Pontifícia Universidade Católica
do Rio de Janeiro

Matheus Kerber Venturelli

Fast and Accurate Simulation of Deformable Solid Dynamics on Coarse Meshes

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática, do Departamento de Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Waldemar Celes Filho

Rio de Janeiro
March 2024

Pontifícia Universidade Católica
do Rio de Janeiro

Matheus Kerber Venturelli

Fast and Accurate Simulation of Deformable Solid Dynamics on Coarse Meshes

Dissertation presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee:

Prof. Waldemar Celes Filho
Advisor
Departamento de Informática – PUC-Rio

Prof. Ivan Fábio Mota de Menezes
Departamento de Engenharia Mecânica – PUC-Rio

Prof. José Alberto Rodrigues Pereira Sardinha
Departamento de Informática – PUC-Rio

Rio de Janeiro, March 15th, 2024

Matheus Kerber Venturelli

Graduated in Computer Engineering from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Works at the Tecgraf Institute of PUC-Rio developing software for simulating the physics of oil reservoirs.

Bibliographic data

Kerber Venturelli, Matheus

Fast and Accurate Simulation of Deformable Solid Dynamics on Coarse Meshes / Matheus Kerber Venturelli; advisor: Waldemar Celes Filho. – 2024.

54 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2024.

Inclui bibliografia

1. Computer Science – Teses. 2. Elementos Finitos. 3. Simulação Numérica. 4. Simulação Baseada em Dados. 5. Aprendizado Profundo. 6. Redes Neurais de Grafo. 7. Dinâmica de Sólidos Deformáveis. 8. Modelo Aproximado. I. Celes Filho, Waldemar. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my parents, friends, and family
for always trusting and motivating me.

Acknowledgments

To CAPES, CNPq, PUC-Rio, and the Tecgraf Institute, for the funds and grants provided, without which this work could not have been carried out.

I thank my colleagues at the Tecgraf Institute for their support, help, and believing in my ability. To Professor Waldemar Celes, my advisor, for guiding and supporting me on this journey. To Jônatas Wehrmann, for his teachings at the beginning of my master's course. To Professor Marco Molinaro, for always supporting and guiding me throughout my academic career. To Professors Ivan Menezes and Paulo Ivson for giving me important advice during my research.

I also thank my parents, family, friends, and girlfriend, Vitória Velloso, for always supporting, motivating, and believing in me during the most difficult moments.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

Abstract

Kerber Venturelli, Matheus; Celes Filho, Waldemar (Advisor). **Fast and Accurate Simulation of Deformable Solid Dynamics on Coarse Meshes**. Rio de Janeiro, 2024. 54p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This thesis introduces a novel hybrid simulator that combines a numerical Finite Element (FE) Partial Differential Equation solver with a Message Passing Neural Network (MPNN) to perform simulations of deformable solid dynamics on coarse meshes. Our work aims to provide accurate simulations with an error comparable to that obtained with more refined meshes in FE discretizations while maintaining computational efficiency by using an MPNN component that corrects the numerical errors associated with using a coarse mesh. We evaluate our model focusing on accuracy, generalization capacity, and computational speed compared to a reference numerical solver that uses 64 times more refined meshes. We introduce a new dataset for this comparison, encompassing three numerical benchmark cases: (i) free deformation after an initial impulse, (ii) stretching, and (iii) torsion of deformable solids. Based on simulation results, the study thoroughly discusses our method's strengths and weaknesses. The study shows that our method corrects an average of 95.4% of the numerical error associated with discretization while being up to 88 times faster than the reference solver. On top of that, our model is fully differentiable in relation to loss functions and can be embedded into a neural network layer, allowing it to be easily extended by future work. Data and code are made available on <https://github.com/Kerber31/fast_coarse_FEM> for further investigations.

Keywords

Finite Elements; Numerical Simulation; Data-driven Simulation; Deep Learning; Graph Neural Networks; Deformable Solid Dynamics; Surrogate Model.

Resumo

Kerber Venturelli, Matheus; Celes Filho, Waldemar. **Simulação Rápida e Precisa de Dinâmica de Sólidos Deformáveis em Malhas Pouco Refinadas**. Rio de Janeiro, 2024. 54p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Esta dissertação introduz um simulador híbrido inovador que combina um resolvedor de Equações Diferenciais Parciais (EDP) numérico de Elementos Finitos (FE) com uma Rede Neural de Passagem de Mensagens (MPNN) para realizar simulações de dinâmicas de sólidos deformáveis em malhas pouco refinadas. Nosso trabalho visa fornecer simulações precisas com um erro comparável ao obtido com malhas mais refinadas em discretizações FE, mantendo a eficiência computacional ao usar um componente MPNN que corrige os erros numéricos associados ao uso de uma malha menos refinada. Avaliamos nosso modelo focando na precisão, capacidade de generalização e velocidade computacional em comparação com um solucionador numérico de referência que usa malhas 64 vezes mais refinadas. Introduzimos um novo conjunto de dados para essa comparação, abrangendo três casos de referência numéricos: (i) deformação livre após um impulso inicial, (ii) alongamento e (iii) torção de sólidos deformáveis. Baseado nos resultados de simulação, o estudo discute as forças e fraquezas do nosso método. O estudo mostra que nosso método corrige em média 95,4% do erro numérico associado à discretização, sendo até 88 vezes mais rápido que o solucionador de referência. Além disso, nosso modelo é totalmente diferenciável em relaçao a funções de custo e pode ser incorporado em uma camada de rede neural, permitindo que seja facilmente estendido por trabalhos futuros. Dados e código estão disponíveis em <https://github.com/Kerber31/fast_coarse_FEM> para investigações futuras.

Palavras-chave

Elementos Finitos; Simulação Numérica; Simulação Baseada em Dados; Aprendizado Profundo; Redes Neurais de Grafo; Dinâmica de Sólidos Deformáveis; Modelo Aproximado.

Table of contents

List of figures

List of tables

List of algorithms

List of Abreviations

PDE – Partial Differential Equation

FEM – Finite Element Method

FE – Finite Element

CFD – Computational Fluid Dynamics

LES – Large Eddy Simulation

PD – Projective Dynamics

DiffPD – Differentiable Projective Dynamics

ANN – Artificial Neural Network

FCNN – Fully Connected Neural Network

DNN – Dense Neural Network

CNN – Convolutional Neural Network

GNN – Graph Neural Network

GCNN – Graph Convolutional Neural Network

MPNN – Message Passing Neural Network

The most beautiful experience we can have is the mysterious. It is the fundamental emotion that stands at the cradle of true art and true science.

Albert Einstein, *The World As I See It.*

1
Introduction

In the realm of physics and engineering, challenges are often modeled through partial differential equations (PDEs), such as the Elastodynamic equation (GONZALEZ; STUART, 2008), or the Navier–Stokes equations (ANDERSON, 2009). Typically, analytical solutions to these PDEs are not possible, thus requiring computational approximation methods. The accuracy of these methods is highly dependent on the quality of the discretization of the domain. Therefore, as coarser discretizations tend to be less accurate, the computational cost for tight error tolerances can be exceedingly high. This motivates the creation of surrogate models capable of delivering faster solutions with reasonable accuracy.

Previous work has shown that Artificial Neural Networks (ANN) are universal approximators (HORNIK; STINCHCOMBE; WHITE, 1989; CYBENKO, 1989); therefore, extensive research has been made into approximating the solution of PDEs using ANNs (URIARTE; PARDO; OMELLA, 2022; MA et al., 2023; ZHANG; CAI; ZHANG, 2023; BERG; NYSTRöM, 2018; SIRIGNANO; SPILIOPOULOS, 2018), which are also known as data-driven PDE solvers. Two main approaches regarding applying ANNs to the solution of PDEs can be found across multiple disciplines of physics. The first one is purely data-driven (FORTUNATO et al., 2022; CAO et al., 2023), and the other is the integration of data-driven methods into the pipeline of a numerical method to create hybrid solvers (MCGREIVY; HAKIM, 2023; PESTOURIE et al., 2023; KOCHKOV et al., 2021; SIRIGNANO; MACART; FREUND, 2020; GONZáLEZ; CHINESTA; CUETO, 2019). Neural Network models present the advantage of being highly parallelizable on GPUs (PASZKE et al., 2019; ABADI et al., 2015), and they can learn to represent nonlinear functions (SHEN et al., 2021).

Further work has been done to create learning representations suitable to different types of data. Convolutional Neural Networks (CNNs) (LECUN et al., 1998) were made to learn functions over structured data, like images (RONNEBERGER; FISCHER; BROX, 2015; REDMON et al., 2016). On the other hand, Graph Convolutional Neural Networks (GCNNs) (KIPF; WELLING, 2017) and Message Passing Neural Networks (MPNN) (GILMER et al., 2017) were developed to learn functions over unstructured graph data, such as geometrical data (MILANO et al., 2020; SHAKIBAJAHROMI; KIM; BREEN, 2024; LIU; NICKEL; KIELA, 2019).

We build upon these works by combining the generalization capacity of MPNNs with a differentiable simulator to create a fully differentiable method that can be embedded into a neural network layer. Our contributions can be summarized as follows:

1. A hybrid method that is reusable on meshes and parameters outside those used in training, and combines graph networks' learning and generalization capabilities with the fast forward and backward times of Differentiable Projective Dynamics (DiffPD) (DU et al., 2021), an approximated numerical solver that uses a surrogate energy function, described in Section 3.1, to approximate the system nonlinearities.;

2. A method capable of performing different kinds of simulations on deformable solid dynamics while correcting an average of 95.4% of the numerical error associated with discretization and being up to 88 times faster when compared to an instance of DiffPD using meshes 64 times more refined;

3. A fully differentiable model that can be embedded into a neural network layer, allowing it to be easily extended by future work;

4. New datasets to train and evaluate data-oriented simulators on coarse mesh deformable solid dynamics;

5. Open Source data and code available on <https://github.com/Kerber31/fast_coarse_FEM>;

2
Related Work

CNNs are extensively used in Computational Fluid Dynamics (CFD) as a way to learn functions over regular grids (ZHANG et al., 2023; WANG; CHANG; ZHANG, 2022; GUAN et al., 2022; TIAN et al., 2020; XU; DURAISAMY, 2020). One important example of their use is in coarse physics simulation, on applications such as Large Eddy Simulations (LES) (KOCHKOV et al., 2021), and simulation of turbulent flows at high Reynolds number (PATHAK et al., 2020). Between those methods, Solver-in-the-Loop (UM et al., 2020) stands out for showing the advantages of including a differentiable simulator in the training loop and analyzing the benefit of including physical gradients in the backpropagation process. Further work has also shown that this coupling creates more accurate resulting simulations than a purely data-oriented simulator (ILLARRAMENDI; BAUERHEIM; CUENOT, 2022) over simulations in CFD.

In solid mechanics, applications tend to avoid CNNs since they restrict the usage of the model to structured data, such as grids or meshes with regular spacing, and even on these meshes, if they are not rectangular shaped, the network needs to receive empty spaces as inputs. This implies the network needs to have a much bigger input size than necessary, making the model very memory and time inefficient, as bigger inputs require more network parameters to store data and feature maps. Also, convolution and pooling operations become more expensive with input size. Therefore, many works use Fully Connected Neural Networks (FCNNs), which can be modeled to learn functions on irregular structures. However, this kind of network cannot process varying numbers of input and output parameters, so works that use them normally train the networks by sampling points on the domain and giving their coordinates as inputs to the network. The training employs one network for each geometry, varying only initial conditions and material parameters on each training sample (DIAO et al., 2023; HU et al., 2024; NING et al., 2023; REZAEI et al., 2022).

Fewer works exist in coarse-graining of deformable solid dynamics, such as (ARORA, 2022), where authors have created an upscaling method based on previous techniques in image supersampling that upscales coarse-grained simulation frames using CNNs. In (HAN et al., 2021), the authors describe a method to train coarse sampled FCNNs to create real-time flexible multibody dynamics simulations. Other work has made a hybrid method to correct a

coarse deformable beam coupled with fluid flow by training one FCNN for every node of the coarse mesh (BAIGES et al., 2020). However, training one FCNN for every node makes them unusable in other meshes, as they become biased towards the node they were trained on. Therefore, these works can be extended by using GNNs to overcome the limitations of using FCNNs and CNNs on unstructured data.

By representing the meshes as graphs where graph vertices and edges represent mesh nodes and edges, previous works have shown the capability of GNNs in learning to simulate soft tissue mechanics while generalizing to new geometries (DALTON; HUSMEIER; GAO, 2023; DALTON; GAO; HUSMEIER, 2022). Other successful cases are using these models to solve PDE-governed forward and inverse problems (GAO; ZAHR; WANG, 2022) and predicting dynamic responses of continuous deformable bodies (CHEN et al., 2024). A method that stands out among GNN-based simulators is $MeshGraphNet$ (PFAFF et al., 2021), a network architecture capable of learning to simulate diverse physical phenomena, including cloth, solids, and fluids, with remarkable accuracy and generalization regarding geometry and initial conditions. Therefore, we choose this model as a base of comparison. Further works have been done in creating multiscale and multi-fidelity methods (FORTUNATO et al., 2022; CAO et al., 2023; BLACK; NAJAFI, 2022) that have better generalization capability and can learn coarse physics simulation as a subproblem in their architecture. We propose extending a numerical simulator with an MPNN architecture similar to $MeshGraphNet$ to create a fast and accurate coarse simulator reusable in other meshes.

