UNIVERSITY OF CALIFORNIA

Los Angeles

Advancing Physics-Based Simulations: Integrating Conventional and Machine-Learning Approaches for Enhanced Computational Efficiency

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Yadi Cao

2024

ABSTRACT OF THE DISSERTATION

Advancing Physics-Based Simulations:
Integrating Conventional and Machine-Learning Approaches for
Enhanced Computational Efficiency

by

Yadi Cao
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2024
Professor Demetri Terzopoulos, Chair

This thesis presents novel approaches to improve the accuracy and efficiency of scientific simulations, particularly those involving complex geometries, intrinsic physical modeling, and demanding computational costs.

The first contribution extends the MPM to unstructured meshes, addressing the challenges of the transfer kernel's gradient continuity and stability issue on any general mesh tesselation. The Unstructured Moving Least Squares MPM (UMLS-MPM) incorporates a diminishing function into the MLS kernel's sample weights, ensuring an analytically continuous function and gradient reconstruction. It is the first-of-its-kind framework in this field. Several numerical test cases demonstrate the method's stability and accuracy.

The second contribution is a hybrid scheme for modeling the interaction between compressible flow, shock waves, and deformable structures. By combining recent advancements in time-splitting compressible flow and Material Point Methods (MPMs), this approach seamlessly integrates Eulerian and Lagrangian/Eulerian methods for monolithic flow-structure interactions. Reflective and penetrable boundary conditions handle deforming boundaries with sub-cell particles, while a mixed-order finite element formulation utilizing B-spline shape functions discretizes the coupled velocity-pressure system. This comprehensive framework accurately captures shock wave propagation, temperature/density-induced buoyancy effects, and topology changes in solids.

The third contribution addresses challenges in learning physical simulations on large-scale meshes using Graph Neural Networks (GNNs). Existing state-of-the-art methods often encounter issues related to over-smoothing and incorrect edge construction during multi-scale adaptation. To overcome these limitations, a novel pooling strategy, termed *bi-stride*, is introduced. This approach, inspired by bipartite graph structures, involves pooling nodes on alternate frontiers of the breadth-first search (BFS), eliminating the need for labor-intensive manual creation of coarser meshes and mitigating incorrect edge problems. The proposed *BSMS-GNN* framework employs non-parametrized pooling and unpooling through interpolations, resulting in a substantial reduction of computational costs and improved efficiency. Experimental results demonstrate the superiority of the *BSMS-GNN* framework in terms of both accuracy and computational efficiency in representative physical simulations on large-scale meshes.

The dissertation of Yadi Cao is approved.

University of California, Los Angeles

2024

To my wife and parents,

for your unwavering love and support

as I hike up the mountain called "life."

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am deeply grateful for the support and guidance provided by many individuals throughout my doctoral journey. Their collective wisdom and encouragement have shaped my experience and achievements.

Foremost, I thank my co-advisor, Professor Chenfanfu Jiang, who provided detailed and patient technical guidance on mathematics, mechanics, coding, and technical writing during the early phase of my journey. His assistance in finding industrial connections has opened new avenues for my research.

My gratitude extends to my co-advisor, Professor Demetri Terzopoulos, for his invaluable guidance and advice on navigating an academic career. His mentorship in developing my research tastes, collaborating effectively, crafting application packages, and mentoring junior graduate students has been pivotal to my career development.

I am thankful to committee member Professor Shaowu Pan for his assistance in pinpointing my research direction, exploring state-of-the-art work in our field, sparking discussions, and then collaborations, and navigating the landscape of funding and grant proposals. I also thank Professors Aditya Grover and Quanquan Gu for their kind suggestions during my thesis defense and on my dissertation.

I thank Professor Liu Yang of UCLA's Mathematics Department. His trust in my research tastes and ambitions led not only to several collaborative projects but also to a trusting and fruitful friendship.

I appreciate Professor Minchen Li of the Computer Science Department at Carnegie Mellon University, who, during his tenure at the UCLA Multiples Lab, provided me with patient and thorough technical guidance in the Finite Element Method and numerical optimization.

I am grateful to Professors Wei Wang and Yizhou Sun of UCLA's Computer Science Department for introducing me into their active ScAI lab. Their openness facilitated many engaging discussions and collaborations with fellow graduate students.

My experience was enriched by Dr. Menglei Chai at Google, who not only mentored me but also gave me several interesting tours around Los Angeles during my internship, making it enjoyable and memorable.

I must also acknowledge my peers at the Multiples Lab—Tianyi Xie, Yidong Zhao, Yunuo Chen, Xuan Li, Yu Fang, Yuxing Qiu, Yu Chang, and Ziyin Qu—and at the ScAI Lab—Zijie Huang, Xiao Luo, Wanjia Zhao, Fang Sun, Yanna Ding, Fred Xu, and Yanqiao Zhu. Additionally, heartfelt thanks go to my invaluable friends Haofan Lu, Tianxiang Li, Wenxin Song, Huan Xu, Lisang Ding, and Yuxuan Liu. Your companionship and our countless fruitful discussions have significantly enriched this special chapter of my life. Thank you.

VITA

2024	PhD in Computer Science, UCLA
2022–2024	Teaching Assistant, Department of Computer Science, UCLA
2023–2024	Student Researcher, Google LLC
2022–2023	Researcher Intern, Snap Inc.
2021–2022	Graduate Student Researcher, Department of Mathematics, UCLA
2021	SDE Intern, Taichi Graphics
2019–2020	CAE Software Research Developer, ShonDynamics
2018	MASc in Mechanical Engineering, University of British Columbia (UBC)
2016	BEng, University of Science and Technology of China (USTC)

PUBLICATIONS

Cao, Y., Zhao, Y., Li, M., Yang, Y., Choo, J., Terzopoulos, D., & Jiang, C. (2023). Material Point Methods on Unstructured Tessellations: A Stable Kernel Approach With Continuous Gradient Reconstruction. *arXiv:2312.10338*, 1–37.

Cao, Y., Chai, M., Li, M., & Jiang, C. (2023). Efficient Learning of Mesh-Based Physical Simulation with Bi-Stride Multi-Scale Graph Neural Network. *Proceedings of the 40th International Conference on Machine Learning*, 202, 3541–3558.

Cao, Y., Chen, Y., Li, M., Yang, Y., Zhang, X., Aanjaneya, M., & Jiang, C. (2022). An Efficient B-Spline Lagrangian/Eulerian Method for Compressible Flow, Shock Waves, and Fracturing Solids. *ACM Transactions on Graphics (TOG)*, **41**(5), 1–13.

Li, X., Cao, Y., Li, M., Yang, Y., Schroeder, C., & Jiang, C. (2022). Plasticitynet: Learning to Simulate Metal, Sand, and Snow for Optimization Time Integration. *Advances in*

Neural Information Processing Systems, 35, 27783–27796.

Huang, Z., Zhao, W., Gao, J., Hu, Z., Luo, X., Cao, Y., Chen, Y., Sun, Y., & Wang, W. (2023). TANGO: Time-Reversal Latent GraphODE for Multi-Agent Dynamical Systems. *NeurIPS Workshop on the Symbiosis of Deep Learning and Differential Equations III*, 1–22

Fang, Y., Li, M., Cao, Y., Li, X., Wolper, J., Yang, Y., & Jiang, C. (2023). Augmented Incremental Potential Contact for Sticky Interactions. *IEEE Transactions on Visualization and Computer Graphics*, 1–13, Date of Publication: 14 July 2023.

CHAPTER 1

Introduction

In the realm of scientific exploration, simulation has emerged as a reliable and rigorous method for conducting research across various domains. From computational fluid dynamics (CFD) (Anderson and Wendt, 1995; Eymard et al., 2000; Reddy and Gartling, 2010; Bridson, 2015) to computational solid elastodynamics (Fung et al., 2001; De Borst et al., 2012; Hughes, 2012; Jiang et al., 2016a), simulations have proven to be invaluable tools in understanding and visualizing intricate physical phenomena. This introduction delves into the dynamic landscape of simulation, highlighting its significance, challenges, and the innovative solutions that drive its advancement.

While simulations hold immense potential, their integration into real-world applications is not without challenges. These challenges include handling complex geometries and boundaries, such as those encountered in computer graphics (Li et al., 2020a; Fang et al., 2021) and CFD (Osher and Fedkiw, 2001; Peskin, 2002; Taira and Colonius, 2007), addressing complex physical phenomena like plasticity (Klár et al., 2016; Jiang et al., 2017a; Gao et al., 2017; Li et al., 2022b,a) and turbulence modeling (Alfonsi, 2009; Germano et al., 1991; Shih et al., 1995; Piomelli, 1999), and dealing with scenarios where only partial observations are available, such as human body modeling. Additionally, simulations often come with demanding computational costs, as exemplified by large eddy simulation (LES) (Piomelli, 1999) and direct numerical simulation (DNS) of CFD (Moin and Mahesh, 1998).

Traditionally, addressing these challenges has relied heavily on the expertise of researchers who meticulously crafted solutions tailored to specific problems. While these expert-driven approaches have proven effective, they often require extensive manual intervention and fine-tuning. In recent years, the field of machine learning has evolved significantly, demonstrating its potential to complement, and in some cases, replace conventional simulation methods as surrogate models (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020; Sun et al., 2020). Additionally, substituting traditional numerical solvers with a physics-informed loss term (Raissi et al., 2019; Karniadakis et al., 2021; Gao et al., 2021, 2022), and learning models that operate at coarser resolutions, can accelerate future simulations (Kochkov et al., 2021). This evolving synergy between simulation and machine learning holds the promise of revolutionizing the application of simulation and computing in real-world scenarios.

This thesis focuses on leveraging both conventional and machine learning approaches to push the boundaries of accuracy and efficiency in scientific computing and simulations. By harnessing the strengths of both methodologies, the aim is to address the challenges associated with simulations involving *complex geometries, intricate and partially unknown modeling, and demanding computational costs*. From traditional material point methods and CFD to the development of state-of-the-art machine learning methods for simulations, each chapter contributes to the overarching goal of enhancing the accuracy and efficiency of simulations. The next section will motivate the major research components of the thesis in greater detail.

1.1 Motivations

1.1.1 Unstructured Moving Least Squares MPMs

The Material Point Method (MPM) (Sulsky et al., 1995) is a hybrid Eulerian-Lagrangian approach, originally introduced to solid mechanics to extend the capabilities of both the Fluid-Implicit Particle (FLIP) method (Brackbill et al., 1988) and the Particle-in-Cell (PIC) method (Harlow, 1962). MPM tracks all physical attributes on a collection of particles, while leveraging a background grid to solve the governing equations. The effectiveness of MPM is well-documented in handling extreme deformations of solid materials including, but not limited to, biological soft tissues (Ionescu et al., 2006; Guilkey et al., 2006), explosive materials (Guilkey et al., 2007; Ma et al., 2009a), sand (Homel et al., 2014; Klár et al., 2016; Tampubolon et al., 2017), and snow (Stomakhin et al., 2013; Gaume et al., 2018, 2019).

In terms of Lagrangian formulations, MPM is classified into total Lagrangian (Vaucorbeil et al., 2020; Vaucorbeil and Nguyen, 2021; Vaucorbeil et al., 2022a,b), where particles are perceived as a static embedding within their initial cells, and updated Lagrangian (Pretti et al., 2023) MPM, wherein a particle's neighboring cell is dynamically updated at each timestep based on the particle's new location. Each approach presents distinct advantages and challenges: The total Lagrangian MPM avoids numerical dissipation errors and artificial fractures, but faces difficulties with significant mesh distortion, whereas the updated Lagrangian MPM demonstrates enhanced robustness in scenarios involving large mesh distortions. Our study concentrates on the updated Lagrangian MPM.

Despite its numerous applications, the updated Lagrangian MPM predominantly utilizes a uniformly-structured background grid. This poses challenges when simulating domains with complex geometries, commonly encountered in mechanical and geotechnical engineering (Fern et al., 2019), due to difficulties in discretizing space conformally. Consequently, several researchers (Wikeckowski, 2004; Beuth et al., 2011; Jassim et al., 2013; Wang et al., 2021) have advocated the adoption of unstructured (2D) triangles or (3D) tetrahedra for discretization, offering flexibility in accommodating geometrically complex boundaries. However, most existing methods on the unstructured mesh rely on a piecewise linear ($\mathcal{C}^0$) basis function (Wikeckowski, 2004; Beuth et al., 2011; Jassim et al., 2013; Wang et al., 2021), leading to discontinuous gradients across element boundaries. Since stress evaluation depends on the gradient of the projection kernel between particles and grids, particles crossing cell boundaries induce oscillations in the stress field, thereby generating substantial numerical errors known as cell-crossing error (Bardenhagen and Kober, 2004).

Efforts to mitigate this cell-crossing error include the generalized interpolation material point (GIMP) method (Bardenhagen and Kober, 2004; Charlton et al., 2017), dual domain

MPM (DDMPM) (Zhang et al., 2011), utilization of high-order basis functions like B-splines (Steffen et al., 2008; Gan et al., 2018), and methods based on moving least squares (MLS) basis functions (Hu et al., 2018; Tran et al., 2019). Despite these advancements, limitations persist, either restricted to structured grids or confined to 2D tessellation with triangle elements (Koster et al., 2021). The cell-crossing error remains a notable challenge in applying general unstructured tessellations within MPM across both 2D and 3D.

