinay Sharma and Olga Fink. A physics-informed graph neural network conserving linear and angular momentum for dynamical systems. *Nature Communications*, 2026. 2

[39] Nicholas Sharp, Cristian Romero, Alec Jacobson, Etienne Vouga, Paul Kry, David I.W. Levin, and Justin Solomon. Data-free learning of reduced-order kinematics. In *ACM SIGGRAPH 2023 Conference Proceedings*, New York, NY, USA, 2023. Association for Computing Machinery. 2, 6

[40] Siyuan Shen, Yin Yang, Tianjia Shao, He Wang, Chenfanfu Jiang, Lei Lan, and Kun Zhou. High-order differentiable autoencoder for nonlinear model reduction. *ACM Trans. Graph.*, 40(4), 2021. 2

[41] Andrew Sosanya and Sam Greydanus. Dissipative hamiltonian neural networks: Learning dissipative and conservative dynamics separately. In *International Conference on Learning Representations (ICLR)*, 2022. 2

[42] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, page 205–214, New York, NY, USA, 1987. Association for Computing Machinery. 1

[43] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58 (1):25–36, 2020. 2

[44] Jiahong Wang, Yinwei Du, Stelian Coros, and Bernhard Thomaszewski. Neural modes: Self-supervised learning of nonlinear modal subspaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 23158–23167, 2024. 2

[45] Max Wardetzky, Miklós Bergou, David Harmon, Denis Zorin, and Eitan Grinspun. Discrete quadratic curvature energies. *Computer Aided Geometric Design*, 24(8):499–518, 2007. Discrete Differential Geometry. 5

[46] Mianlun Zheng, Yi Zhou, Duygu Ceylan, and Jernej Barbic. A deep emulator for secondary motion of 3d characters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5932–5940, 2021. 1, 2