Regarding numerical simulators, recently, extensive work has been done on differentiable simulators to be used in the solution of inverse physical problems and training of control neural networks for soft robots (HU et al., 2019; HUANG et al., 2021; BELBUTE-PERES et al., 2018; HAHN et al., 2019). These solvers must be fast because the processes of finding optimal parameters on inverse problems and training control neural networks are gradient-based nonlinear optimization processes that require running simulations multiple times. Therefore, a differentiable version of Projective Dynamics (BOUAZIZ et al., 2014), namely Differentiable Projective Dynamics (DiffPD) (DU et al., 2021), has been created as a way to perform fast differentiable simulations by taking advantage of the efficient local-global optimization process of Projective Dynamics, which became a state-of-the-art method for these tasks (MA et al., 2021; QIAO et al., 2021).

3
Numerical Simulation Background

In this chapter, we review the formulation for forward PD defined by (BOUAZIZ et al., 2014; LIU; BOUAZIZ; KAVAN, 2017) and backward PD defined by (DU et al., 2021) that compose DiffPD. This formulation is later used to compose our hybrid method in Section 5.2.

3.1
Forward Projective Dynamics

Let $\mathbf{p_i} \in \mathbb{R}^3$ be the position and $\mathbf{v_i} \in \mathbb{R}^3$ be the velocity of the n 3D nodes in a FEM discretization of a deformable solid at the i-th time step. (DU et al., 2021; BOUAZIZ et al., 2014; LIU; BOUAZIZ; KAVAN, 2017) show that we can define an implicit integration scheme:

$$\mathbf{p}_{i+1} = \mathbf{p}_i + h\mathbf{v}_{i+1}, \tag{3-1}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + h\mathbf{M}^{-1}[\mathbf{f}_{int}(\mathbf{p}_{i+1}) + \mathbf{f}_{ext}], \tag{3-2}$$

where $\mathbf{M} \in \mathbb{R}^{3nx3n}$ is a mass matrix, $\mathbf{f}_{ext}$ and $\mathbf{f}_{int}$ are the sum of external and internal forces, and h is the time step size. By substituting Eq. (3-2) in Eq. (3-1), we obtain the following nonlinear system of equations (DU et al., 2021; BOUAZIZ et al., 2014):

$$\mathbf{q}_i = \mathbf{p}_i + h\mathbf{v}_i + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}, \tag{3-3}$$

$$\frac{1}{h^2}\mathbf{M}(\mathbf{p}_{i+1} - \mathbf{q}_i) - \mathbf{f}_{int}(\mathbf{p}_{i+1}) = \mathbf{0}. \tag{3-4}$$

This system can be used to find $\mathbf{p}$, with a given $\mathbf{q}$, at each time step. We can convert the system to an optimization problem (BOUAZIZ et al., 2014; STUART; HUMPHRIES, 1996; DU et al., 2021):

$$\min_{\mathbf{p}_{i+1}} \gamma(\mathbf{p}_{i+1}), \tag{3-5}$$

$$\gamma(\mathbf{p}_{i+1}) = \frac{1}{2h^2}(\mathbf{p}_{i+1} - \mathbf{q}_i)^\top \mathbf{M}(\mathbf{p}_{i+1} - \mathbf{q}_i) + W(\mathbf{p}_{i+1}), \tag{3-6}$$

$$\mathbf{f}_{int} = -\nabla W, \tag{3-7}$$

where W is the potential energy associated with the internal force. Setting the gradient of the optimization in Eq. (3-5) to zero leads to the system of equations in Eqs. (3-3) and (3-4), as $\nabla\gamma(\mathbf{p})$ is the left side of Eq. (3-4):

$$\nabla\gamma(\mathbf{p}_{i+1}) = \nabla[\frac{1}{2h^2}(\mathbf{p}_{i+1} - \mathbf{q}_i)^\top \mathbf{M}(\mathbf{p}_{i+1} - \mathbf{q}_i)] + \nabla W(\mathbf{p}_{i+1}) \tag{3-8}$$

$$\nabla\gamma(\mathbf{p}_{i+1}) = \frac{1}{h^2}\mathbf{M}(\mathbf{p}_{i+1} - \mathbf{q}_i) - \mathbf{f}_{int}(\mathbf{p}_{i+1}). \tag{3-9}$$

Typically, the minimization in Eq. (3-5) is done using Newton's Method, which requires solving a linear system of equations for each step, as the Hessian changes every iteration, thus making the process computationally expensive. To simplify the notation on further formulation, we will drop the subscripts in $\mathbf{p}_{i+1}$ and $\mathbf{q}_i$.

The key aspect of PD that accelerates this process is the use of a sum of quadratic functions to define the potential energy W that decouples the nonlinearity in material models (BOUAZIZ et al., 2014):

$$\widetilde{W}_k(\mathbf{p}, \mathbf{c}_k) = \frac{\omega_k}{2}\|\mathbf{Q}_k\mathbf{p} - \mathbf{c}_k\|_2^2, \tag{3-10}$$

$$W_k = \min_{\mathbf{c}_k \in \mathcal{M}_k} \widetilde{W}_k(\mathbf{p}, \mathbf{c}_k), \tag{3-11}$$

$$W(\mathbf{p}) = \sum_k W_k(\mathbf{p}), \tag{3-12}$$

where k is the node number. In this system, W_k projects a linear transformation of $\mathbf{p}$, $\mathbf{Q}_k\mathbf{p}$, into its closest point $\mathbf{c}_k$ in the constraint manifold $\mathcal{M}_k$, and scales the squared distance $\|\mathbf{Q}_k\mathbf{p} - \mathbf{c}_k\|_2^2$ by a stiffness ω_k.

With this definition, PD proposes a surrogate objective function, solved by a local-global optimization process:

$$\widetilde{\gamma}(\mathbf{p}, \mathbf{c}) = \frac{1}{2h^2}(\mathbf{p} - \mathbf{q})^\top \mathbf{M}(\mathbf{p} - \mathbf{q}) + \sum_k \widetilde{W}_k(\mathbf{p}, \mathbf{c}_k), \tag{3-13}$$

where $\mathbf{c}$ stacks up all $\mathbf{c}_k$ from each W_k. In the local step, PD fixes the current $\mathbf{p}$, then projects $\mathbf{Q}_k\mathbf{p}$ onto the constraint manifold $\mathcal{M}_k$ to obtain $\mathbf{c}_k$ in each W_k. This can be massively parallelized for all W_k. The global step, $\widetilde{\gamma}$ is minimized over $\mathbf{p}$ with $\mathbf{c}$ fixed, which is a quadratic function and can be solved analytically by the system of equations originated by $\nabla\widetilde{\gamma} = \mathbf{0}$:

$$\underbrace{(\frac{1}{h^2}\mathbf{M} + \sum_k \omega_k \mathbf{Q}_k^\top \mathbf{Q}_k)}_{\mathbf{G}}\,\mathbf{p} = \frac{1}{h^2}\mathbf{M}\mathbf{q} + \sum_k \omega_k \mathbf{Q}_k^\top \mathbf{c}_k. \tag{3-14}$$

This local-global optimization process ensures that $\widetilde{\gamma}$ decreases monotonically. Also, it can be shown that $\nabla_{\mathbf{p}}\widetilde{\gamma} = \nabla\gamma$ upon convergence (LIU; BOUAZIZ; KAVAN, 2017) and, as $\widetilde{\gamma}$ is lower bound by 0, the method is guaranteed to converge to a local minimum of $\widetilde{\gamma}$, where the saddle point condition $\nabla\widetilde{\gamma} = \mathbf{0}$ is satisfied.

What guarantees efficiency in forward PD is that the matrix $\mathbf{G}$ in the global step is symmetric positive definite and constant, thus being prefactorizable at the beginning of the simulation, which leads the forward simulation to only require back-substitution.

3.2 Backward Projective Dynamics

For backpropagation, (DU et al., 2021) demonstrates that we need to obtain the gradient of a loss function $L(\mathbf{p})$ in relation to $\mathbf{p}$ and $\mathbf{q}$, $\frac{\partial L}{\partial \mathbf{p}}$ and $\frac{\partial L}{\partial \mathbf{q}}$. Then, it is possible to backpropagate through multiple time steps by backpropagating through every ($\mathbf{p}$, $\mathbf{q}$) pair for all time steps.

We need to obtain $\frac{\partial L}{\partial \mathbf{q}}$. For that, the chain rule can be used:

$$\frac{\partial L}{\partial \mathbf{q}} = \frac{\partial L}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{q}}, \tag{3-15}$$

as for $\frac{\partial \mathbf{p}}{\partial \mathbf{q}}$, we can obtain it using the fact that $\mathbf{p}$ and $\mathbf{q}$ are implicitly constrained by $\nabla\gamma(\mathbf{p}) = \mathbf{0}$, and differentiate with respect to $\mathbf{q}$ to obtain the following equation (DU et al., 2021):

$$\frac{\partial \nabla\gamma(\mathbf{p})}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{q}} + \frac{\partial \nabla\gamma(\mathbf{p})}{\partial \mathbf{q}} = \mathbf{0}, \tag{3-16}$$

$$\frac{\partial}{\partial \mathbf{p}}\left[\frac{1}{h^2}\mathbf{M}(\mathbf{p}-\mathbf{q}) + \nabla W(\mathbf{p})\right]\frac{\partial \mathbf{p}}{\partial \mathbf{q}} + \frac{\partial}{\partial \mathbf{q}}\left[\frac{1}{h^2}\mathbf{M}(\mathbf{p}-\mathbf{q}) + \nabla W(\mathbf{p}\right] = \mathbf{0}, \tag{3-17}$$

$$\left[\frac{1}{h^2}\mathbf{M} + \nabla^2 W(\mathbf{p})\right]\frac{\partial \mathbf{p}}{\partial \mathbf{q}} - \frac{1}{h^2}\mathbf{M} = \mathbf{0}, \tag{3-18}$$

$$\nabla^2\gamma(\mathbf{p})\frac{\partial \mathbf{p}}{\partial \mathbf{q}} - \frac{1}{h^2}\mathbf{M} = \mathbf{0}, \tag{3-19}$$

$$\frac{\partial \mathbf{p}}{\partial \mathbf{q}} = \frac{1}{h^2}[\nabla^2\gamma(\mathbf{p})]^{-1}\mathbf{M}. \tag{3-20}$$

Now, we can combine Eqs. (3-15) and (3-20) to find $\frac{\partial L}{\partial \mathbf{q}}$:

$$\frac{\partial L}{\partial \mathbf{q}} = \frac{\partial L}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{q}} = \frac{1}{h^2}\frac{\partial L}{\partial \mathbf{p}}[\nabla^2\gamma(\mathbf{p})]^{-1}\mathbf{M}. \tag{3-21}$$

As computing the inverse of $\nabla^2\gamma(\mathbf{p})$ is expensive, (DU et al., 2021) separates this equation into the system:

$$\frac{\partial L}{\partial \mathbf{q}} = \frac{\mathbf{M}}{h^2}\mathbf{x}^\top, \tag{3-22}$$

$$\nabla^2\gamma(\mathbf{p})\mathbf{x} = \left(\frac{\partial L}{\partial \mathbf{p}}\right)^\top, \tag{3-23}$$

where $\nabla^2\gamma(\mathbf{p})$ is symmetric, therefore, we drop its transpose. Implementing Eq. (3-23) is computationally expensive, because $\nabla^2\gamma(\mathbf{p})$ is recalculated and refactorized at all time steps. Therefore, (DU et al., 2021) proposes a PD-based

backpropagation method by decoupling $\nabla^2\gamma(\mathbf{p})$ into a parallelizable nonlinear component and a constant global matrix. The first step for that is to write the equation for $\nabla^2\gamma(\mathbf{p})$:

$$\nabla^2\gamma(\mathbf{p}) = \frac{1}{h^2}\mathbf{M} + \nabla^2 W(\mathbf{p}), \tag{3-24}$$

now, we need to write the equation for $\nabla^2 W(\mathbf{p})$:

$$\nabla W(\mathbf{p}) = \sum_k \omega_k (\mathbf{Q}_k - \frac{\partial \mathbf{c}_k}{\partial \mathbf{p}})^\top (\mathbf{Q}_k\mathbf{p} - \mathbf{c}_k) \Longrightarrow \sum_k \omega_k \mathbf{Q}_k^\top (\mathbf{Q}_k\mathbf{p} - \mathbf{c}_k), \tag{3-25}$$

$$\nabla^2 W(\mathbf{p}) = \sum_k \omega_k \mathbf{Q}_k^\top \mathbf{Q}_k - \sum_k \omega_k \mathbf{Q}_k^\top \frac{\partial \mathbf{c}_k}{\partial \mathbf{p}}. \tag{3-26}$$

In Eq. (3-25), $\frac{\partial \mathbf{c}_k}{\partial \mathbf{p}}$ can be ignored according to the envelope theorem (LIU; BOUAZIZ; KAVAN, 2017). With this definition, we can substitute Eq. (3-26) into Eq. (3-24):

$$\nabla^2\gamma(\mathbf{p}) = \underbrace{\frac{1}{h^2}\mathbf{M} + \sum_k \omega_k \mathbf{Q}_k^\top \mathbf{Q}_k}_{\mathbf{G}} - \sum_k \omega_k \mathbf{Q}_k^\top \frac{\partial \mathbf{c}_k}{\partial \mathbf{p}}, \tag{3-27}$$

$$\mathbf{H} = \sum_k \omega_k \mathbf{Q}_k^\top \frac{\partial \mathbf{c}_k}{\partial \mathbf{p}}, \tag{3-28}$$

$$\nabla^2\gamma(\mathbf{p}) = \mathbf{G} - \mathbf{H}, \tag{3-29}$$

where $\mathbf{G}$ is the matrix defined in Eq. (3-14). An iterative process can be defined to calculate the auxiliary vector $\mathbf{x}$ in Eq. (3-23) by using $\nabla^2\gamma(\mathbf{p}) = \mathbf{G} - \mathbf{H}$:

$$\mathbf{G}\mathbf{x}^{j+1} = \mathbf{H}\mathbf{x}^j + \left(\frac{\partial L}{\partial \mathbf{p}}\right)^\top, \tag{3-30}$$

where j is the iteration number. To solve this system, a local-global solve is used, where the local solve computes $\mathbf{H}\mathbf{x}^j$ across all W_k, then the global step solves $\mathbf{x}^{j+1}$ by performing back-substitution in $\mathbf{G}$. Since $\mathbf{G}$ was used in forward simulation, its Cholesky factorization was already calculated and can be reused. (DU et al., 2021) shows that this iterative algorithm converges for any initial guess $\mathbf{x}^0$ if and only if the spectral radius of $\mathbf{G}^{-1}\mathbf{H}$ is smaller than 1, and the authors leave a general proof of whether this is true for all forms of W_k to future work, but argue that convergence issues were never seen in their experiments, which indicates that this condition is likely satisfied.