1.1.2 Shockwave and Compressible Flow Simulation with the MPM

Supersonic motions and the detonation of explosive devices give rise to shock waves propagating through the air. Characterized by rapid pressure changes, shock waves carry an immense amount of energy having destructive impacts on structures, including rock fragmentation, organic tissue rupture, and soil displacement. Accurately simulating these phenomena presents formidable challenges in efficiently handling both compressible flow and its intricate two-way interaction with solids, particularly under conditions of extreme deformation and topological changes. The ignition of explosive materials in fluids leads to rapid motion exceeding the speed of sound, generating high-energy shock wavefronts that cause extensive material disruption. These phenomena pose multifaceted challenges to existing numerical simulation schemes.

In the domain of computer graphics, many established methods focus on simulating incompressible fluids. Among these, hybrid Lagrangian/Eulerian approaches, such as Particle-in-Cell (PIC) methods (Brackbill et al., 1988; Bridson, 2015; Jiang et al., 2015, 2017b; Fu et al., 2017), are widely adopted. PIC methods track fluid motion through particles, facilitating the creation of initial fluid volumes or emission of fluids from sources. Additionally, the time-splitting scheme (Chorin, 1967; Bridson, 2015) decouples the nonlinear advection step from linear steps, enabling efficient solution of each step. Various Eulerian advection schemes have been developed to accommodate large time steps (Stam, 1999; Kim et al., 2005; Qu et al., 2019). Notably, the advection process inherently conserves mass when using particle-based fluid representation. Conversely, enforcing incompressibility often involves pressure projection, entailing the solution of a pressure Poisson's equation. The pressure projection system is typically symmetric positive definite (SPD) (Batty et al., 2007; Bridson, 2015) when a uniform grid represents constant fluid density. For liquids with free surfaces, domain sparsity (Setaluri et al., 2014; Wu et al., 2018) is frequently exploited to reduce memory usage and computational cost.

Adapting incompressible flow solvers, computer graphics researchers have devised methods for modeling explosion effects by introducing artificial volume changes. A prevalent approach involves "divergence control" to enforce a specified source value for pressure divergence, effectively simulating non-uniform density. This technique has been used to mimic the appearance of expanding smoke plumes (Feldman et al., 2003; Takeshita et al., 2003). The Boussinesq approximation (Spiegel and Veronis, 1960) is another method for simulating buoyancy-like effects. Procedural explosion models based on grid-based incompressible flow simulations have also been developed (Kawada and Kanai, 2011). However, these approaches often neglect various physical quantities except velocity, potentially yielding misleading effects. The Boussinesq approximation-based methods, in particular, rely on a temperature field-based buoyancy model, which may fall outside its reliable regime, leading to convergence issues. Despite their visually plausible outcomes, these methods lack physical accuracy, and the quality of their results relies on artificial parameter tuning.

Unfortunately, extending from incompressible to compressible flow is far from straightforward, introducing several challenges:

- The utilization of particles and sparse grids to represent the fluid domain is no longer viable due to the presence of ambient air, leading to a substantial increase in the number of degrees of freedom (DOFs).
- The preference for direct one-step methods over operator splitting arises due to internal relations, such as the Equation of State, between pressure and other conserved variables (Forrer and Jeltsch, 1998; Forrer and Berger, 1999; Monasse et al., 2012). This introduces additional complexities when handling nonlinear terms using various methods like characteristic decomposition (Deconinck et al., 1993; Fey,

1998).

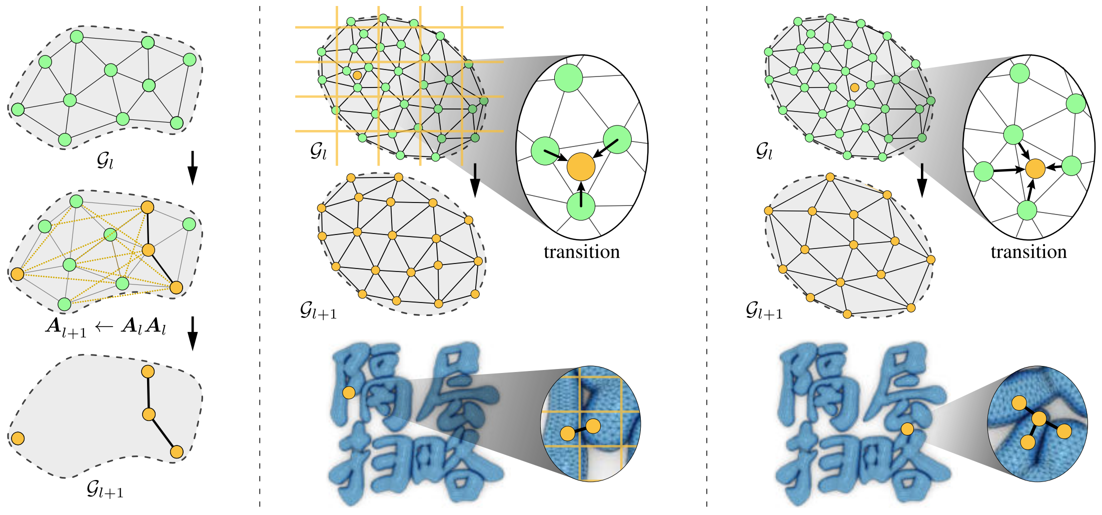
- More accurate advection schemes, such as the Weighted Essentially Non-Oscillatory (WENO) scheme (Shu and Osher, 1988; Liu et al., 1994), require conservatism and often necessitate small time steps strictly bounded by a small CFL number and the sound speed.
- The linear system associated with implicit integration is not guaranteed to be SPD due to differing densities and resultant coefficients in the mass matrix.

Given the inevitable increase in the number of DOFs, much research in the compressible flow domain has focused on addressing the last three challenges.

1.1.3 Bi-Stride Multi-Scale GNNs for Mesh-Based Physical Simulation

Simulating physical systems through numerical solutions of partial differential equations (PDEs) is a fundamental pursuit in science and engineering, with diverse applications spanning solid mechanics (Jiang et al., 2016b; Li et al., 2020a), fluid dynamics (Bridson, 2015; Cao and Li, 2018), aerodynamics (Cao et al., 2022), and heat transfer (Cao et al., 2019). However, conventional numerical solvers often entail significant computational expense, particularly in time-sensitive scenarios like iterative design optimization, where rapid online inference is imperative.

In recent years, the machine learning community has exhibited keen interest in enhancing efficiency or substituting traditional solvers with learned models. These endeavors encompass holistic frameworks (Grzeszczuk et al., 1998; Obiols-Sales et al., 2020) as well as those incorporating physics-informed losses (Raissi et al., 2019; Karniadakis et al., 2021; Sun et al., 2020). Numerous existing initiatives leverage convolutional neural networks (CNNs) (Fukushima and Miyake, 1982) for physical systems situated on two- or three-dimensional structured grids (Guo et al., 2016; Tompson et al., 2017; Kim et al., 2019; Fotiadis et al., 2020). Nevertheless, their rigid dependence on regular domain shapes poses challenges for application on unstructured meshes. While it is feasible to deform uncomplicated irregular domains into rectangular shapes to accommodate CNNs (Gao



(a) Learnable Pooling (b) Pooling by Rasterization (c) Pooling by Spatial Proximity

Figure 1.1: Challenges of existing multi-level GNNs. (a) Learnable pooling (Gao and Ji, 2019) may result in a loss of connectivity even after the 2nd-order enhancement. (b) Pooling by rasterization (Lino et al., 2021, 2022a,b) and (c) by spatial proximity (Liu et al., 2021; Fortunato et al., 2022) can lead to incorrect connections across boundaries at the coarser level.

et al., 2021; Li et al., 2022c), the obstacle persists for domains with intricate topologies, commonly encountered in practical scenarios.

As a consequence, the utilization of graph neural networks (GNNs) in physics-based simulations on unstructured meshes has recently garnered substantial attention (Battaglia et al., 2018; Sanchez-Gonzalez et al., 2018; Belbute-Peres et al., 2020; Pfaff et al., 2020; Sanchez-Gonzalez et al., 2020; Harsch and Riedelbauch, 2021; Gao et al., 2022). The rudimentary GNN approach involves stacking multiple message-passing (MP) layers to model information propagation across space. However, as the graph size increases, this strategy confronts two primary challenges: (1) *Complexity*; with both the quantity of nodes for processing and MP iterations increasing linearly, a quadratic complexity becomes inevitable for both computation time and memory consumption of the computational graph (Fortunato et al., 2022). (2) *Oversmoothing*; graph convolution can be perceived as a low-pass filter that dampens higher-frequency signals (Chen et al., 2020; Li et al., 2020b). Stacked MP layers then iteratively project information onto the graph's Eigenspace, effectively smudging out all higher-frequency signals, thereby complicating training.

To overcome these limitations, researchers have commenced introducing multi-scale GNNs (MS-GNNs) for physics-based simulation (Li et al., 2020b; Liu et al., 2021; Lino et al., 2021; Fortunato et al., 2022; Lino et al., 2022b,a). The multi-scale approach mitigates oversmoothing by constructing sub-level graphs at coarser resolutions, promoting longer-range interactions and curtailing MP iterations. Existing methods for constructing the multi-scale structure encompass utilizing spatial proximity to generate sub-level graphs at coarser levels (Lino et al., 2021; Liu et al., 2021; Lino et al., 2022a), applying Guillard's coarsening algorithm (Guillard, 1993; Lino et al., 2022b), manually generating coarser meshes for the original geometry (Liu et al., 2021; Fortunato et al., 2022), or randomly pooling nodes and applying adjacency matrix factorization (Li et al., 2020b). However, these solutions suffer respective limitations. For instance, learnable or random pooling can introduce artificial partitions in the sub-level graphs (Figure 1.1(a)), even with adjacency enhancement, obstructing information exchange across partitions; spatial proximity can result in erroneous edges across coarser level boundaries (Figure 1.1(b) and (c)); Guillard's algorithm is restricted to 2D triangle meshes; and manually generating thousands of meshes is excessively labor-intensive.

1.2 Contributions

In Chapter 2, we introduce an innovative MPM framework that effectively addresses the challenges discussed in Section 1.1.1 with theoretical guarantees on the continuous kernel and gradient reconstructions for MPM on the general mesh. Our contributions are manifold (Cao et al., 2023b):

- We introduce a novel material point method based on an MLS reconstruction process that resolves cross-cell errors in general unstructured tessellations, representing a pioneering effort in this field.
- This achievement was realized by incorporating a diminishing function into the sample weights of the MLS kernel. We provide an analytical proof of our approach, establishing a theoretical bound for its effectiveness.

- Additionally, we present comprehensive numerical experiments that demonstrate the versatility and ease of integrating our method into existing MPM frameworks.

In Chapter 3, we introduce a novel method for simulating potentially destructive interactions between compressible flow (shock waves) and nonlinear elastoplastic solids. Our technical contributions (Cao et al., 2022) include:

- A hybrid Eulerian Finite Element and Lagrangian/Eulerian Material Point scheme for monolithic modeling of compressible flow and nonlinear structural interactions;
- A mixed treatment of reflective and passable interfaces inspired by porous media theory, enabling stable *sub-grid* resolution—a crucial element for modeling gas and fragment interactions accurately; and
- A new mixed-order finite element discretization utilizing B-spline kernels with *mismatching* interface pressure orders and (thus) *non-staggered* solid/fluid degrees of freedom, avoiding additional interpolation steps and resulting in a diagonally dominant SPD system for velocity-pressure unknowns.

In contrast to the method of Kwatra et al. (2010), which couples staggered Marker-and-Cell (MAC) grid finite difference-based fluids with purely Lagrangian solids, our approach is based on the non-staggered grid. This eliminates extrapolation during the advection step for compressible flow. Our framework relies on operator splitting and remains stable for large time step sizes determined by pressure gradients and maximum velocities.

In Chapter 4, we discern that all the limitations of multi-scale graph neural networks stem from immature operations, specifically pooling and establishing graph connections at coarser levels. We devise operations that 1) uphold accurate connections at coarser levels, 2) abstain from introducing edges that blur boundaries, 3) remain universally applicable to any mesh type, and 4) are automated. We tackle these challenges with two progressive contributions (Cao et al., 2023a):

- First, we introduce a novel yet straightforward pooling strategy, termed *bi-stride*. Bi-stride is inspired by bi-partition determination in a directed acyclic graph (DAG).

It pools all nodes on alternate breadth-first search (BFS) frontiers, such that a 2nd-powered adjacency enhancement ($\boldsymbol{A} \leftarrow \boldsymbol{A}^2$, where $\boldsymbol{A}$ is the adjacency matrix of the graph) conserves all correct connectivity. Bi-stride exclusively leverages the input mesh, obviating the need for spatial proximity, is universally applicable to any mesh type, and is fully automated.

- Second, bi-stride pooling preserves direct connections between pooled and unpooled nodes; leveraging this advantage, a single MP operation suffices for information exchange between pooled and unpooled nodes before transitioning to the adjacent level; we also devise a non-parameterized aggregation and propagation method, akin to interpolation in U-Net, to manage the transition between adjacent levels. These simplifications significantly curtail computational requisites compared to state-of-the-art approaches.

In unison, these two contributions give rise to our Bi-Stride Multi-Scale GNN (BSMS-GNN), an innovative framework representing a notable advance in the domain of learned mesh-based simulations that are especially suitable for deployment in genuine industrial applications where meshes frequently feature intricate geometry and considerable size.

CHAPTER 2

Unstructured Moving Least Squares Material Point Method

The Material Point Method (MPM) is a hybrid Eulerian-Lagrangian simulation technique for solid mechanics with significant deformation. Structured background grids are commonly employed in the standard MPM, but they may give rise to several accuracy problems in handling complex geometries. When using (2D) unstructured triangular or (3D) tetrahedral background elements, however, significant challenges arise (*e.g.*, cell-crossing error). Substantial numerical errors develop due to the inherent $\mathcal{C}^0$ continuity property of the interpolation function, which causes discontinuous gradients across element boundaries. Prior efforts in constructing $\mathcal{C}^1$ continuous interpolation functions have either not been adapted for unstructured grids or have only been applied to 2D triangular meshes. In this study, an Unstructured Moving Least Squares MPM (UMLS-MPM) is introduced to accommodate 2D and 3D simplex tessellation. The central idea is to incorporate a diminishing function into the sample weights of the MLS kernel, ensuring an analytically continuous velocity gradient estimation. Numerical analyses confirm the method's capability in mitigating cell crossing inaccuracies and realizing expected convergence.

2.1 Methodology

2.1.1 Governing Equations

Following standard continuum mechanics (Bonet and Wood, 2008), consider the mapping $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{X}, t)$, which maps points from the (reference) material configuration, represented by $\boldsymbol{X}$, to their corresponding locations in the (current) spatial configuration, represented by $\boldsymbol{x}$. In this framework, velocity is defined in two different but equivalent manners. On the one hand, $\boldsymbol{V}(\boldsymbol{X}, t) = \frac{\partial \boldsymbol{x}}{\partial t}(\boldsymbol{X}, t)$ defines the Lagrangian velocity in the material configuration. On the other hand, the Eulerian velocity in the spatial configuration, is denoted by $\boldsymbol{v}(\boldsymbol{x}, t) = \boldsymbol{V}(\boldsymbol{\phi}^{-1}(\boldsymbol{x}, t), t)$. Furthermore, the deformation experienced by the material points is quantified using the deformation gradient, given by $\boldsymbol{F}(\boldsymbol{X}, t) = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}}(\boldsymbol{X}, t)$. The determinant of this gradient, represented by J, is also crucial as it provides insights into volumetric changes associated with the deformation process.

Given these definitions, the conservation equations for mass and momentum (neglecting external forces) are (Zhang et al., 2016; Bonet and Wood, 2008)

$$\begin{aligned} \rho J &= \rho_0, \\ \rho \frac{D\boldsymbol{v}}{Dt} &= \nabla \cdot \boldsymbol{\sigma}, \end{aligned} \tag{2.1}$$

where ρ represents the density, D/Dt is the material derivative, and

$$\boldsymbol{\sigma} = \frac{1}{J} \boldsymbol{P} \boldsymbol{F}^T. \tag{2.2}$$

is the Cauchy stress tensor, which is related to the first Piola-Kirchhoff stress $\boldsymbol{P} = \frac{\partial \Psi}{\partial \boldsymbol{F}}$, where Ψ denotes the strain energy density. The evolution of the deformation gradient is given by

$$\dot{\boldsymbol{F}} = (\nabla \boldsymbol{v}) \boldsymbol{F}. \tag{2.3}$$

Consider a domain represented by Ω. Boundaries on which the displacement is known, represented as $\partial\Omega_u$, are governed by the Dirichlet boundary condition

$$x_k(\boldsymbol{x},t) = \bar{x}_k(\boldsymbol{x},t), \quad \forall \boldsymbol{x} \in \partial\Omega_u, \tag{2.4}$$

where $\bar{x}_k$ denotes the predetermined displacement for component k. Boundaries on which the tractions (forces per unit area) are predefined, represented as $\partial\Omega_\tau$, adhere to the Neumann boundary condition

$$\sigma_{kl}(\boldsymbol{x},t)n_l = \bar{\tau}_k(\boldsymbol{x},t), \quad \forall \boldsymbol{x} \in \partial\Omega_\tau, \tag{2.5}$$

where $\bar{\tau}_k$ is the prescribed traction for component k, and $\sigma_{kl}(\boldsymbol{x},t)n_l$ represents the traction inferred from the stress tensor σ_{kl} acting in the direction of the outward unit normal vector n_l. For ease of reference and notational clarity in our framework, the subscripts k and l refer to components k and l of any given vector or tensor.

To solve the conservation equations for mass and momentum within the MPM framework, one often turns to the weak form. Specifically, a continuous test function ϕ, which vanishes on $\partial\Omega_u$, is employed. Then, both sides of the equation are multiplied by ϕ and integrated over the domain Ω:

$$\int_\Omega \phi\rho\ddot{x}_k d\Omega = \int_{\partial\Omega_\tau} \phi\tau_k dA - \int_\Omega \frac{\partial\phi}{\partial\boldsymbol{x}_l}\sigma_{kl} d\Omega. \tag{2.6}$$

At this juncture, integration by parts and the Gauss integration theorem are utilized, nullifying the contributions on $\partial\Omega_u$ due to the vanishing of the test function on this boundary subset.

In the standard implementation of the MPM, physical quantities are retained at material points and then projected onto background grids for further computation. Equation (2.6) is discretized within the grid space by leveraging the Finite Element Method (FEM) and then solved using either implicit or explicit time integration schemes. This article focuses on the explicit symplectic Euler time integration method. While the extension to implicit methods is possible and indeed straightforward, that would be

Algorithm 1 Explicit MPM.

```
 1: Determine material point-grid connectivity, calculate kernel functions $w_{ip}$
 2: **P2G:**
        Nodal mass: $m_i = \sum_p \rho_p V_p w_{ip}$
        Nodal momentum: $\boldsymbol{p}_i = \sum_p \boldsymbol{v}_p \rho_p V_p w_{ip}$
        Nodal velocity: $\boldsymbol{v}_i = \boldsymbol{p}_i / m_i$
 3:     Internal force: $\boldsymbol{f}_i^{\text{int}} = -\sum_p V_p \boldsymbol{\sigma}_p \nabla w_{ip}$
 4:     Gravity: $\boldsymbol{f}_i^{\text{ext}} = \sum_p w_{ip} m_p \mathbf{g}_p$
 5:     Nodal force: $\boldsymbol{f}_i = \boldsymbol{f}_i^{\text{ext}} + \boldsymbol{f}_i^{\text{int}}$
 6: **Deformation of grid:**
        Updated nodal accelerations: $\ddot{\boldsymbol{x}}_i = \boldsymbol{f}_i / m_i$
        Update nodal velocities: $\tilde{\boldsymbol{v}}_i = \boldsymbol{v}_i + \Delta t \ddot{\boldsymbol{x}}_i$
        Enforce Dirichlet conditions: $\ddot{\boldsymbol{x}}_i = 0$ and $\boldsymbol{f}_i = 0$
 7: **G2P:**
        Update point velocities: $\boldsymbol{v}_p^{\Delta t} = \boldsymbol{v}_p + \Delta t \sum_i w_{ip} \ddot{\boldsymbol{x}}_i$
        Update point positions: $\boldsymbol{x}_p^{\Delta t} = \sum_i w_{ip} \tilde{\boldsymbol{x}}_i$
 8: Update deformation gradient: $\boldsymbol{F}_p^{\Delta t} = \left(\boldsymbol{I} + \sum_i (\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i)(\nabla w_{ip})^T\right) \boldsymbol{F}_p$
 9: Update point volume: $V_p^{\Delta t} = \det(\boldsymbol{F}_p^{\Delta t}) V_p^0$
10: Update point stresses: $\boldsymbol{\sigma}_p = C(\boldsymbol{F}_p)$
11: **Enforce plasticity, reset grid deformation, advance to next timestep**
```

orthogonal to the contribution of the article.

2.1.2 Explicit MPM Pipeline

The explicit MPM pipeline in each time step is delineated into four main stages: (1) the transfer of material point quantities to the background grid, commonly known as Particle-To-Grid (P2G), (2) the computation of the system's evolution on this background grid, (3) the back-transfer of the evolved grid quantities to the material points, designated as Grid-To-Particle (G2P), and (4) the execution of supplementary processing tasks, specifically on strain and/or stress to incorporate effects such as elastoplasticity return mapping and material hardening. Finally, the hypothetical deformation incurred on the background grid is reset for the next computational cycle. Algorithm 1 presents an overview of the MPM pipeline, and the main stages are elaborated below.

Stage 1: P2G In the MPM, discrete material points represent physical attributes such as mass, position, and velocity. For a given particle, labeled p, and a grid node, labeled i, the interpolation function's value associated with node i, evaluated at the spatial position $\boldsymbol{x}_p$ of particle p, is represented as w_{ip}. Similarly, the gradient of this interpolation function, evaluated at the same location, is denoted as ∇w_{ip}. In the explicit MPM framework, the lumped grid mass is defined as $m_i = \sum_p \rho_p V_p w_{ip}$, where ρ_p represents the density and V_p the volume of each material point. This definition facilitates the momentum calculation on the grid, expressed as

$$m_i \ddot{\boldsymbol{x}}_i = \boldsymbol{f}_i^{\text{int}} + \boldsymbol{f}_i^{\text{ext}}, \tag{2.7}$$

where $\ddot{\boldsymbol{x}}_i$ is the acceleration of grid node i. The internal forces $\boldsymbol{f}_i^{\text{int}}$ and external forces $\boldsymbol{f}_i^{\text{ext}}$ acting on the grid nodes are as follows:

$$\boldsymbol{f}_i^{\text{int}} = -\sum_p V_p \boldsymbol{\sigma}_p \nabla w_{ip}, \tag{2.8}$$

$$\boldsymbol{f}_i^{\text{ext}} = \sum_p m_p w_{ip} \boldsymbol{b}_p + \sum_p m_p w_{ip} \boldsymbol{g}_p. \tag{2.9}$$

The stress tensor $\boldsymbol{\sigma}$ is linked to the deformation gradient $\boldsymbol{F}$ through the constitutive relation

$$\boldsymbol{\sigma} = C(\boldsymbol{F}), \tag{2.10}$$

which defines how material deformation influences internal force.

Stage 2: Evolution on the Background Grids Nodal accelerations are computed using (2.7). To update the velocities and positions of the grid nodes, a symplectic Euler time integrator is employed:

$$\tilde{\boldsymbol{v}}_i = \boldsymbol{v}_i + \Delta t \ddot{\boldsymbol{x}}_i \quad \text{(Velocity Update)}, \tag{2.11}$$

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \Delta t \tilde{\boldsymbol{v}}_i \quad \text{(Position Update)}. \tag{2.12}$$

These equations ensure a time-stepped progression of the grid nodes where the time step size Δt is chosen based on the CFL condition.

Stage 3: G2P The FLIP scheme (Brackbill et al., 1988) is utilized for all the experiments discussed in Section 2.3. In FLIP, the material point positions and velocities are updated as

$$x_p^{\Delta t} = \sum_i w_{ip} \tilde{\boldsymbol{x}}_i, \tag{2.13}$$

$$v_p^{\Delta t} = v_p + \Delta t \sum_i w_{ip} \ddot{\boldsymbol{x}}_i. \tag{2.14}$$

Subsequently, the evolution of the deformation gradient $\boldsymbol{F}$ in (2.3) is discretized as

$$\boldsymbol{F}_p^{\Delta t} = \left(\boldsymbol{I} + \sum_i (\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i)(\nabla w_{ip})^T \right) \boldsymbol{F}_p. \tag{2.15}$$

With an initial $\boldsymbol{F}^0 = \boldsymbol{I}$, material point volumes are updated as

$$V_p^{\Delta t} = \det(\boldsymbol{F}_p^{\Delta t}) V_p^0. \tag{2.16}$$

Stage 4: Post-Processing and Resetting the Background Grid This stage encompasses all post-processing tasks such as plasticity return mapping and hardening (Simo and Hughes, 2006). In the updated Lagrangian MPM, the grid is reset to a non-deformed state. This can be done by not updating grid positions while discarding other grid information such as velocity and acceleration.

2.1.3 Transfer Kernel

In the MPM, the transfer kernel is vital for relaying particle information to adjacent grid nodes. Techniques such as the B-spline MPM (Steffen et al., 2008) and GIMP (Bardenhagen and Kober, 2004) use a specific compact support function to smoothly influence nearby grid nodes, whereas methods like Moving Least Squares MPM (MLS-

MPM) (Hu et al., 2018) determine the kernel implicitly, based on the proximity of nodes. However, both strategies follow a similar workflow: (1) identifying the set of nearby nodes, and (2) calculating the weights (transfer kernel) for each node in relation to a particle in its influence zone.

This section first introduces the general MLS reconstruction process and the application of MLS-MPM with a comprehensive linear polynomial basis. This is followed by a discussion on a superficially straightforward extension of MLS-MPM to unstructured meshes, highlighting the aforementioned steps of identifying nearby nodes and computing the transfer weights. We then delve into the desirable properties of the kernel, emphasizing why the naive extension fails to yield continuous gradient reconstructions when particles cross cell boundaries. Finally, we offer a solution addressing the issue of discontinuous gradient reconstructions and propose UMLS-MPM.