Other then the PD-like local-global solve, (DU et al., 2021) demonstrates that it is possible to apply a similar numerical optimization technique to the quasi-Newton method in (LIU; BOUAZIZ; KAVAN, 2017) by using the following objective function:

$$r(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \nabla^2\gamma(\mathbf{p})\mathbf{x} - \frac{\partial L}{\partial \mathbf{p}}\mathbf{x}, \tag{3-31}$$

where $\nabla r(\mathbf{x}) = \mathbf{0}$ is Eq. (3-23) and both $\nabla^2\gamma(\mathbf{p})$ and $\frac{\partial L}{\mathbf{p}}$ were already calculated in forward simulation.

The Newton's Method update rule for this critical-point problem can be summarized as:

$$\nabla r(\mathbf{x}) = \mathbf{0}, \tag{3-32}$$

$$\nabla r(\mathbf{x}) = \nabla r(\mathbf{x}^j + \Delta\mathbf{x}) \approx \nabla r(\mathbf{x}^j) + \nabla^2 r(\mathbf{x}^j)\Delta\mathbf{x}, \tag{3-33}$$

$$\nabla^2 r(\mathbf{x}^j)\Delta\mathbf{x} \approx -\nabla r(\mathbf{x}^j), \tag{3-34}$$

$$\Delta\mathbf{x} = -[\nabla^2 r(\mathbf{x}^j)]^{-1}\nabla r(\mathbf{x}^j), \tag{3-35}$$

where the update at each iteration is defined by $\mathbf{x}^{j+1} = \mathbf{x}^j + \Delta\mathbf{x}$, and $\nabla^2 r(\mathbf{x}) = \nabla^2\gamma(\mathbf{p}) = \mathbf{G} - \mathbf{H}$. If the hessian $\nabla^2 r(\mathbf{x})$ is approximated using $\mathbf{G}$, then the following quasi-Newton update rule is derived:

$$\mathbf{x}^{j+1} = \mathbf{x}^j - \mathbf{G}^{-1}\nabla r(\mathbf{x}) \tag{3-36}$$

$$\mathbf{x}^{j+1} = \mathbf{x}^j - \mathbf{G}^{-1}\left[(\mathbf{G} - \mathbf{H})\mathbf{x}^j - \left(\frac{\partial L}{\partial \mathbf{p}}\right)^\top\right] \tag{3-37}$$

$$\mathbf{x}^{j+1} = \mathbf{G}^{-1}\mathbf{H}\mathbf{x}^j + \mathbf{G}^{-1}\left(\frac{\partial L}{\partial \mathbf{p}}\right)^\top, \tag{3-38}$$

that matches the iteration process described in Eq. (3-30). Consequently, the local-global solver introduced by (DU et al., 2021) can be understood as employing a basic quasi-Newton approach, utilizing a fixed Hessian estimate denoted by $\mathbf{G}$.

4
Deep Learning Background

4.1
Fully Connected Neural Network

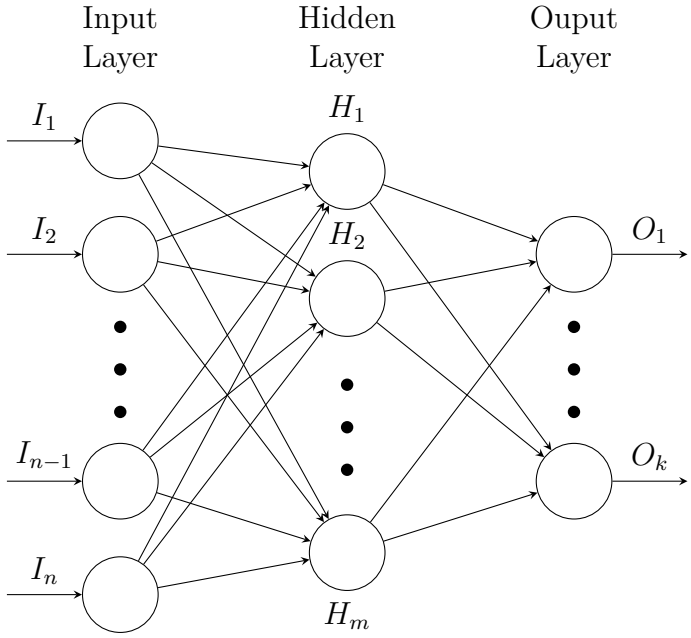

Figure 4.1: Architecture of a Fully Connected Neural Network (FCNN): This diagram illustrates a FCNN, composed of three key layers: the Input Layer $(I_1, I_2, \ldots)$, where each neuron represents an input feature; the Hidden Layer $(H_1, H_2, \ldots)$, which processes inputs via weighted connections to learn complex patterns; and the Output Layer $(O_1, O_2, \ldots)$, that outputs the network's final decision. Arrows signify synaptic weights between neurons.

Fully connected neural networks (FCNNs), or dense neural networks (DNNs), are characterized by their layered architecture, where each neuron in a layer is connected to all neurons in the subsequent layer (Figure 4.1). The network comprises L layers, with each layer l containing N_l neurons. The output of each neuron is the weighted sum of its inputs and a bias term transformed by a nonlinear activation function. The weights and biases for each layer l are represented by a weight matrix $\mathbf{\Theta}_l$ of dimensions $N_l \times N_{l-1}$ and a bias vector $\mathbf{b}_l$ of dimension N_l. The following equation defines forward propagation in the network:

$$\mathbf{a}^{[l]} = f(\mathbf{\Theta}_l \mathbf{a}^{[l-1]} + \mathbf{b}_l), \tag{4-1}$$

where $\mathbf{a}^{[l-1]}$ is the activation from the previous layer and f is the activation function.

This framework enables the network to learn complex mappings from inputs to outputs, contingent upon the choice of activation functions, loss functions, and optimization algorithms used during the training process.

4.2 Message Passing Neural Network

In message-passing networks, each node $\mathbf{n}_i$ in the graph holds a feature vector $\mathbf{h}_i$. The core idea is to iteratively update the state of each node by aggregating and transforming the feature vectors of its neighboring nodes. The update at each iteration k can be mathematically described as follows:

$$\mathbf{h}_i^{k+1} = f\left(\mathbf{h}_i^k, \bigoplus_{\mathbf{n}_j \in N(\mathbf{n}_i)} g(\mathbf{h}_i^k, \mathbf{h}_j^k, \mathbf{e}_{ij})\right), \tag{4-2}$$

where $N(\mathbf{n}_i)$ denotes the set of neighbors of node $\mathbf{n}_i$, $\mathbf{e}_{ij}$ represents the edge features between nodes $\mathbf{n}_i$ and $\mathbf{n}_j$, $\bigoplus$ is an aggregation function (such as sum, mean, or max), and f and g are learnable transformation functions, implemented as FCNNs. This process is depicted in Figure 4.2.

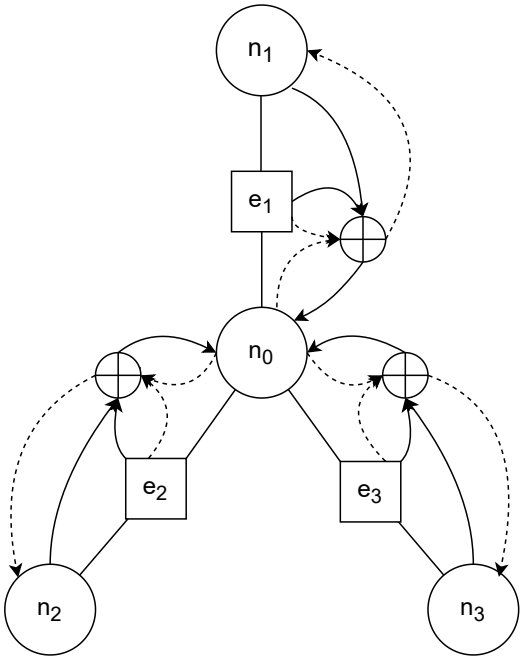

Figure 4.2: Visual representation of the message passing operation in an MPNN layer. Each node n_i propagates its features, aggregated with the features of the edge e that connects them, to its neighbor nodes n_j.

The network typically operates in two phases: the message-passing phase, where nodes exchange information, and the readout phase, where the updated node features are aggregated to form the final output of the network. Depend-

ing on the specific task, this output could be node-level predictions, edge-level predictions, or a graph-level representation.

4.3 Numerical Challenges in Message Passing Neural Networks

Despite the efficacy of Message Passing Neural Networks (MPNNs) in learning graph-structured data, several numerical challenges can impede their performance and stability. We discuss these challenges with respect to the notation used for describing MPNNs in our framework.

4.3.1 Vanishing and Exploding Gradients

One of the primary numerical challenges in deep MPNNs, namely the vanishing and exploding gradients (BENGIO; SIMARD; FRASCONI, 1994; LUKOVNIKOV; LEHMANN; FISCHER, 2020), can be attributed to the characteristics of the activation functions used within the FCNNs that implement the learnable transformation functions f and g. These issues impact the network's learning capability and stability as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}^1} = \prod_{l=1}^{L} \left(\frac{\partial \sigma(h^l)}{\partial h^l} \cdot \frac{\partial h^{l+1}}{\partial \mathbf{\Theta}^l} \right) \cdot \frac{\partial \mathcal{L}}{\partial h^{L+1}} \tag{4-3}$$

where $\frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}^1}$ represents the gradient of the loss function $\mathcal{L}$ with respect to the weights of the first layer $\mathbf{\Theta}^1$, $\sigma(h^l)$ denotes the activation function applied at layer l, and $\frac{\partial \sigma(h^l)}{\partial h^l}$ represents the derivative of the activation function with respect to the pre-activation output at layer l. This derivative plays a crucial role in determining the magnitude of gradients during backpropagation. Depending on the properties of $\frac{\partial \sigma(h^l)}{\partial h^l}$, gradients can either vanish or explode. To quantify the relationship between the activation function's derivative and the gradient magnitude, consider the following proportional relationship:

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}^1} \right\| \propto \prod_{l=1}^{L} \left| \frac{\partial \sigma(h^l)}{\partial h^l} \right| \tag{4-4}$$

Eq. 4-4 illustrates that the modulus of the gradient of the loss function with respect to the weights of the first layer is proportional to the product of the modulus of the derivative of the activation function across all layers. This relationship highlights how the characteristics of the activation function influence the phenomena of vanishing and exploding gradients in deep MPNNs.

Vanishing Gradients: Activation functions with derivatives that tend to produce values less than 1 (e.g., sigmoid or tanh in their saturation regions) can cause the gradients to diminish as they are propagated back through the

layers. This reduces the ability of the network to update its weights effectively, hindering the learning process:

$$\left|\frac{\partial\sigma(h^l)}{\partial h^l}\right| < 1 \implies \left\|\frac{\partial\mathcal{L}}{\partial\mathbf{\Theta}^1}\right\| \to 0, \tag{4-5}$$

resulting in the network's inability to learn complex patterns or improve performance over iterations.

Exploding Gradients: Conversely, if the activation functions or their compositions in the FCNNs amplify the gradients (for instance, through repeated multiplication in deep networks), the gradients can increase exponentially through the layers:

$$\left|\frac{\partial\sigma(h^l)}{\partial h^l}\right| > 1 \implies \left\|\frac{\partial\mathcal{L}}{\partial\mathbf{\Theta}^1}\right\| \to \infty, \tag{4-6}$$

causing numerical instability and making the training process divergent.

The choice of activation functions within the FCNNs of MPNNs is thus crucial for mitigating the risks associated with vanishing and exploding gradients. Employing activation functions with well-behaved derivatives across the relevant input domain, or incorporating normalization techniques and gradient clipping strategies, can significantly alleviate these issues, ensuring stable and effective learning dynamics in deep MPNN architectures.

4.3.2 Over-Smoothing

Over-smoothing (KERIVEN, 2022; CHEN et al., 2020) is a phenomenon observed in MPNNs as a direct consequence of the iterative aggregation and transformation process across layers. As the number of iterations k increases, the node features $\mathbf{h}_i^{(k)}$ tend to converge to a homogeneous state, making them indistinguishable from one another. This effect is particularly pronounced in deep networks, where the repeated application of the aggregation and transformation functions leads to a saturation of the node feature space. Mathematically, the over-smoothing process can be described as:

$$\lim_{k\to\infty} \mathbf{h}_i^k = \mathbf{h}_{\text{uniform}}, \quad \forall i \in [1, ..., N], \tag{4-7}$$

where N is the number of nodes in the graph, i is the node index, and $\mathbf{h}_{\text{uniform}}$ is a state where all node features converge to a uniform vector, reducing the discriminative power of the network. This uniformity results from the excessive blending of features, where the distinctiveness of local neighborhood structures is lost. The primary cause of over-smoothing is the inherent design of MPNNs to repeatedly mix the features of neighboring nodes, which, while intended to capture the graph structure, inadvertently leads to feature homogenization.

4.3.3 Countering Over-Smoothing and Gradient Problems

To counter over-smoothing and gradient problems in Message Passing Neural Networks (MPNNs), several strategies can be employed, including the use of residual connections, the Rectified Linear Unit (ReLU) activation function, and normalization techniques such as data normalization and batch/layer normalizations.

4.3.3.1 Residual Connections

Residual connections are instrumental in alleviating the over-smoothing issue by allowing for the direct flow of information across layers. By adding the input of a layer to its output, residual connections help preserve the original feature information and the gradient magnitude, thereby maintaining feature diversity and combating the vanishing/exploding gradient phenomenon:

$$\mathbf{h}_i^{k+1} = \mathbf{h}_i^k + f\left(\mathbf{h}_i^k, \bigoplus_{\mathbf{n}_j \in N(\mathbf{n}_i)} g(\mathbf{h}_i^k, \mathbf{h}_j^k.\mathbf{e}_{ij})\right), \tag{4-8}$$

4.3.3.2 ReLU and Normalization

The Rectified Linear Unit (ReLU) activation function plays a critical role in mitigating the vanishing gradient problem. By providing a linear response for positive inputs while zeroing negative inputs, ReLU ensures that the gradient remains sufficiently large during backpropagation, preventing it from vanishing. The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x). \tag{4-9}$$

Its application within the transformation functions f and g of the MPNN not only aids in maintaining healthy gradient flow but also introduces non-linearity without saturating, unlike sigmoid or tanh functions.

Normalization techniques, including data normalization and batch/layer normalization, are critical in ensuring that the ReLU activation functions operate effectively by centering the data around zero. Data normalization adjusts the input features to have zero mean and unit variance, ensuring that the activations are not pushed to the extremes of the ReLU function. Similarly, batch normalization adjusts the activations of a layer for each mini-batch, ensuring that the inputs to the ReLU function are centered and scaled appropriately. On the other hand, layer normalization performs this adjustment across all features in a single layer for each training example, which

is particularly beneficial for MPNNs that deal with variable-sized inputs due to the differing neighborhood sizes. Both normalization techniques ensure that the network benefits from the non-saturating properties of ReLU, facilitating stable and efficient training:

$$\mathbf{h}' = \frac{\mathbf{h} - \mu_{\mathbf{h}}}{\sigma_{\mathbf{h}}}, \tag{4-10}$$

where $\mathbf{h}'$ is the normalized version of the feature vector $\mathbf{h}$, with $\mu_{\mathbf{h}}$ and $\sigma_{\mathbf{h}}$ representing the mean and standard deviation of the features, respectively, for data normalization. The mean and variance are computed differently for batch and layer normalization, but the goal remains to ensure that the inputs to the activation functions facilitate effective learning.

By combining these approaches, MPNNs can effectively counteract the challenges posed by over-smoothing and vanishing/exploding gradient problems, leading to more stable and robust graph representation learning.

4.4 MeshGraphNet

To create a data-oriented simulator using graph networks, it is first necessary to interpret data as graphs. In the case of finite element simulations, we define a mesh M containing m nodes connected by the edge set E. To interpret this mesh as a graph, in (PFAFF et al., 2021), the authors encode M into a multigraph $G = (N, E^M, E^W)$, where mesh vertices become graph nodes N and mesh edges become bidirectional graph edges E^M. Another type of edge is added to the multigraph, namely the world edges E^W, that encode further dynamic information such as collisions, self-collisions, and contact. These edges are added by proximity; therefore, given two nodes $\mathbf{n}_i$ and $\mathbf{n}_j$ and a radius r_W, if $\|\mathbf{n}_i - \mathbf{n}_j\| < r_W$ a world edge $e_{ij}^W \in E^W$ is added.

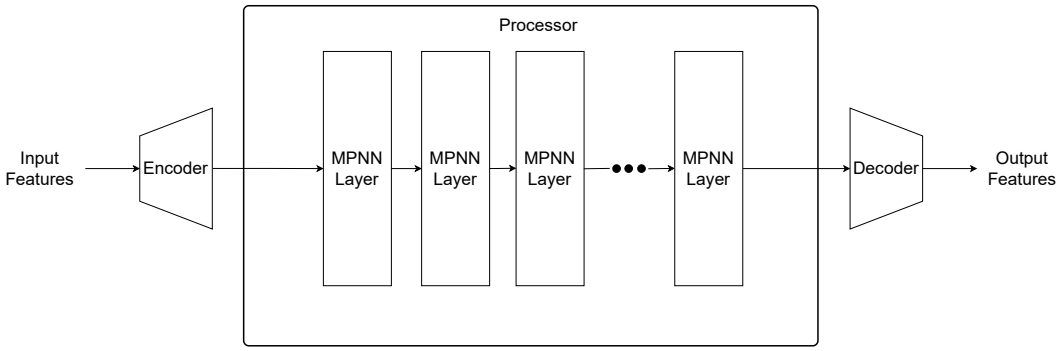

Figure 4.3: Visual representation of a $MeshGraphNet$. The input features are fed to the Encoder FCNN, which transforms them into the latent features subsequently fed to the Processor network. Then, the processor network applies a series of message-passing steps to obtain the latent features that are fed to the Decoder FCNN. Then, the Decoder transforms the latent features into the output features.

The architecture is divided into three parts: Encoder, Processor, and Decoder (Figure 4.3). The Encoder encodes features into graph nodes N and edges E^M and E^W. Spatial invariance is achieved by representing positional features as relative edge features. The encoder embeds relative displacement vectors in mesh space $\mathbf{u}_i - \mathbf{u}_j$ and their norms $||\mathbf{u}_i - \mathbf{u}_j||$ into features for mesh edges $e_{ij}^M \in E^M$ and world edges $e_{ij}^W \in E^W$. Additionally, all remaining dynamical features $\mathbf{d}_i$, along with a one-hot encoding of node types, are provided as node features in $\mathbf{n}_i$.

The concatenation of these features is then transformed by encoder FCNNs into a latent vector of size 128 at each node and edge, forming the encoded state that subsequent network layers will process. The specific FCNNs for mesh edges, world edges, and nodes are denoted as $\mathcal{E}^M$, $\mathcal{E}^W$, and $\mathcal{E}^V$, respectively. This encoding process lays the foundation for the Processor, a message-passing network composed of L_p message-passing layers with skip connections to avoid vanishing gradients.

The decoder network uses an FCNN to recover the output features $\mathbf{o}_i$ from the latent features $\mathbf{n}_i$ of the last processing layer. Then, $\mathbf{o}_i$ is interpreted as either the first or higher-order derivative of $\mathbf{d}_i$ and integrated using a forward-Euler integrator to compute the next step in time of the dynamical features $\mathbf{d}_i^{t+1}$.

5
Methodology

We present a novel hybrid simulation method that synergizes a differentiable numerical simulator with a neural network to enhance accuracy in simulations involving coarse FEM discretization.

5.1
Formulation

First, we introduce a general formulation for our problem based on (UM et al., 2020), defining four manifolds:

- The reference $\mathcal{R}$ manifold is where lie the solutions $\mathbf{r}^t$, where t is the time step number, to a given PDE obtained using a differentiable simulator $S(\mathbf{x}, \Delta t)$, where Δt is the time step size and $\mathbf{x}$ is an input state, with a refined mesh $M_{\mathcal{R}}$;
- The down-sampled $\mathcal{D}$ manifold is the solution manifold obtained by applying a down-sampling operation $\mathcal{T}_{\mathcal{D}}$ that projects $\mathbf{r}^t$ from $M_{\mathcal{R}}$ into a coarse mesh $M_{\mathcal{U}}$ to obtain down-sampled solutions $\mathbf{d}^t$;
- The coarse $\mathcal{U}$ manifold is where lie the solutions $\mathbf{u}^t$ to the PDE using $S(\mathbf{x}, \Delta t)$ with $M_{\mathcal{U}}$.
- Finally, the corrected manifold $\mathcal{C}$ is where lie the solutions obtained by summing $\mathbf{u}^t$ to a correction function approximated by a neural network $\mathcal{N}(\mathbf{x}|\hat{\theta})$, where $\hat{\theta}$ is a vector that contains the estimated network parameters, to obtain the corrected state $\tilde{\mathbf{u}}^t$;

As a simplification, we will use $S_{\mathcal{M}}(\mathbf{x}, \Delta t)$ as the simulator using a mesh $M_{\mathcal{M}}$, where $\mathcal{M}$ is the manifold.

To summarize the forward computation, we have the following equations (UM et al., 2020):

$$\mathbf{u}^{t+1} = S_{\mathcal{U}}(\mathbf{u}^t, \Delta t), \tag{5-1}$$

$$\tilde{\mathbf{u}}^{t+1} = \mathbf{u}^{t+1} + \mathcal{N}(\mathbf{u}^{t+1}|\hat{\theta}), \tag{5-2}$$

that can be repeated for multiple time steps, generating the state $\tilde{\mathbf{u}}^{t+k}$, where k is the number of time steps. Our goal is to make the error $\epsilon = \|\tilde{\mathbf{u}}^t - \mathbf{d}^t\|_2$ as close as possible to zero. In other words, $\mathcal{C}$ should be a close approximation of $\mathcal{D}$. Therefore, we can define the following training objective:

$$\arg\min_{\theta} \underbrace{\sum_{i=1}^{k} \left\| \left[\mathbf{u}^{t+i} + \mathcal{N}(\mathbf{u}^{t+i}|\theta) \right] - \mathbf{d}^{t+i} \right\|_2}_{L(\mathbf{u},\mathbf{d})}. \tag{5-3}$$

The backpropagation procedure integrates the network gradients and the simulator gradients to enable full differentiability (THUEREY et al., 2021):

$$\frac{\partial L}{\partial \theta} = \sum_{n} \sum_{i=1}^{k} \left[\frac{\partial L}{\partial \tilde{\mathbf{u}}_n^k} \left(\prod_{j=k}^{i+1} \frac{\partial \tilde{\mathbf{u}}_n^j}{\partial \mathbf{u}_n^j} \frac{\partial \mathbf{u}_n^j}{\partial \tilde{\mathbf{u}}_n^{j-1}} \right) \frac{\partial \tilde{\mathbf{u}}_n^i}{\partial \theta} \right], \tag{5-4}$$

where k is the number of time steps taken before the backward operation, and n is the mini-batch number.

5.2 Implementation Details

To implement this framework, we use DiffPD (DU et al., 2021) as $S_{\mathcal{M}}(\mathbf{x}, \Delta t)$ and calculate the simulator gradients in Eq. (5-4) using Eqs. (3-21) and (3-30). We use the implementation of DiffPD provided by (DU et al., 2021) in their GitHub repository, which is CPU-based in C++ using the Eigen library (GUENNEBAUD; JACOB et al., 2010).

As for the network $\mathcal{N}(\mathbf{x}|\theta)$, we use a variation of *MeshGraphNet* (PFAFF et al., 2021). We implement $\mathcal{N}(\mathbf{x}|\theta)$ with 15 message passing layers and all FCNNs in the encoder, processor, and decoder with four layers and ReLU activation functions. All FCNN outputs are normalized with a LayerNorm. This configuration was chosen because it achieved the best results in (PFAFF et al., 2021).

Furthermore, we do not implement collisions or self-collisions, so we do not add world edges by proximity. For the node features, we use the material Young's Modulus E and Poisson's Ratio ν, as well as node velocities, external forces, and node types encoded as one-hot vectors. For the node types, we define two possibilities: either the node is free or fixed. Edge features are not modified and remain the same as defined in Section 4.4. For the output feature, we predict a position correction vector.