2.1.3.1 Introduction to General MLS and MLS-MPM

The essential concept of Moving Least Squares (MLS) is to approximate a function u at a point $\boldsymbol{z}$ within the continuous domain surrounding $\boldsymbol{x}$. This approximation is achieved by employing a polynomial least-squares fit of u based on its sampled values at specific points $\boldsymbol{x}_i$, where each u_i denotes the value of u at $\boldsymbol{x}_i$. The functional reconstruction is given by

$$u(\boldsymbol{z}) = \boldsymbol{P}^T(\boldsymbol{z}-\boldsymbol{x})\boldsymbol{w}(\boldsymbol{x}), \tag{2.17}$$

where $\boldsymbol{P}(\boldsymbol{z}-\boldsymbol{x}) = [p_0(\boldsymbol{z}-\boldsymbol{x}), p_1(\boldsymbol{z}-\boldsymbol{x}), \dots, p_l(\boldsymbol{z}-\boldsymbol{x})]^T$ represents the polynomial basis, while $\boldsymbol{w}(\boldsymbol{x}) = [w_0(\boldsymbol{x}), w_1(\boldsymbol{x}), \dots, w_l(\boldsymbol{x})]$ comprises the corresponding coefficients, and l indicates the number of polynomial basis functions. The coefficients $\boldsymbol{w}(\boldsymbol{x})$ are determined by minimizing the cumulative weighted reconstruction errors at the sampled points. This is done by substituting $\boldsymbol{z} \leftarrow \boldsymbol{x}_i$ into (2.17) for each sample:

$$\boldsymbol{w}(\boldsymbol{x}) = \operatorname{argmin} \sum_{i \in \mathcal{B}_{\boldsymbol{x}}} d(\boldsymbol{x}_i - \boldsymbol{x})\, ||u_i - \boldsymbol{P}^T(\boldsymbol{x}_i - \boldsymbol{x})\boldsymbol{w}(\boldsymbol{x})||^2. \tag{2.18}$$

Here d represents the inverse of a separation function, which is typically a positive value that decreases with increasing separation distance. It acts as the weight for the reconstruction error at each sample. The set $\mathcal{B}_{\boldsymbol{x}}$ encompasses the local region around $\boldsymbol{x}$ where this weighting function is non-zero; i.e., $d(\boldsymbol{x}_i - \boldsymbol{x}) > 0$.

This minimization leads to the following solution for $\boldsymbol{w}$:

$$\boldsymbol{w}(\boldsymbol{x}) = \boldsymbol{M}^{-1}(\boldsymbol{x})\boldsymbol{b}(\boldsymbol{x}), \tag{2.19}$$

where $\boldsymbol{M}(\boldsymbol{x}) = \sum_{i\in\mathcal{B}_{\boldsymbol{x}}} d(\boldsymbol{x}_i-\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x}_i-\boldsymbol{x})\boldsymbol{P}^T(\boldsymbol{x}_i-\boldsymbol{x})$ and $\boldsymbol{b} = \sum_{i\in\mathcal{B}_{\boldsymbol{x}}} d(\boldsymbol{x}_i-\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x}_i-\boldsymbol{x})u_i$. Substituting (2.19) into (2.17), we obtain the reconstruction

$$u(\boldsymbol{x}) = \sum_{i\in\mathcal{B}_{\boldsymbol{x}}} d(\boldsymbol{x}_i - \boldsymbol{x})\boldsymbol{P}^T(\boldsymbol{z} - \boldsymbol{x})\boldsymbol{M}^{-1}(\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x}_i - \boldsymbol{x})u_i. \tag{2.20}$$

The Linear Polynomial Basis Case A special case involves using a complete linear polynomial basis, as is done in MLS-MPM (Hu et al., 2018), where $\boldsymbol{P}(\boldsymbol{x}_i - \boldsymbol{x}) = [1, (\boldsymbol{x}_i-\boldsymbol{x})^T]^T$. Then (2.20) can reconstruct the function value $\hat{u}$ and provide an estimation of the gradient $\nabla\hat{u}$ at $\boldsymbol{x}$ as follows:

$$\begin{bmatrix} \hat{u} \\ \nabla\hat{u} \end{bmatrix} = \boldsymbol{M}^{-1}(\boldsymbol{x})\boldsymbol{Q}^T(\boldsymbol{x})\boldsymbol{D}(\boldsymbol{x})\boldsymbol{u}, \tag{2.21}$$

where $\boldsymbol{u} = [u_1, \dots, u_N]^T$ is the stacked sampled values, $\boldsymbol{Q}(\boldsymbol{x}) = [\boldsymbol{P}(\boldsymbol{x}_1 - \boldsymbol{x}), \dots, \boldsymbol{P}(\boldsymbol{x}_N - \boldsymbol{x})]^T$ is the stacked basis for every sample, $\boldsymbol{D}(\boldsymbol{x})$ is the diagonal sample weighting matrix with $\boldsymbol{D}_{i,i}(\boldsymbol{x}) = d(\boldsymbol{x}_i - \boldsymbol{x})$, and $\boldsymbol{M}(\boldsymbol{x}) = \boldsymbol{Q}(\boldsymbol{x})^T\boldsymbol{D}(\boldsymbol{x})\boldsymbol{Q}(\boldsymbol{x})$. We adopt the linear basis.

2.1.3.2 Extending MLS-MPM Onto Unstructured Meshes

We select MLS-MPM as the foundational approach due to the inherent versatility of MLS, which enables application to adjacent nodes without reliance on specific topological or positional constraints. Our implementation and experimental work have focused on triangular and tetrahedral meshes. Nonetheless, it is worth noting that our method can

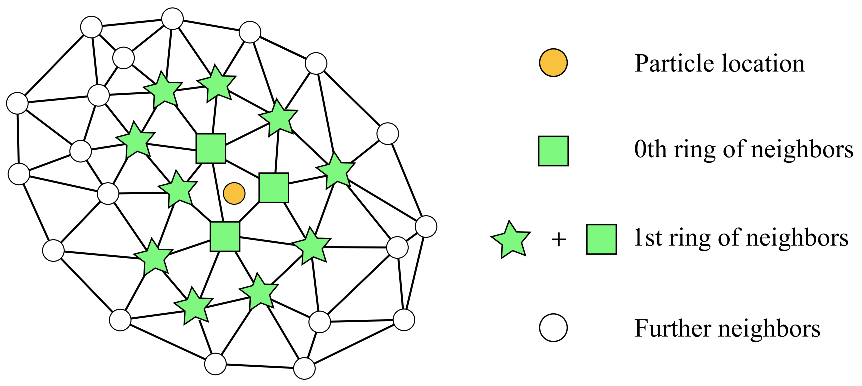
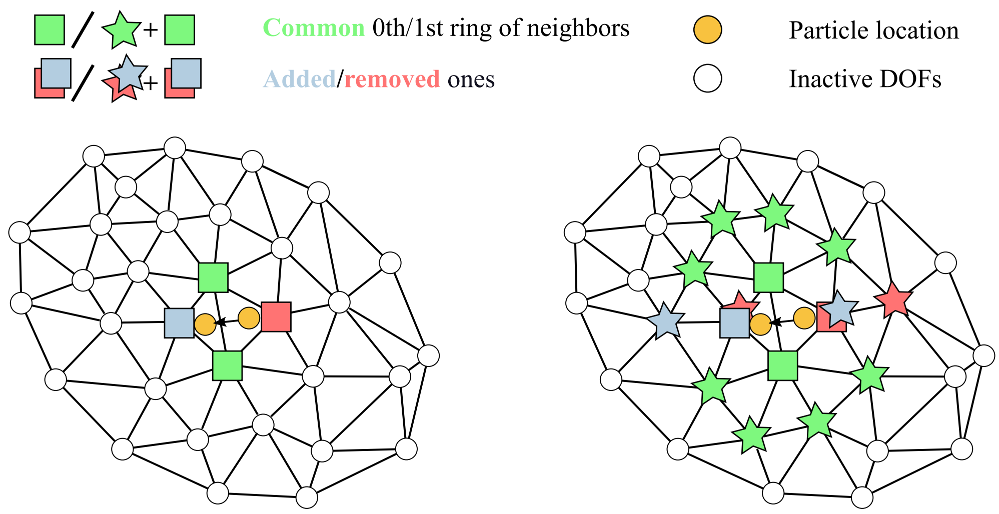


Figure 2.1: Schematic plot of the zeroth and first ring of neighbors.

easily be extended to other kinds of unstructured grid tessellations by designing a smooth and locally diminishing function η_v compatible with the tessellation, such as the one we design below in (2.29) for simplex cells.

Identifying Nearby Nodes Around a Particle To determine the nearby vertices for a given particle p, we first locate the cell that encompasses p and refer to its vertices as $\mathcal{N}_p^0$, representing the 0-ring neighbors of p. Then, we define $\mathcal{N}_p^1$ as the 1-ring neighbors, which comprise all nodes connected to $\mathcal{N}_p^0$. Note that $\mathcal{N}_p^0$ is a subset of $\mathcal{N}_p^1$. Similarly, we can define $\mathcal{N}_p^2, \ldots$ in an analogous manner, as illustrated in Figure 2.1.

Ring Level Selection for Nearby Nodes When a specific level of ring neighbors is chosen as the nearby nodal degrees of freedom, a natural question arises: What is the minimum number of rings required to satisfy the desired properties of the MPM kernel? Assume $\mathcal{N}_p^0$ is selected, meaning the particle only affects the vertices in the cell where it currently resides, and at least $\mathcal{C}^0$ continuity is required for the kernel. This scenario leads to the kernel degenerating, which is characterized by the kernel affecting merely the shared edge in 2D or the shared face in 3D when the particle transitions across cell boundaries, as depicted in Figure 2.2a. Conversely, opting for 1-ring neighbors, $\mathcal{N}_p^1$, effectively circumvents this issue, ensuring a non-degenerate kernel interaction as illustrated in Figure 2.2b.



(a) If 0th ring of neighbors are DOFs

(b) Expand to use 1th ring of neighbors

Figure 2.2: The selection of different ring of neighbors. (a) When $\mathcal{N}_p^0$ alone is selected as the active Degrees of Freedom (DOFs), as a particle crosses the cell edge, the DOFs indicated by the blue and red boxes are added or removed, respectively. Consequently, the weights on these DOFs must approach zero to ensure $\mathcal{C}^0$ continuity, resulting in kernel degeneration along the edge. (b) Advancing to $\mathcal{N}_p^1$ addresses this issue by incorporating a sufficient number of DOFs to fully encompass the particles.

Computing the Weights For conciseness and consistency, we will omit the function arguments and subscripts, such as $\boldsymbol{x}$ and i in (2.21). Instead, since we project onto the mesh vertex v, we will hereafter use the subscript v and obtain the following rewritten form of (2.21):

$$\begin{bmatrix} \hat{u}_p \\ \nabla \hat{u}_p \end{bmatrix} = \boldsymbol{M}_{vp}^{-1} \boldsymbol{Q}_{vp}^T \boldsymbol{D}_{vp} \boldsymbol{u}_{vp}, \tag{2.22}$$

where $\boldsymbol{u}_{vp} = [u_{v_1}, \ldots, u_{v_n}]^T$ is the stacked vertex values with $v_i \in \mathcal{N}_p^1$, $i \in [1,\ n]$ and the matrix $\boldsymbol{M}_{vp} = \boldsymbol{Q}_{vp}^T \boldsymbol{D}_{vp} \boldsymbol{Q}_{vp}$ with stacked vertex basis $\boldsymbol{Q}_{vp} = [\boldsymbol{P}(\boldsymbol{x}_{v_1} - \boldsymbol{x}_p), \ldots, \boldsymbol{P}(\boldsymbol{x}_{v_n} - \boldsymbol{x}_p)]^T$ and diagonal sample weighting matrix $\boldsymbol{D}_{vp}$ such that $\boldsymbol{D}_{pv,i,i} = d_{vp} = d(\boldsymbol{x}_{v_1} - \boldsymbol{x}_p)$, where d is chosen to be a B-spline function.

2.1.3.3 Required Properties for the Transfer Kernel

Despite implementing a broader interpolation kernel, the reconstruction method described in Section 2.1.3.2 is not directly applicable as it still suffers from cell crossing errors. To grasp this problem, consider the essential desirable properties for an MPM kernel:

1. The kernel must be a non-negative partition of unity. This means that the sum of the kernel weights for all nearby vertices to a particle should equal 1; *i.e.*, $\sum_{v\in\mathcal{N}_p^1} w_v = 1$, with each individual weight $w_v \geq 0, \forall v \in \mathcal{N}_p^1$.
2. There should be a continuous reconstruction of both the function value and gradient as the particle transitions across the cell boundary.

Methods such as B-spline MPM and GIMP are specifically designed to fulfill these requirements. With MLS-MPM, the partition of unity is inherently assured by the characteristics of MLS (Levin, 1998). The non-negativity of this partition additionally depends on preventing sample degeneration, a requirement met in MLS-MPM due to its use of uniform background nodes. Furthermore, MLS-MPM ensures continuous reconstruction by utilizing a B-spline for sample weighting, which provides $\mathcal{C}^1$ continuity.

A key insight from MLS-MPM is that whenever a grid node is added or removed from the nearby node set, its B-spline weighting function also smoothly approaches zero, ensuring its influence on the assembly of $\boldsymbol{M}$ and $\boldsymbol{Q}$ in (2.21) is infinitesimal and, hence, the continuity of the reconstruction. However, since our method determines nearby vertices based on the ring of neighbors rather than proximity, a uniform sample weighting function cannot guarantee diminishing influence for the added or removed vertices. Consequently, the abrupt changes in influence of these vertices during the MLS assembly yield discontinuous reconstruction. This issue will be addressed in the next section.

2.1.3.4 Remedying Discontinuous Reconstruction Across the Cell Boundary

To mitigate the abrupt influence changes from vertices being added or removed during cell crossings, an intuitive solution is to diminish their impact on the MLS assembly.

This can be accomplished by adjusting the kernel to predominantly rely on vertices that remain common before and after crossing the cell boundary. Intuitivly, we can multiply any initial sample weighting function d_{vp} that is $\mathcal{C}^1$ by a smooth diminishing function η_{vp}; *i.e.*, $d_{vp} \leftarrow \eta_{vp} d_{vp}$. Here, η_{vp} approaches zero for vertices that are added or removed from $\mathcal{N}_p^1$ during the cell crossing. The formal proof is provided below.

For conciseness, we drop the subscripts $_{vp}$ in the following proofs. Assume there exists a smooth, diminishing function η for the nodes added or removed from the set of nearby nodes $\mathcal{N}^1$ when a particle crosses the boundary of a cell. As such, we need to prove that our kernel value and gradient estimation is continuous across the boundary.

Proposition 1. *The kernel value and gradient estimation generated by the diminished sample weighted MLS is continuous across cell boundaries.*

Proof. Let $\mathcal{N}_{o,n}^1$ be the sets of nearby nodes before/after the particle p crosses the common edge between the old/new cells $\mathcal{N}_{o,n}^0$. Here, the subscripts o, n denote the old/new cell, respectively, and the superscripts $0, 1$ indicate the ring-0/1 neighbors of the cell, respectively. Let $\boldsymbol{x}_p^{o,n}$ be the position of particle p before/after the crossing and $||\boldsymbol{x}_p^n - \boldsymbol{x}_p^o|| = \mathcal{O}(\epsilon)$. Define the common node set $\mathcal{N}_c^1 = \mathcal{N}_o^1 \cap \mathcal{N}_n^1$, the added node set $\mathcal{N}_a^1 = \mathcal{N}_n^1 \setminus \mathcal{N}_c^1$, and the removed node set $\mathcal{N}_r^1 = \mathcal{N}_o^1 \setminus \mathcal{N}_c^1$. Since η is locally diminishing for $v \in \mathcal{N}_{a,r}^1$, we have a positive value K_1 such that $\eta = \mathcal{O}(K_1 \epsilon) = \mathcal{O}(\epsilon)$. The pertubation for the assembled matrix $\boldsymbol{M}$ before/after the particle p crosses an edge is

$$\delta \boldsymbol{M} = \sum_{v \in \mathcal{N}_c^1} \delta(\eta d \boldsymbol{P}\boldsymbol{P}^T) + \sum_{v \in \mathcal{N}_a^1} \eta d \boldsymbol{P}\boldsymbol{P}^T - \sum_{v \in \mathcal{N}_r^1} \eta d \boldsymbol{P}\boldsymbol{P}^T, \tag{2.23}$$

where the first term is continuous by construction since every factor is smooth; *i.e.*,

$||\delta(\eta d\boldsymbol{P}\boldsymbol{P}^T)|| = \mathcal{O}(\epsilon)$. For the second and third terms, since $\eta = \mathcal{O}(\epsilon)$, we have

$$\begin{aligned} ||\delta\boldsymbol{M}|| &\leq \sum_{v\in\mathcal{N}_c^1} ||\delta(\eta d\boldsymbol{P}\boldsymbol{P}^T)|| + \sum_{v\in\mathcal{N}_a^1} ||\eta d\boldsymbol{P}\boldsymbol{P}^T|| + \sum_{v\in\mathcal{N}_r^1} ||\eta d\boldsymbol{P}\boldsymbol{P}^T|| \\ &\leq \left[|\mathcal{N}_c^1| + \left(|\mathcal{N}_a^1| + |\mathcal{N}_r^1|\right) \max_{v\in\mathcal{N}_{a,r}^1} ||d\boldsymbol{P}\boldsymbol{P}^T||\right]\mathcal{O}(\epsilon) \\ &= \mathcal{O}(|\mathcal{N}^1|h^2\epsilon) \\ &= \mathcal{O}(\epsilon). \end{aligned} \tag{2.24}$$

Here, as long as the mesh has a reasonably good quality, $|\mathcal{N}^1|$ is finite and small; *i.e.*, there is a finite and small amount of ring-1 neighbors. Also, h, a constant, is the support radius of the kernel, outside of which the weight is zero. In all, both $|\mathcal{N}^1|$ and h can be omitted in the analysis.

The perturbation of the inverse matrix is given by

$$\begin{aligned} ||\delta\boldsymbol{M}^{-1}|| &= ||(\boldsymbol{M} + \delta\boldsymbol{M})^{-1} - \boldsymbol{M}^{-1}|| \\ &= ||\boldsymbol{M}^{-1} - \boldsymbol{M}^{-1}\delta\boldsymbol{M}\boldsymbol{M}^{-1} + \mathcal{O}(||\delta\boldsymbol{M}||^2) - \boldsymbol{M}^{-1}|| \\ &= ||\boldsymbol{M}^{-1}\delta\boldsymbol{M}\boldsymbol{M}^{-1} + \mathcal{O}(\epsilon^2)|| \\ &\leq ||\boldsymbol{M}^{-1}\delta\boldsymbol{M}\boldsymbol{M}^{-1}|| + \mathcal{O}(\epsilon^2) \\ &\leq ||\boldsymbol{M}^{-1}||^2 \cdot ||\delta\boldsymbol{M}|| + \mathcal{O}(\epsilon^2) \\ &= \frac{||\delta\boldsymbol{M}||}{\sigma(\boldsymbol{M})^2_{\min}} + \mathcal{O}(\epsilon^2) \\ &= \mathcal{O}\left(\frac{\epsilon}{\sigma(\boldsymbol{M})^2_{\min}}\right) + \mathcal{O}(\epsilon^2) \\ &= \mathcal{O}\left(\frac{\epsilon}{\sigma(\boldsymbol{M})^2_{\min}}\right), \end{aligned} \tag{2.25}$$

where $\sigma(\boldsymbol{M})_{\min}$ is the minimum singular value of $\boldsymbol{M}$.

Similarly, for the perturbation in the assembled vector $\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}$ before/after the particle crossing is

$$\begin{aligned}
||\delta\left(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}\right)|| &= ||\sum_{v\in\mathcal{N}_c^1}\delta(\eta du\boldsymbol{P}) + \sum_{v\in\mathcal{N}_a^1}\eta du\boldsymbol{P} - \sum_{v\in\mathcal{N}_r^1}\eta du\boldsymbol{P}|| \\
&\leq \sum_{v\in\mathcal{N}_c^1}||\delta(\eta du\boldsymbol{P})|| + \sum_{v\in\mathcal{N}_a^1}||\eta du\boldsymbol{P}|| + \sum_{v\in\mathcal{N}_r^1}||\eta du\boldsymbol{P}|| \\
&\leq \left[|\mathcal{N}_c^1| + \left(|\mathcal{N}_a^1| + |\mathcal{N}_r^1|\right)\max_{v\in\mathcal{N}_{a,r}^1}||du\boldsymbol{P}||\right]\mathcal{O}(\epsilon) \\
&= \mathcal{O}(|\mathcal{N}^1|h\epsilon) \\
&= \mathcal{O}(\epsilon).
\end{aligned} \tag{2.26}$$

Furthermore, we can establish the following bound for the assembled vector $\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}$:

$$\begin{aligned}
||\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}|| &= ||\sum_{v\in\mathcal{N}^1}\eta du\boldsymbol{P}|| \\
&\leq |\mathcal{N}^1|\cdot\max_{v\in\mathcal{N}^1}||\eta du\boldsymbol{P}|| \\
&= \mathcal{O}(|\mathcal{N}^1|h) \\
&= \mathcal{O}(1).
\end{aligned} \tag{2.27}$$

Finally, the perturbation for $\left[\hat{u}_p, \nabla\hat{u}_p^T\right]^T$ from (2.22) is

$$\begin{aligned}
\left[\hat{u}_p, \nabla\hat{u}_p^T\right]^T &= ||\delta(\boldsymbol{M}^{-1}\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u})|| \\
&= ||\delta\boldsymbol{M}^{-1}\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u} + \boldsymbol{M}^{-1}\delta\left(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}\right)|| \\
&\leq ||\delta\boldsymbol{M}^{-1}\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}|| + ||\boldsymbol{M}^{-1}\delta\left(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}\right)|| \\
&\leq ||\delta\boldsymbol{M}^{-1}||\cdot||\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}|| + ||\boldsymbol{M}^{-1}||\cdot||\delta\left(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{u}\right)|| \\
&= \mathcal{O}\left(\left(\frac{1}{\sigma(\boldsymbol{M})_{\min}^2} + \frac{1}{\sigma(\boldsymbol{M})_{\min}}\right)\epsilon\right).
\end{aligned} \tag{2.28}$$

In the incomplete singular value decomposition of $\boldsymbol{M}$, the singular values will always be non-negative. And if the surrounding nodes are not degenerate, the minimum singular value $\sigma(\boldsymbol{M})_{\min}$ will always be positive and the condition number of $\boldsymbol{M}$ is bounded. Therefore, as long as the mesh is of reasonably good quality, both the function value and gradient estimation is $\mathcal{C}^0$ across the boundary. $\square$

Note the above conclusion holds for any general mesh tessellations. For simplex elements, we design the following η_{vp}:

$$\eta_{vp} = \sum_{n \in \mathcal{N}_p^0} B_{np} \boldsymbol{A}_{v,n}, \tag{2.29}$$

where, for particle p, B_{np} represents the barycentric weight for $n \in \mathcal{N}_p^0$, and $\boldsymbol{A}$ is the mesh's adjacency matrix. Geometrically, for a nearby vertex $v \in \mathcal{N}_p^1$, η_{vp} denotes the sum of the barycentric weights for all vertices $n \in \mathcal{N}_p^0$ that are directly connected to v. Note that $\eta_{vp} = 1$, $\forall v \in \mathcal{N}_p^0$. Next, we prove that (2.29) is locally diminishing for $\mathcal{N}_{a,r}^1$ when the particle crosses the edge. Note that the proof is presented in 2D when a particle crosses an edge; the extension to 3D and other crossing cases is straightforward.

Proposition 2. *The function η in (2.29) is locally diminishing for $\forall v \in \mathcal{N}_{a,r}^1$.*

Proof. Formally, we must prove that for any $v \in \mathcal{N}_{a,r}^1$, when $\boldsymbol{x}_p$ crosses the edge of a triangle and $||\boldsymbol{x}_p^n - \boldsymbol{x}_p^o|| = \mathcal{O}(\epsilon)$, the smoothing function $\eta = \mathcal{O}(\epsilon)$.

Denote the edge that the particle is crossing as e and the portion of $||\boldsymbol{x}_p^n - \boldsymbol{x}_p^o||$ in the new/old cell as $L^{n,o}$. Trivially,

$$\begin{aligned} L^{n,o} &\leq L^n + L^o \\ &= ||\boldsymbol{x}_p^n - \boldsymbol{x}_p^o|| \\ &= \mathcal{O}(\epsilon). \end{aligned} \tag{2.30}$$

Then, referring to Figure 2.3, let the far-away node not on the edge but in the new/old cell be v_{far} (*i.e.*, $v_{far}^{n,o} \notin e \wedge v_{far}^{n,o} \in \mathcal{N}_{o,n}^0$) and the height from a node v to an edge e be $H(v, e)$. Since the height is orthogonal to the edge, we have $H(\boldsymbol{x}_p^{n,o}, e) \leq L^{n,o} = \mathcal{O}(\epsilon)$. Consider the barycentric coordinate contributed by the far-away node, in the new/old