The network is implemented in Pytorch (PASZKE et al., 2019) and PyTorch Geometric (FEY; LENSSEN, 2019). We train the network using batch size $n = 1$ and $k = 1$ time steps and the Adam optimizer (KINGMA; BA, 2015). All input, output, and target features are normalized to unit variance, and zero mean using dataset statistics. Additionally, we scale $\tilde{\mathbf{u}}^t$ and $\mathbf{d}^t$ by multiplying them by a constant $s = 1000$ before measuring the loss (Eq. (5-3)) to further mitigate the problem of vanishing gradients.

We train our model for 20 epochs on each dataset described in Chapter 6. We also implement the option of using noise in the training:

$$\tilde{\mathbf{x}}^t = \mathbf{x}^t + \mu\tau, \tag{5-5}$$

$$\tilde{\mathbf{x}}^{t+1} = \mathbf{x}^{t+1}, \tag{5-6}$$

where $\mathbf{x}^{t+1}$ is the target position and $\tilde{\mathbf{x}}^{t+1}$ is the target position modified by noise, $\mu \in \mathbb{R}$ is a random variable sampled from a normal distribution with zero mean and unit variance, scaled by another parameter τ. This way, the model's target includes correcting the noise added to the input positions.

6
Dataset

We introduce three novel datasets to facilitate the training and evaluation of models in coarse deformable solid dynamics simulations. Each dataset is bifurcated into two subsets: one designated for training and the other for evaluation. The training subsets encompass 1000 simulations each, while the evaluation subsets contain 100 simulations. The datasets are specialized to represent three fundamental benchmarks: (i) free deformation following an initial impulse, (ii) stretching, and (iii) torsion of deformable solids. We chose these three types of simulations as we observe them to be prevalent in the literature to evaluate simulation models for deformable solid dynamics (DU et al., 2021; LUO et al., 2020; LI; BAZANT; ZHU, 2021). For simplicity, we name dataset (i) *Impulse*, (ii) *Stretch*, and (iii) *Torsion*. All datasets are generated in less than a day by using multithreading.

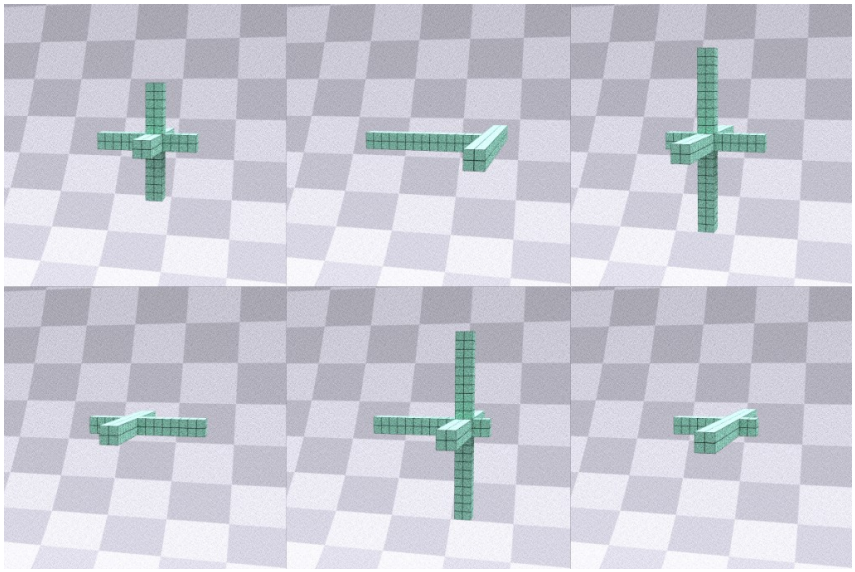

Figure 6.1: *CrossMeshes* are composed of a central beam with either two or four symmetric perpendicular ramifications centered at a random position on the central beam. The refined meshes' number of nodes varies from 7857 to 18225 nodes, while the coarse counterparts vary from 225 to 513.

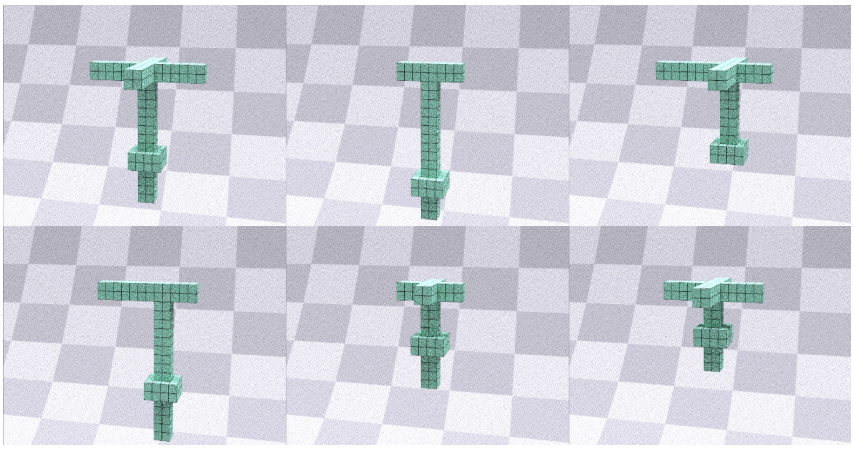

Figure 6.2: $WeightMeshes$ are composed of one or two perpendicular support beams and another pendular beam with an inflation in the middle, which we refer to as the weight. To add more variation to the data, we move the pendular beam together with one of the support beams over the other support beam. We also move the weight over the pendular beam and vary the sizes of all beams. The refined meshes' number of nodes varies from 8433 to 16857 nodes, while the coarse counterparts vary from 237 to 471.

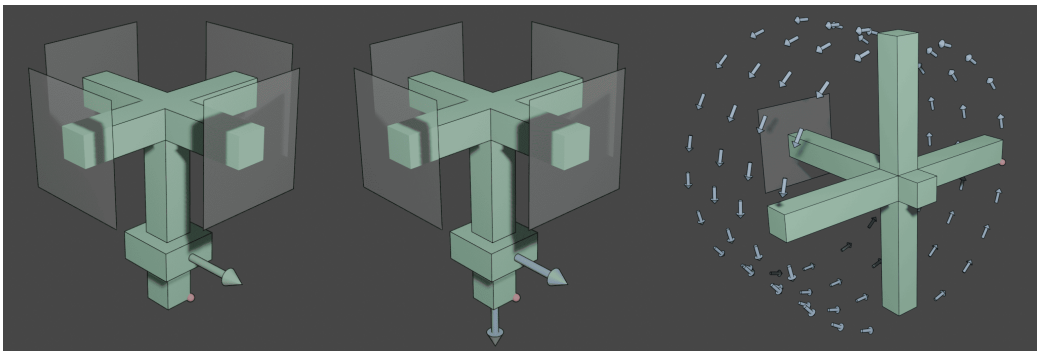

Figure 6.3: Visual representation of the data in the $Impulse$ (left), $Stretch$ (middle), and $Torsion$ (right) datasets. The blue arrows represent forces, while the transparent planes are boundary conditions, and the red sphere is the node used as reference in the analysis presented in Chapter 7.

The datasets incorporate two types of meshes, which we call $Cross\,Meshes$ (Fig. 6.1) and $Weight\,Meshes$ (Fig. 6.2). We use hexahedral elements, and the refined meshes $M_{\mathcal{R}}$ have 64 times more elements than the coarse meshes $M_{\mathcal{U}}$. When generating them, we create $M_{\mathcal{R}}$ by subdividing $M_{\mathcal{U}}$'s elements. When creating the $Impulse$ and $Stretch$ datasets, we run simulations for 60 frames and use 200 meshes for train and 50 for validation. For $Torsion$, we run them for 120 frames, using 100 meshes for the train and 40 for validation. This difference is because the torsion phenomenon needs more frames to be properly evaluated, and the characteristics of cross meshes

allow us to create fewer meshes that lead to simulations that show the desired phenomenon well enough to be properly evaluated. On all datasets, we need to obtain $M_{\mathcal{D}}$ to measure the loss function (Eqn. (5-3)). We perform the downsampling operation by finding the nodes in $M_{\mathcal{R}}$ and $M_{\mathcal{U}}$ that have the same spacial coordinates when the meshes are undeformed, and stacking them on the matrix $\mathbf{N}_c$.

Cross Meshes are designed to be used in the *Torsion* dataset, as their configurations are favorable to induce different torques on the central beam of the meshes since the *Torsion* dataset is generated by subjecting the meshes to a rotational force field (Fig. 6.3) with the following equation:

$$f_x = \alpha, \quad f_y = -z - ky, \quad f_z = y - kz \tag{6-1}$$

$$\mathbf{f}(\mathbf{x}) = (f_x(\mathbf{x}), f_y(\mathbf{x}), f_z(\mathbf{x})), \tag{6-2}$$

$$\hat{\mathbf{f}}(\mathbf{x}) = \sigma \frac{\mathbf{f}(\mathbf{x})}{\|\mathbf{f}(\mathbf{x})\|}, \tag{6-3}$$

where k is a parameter that controls how much the field will attract to its center, σ is a force scaling parameter, and α is a stretching parameter introduced to avoid instability. $\mathbf{f}$ is the force field, $\hat{\mathbf{f}}$ is the normalized and scaled force field, and $\mathbf{x}$ is a vector where the field is sampled. The field can also be sampled using a matrix of stacked points if we also stack node positions in a matrix and use the following equation:

$$\hat{\mathbf{F}}(\mathbf{X}) = \begin{bmatrix} \hat{\mathbf{f}}(\mathbf{x_1}) \\ \hat{\mathbf{f}}(\mathbf{x_2}) \\ \vdots \\ \hat{\mathbf{f}}(\mathbf{x_n}) \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \tag{6-4}$$

We use Algorithm 1 to generate this dataset. In this algorithm, we use $\mathbf{v}_{\mathcal{M}}^{(t)}$ and $\mathbf{p}_{\mathcal{M}}^{(t)}$ as velocities and positions in the manifold $\mathcal{M}$. At each frame, a time step t is taken from the vector $\mathbf{t}_s$, which is defined as $\{1, 2, 3, 4, ..., n\}$ where n is the number of time steps in the simulation. $S_{\mathcal{N}}(\mathbf{p}_{\mathcal{D}}^{(t)}, \mathbf{v}_{\mathcal{D}}^{(t)}, \mathbf{f_n}, \Delta t)$ is a simulator using a mesh on manifold $\mathcal{N}$, $M_{\mathcal{N}}$, and receiving a set of forces $\mathbf{f_n}$, a time step size Δt, other than positions and velocities to be updated.

Algorithm 1: *Torsion* Dataset Generation Procedure

$P_{\mathcal{U}} \leftarrow \{\}$
$P_{\mathcal{R}} \leftarrow \{\}$
for $t \in \mathbf{t}_s$ **do**
    $\mathbf{f_n} \leftarrow \tilde{\mathbf{F}}(\mathbf{N}_c)$
    $\mathbf{p}_{\mathcal{U}}^{(t+1)}, \mathbf{v}_{\mathcal{U}}^{(t+1)} \leftarrow S_{\mathcal{U}}(\mathbf{p}_{\mathcal{D}}^{(t)}, \mathbf{v}_{\mathcal{D}}^{(t)}, \mathbf{f_n}, \Delta t)$
    $\mathbf{p}_{\mathcal{R}}^{(t+1)}, \mathbf{v}_{\mathcal{R}}^{(t+1)} \leftarrow S_{\mathcal{R}}(\mathbf{p}_{\mathcal{R}}^{(t)}, \mathbf{v}_{\mathcal{R}}^{(t)}, \mathbf{f_n}, \Delta t)$
    $P_{\mathcal{U}} \leftarrow P_{\mathcal{U}} \cup \{\mathbf{p}_{\mathcal{U}}^{(t+1)}\}$
    $P_{\mathcal{R}} \leftarrow P_{\mathcal{R}} \cup \{\mathbf{p}_{\mathcal{R}}^{(t+1)}\}$
    $\mathbf{p}_{\mathcal{D}}^{(t+1)}, \mathbf{v}_{\mathcal{D}}^{(t+1)} \leftarrow \mathcal{T}_{\mathcal{D}}\mathbf{p}_{\mathcal{R}}^{(t+1)}, \mathcal{T}_{\mathcal{D}}\mathbf{v}_{\mathcal{R}}^{(t+1)}$
end for

The *Weight Meshes* are designed for the *Impulse* and *Stretch* datasets. These meshes have a small inflation in the middle that can be adjusted in height and serves to change the weight distribution of the mesh. We build this dataset by applying forces on the borders of this part because the possibility of it being at multiple heights creates more variation in the resulting simulations.

We use Algorithm 2 to generate the data for these datasets. The difference between this algorithm and Algorithm 1 is that the set of forces is predefined and time-dependent, being called $\mathbf{f_n}^{(t)}$. For the *Impulse* dataset, this set comprises an initial nonzero-valued force and subsequent zero-valued forces. For the *Stretch* dataset, a force is generated by combining a component parallel to the central axis of the mesh where the weight is located (Fig. 6.3), to create the stretching of the mesh, and another component parallel to the mesh to generate more variation to the data.