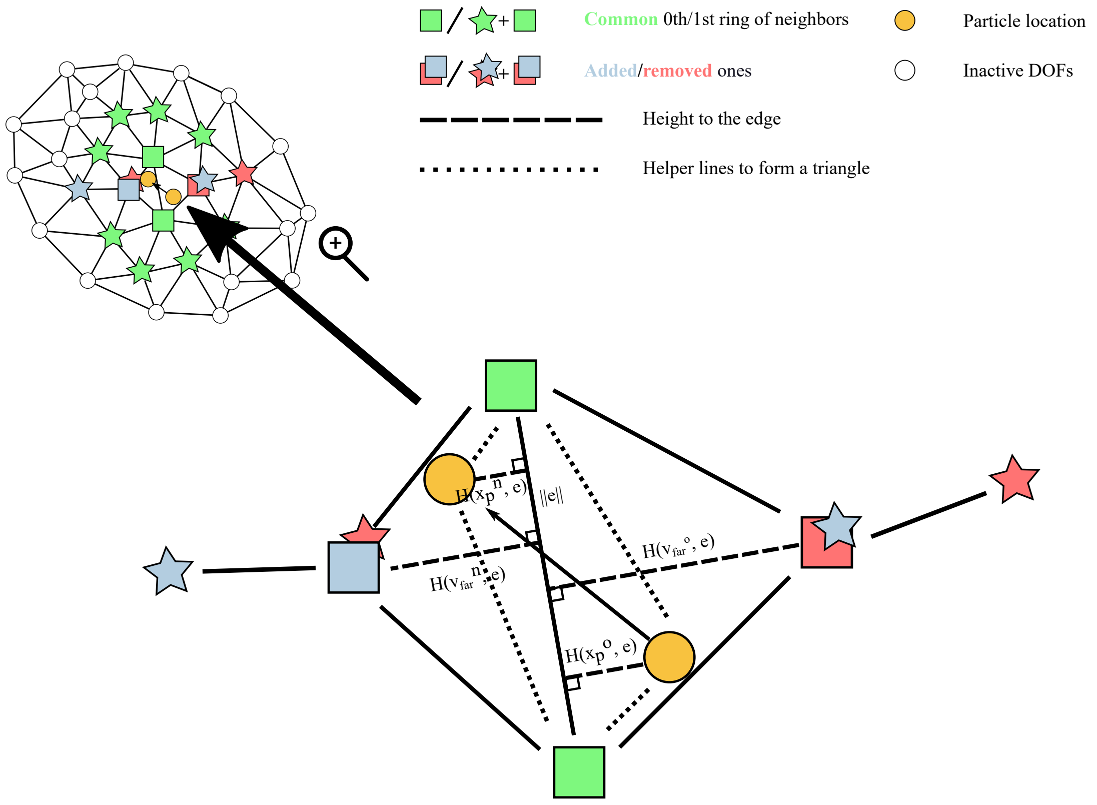


Figure 2.3: A visual proof that η diminishes locally. As described in (2.31), for every vertex v within the first ring of neighbors, $\mathcal{N}^1_{a,r}$. The dashed line denotes the perpendicular height from a given position to the shared edge. Dotted lines are drawn to construct a triangle between the point $\boldsymbol{x}_p$ and the edge, facilitating the computation of the barycentric coordinates.

cell respectively, for $\boldsymbol{x}_p$:

$$
\begin{aligned}
B^{n,o}_{v_{far}} &= \frac{H(\boldsymbol{x}^{n,o}_p, e)\cdot||e||}{H(v^{n,o}_{far}, e)\cdot||e||} \\
&= \frac{H(\boldsymbol{x}^{n,o}_p, e)}{H(v^{n,o}_{far}, e)} \\
&= \mathcal{O}(\frac{\epsilon}{H(v^{n,o}_{far}, e)}) \\
&= \mathcal{O}(\epsilon).
\end{aligned}
\tag{2.31}
$$

Finally, if a node is added/removed during the particle crossing (*i.e.*, $v \in \mathcal{N}^1_{a,r}$), this means that v is only connected to the far-away nodes $v^{n,o}_{far}$ but not to the edge e; *i.e.*,

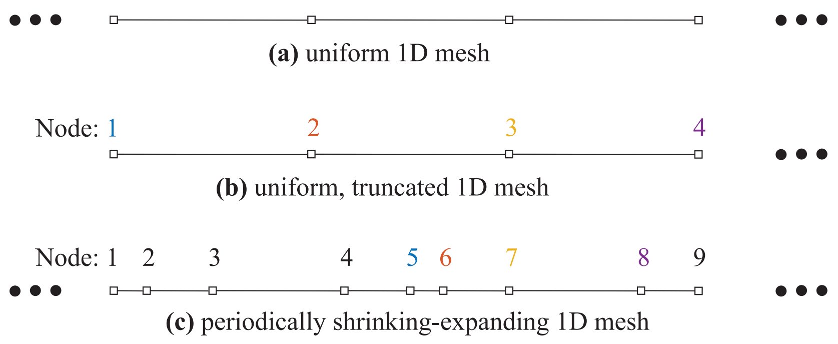

(a) uniform 1D mesh
Node: 1 2 3 4
(b) uniform, truncated 1D mesh
Node: 1 2 3 4 5 6 7 8 9
(c) periodically shrinking-expanding 1D mesh

Figure 2.4: Different 1D meshes used for verification of kernel reconstruction. (a) Uniform. (b) Uniform but truncated. (c) Periodically shrinking/expanding.

$A_{v,v^{n,o}_{far}} = 1, \forall v \in \mathcal{N}^1_{a,r}$, otherwise $A_{v,n} = 0, \forall n \in e \wedge \forall v \in \mathcal{N}^1_{a,r}$. Hence,

$$\begin{aligned}
\eta &= \sum_{n \in \mathcal{N}^0} B_n A_{v,n} \\
&= B^{n,o}_{v_{far}} A_{v,v^{n,o}_{far}} + \sum_{n \in e} B_n A_{v,n} \\
&= B^{n,o}_{v_{far}} \cdot 1 + \sum_{n \in e} B_n \cdot 0 \\
&= B^{n,o}_{v_{far}} \\
&= \mathcal{O}(\epsilon), \quad \forall v \in \mathcal{N}^1_{a,r}.
\end{aligned} \tag{2.32}$$

This concludes the proof. □

2.1.3.5 Verification of the Proposed Kernel

To verify that the proposed method can produce continuous reconstruction, analytical and numerical solutions of the kernels are produced on 1D and 2D meshes, respectively.

For the 1D case, the first basic verification is conducted on a uniform mesh, as the setup in Figure 2.4a. Here, each cell has a length of 1, so is the unit support length for the B-spline as the sample weights. The analytical solution for the uniform 1D mesh, obtained using `Mathematica 2023`, is as follows, where the kernel value is denoted as

$$f = \begin{cases} \frac{0.25(0.5-x)^2x}{x^5-6x^4+13.5x^3-13.75x^2+6.3125x-0.5625}, & 0.5 < x \leq 1 \\ \frac{-x^6+5x^5-9.5x^4+8.5x^3-3.3125x^2+0.3125x+0.0625}{-x^5+4x^4-5.5x^3+3.25x^2-1.3125x+1.0625}, & 1 < x \leq 1.5 \\ \frac{x^6-12x^5+58.5x^4-148.25x^3+205.563x^2-147.063x+42.25}{x^5-11x^4+47.5x^3-100.25x^2+103.313x-41.125}, & 1.5 < x \leq 2 \\ \frac{x^6-12x^5+58.5x^4-147.75x^3+202.563x^2-141.438x+39}{-x^5+9x^4-31.5x^3+53.75x^2-45.3125x+16.125}, & 2 < x \leq 2.5 \\ \frac{x^6-19x^5+149.5x^4-623.5x^3+1453.31x^2-1794.19x+915.687}{-x^5+16x^4-101.5x^3+318.75x^2-495.313x+304.188}, & 2.5 < x \leq 3 \\ \frac{-0.25x^3+2.75x^2-10.0625x+12.25}{-x^5+14x^4-77.5x^3+212.25x^2-288.313x+156.688}, & 3 < x \leq 3.5 \\ 0, & \text{Otherwise,} \end{cases} \tag{2.33}$$

and the gradient estimation is denoted as

$$g = \begin{cases} -\frac{1(0.5-x)^2x(x-2)}{x^5-6x^4+13.5x^3-13.75x^2+6.3125x-0.5625}, & 0.5 < x \leq 1 \\ \frac{-x^5+5x^4-10.5x^3+11.5x^2-5.8125x+1.0625}{-x^5+4x^4-5.5x^3+3.25x^2-1.3125x+1.0625}, & 1 < x \leq 1.5 \\ \frac{-x^5+9x^4-33.5x^3+65.75x^2-67.3125x+27.625}{-x^5+11x^4-47.5x^3+100.25x^2-103.313x+41.125}, & 1.5 < x \leq 2 \\ \frac{-x^5+11x^4-49.5x^3+112.25x^2-125.313x+53.625}{x^5-9x^4+31.5x^3-53.75x^2+45.3125x-16.125}, & 2 < x \leq 2.5 \\ \frac{-x^5+15x^4-90.5x^3+274.5x^2-417.813x+254.188}{x^5-16x^4+101.5x^3-318.75x^2+495.313x-304.188}, & 2.5 < x \leq 3 \\ \frac{x^4-13x^3+62.25x^2-129.5x+98}{-x^5+14x^4-77.5x^3+212.25x^2-288.313x+156.688}, & 3 < x \leq 3.5 \\ 0, & \text{Otherwise.} \end{cases} \tag{2.34}$$

Figure 2.5a shows the continous kernel reconstruction with the diminishing function η, while Figure 2.5b, as an ablation, shows that the discontinuous reconstruction even for the simplest uniform mesh, proving the necessity of η.

Note that the presence of a particle within a boundary element, as depicted in Figure 2.4b, can lead to a negative weight value for the most interior node. This phenomenon is exemplified by Node 3 in Figure 2.6, where the kernel degenerates due to the absence of a first ring of neighboring elements on the boundary side during MLS sampling. To remedy the problem of negative kernel values and prevent numerical

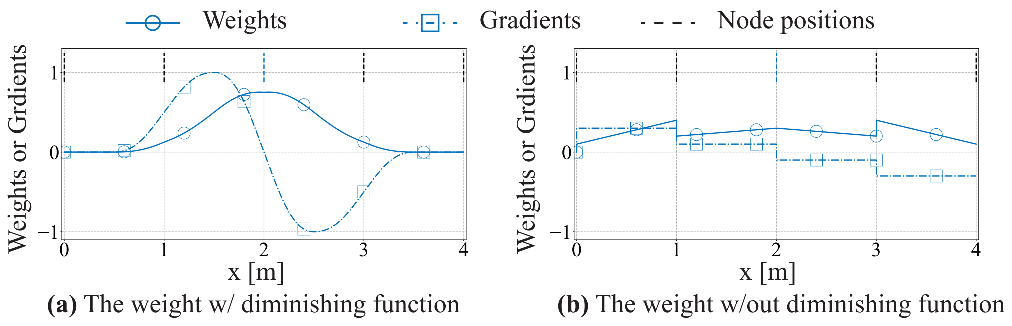


Figure 2.5: Ablation study of reconstruction with or without diminishing on 1D mesh. Comparison of kernel values and gradient estimations on a uniform 1D mesh (a) with and (b) without applying the diminishing function.

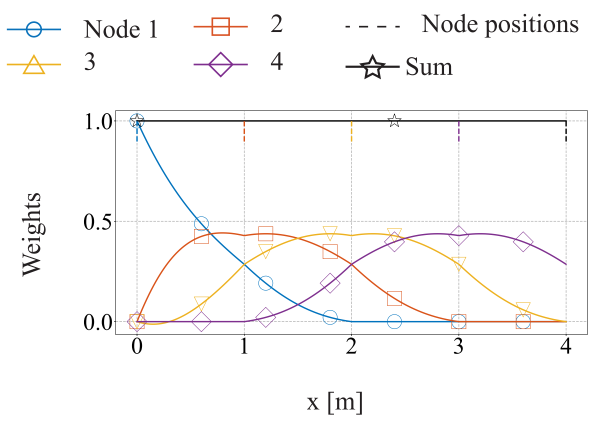


Figure 2.6: A failure example when there is no extra layer near the boundary. The negative weight for Node 3 (yellow) when the particle is in the boundary cell and there is no extra layer.

instabilities (Andersen and Andersen, 2010), drawing an additional layer of elements beyond the original boundary is recommended in practice.

The next verification is on a periodically shrinking and expanding 1D mesh (Figure 2.4c). The mesh contains cyclic cell sizes of $[\dots, 1, R, R^2, R, 1, \dots]$ designed to mimic the transition between varying resolutions. The size transition ratios tested range from 1.1 to 1.5 so as to correspond with the typical transition ratios in FEM analysis. Kernel reconstructions are conducted on Nodes 5, 6, 7, and 8 as they can present a full cycle. As shown in Figure 2.7, both the kernel and the gradient estimations are piece-wise $\mathcal{C}^1$.