Algorithm 2: *Impulse* and *Stretch* Datasets Generation Procedure

$P_{\mathcal{U}} \leftarrow \{\}$
$P_{\mathcal{R}} \leftarrow \{\}$
for $t \in \mathbf{t}_s$ **do**
    $\mathbf{p}_{\mathcal{U}}^{(t+1)}, \mathbf{v}_{\mathcal{U}}^{(t+1)} \leftarrow S_{\mathcal{U}}(\mathbf{p}_{\mathcal{D}}^{(t)}, \mathbf{v}_{\mathcal{D}}^{(t)}, \mathbf{f_n}^{(t)}, \Delta t)$
    $\mathbf{p}_{\mathcal{R}}^{(t+1)}, \mathbf{v}_{\mathcal{R}}^{(t+1)} \leftarrow S_{\mathcal{R}}(\mathbf{p}_{\mathcal{R}}^{(t)}, \mathbf{v}_{\mathcal{R}}^{(t)}, \mathbf{f_n}^{(t)}, \Delta t)$
    $P_{\mathcal{U}} \leftarrow P_{\mathcal{U}} \cup \{\mathbf{p}_{\mathcal{U}}^{(t+1)}\}$
    $P_{\mathcal{R}} \leftarrow P_{\mathcal{R}} \cup \{\mathbf{p}_{\mathcal{R}}^{(t+1)}\}$
    $\mathbf{p}_{\mathcal{D}}^{(t+1)}, \mathbf{v}_{\mathcal{D}}^{(t+1)} \leftarrow \mathcal{T}_{\mathcal{D}}\mathbf{p}_{\mathcal{R}}^{(t+1)}, \mathcal{T}_{\mathcal{D}}\mathbf{v}_{\mathcal{R}}^{(t+1)}$
end for

7
Results and Discussion

In this chapter, we discuss our training and validation results, comparing our model with the reference $S_{\mathcal{R}}(\mathbf{x}, \Delta t)$ and coarse $S_{\mathcal{U}}(\mathbf{x}, \Delta t)$ solvers. To perform this analysis, we use a relative error metric:

$$MSE(\mathbf{P}_M, \mathbf{P}_{\mathcal{D}}) = \frac{1}{n} \sum_{t=1}^{n} \left(\mathbf{p}_M^t - \mathbf{p}_{\mathcal{D}}^t \right)^2, \tag{7-1}$$

$$E_M = 100 \left[\frac{MSE(\mathbf{P}_{\mathcal{U}}, \mathbf{P}_{\mathcal{D}}) - MSE(\mathbf{P}_M, \mathbf{P}_{\mathcal{D}})}{MSE(\mathbf{P}_{\mathcal{U}}, \mathbf{P}_{\mathcal{D}})} \right], \tag{7-2}$$

$$\Omega_M = \frac{1}{n} \sum_i E_M^i, \tag{7-3}$$

where $\mathbf{P}_M$ is a matrix containing the positions p_M^t at time steps $t \in \mathbf{t}_s$, generated by a model M. $\mathbf{P}_{\mathcal{D}}$ is a matrix containing the positions $\mathbf{p}_{\mathcal{D}}^t$, that are the downsampled reference positions, contained in the manifold $\mathcal{D}$. The metric $\Omega_M \in (-\infty, 100]$ is the average of E_M^i applied to all n trajectories in a dataset and can be interpreted as the percentage of discretization error correction, where positive values of Ω_M indicate a percentual decrease in the error, while negative values indicate a percentual increase in the error. We use these metrics to measure how much of the discretization error was corrected and how close our model is to the reference.

We train and evaluate the models in a computer with a single NVIDIA A100 GPU with 40GB of VRAM, two AMD EPYC 7352 24-core processors, and 512GB of RAM. In this setting, our training times average two days for the *Impulse* and *Stretch* datasets and three days for the *Torsion* dataset when training for 20 epochs. We train the neural network with the simulator in the training loop, showing that our training times are reasonable and our model can be integrated into a neural network layer to be later expanded by future work. Figure 7.1 shows the training loss graphs for our model's training.

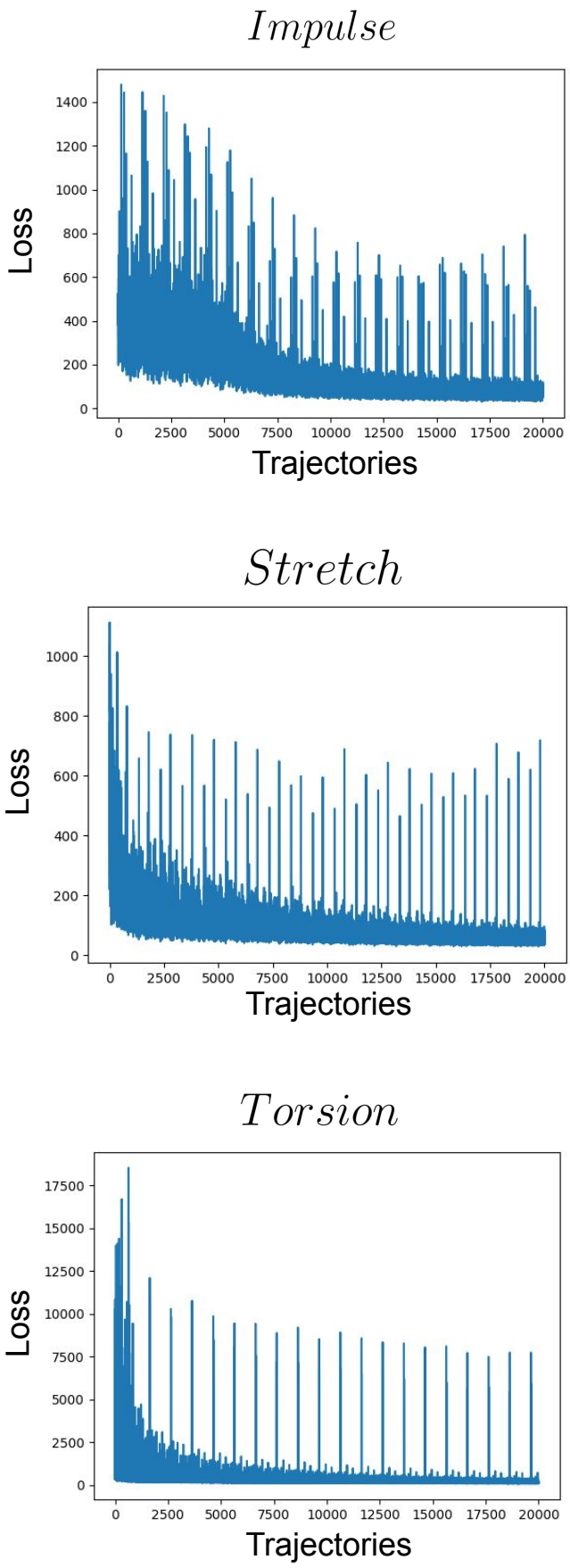

Figure 7.1: Trajectory-wise training losses for the *Impulse*, *Stretch*, and *Torsion* datasets.

To quantify the performance of our hybrid simulator, we compare it to $S_{\mathcal{R}}(\mathbf{x}, \Delta t)$ and $S_{\mathcal{U}}(\mathbf{x}, \Delta t)$ using Eqn. (7-3) on the trajectories of the $Impulse$, $Stretch$ and $Torsion$ validation datasets. Figures 7.2, 7.3 and 7.4 show the trajectories of a mesh node (depicted in Figure 6.3) of curated samples from our validation datasets predicted by the reference simulator, the coarse simulator, and our hybrid method. Figures 7.5, 7.6 and 7.7 show frames of the same simulations as the graphs in Figures 7.2, 7.3 and 7.4, comparing the simulations generated by our method to $S_{\mathcal{R}}(\mathbf{x}, \Delta t)$ and $S_{\mathcal{U}}(\mathbf{x}, \Delta t)$.

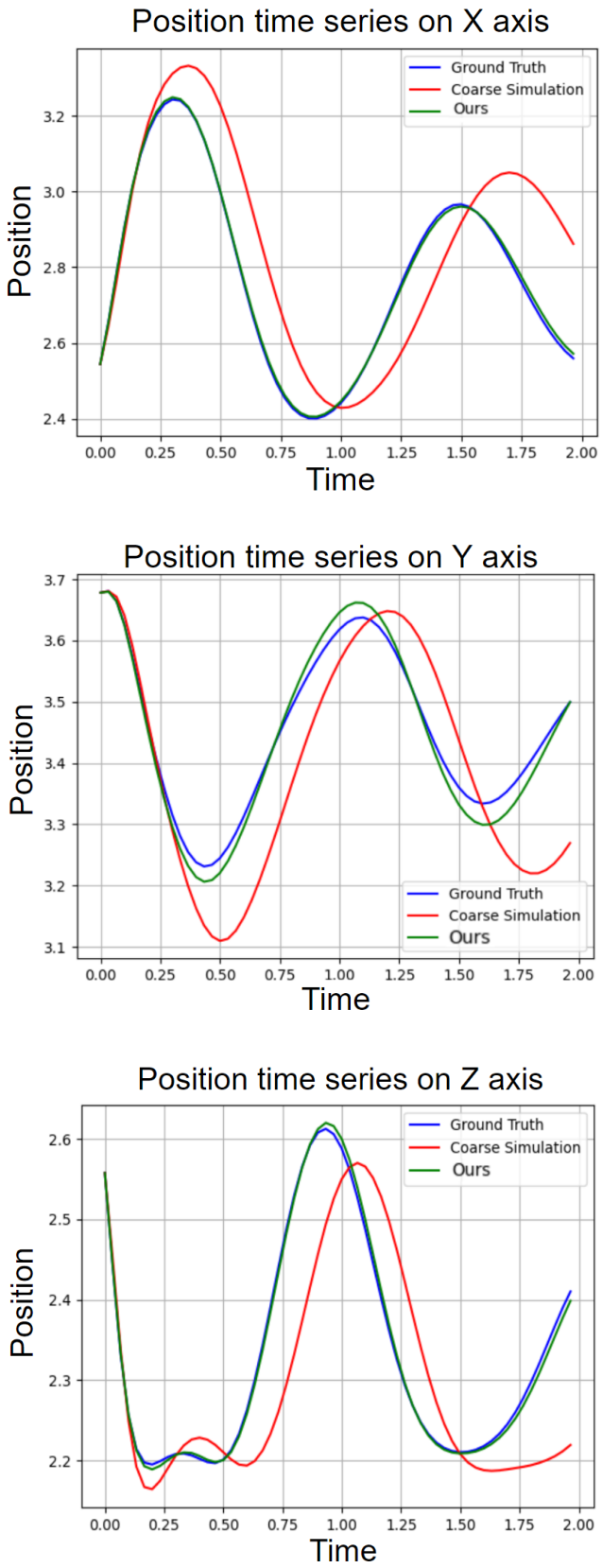

Figure 7.2: Comparison of node trajectories resulting from the application of all simulators in the first trajectory of the *Impulse* validation dataset.

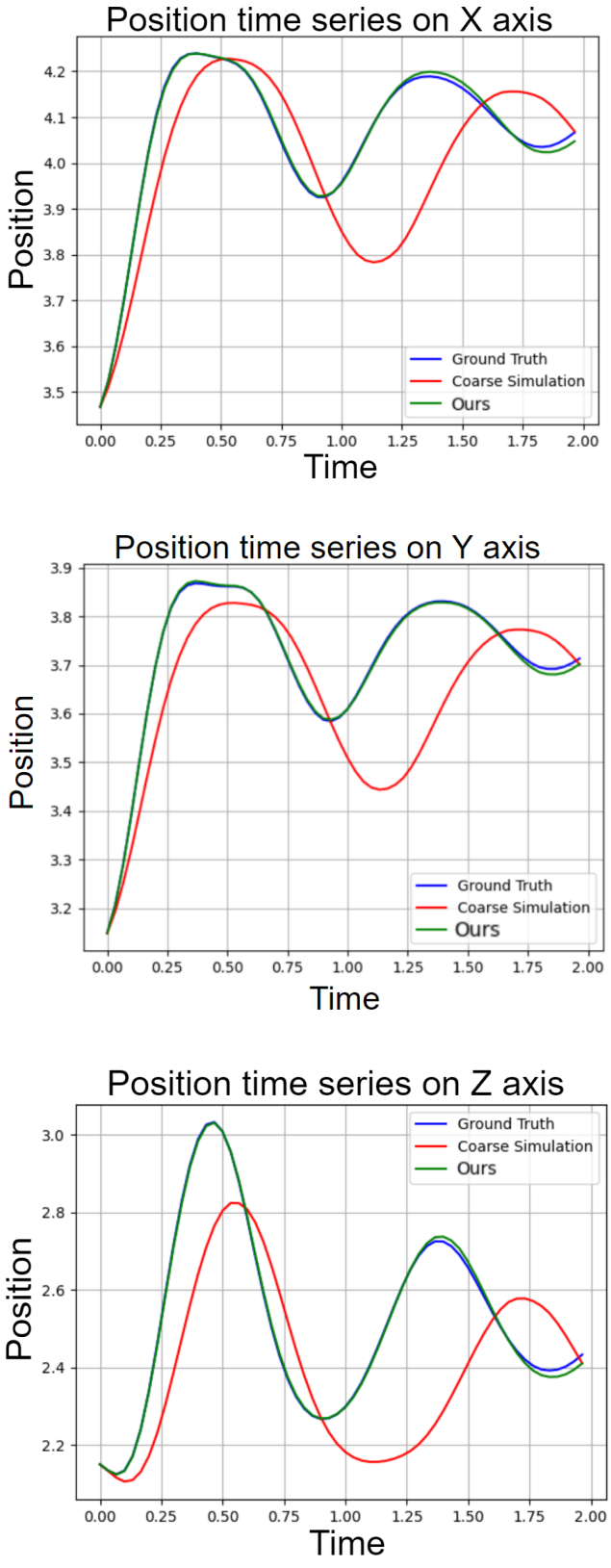

Figure 7.3: Comparison of node trajectories resulting from the application of all simulators in the first trajectory of the *Stretch* validation dataset.

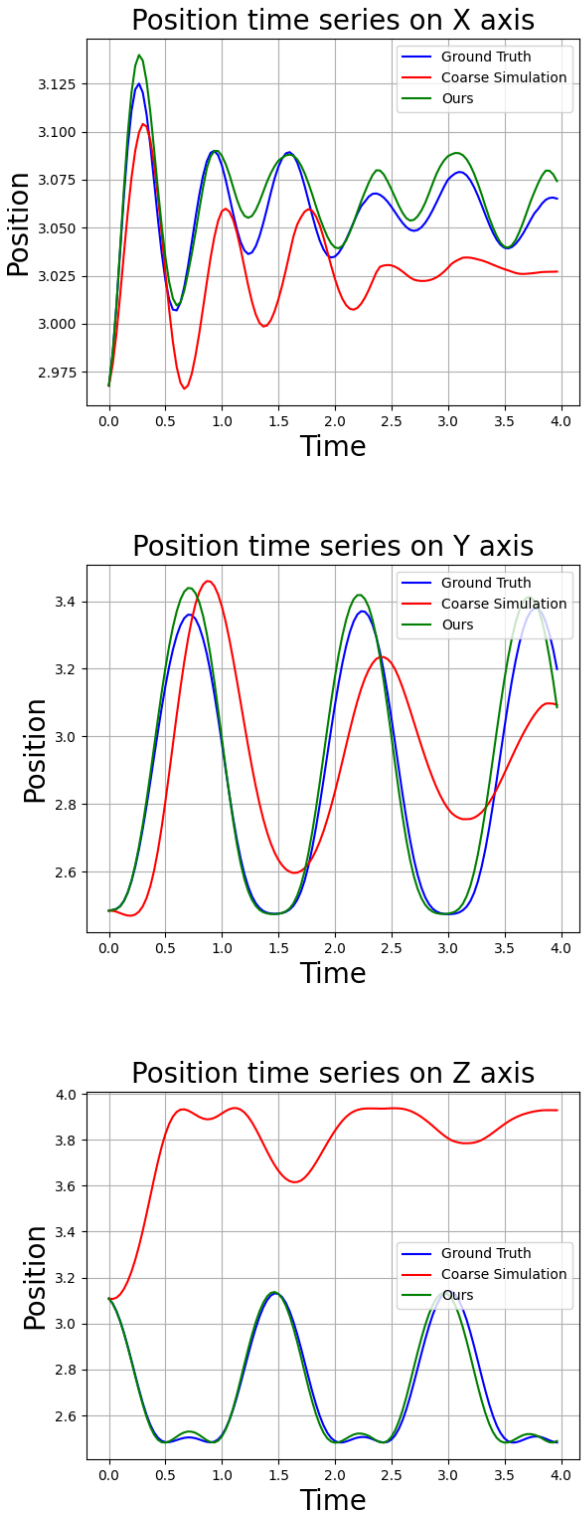

Figure 7.4: Comparison of node trajectories resulting from the application of all simulators in the first trajectory of the *Torsion* validation dataset.

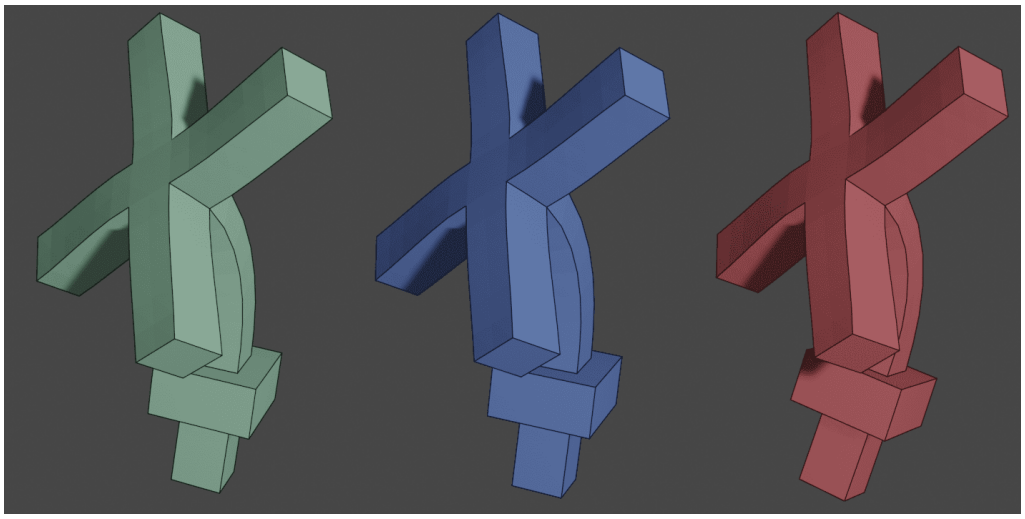

Figure 7.5: Comparison of the same frame from the first trajectory of the *Impulse* validation dataset, generated by our model (green), $S_{\mathcal{R}}(\mathbf{x}, \Delta t)$ (blue), and $S_{\mathcal{U}}(\mathbf{x}, \Delta t)$ (red).

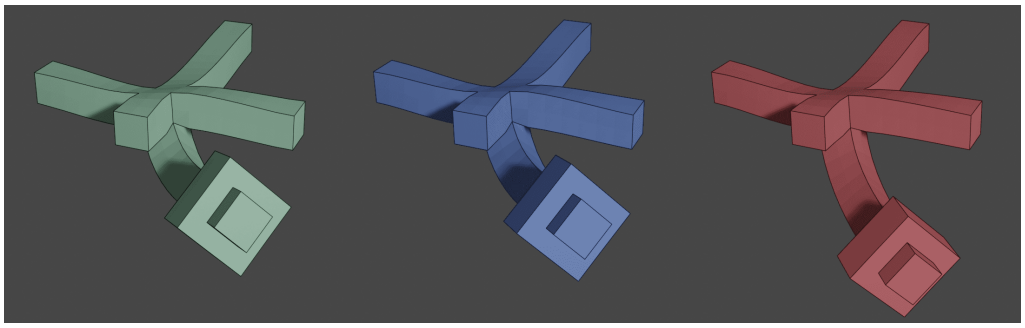

Figure 7.6: Comparison of the same frame from the second trajectory of the *Stretch* validation dataset, generated by our model (green), $S_{\mathcal{R}}(\mathbf{x}, \Delta t)$ (blue), and $S_{\mathcal{U}}(\mathbf{x}, \Delta t)$ (red).

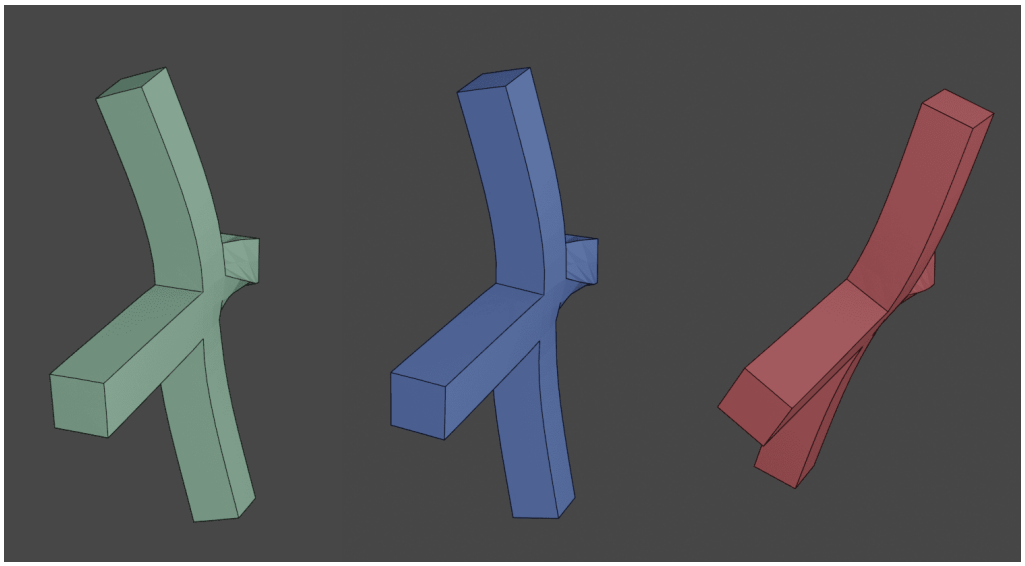

Figure 7.7: Comparison of the same frame from the fifth trajectory of the *Torsion* validation dataset, generated by our model (green), $S_{\mathcal{R}}(\mathbf{x}, \Delta t)$ (blue), and $S_{\mathcal{U}}(\mathbf{x}, \Delta t)$ (red).

Table 7.1 summarizes each solver's accuracy and computational efficiency. Our method significantly outperforms the coarse simulator in terms of accuracy, achieving an average numerical error correction of 95.4%. Furthermore, it demonstrates a computational speed up to 88 times faster than the reference solver, highlighting the efficiency of integrating MPNNs with FE simulation methods. The results in Table 7.1 also demonstrate that our method is faster and more accurate than an instance of DiffPD that uses a mesh eight times more refined than the coarse mesh used in $S_{\mathcal{C}}(\mathbf{x}, \Delta t)$, which we call *Intermediate Simulator*.

Method	*Impulse*		*Stretch*		*Torsion*	
	Ω_M	Speedup	Ω_M	Speedup	Ω_M	Speedup
Reference	100.0	1x	100.0	1x	100.0	1x
Intermediate	83.1 ± 0.04	17.57x	81.4 ± 0.02	12.99x	54.5 ± 7.4	12.89x
Coarse	0.0	147.21x	0.0	162.27x	0.0	173.22x
Ours	95.9 ± 0.006	81.23x	99.1 ± 0.52	81.17x	91.3 ± 3.9	88.85x

Table 7.1: Accuracy and computational speed comparison across different datasets. In the table $Speedup = \frac{t_{\mathcal{R}}}{t_{\mathcal{M}}}$, where $t_{\mathcal{R}}$ is the total time $S_{\mathcal{R}}(\mathbf{x}, \Delta t)$ took to perform the simulations of the validation dataset, and $t_{\mathcal{M}}$ is the time the method $\mathcal{M}$ took to perform the same task. Our method substantially improves accuracy when compared to the baselines and speed when compared to the Intermediate and Reference simulators, demonstrating its potential for efficient and precise simulations.

The results underscore the effectiveness of our hybrid approach in addressing the limitations associated with coarse mesh discretizations in FE

simulations. By leveraging the corrective capabilities of MPNNs, our method reduces numerical errors and achieves substantial gains in computational efficiency. Integrating our method into a neural network layer further enhances its versatility, allowing for seamless extensions in future work.

8
Conclusion

This work introduces a novel hybrid simulator that combines an MPNN with a differentiable numerical simulator, notably correcting an average of 95.4% of the numerical error associated with discretization and being up to 88 times faster than the reference solver.

Moreover, providing three new datasets and making our code publicly available encourages reproducibility and facilitates further advancements in the field. Despite the promising outcomes, we acknowledge certain limitations, such as the potential for overfitting. Also, the spikes in the graphs of Figure 7.1 indicate the presence of outliers in the datasets. We encourage future work to expand our training and evaluation datasets to contain a broader range of meshes and simulation cases to mitigate these problems.

Future work can also explore extensions of our model, such as multi-fidelity and multi-scale methods since it is fully differentiable and can be embedded into a neural network layer. Furthermore, as we use a generic formulation, future work can explore the capabilities of our hybrid model to learn other physical properties and behaviors, like thermodynamic expansion and solid-fluid interactions, as well as other correction functions, such as simulated-to-real and linear-to-nonlinear.

In conclusion, this study demonstrates the potential of hybrid simulation methods in achieving high accuracy and efficiency in modeling deformable solids. Our approach sets a new benchmark for computational simulations, offering a viable solution to the trade-off between mesh resolution and computational demand.

9
Bibliography

ABADI, M. et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

ANDERSON, J. Governing equations of fluid dynamics. In: _____. **Computational Fluid Dynamics**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 15–51. ISBN 978-3-540-85056-4. Disponível em: <https://doi.org/10.1007/978-3-540-85056-4_2>.

ARORA, R. Physrnet: Physics informed super-resolution network for application in computational solid mechanics. In: **2022 IEEE/ACM International Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S)**. [S.l.: s.n.], 2022. p. 13–18.