The final verification and ablation tests were performed on a 2D unstructured mesh featuring "&" shapes. The comparison between scenarios with and without the use of η,

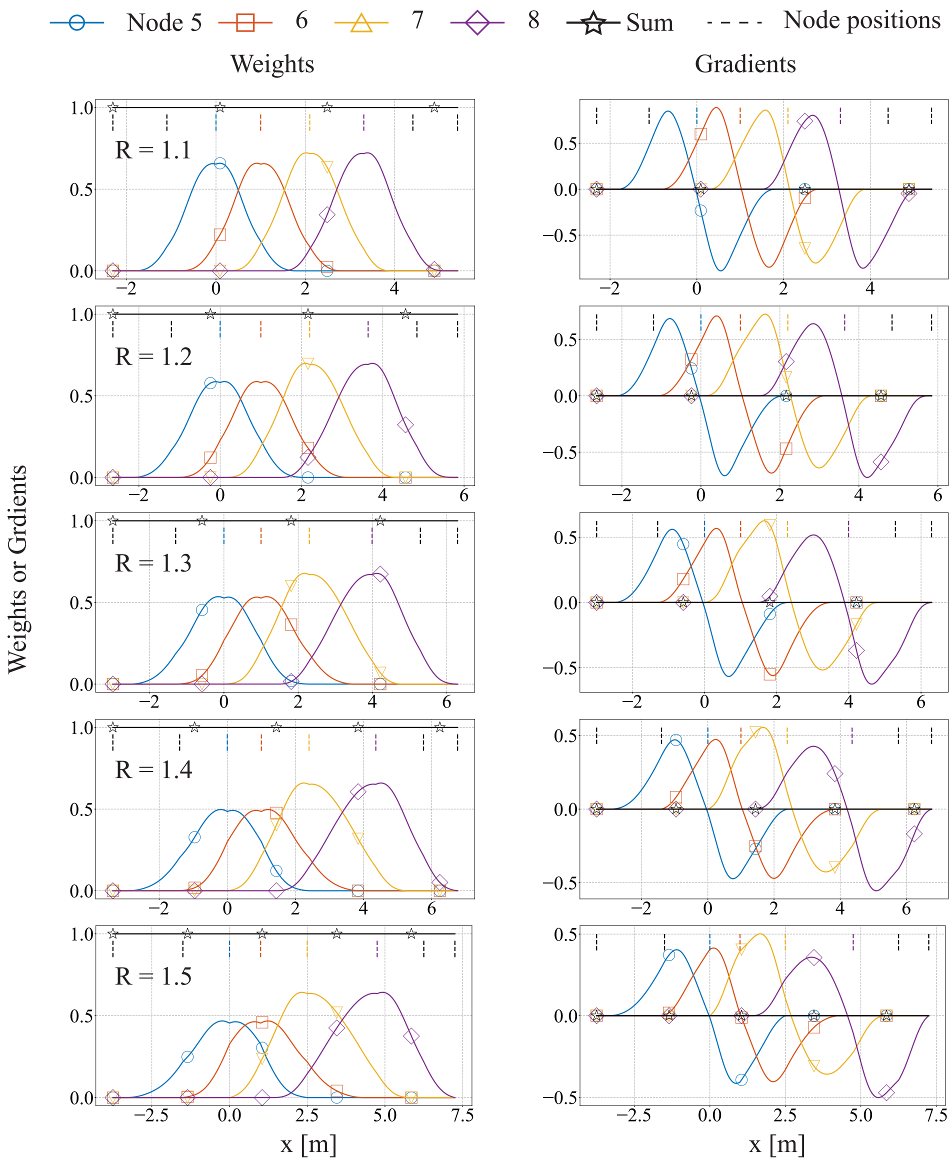
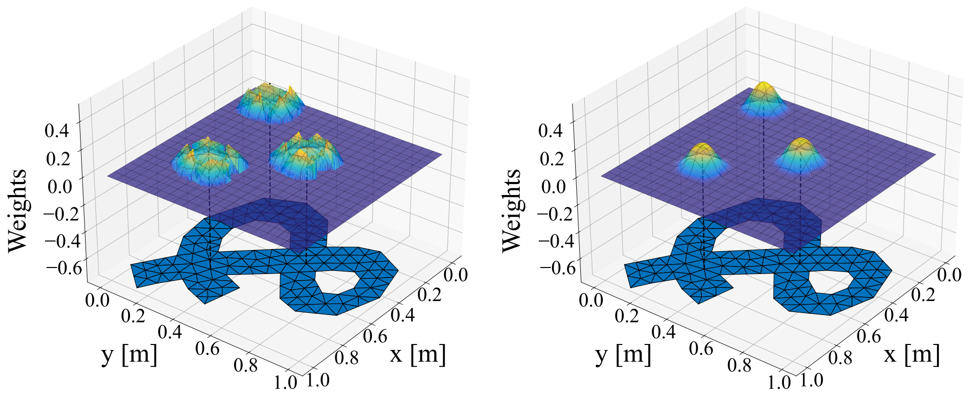


Figure 2.7: Reconstruction on a periodically shrinking/expanding 1D mesh. Kernel values (left column) and gradient estimations (right column).



(a) The weight w/out diminishing function **(b)** The weight w/ diminishing function

Figure 2.8: Ablation study of reconstruction with or without diminishing on 2D mesh. (a) without and (b) with the application of the diminishing function.

as shown in Figure 2.8a and Figure 2.8b respectively, validates the importance of η and the effectiveness of the proposed method in managing unstructured meshes.

2.2 Conservation of the Linear and Affine Momentum When Combined With Affine Particle-in-Cell

In the previous section, we numerically verified the continuous reconstruction capabilities of our approach. Before we move into conducting numerical test cases, it is noteworthy to highlight that UMLS-MPM conforms to the partition of unity and conserves the linear basis, assured by the properties of Moving Least Squares (MLS) (Levin, 1998), specifically:

$$\sum_i w_{ip}^n = 1,$$

$$\sum_i w_{ip}^n \boldsymbol{x}_i^n = \boldsymbol{x}_p^n,$$

$$\sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) = 0,$$

As a result, incorporating UMLS-MPM into the Affine Particle-In-Cell (APIC) framework (Jiang et al., 2015, 2017b) will conserve both total linear and total angular momentum of the system. This implies a significant advantage of our methodology: that it can be seamlessly integrated with any existing MPM framework to broaden its application on unstructured meshes.

For self-completeness, a simple introduciton to APIC is provided herein, while a more thorough proof and explanation are available in the original APIC paper (Jiang et al., 2015).

In APIC, mass m_p, position $\boldsymbol{x}_p$, velocity $\boldsymbol{v}_p$, and an affine matrix $\boldsymbol{B}_p = \sum_i w_{ip} v_i (\boldsymbol{x}_i - \boldsymbol{x}_p)^T$ are stored and tracked on particles. Then,

Definition 2.2.1. The total linear momentum on grids is

$$\boldsymbol{P}_i^{tot} = \sum_i m_i \boldsymbol{v}_i.$$

Definition 2.2.2. The total linear momentum on particles is

$$\boldsymbol{P}_p^{tot} = \sum_p m_p \boldsymbol{v}_p.$$

Definition 2.2.3. The total angular momentum on grids is

$$\boldsymbol{I}_i^{tot} = \sum_i \boldsymbol{x}_i \times m_i \boldsymbol{v}_i.$$

Definition 2.2.4. The total angular momentum on particles is

$$\boldsymbol{I}_p^{tot} = \sum_p \boldsymbol{x}_p \times m_p \boldsymbol{v}_p + \sum_p m_p (\boldsymbol{B}_p)^T : \epsilon,$$

where ϵ is the Levi-Civita permutation tensor, and for any matrix $\boldsymbol{A}$, the contraction $\boldsymbol{A} : \epsilon = \sum_{\alpha\beta} A_{\alpha\beta} \epsilon_{\alpha\beta\gamma}$, which is usually used to transition from a cross product into the tensor product $\boldsymbol{u} \times \boldsymbol{v} = (\boldsymbol{v}\boldsymbol{u}^T)^T : \epsilon$. Also note that for the total angular momentum of the particles: 1) the grid node locations can be perceived as the sample points of a rotating mass centered at the material particle location, and 2) the total angular momentum comprises both that of the center and that of the affine-rotation of the grids around the center.

APIC P2G is given by

$$\begin{aligned} m_i^n &= \sum_p w_{ip}^n m_p \\ \boldsymbol{D}_p^n &= \sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)(\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T \\ m_i^n \boldsymbol{v}_i^n &= \sum_p w_{ip}^n m_p (\boldsymbol{v}_p^n + \boldsymbol{B}_p^n (\boldsymbol{D}_p^n)^{-1} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \end{aligned} \tag{2.35}$$

with G2P given by

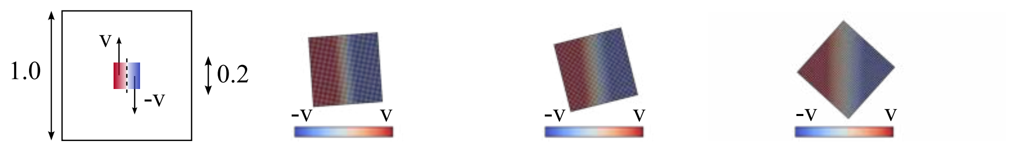
$$\begin{aligned} \boldsymbol{v}_p^{n+1} &= \sum_i w_{ip}^n \tilde{v}_i^{n+1} \\ \boldsymbol{B}_p^{n+1} &= \sum_i w_{ip}^n \tilde{v}_i^{n+1} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T, \end{aligned} \tag{2.36}$$

where the superscript $\tilde{}$ means the intermediate value after the update on grids but before the G2P process.

2.2.1 Numerical Validation

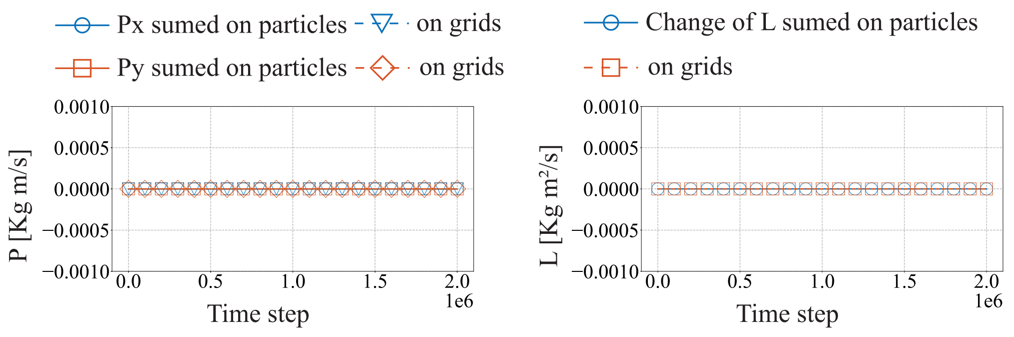
A numerical validation as in (Jiang et al., 2015) is also conducted here to verify these conservations. A square with a side length of $l = 0.2$ is discretized with 20×20 particles. The physical properties of the square are as follows: $E = 1 \times 10^4\,\text{Pa}$, $\nu = 0.3$, and $\rho = 1.0\,\text{kg/m}^3$. Initially, the square is divided into two halves by a hypothetical vertical line through the middle. The left half is initialized with an upward velocity $\mathbf{v} = (1, 0)\,\text{m/s}$, while the right half is initialized with a downward velocity $\mathbf{v} = (-1, 0)\,\text{m/s}$. The schematic plot of the setup is presented in Figure 2.9a.

The background mesh is generated using Delaunay triangulation with a target element size of $0.01\,\text{m}$ in a $1 \times 1\,\text{m}^2$ box. The simulation is run for 1×10^6 time steps with a time step size of $1 \times 10^{-5}\,\text{s}$. Three typical timestamps are plotted in Figure 2.9b–d.



(a) setup **(b)** step 2E4 **(c)** step 6E4 **(d)** step 2E5

Figure 2.9: Setup and snapshots of a rotating elastic square.



(a) Linear momentum **(b)** Angular momentum

Figure 2.10: Conservation of the rotating elastic square system. Logs of (a) linear and (b) angular momentum of the rotating cube experiment after 10^6 time steps.

The proposed conservation is accurately illustrated in Figure 2.10a–b, with only round-off errors on the order of 1×10^{-15} and 1×10^{-7} for the total linear and affine momentum of the system, respectively.

2.3 Experiments and Results

To demonstrate and assess the effectiveness of our approach, particularly its reduced cross-cell error owing to the continuous gradient reconstruction, we have chosen representative test cases from prior related studies. Our benchmarking relies on analytical solutions when feasible, or alternatively, on the standard B-spline MPM at a sufficiently high resolution. All experiments were carried out on a single PC equipped with an Intel® Core™ i9-10920X CPU.

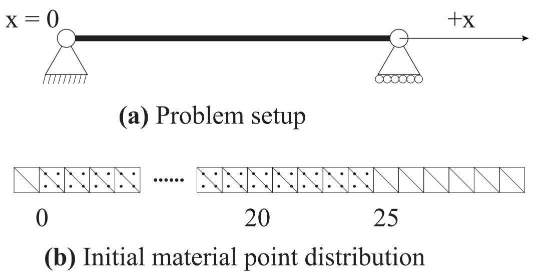


Figure 2.11: Setup of the 1D bar vibration test.

2.3.1 1D Vibrating Bar

Consider the 1D vibration bar problem shown in Figure 2.11a (Wilson et al., 2021). The left end of the bar is fixed and the right has a sliding condition in the x direction. The physical properties of the bar are: $E = 100\,\mathrm{Pa}$, $\nu = 0$, $L = 25\,\mathrm{m}$, and $\rho = 1\,\mathrm{kg/m^3}$. The initial velocity conditions are $\dot{u}(x, t = 0) = v_0 \sin(\beta_1 x)$ with $\beta_1 = \frac{\pi}{2L}$.

The analytical expression of the center of mass in this problem is

$$x(t)_{CM} = \frac{L}{2} + \frac{v_0}{\beta_1 L \omega_1} \sin(\omega_1 t), \tag{2.37}$$

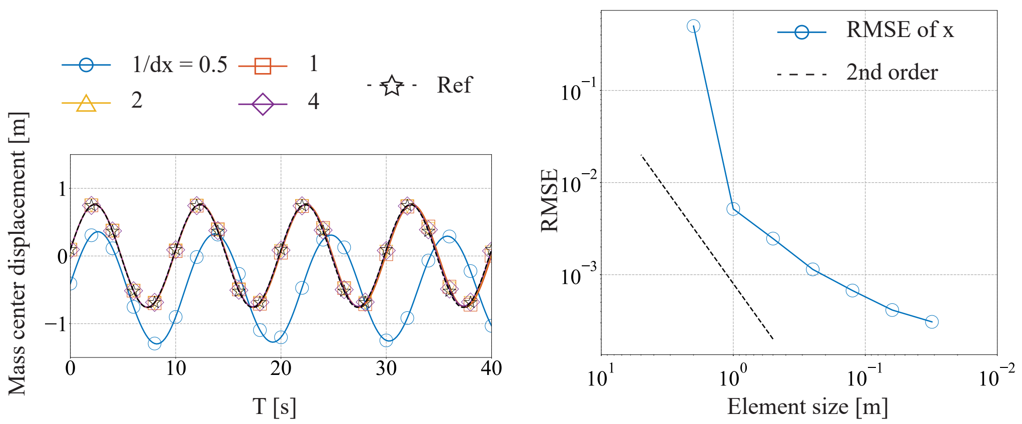
and

$$\dot{u}(t)_{CM} = \frac{v_0}{\beta_1 L} \cos(\omega_1 t), \tag{2.38}$$

with $\omega_1 = \beta_1 \sqrt{E/\rho}$.