BAIGES, J. et al. A finite element reduced-order model based on adaptive mesh refinement and artificial neural networks. **International Journal for Numerical Methods in Engineering**, v. 121, n. 4, p. 588–601, 2020. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.6235>.

BELBUTE-PERES, F. de A. et al. End-to-end differentiable physics for learning and control. In: BENGIO, S. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2018. v. 31. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2018/file/842424a1d0595b76ec4fa03c46e8d755-Paper.pdf>.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **IEEE Transactions on Neural Networks**, v. 5, n. 2, p. 157–166, 1994.

BERG, J.; NYSTRöM, K. A unified deep artificial neural network approach to partial differential equations in complex geometries. **Neurocomputing**, v. 317, p. 28–41, 2018. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S092523121830794X>.

BLACK, N.; NAJAFI, A. R. Learning finite element convergence with the multi-fidelity graph neural network. **Computer Methods in Applied Mechanics and Engineering**, v. 397, p. 115120, 2022. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S004578252200305X>.

BOUAZIZ, S. et al. Projective dynamics: Fusing constraint projections for fast simulation. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 33, n. 4, jul 2014. ISSN 0730-0301. Disponível em: <https://doi.org/10.1145/2601097.2601116>.

CAO, Y. et al. Efficient learning of mesh-based physical simulation with bi-stride multi-scale graph neural network. In: **Proceedings of the 40th International Conference on Machine Learning**. [S.l.]: JMLR.org, 2023. (ICML'23).

CHEN, D. et al. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 34, n. 04, p. 3438–3445, Apr. 2020. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/5747>.

CHEN, Q. et al. Predicting dynamic responses of continuous deformable bodies:a graph-based learning approach. **Computer Methods in Applied Mechanics and Engineering**, v. 420, p. 116669, 2024. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782523007922>.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals and Systems**, v. 2, n. 4, p. 303–314, Dec 1989. ISSN 1435-568X. Disponível em: <https://doi.org/10.1007/BF02551274>.

DALTON, D.; GAO, H.; HUSMEIER, D. Emulation of cardiac mechanics using graph neural networks. **Computer Methods in Applied Mechanics and Engineering**, v. 401, p. 115645, 2022. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782522006004>.

DALTON, D.; HUSMEIER, D.; GAO, H. Physics-informed graph neural network emulation of soft-tissue mechanics. **Computer Methods in Applied Mechanics and Engineering**, v. 417, p. 116351, 2023. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782523004759>.

DIAO, Y. et al. Solving multi-material problems in solid mechanics using physics-informed neural networks based on domain decomposition technology. **Computer Methods in Applied Mechanics and Engineering**, v. 413, p. 116120, 2023. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S004578252300244X>.

DU, T. et al. Diffpd: Differentiable projective dynamics. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 41, n. 2, nov 2021. ISSN 0730-0301. Disponível em: <https://doi.org/10.1145/3490168>.

FEY, M.; LENSSEN, J. E. **Fast Graph Representation Learning with PyTorch Geometric**. 2019. Cite arxiv:1903.02428. Disponível em: <http://arxiv.org/abs/1903.02428>.

FORTUNATO, M. et al. Multiscale meshgraphnets. In: **ICML 2022 2nd AI for Science Workshop**. [s.n.], 2022. Disponível em: <https://openreview.net/forum?id=G3TRIsmMhhf>.

GAO, H.; ZAHR, M. J.; WANG, J.-X. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. **Computer Methods in Applied Mechanics and Engineering**, v. 390, p. 114502, 2022. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782521007076>.

GILMER, J. et al. Neural message passing for quantum chemistry. In: PRECUP, D.; TEH, Y. W. (Ed.). **Proceedings of the 34th International Conference on Machine Learning**. PMLR, 2017. (Proceedings of Machine Learning Research, v. 70), p. 1263–1272. Disponível em: <https://proceedings.mlr.press/v70/gilmer17a.html>.

GONZALEZ, O.; STUART, A. M. **A First Course in Continuum Mechanics**. [S.l.]: Cambridge University Press, 2008. (Cambridge Texts in Applied Mathematics).

GONZáLEZ, D.; CHINESTA, F.; CUETO, E. Learning corrections for hyperelastic models from data. **Frontiers in Materials**, v. 6, 2019. ISSN 2296-8016. Disponível em: <https://www.frontiersin.org/articles/10.3389/fmats.2019.00014>.

GUAN, Y. et al. Stable a posteriori les of 2d turbulence using convolutional neural networks: Backscattering analysis and generalization to higher re via transfer learning. **Journal of Computational Physics**, v. 458, p. 111090, 2022. ISSN 0021-9991. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999122001528>.

GUENNEBAUD, G.; JACOB, B. et al. **Eigen v3**. 2010. Http://eigen.tuxfamily.org.

HAHN, D. et al. Real2sim: Visco-elastic parameter estimation from dynamic motion. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 6, nov 2019. ISSN 0730-0301. Disponível em: <https://doi.org/10.1145/3355089.3356548>.

HAN, S. et al. A dnn-based data-driven modeling employing coarse sample data for real-time flexible multibody dynamics simulations. **Computer Methods in Applied Mechanics and Engineering**, v. 373, p. 113480, 2021. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782520306654>.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Networks**, v. 2, n. 5, p. 359–366, 1989. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/0893608089900208>.

HU, W.-F. et al. A shallow physics-informed neural network for solving partial differential equations on static and evolving surfaces. **Computer Methods in Applied Mechanics and Engineering**, v. 418, p. 116486, 2024. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782523006102>.

HU, Y. et al. Chainqueen: A real-time differentiable physical simulator for soft robotics. In: **2019 International Conference on Robotics and Automation (ICRA)**. IEEE Press, 2019. p. 6265–6271. Disponível em: <https://doi.org/10.1109/ICRA.2019.8794333>.

HUANG, Z. et al. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In: **International Conference on Learning Representations**. [s.n.], 2021. Disponível em: <https://openreview.net/forum?id=xCcdBRQEDW>.

ILLARRAMENDI, E. A.; BAUERHEIM, M.; CUENOT, B. Performance and accuracy assessments of an incompressible fluid solver coupled with a deep convolutional neural network. **Data-Centric Engineering**, v. 3, p. e2, 2022.

KERIVEN, N. Not too little, not too much: a theoretical analysis of graph (over)smoothing. In: KOYEJO, S. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2022. v. 35, p. 2268–2281. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2022/file/0f956ca6f667c62e0f71511773c86a59-Paper-Conference.pdf>.

KINGMA, D.; BA, J. Adam: A method for stochastic optimization. In: **International Conference on Learning Representations (ICLR)**. San Diega, CA, USA: [s.n.], 2015.

KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. In: **International Conference on Learning Representations**. [s.n.], 2017. Disponível em: <https://openreview.net/forum?id=SJU4ayYgl>.

KOCHKOV, D. et al. Machine learning–accelerated computational fluid dynamics. **Proceedings of the National Academy of Sciences**, v. 118, n. 21, p. e2101784118, 2021. Disponível em: <https://www.pnas.org/doi/abs/10.1073/pnas.2101784118>.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998.

LI, W.; BAZANT, M. Z.; ZHU, J. A physics-guided neural network framework for elastic plates: Comparison of governing equations-based and energy-based approaches. **Computer Methods in Applied Mechanics and Engineering**, v. 383, p. 113933, 2021. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S004578252100270X>.

LIU, Q.; NICKEL, M.; KIELA, D. Hyperbolic graph neural networks. In: WALLACH, H. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2019. v. 32. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2019/file/103303dd56a731e377d01f6a37badae3-Paper.pdf>.

LIU, T.; BOUAZIZ, S.; KAVAN, L. Quasi-newton methods for real-time simulation of hyperelastic materials. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 36, n. 3, may 2017. ISSN 0730-0301. Disponível em: <https://doi.org/10.1145/2990496>.

LUKOVNIKOV, D.; LEHMANN, J.; FISCHER, A. **Improving the Long-Range Performance of Gated Graph Neural Networks**. 2020.

LUO, R. et al. Nnwarp: Neural network-based nonlinear deformation. **IEEE Transactions on Visualization and Computer Graphics**, v. 26, n. 4, p. 1745–1759, 2020.

MA, P. et al. Learning neural constitutive laws from motion observations for generalizable pde dynamics. In: **Proceedings of the 40th International Conference on Machine Learning**. [S.l.]: JMLR.org, 2023. (ICML'23).

MA, P. et al. Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. **ACM Transactions on Graphics (TOG)**, ACM New York, NY, USA, v. 40, n. 4, p. 132, 2021.

MCGREIVY, N.; HAKIM, A. **Invariant preservation in machine learned PDE solvers via error correction**. 2023.

MILANO, F. et al. Primal-dual mesh convolutional neural networks. In: LAROCHELLE, H. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2020. v. 33, p. 952–963. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2020/file/0a656cc19f3f5b41530182a9e03982a4-Paper.pdf>.

NING, L. et al. A peridynamic-informed neural network for continuum elastic displacement characterization. **Computer Methods in Applied Mechanics and Engineering**, v. 407, p. 115909, 2023. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782523000324>.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. In: _____. **Proceedings of the 33rd International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2019.

PATHAK, J. et al. **Using Machine Learning to Augment Coarse-Grid Computational Fluid Dynamics Simulations**. 2020.

PESTOURIE, R. et al. Physics-enhanced deep surrogates for partial differential equations. **Nature Machine Intelligence**, v. 5, n. 12, p. 1458–1465, Dec 2023. ISSN 2522-5839. Disponível em: <https://doi.org/10.1038/s42256-023-00761-y>.

PFAFF, T. et al. Learning mesh-based simulation with graph networks. In: **International Conference on Learning Representations**. [s.n.], 2021. Disponível em: <https://openreview.net/forum?id=roNqYL0_XP>.

QIAO, Y.-L. et al. Differentiable simulation of soft multi-body systems. In: BEYGELZIMER, A. et al. (Ed.). **Advances in Neural Information Processing Systems**. [s.n.], 2021. Disponível em: <https://openreview.net/forum?id=j3fpZLKcXF>.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016.

REZAEI, S. et al. A mixed formulation for physics-informed neural networks as a potential solver for engineering problems in heterogeneous domains: Comparison with finite element method. **Computer Methods in Applied Mechanics and Engineering**, v. 401, p. 115616, 2022. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782522005722>.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: NAVAB, N. et al. (Ed.). **Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015**. Cham: Springer International Publishing, 2015. p. 234–241. ISBN 978-3-319-24574-4.

SHAKIBAJAHROMI, B.; KIM, E.; BREEN, D. E. Rimeshgnn: A rotation-invariant graph neural network for mesh classification. In: **Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)**. [S.l.: s.n.], 2024. p. 3150–3160.

SHEN, S. et al. High-order differentiable autoencoder for nonlinear model reduction. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 40, n. 4, jul 2021. ISSN 0730-0301. Disponível em: <https://doi.org/10.1145/3450626.3459754>.

SIRIGNANO, J.; MACART, J. F.; FREUND, J. B. Dpm: A deep learning pde augmentation method with application to large-eddy simulation. **Journal of Computational Physics**, v. 423, p. 109811, 2020. ISSN 0021-9991. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999120305854>.

SIRIGNANO, J.; SPILIOPOULOS, K. Dgm: A deep learning algorithm for solving partial differential equations. **Journal of Computational Physics**, v. 375, p. 1339–1364, 2018. ISSN 0021-9991. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999118305527>.

STUART, A. M.; HUMPHRIES, A. R. Dynamical systems and numerical analysis. In: . [s.n.], 1996. Disponível em: <https://api.semanticscholar.org/CorpusID:117039154>.

THUEREY, N. et al. **Physics-based Deep Learning**. WWW, 2021. Disponível em: <https://physicsbaseddeeplearning.org>.

TIAN, J. et al. Surrogate permeability modelling of low-permeable rocks using convolutional neural networks. **Computer Methods in Applied Mechanics and Engineering**, v. 366, p. 113103, 2020. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782520302875>.

UM, K. et al. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. In: **Proceedings of the 34th International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2020. (NIPS'20). ISBN 9781713829546.

URIARTE, C.; PARDO, D.; OMELLA Ángel J. A finite element based deep learning solver for parametric pdes. **Computer Methods in Applied Mechanics and Engineering**, v. 391, p. 114562, 2022. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782521007374>.

WANG, N.; CHANG, H.; ZHANG, D. Surrogate and inverse modeling for two-phase flow in porous media via theory-guided convolutional neural network. **Journal of Computational Physics**, v. 466, p. 111419, 2022. ISSN 0021-9991. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999122004818>.

XU, J.; DURAISAMY, K. Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics. **Computer Methods in Applied Mechanics and Engineering**, v. 372, p. 113379, 2020. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782520305648>.

ZHANG, Z. et al. A physics-informed convolutional neural network for the simulation and prediction of two-phase darcy flows in heterogeneous porous media. **Journal of Computational Physics**, v. 477, p. 111919, 2023. ISSN 0021-9991. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999123000141>.

ZHANG, Z.-Y.; CAI, S.-J.; ZHANG, H. A symmetry group based supervised learning method for solving partial differential equations. **Computer Methods in Applied Mechanics and Engineering**, v. 414, p. 116181, 2023. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782523003055>.