The original experiments in (Wilson et al., 2021) included two velocity settings: $v_0 = 0.1\,\mathrm{m/s}$ and $v_0 = 0.75\,\mathrm{m/s}$. The lower velocity setting, $v_0 = 0.1\,\mathrm{m/s}$, was utilized solely for validation against the linear kernel MPM, as it does not involve cell crossings. Here, we focus on the higher-velocity setting to assess the effectiveness of UMLS-MPM in addressing cell-crossing errors.

Figure 2.12 presents the convergence rate of UMLS-MPM with grid refinement. Specifically, Figure 2.12a shows that, with the exception of the coarsest resolution $dx = 2\,\mathrm{m}$, UMLS-MPM consistently achieves high accuracy, with a maximum root mean square error (RMSE) of 0.554% in particle displacements. Figure 2.12b indicates that the convergence



(a) Mass center displacement **(b)** Convergence rate plot

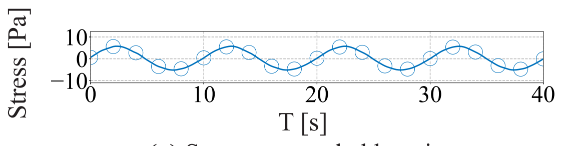
Figure 2.12: Displacement and convergence rate of the 1D vibration bar. (a) The center of mass displacement of the bar and (b) convergence rate of the RMSE of particle displacements.

rate is approximately second order on coarser grids, but it starts to level off on finer grids due to mounting temporal errors, aligning with established MPM theory (Jiang et al., 2016b).

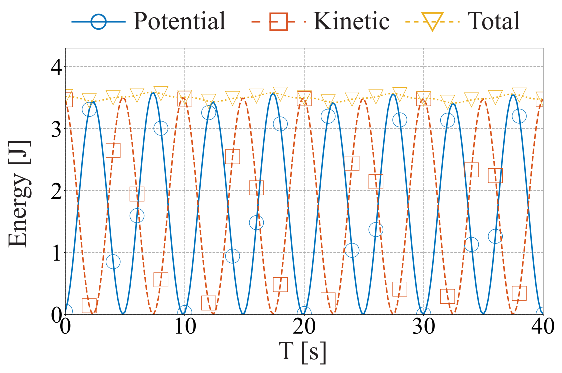
Figure 2.13a displays the stress profile for a particle located at $x_0 = 12.75\,\mathrm{m}$, which undergoes the most frequent cell crossings during its vibrational motion. The outcomes achieved with UMLS-MPM showcase a remarkable level of smoothness and precision. Figure 2.13b illustrates the energy dynamics for the entire system, revealing that the system's energy is largely conserved throughout the simulation, with only slight fluctuations. These findings collectively underscore the robustness and precision of UMLS-MPM in managing intense cell crossings by particles.

2.3.2 2D Collision Disks

Next, we considered the problem of two colliding elastic disks shown in Figure 2.14a (Wilson et al., 2021). The physical properties of the disks are: $E = 1000\,\mathrm{Pa}$, $\nu = 0.3$, $\rho = 1000\,\mathrm{kg/m^3}$, and $\boldsymbol{v} = \pm(0.1, 0.1)\,\mathrm{m/s}$ for the left and right disks, respectively. Each disk was discretized with 462 material points using the triangle mesh of a disk. The background mesh was generated using Delaunay triangulation with a target element size

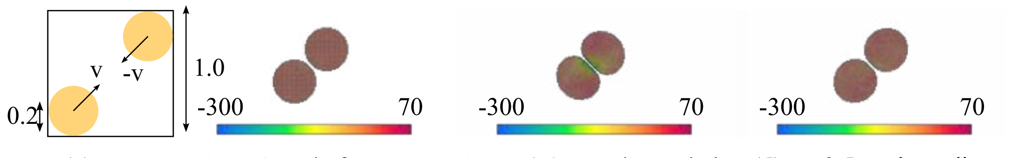


(a) Stress at sampled location



(b) Energy

Figure 2.13: The stress of sampled particle and system energy of the 1D vibration bar. (a) The stress at the sampled particle closest to [17.5, 0.5] and (b) the system energy.

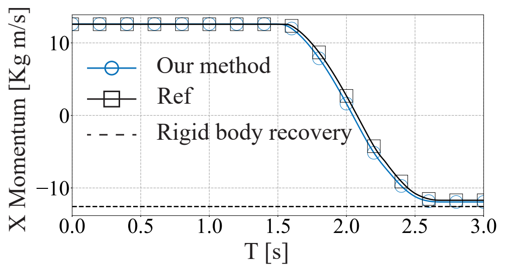


(a) setup **(b)** t = 1.5s; before contact **(c)** t = 2.0s; total retardation **(d)** t = 2.5s; rebounding

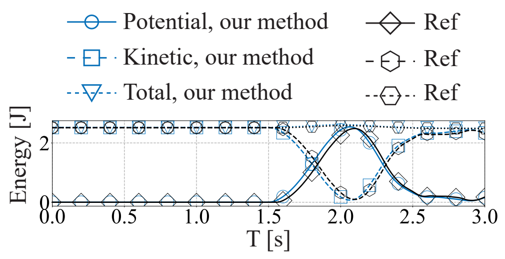
Figure 2.14: Setup and snapshots of the 2D collision disks. (a) Problem setup, (b)–(d) Snapshots of the simulation at 1.5s, 2.0s, and 2.5s.

of 0.025 m. We plot key snapshots of the simulation in Figure 2.14b–d, with the impact at 1.5 s, total retardation right before 2.0 s, and rebounding separation right before 2.5 s.

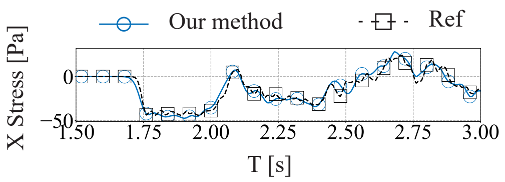
Quantitative results for the collision disks are presented in Figure 2.15. In Figure 2.15a, a comparison of momentum recovery during collision between UMLS-MPM and the B-spline MPM with sufficiently high resolution is shown. While a perfect momentum recovery, such as that in the rigid collision (dashed gray line in Figure 2.15a), is not expected, UMLS-MPM approaches this limit effectively. Similarly, Figure 2.15b displays the kinetic energy recovery during the collision. The results indicate that UMLS-MPM effectively preserves the system energy. Figure 2.15c illustrates the stress log at the center particle of the left disk. The results align perfectly with the reference, but only for negligible fluctuations, showing that UMLS-MPM does not generate spurious stress



(a) X momentum log

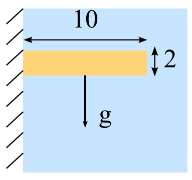


(b) Energy log

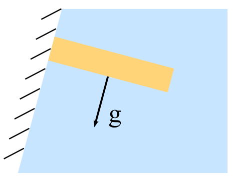


(c) Stress log at sampled point

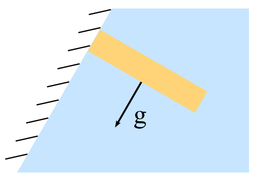
Figure 2.15: System momentum, energy, and sampled stress of the 2D collision disks. (a) The momentum in the x-direction of the left disk, (b) the energies of the system, and (c) the stress at the sampled particle closest to the center of the left disk.



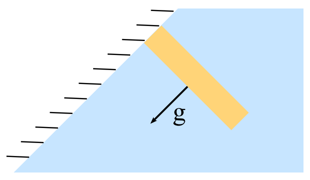
(a) zero rotation



(b) rotated 15$^{\text{O}}$



(c) rotated 30$^{\text{O}}$



(d) rotated 45$^{\text{O}}$

Figure 2.16: Setup of the 2D cantilever problem under different rotation angles. (a) 0°, (b) 15°, (c) 30°, and (d) 45°.

oscillations either from the collision or cell crossings.

2.3.3 2D Cantilever With Rotations

Although an unstructured mesh offers the adaptability to match any boundary shape, the cell orientation, or a different tessellation, can potentially affect accuracy. To illustrate the precision of our method under various rotation angles, we examined the case of a cantilever under its own weight, as shown in Figure 2.16a (Wilson et al., 2021). The cantilever's physical characteristics are as follows: length $l = 10\,\text{m}$, height $h = 2\,\text{m}$, gravitational acceleration $g = 9.81\,\text{m/s}^2$, Young's modulus $E = 100000\,\text{Pa}$, Poisson's ratio $\nu = 0.29$, and density $\rho = 2\,\text{kg/m}^3$. The cantilever was discretized with uniformly spaced particles

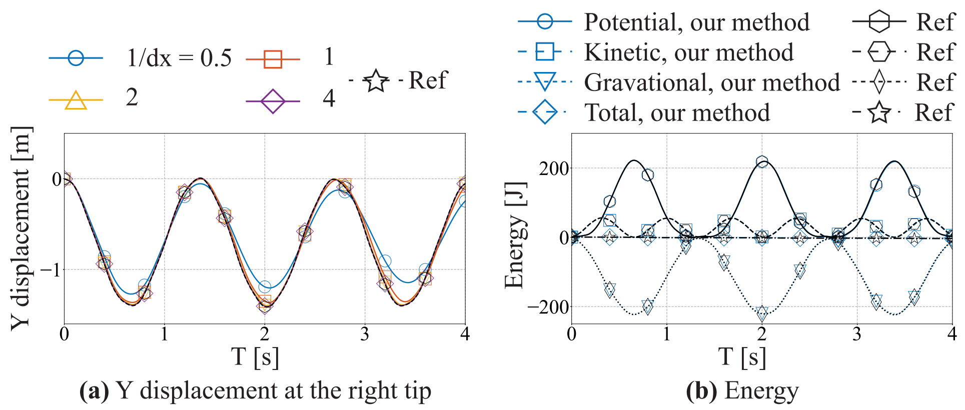


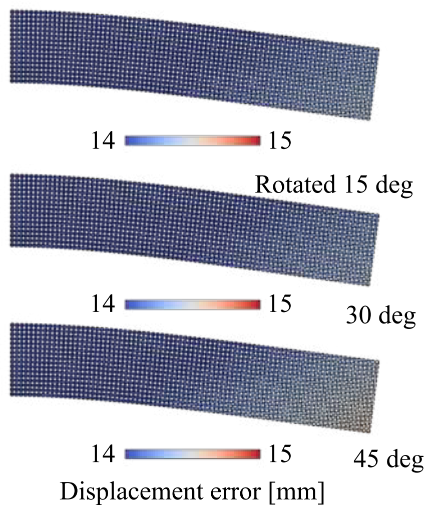
Figure 2.17: Displacements and system energy of the 2D cantilever without rotation. Plots of (a) the displacement in the y-direction at the right tip of the cantilever and (b) the energies of the system.

in both directions. We created the background mesh using Delaunay triangulation, aiming for an element size of 0.5 m. Additionally, we rotated the mesh of the cantilever by angles of 15°, 30°, and 45° to showcase the resilience of our method to rotation, as depicted in Figure 2.16b–d.

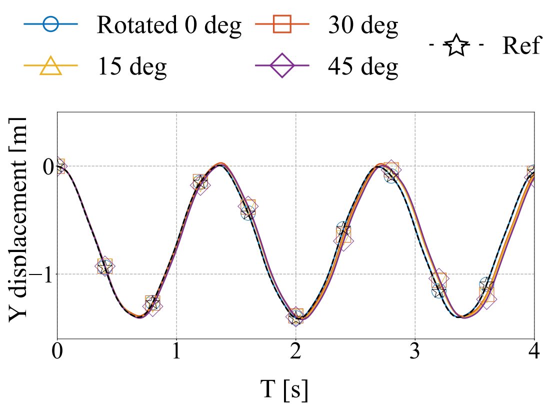
Figure 2.17a illustrates the spatial convergence of the y-displacement at the right tip of the cantilever beam under grid refinement. Notably, except for the coarse resolutions of $dx = 2$ m and $dx = 1$ m, errors for all finer resolutions are negligible. Therefore, a resolution of $dx = 0.5$ m was employed to ensure sufficient accuracy for all subsequent plots in this experiment. Figure 2.17b demonstrates that UMLS-MPM effectively conserves energy, aligning with the reference B-spline MPM.

Figure 2.18a shows snapshots of the cantilever with different initial mesh rotation angles. The results indicate that UMLS-MPM is robust under mesh rotation with only minor visible errors. Figure 2.18b quantitatively compares the y-displacement at the right tip. The results align well overall with both zero rotation and the reference, with errors of 1.27%, 2.18%, and 4.72% for 15°, 30°, and 45° rotation, respectively.

The convergence rate of UMLS-MPM is demonstrated in Figure 2.19. The results indicate that for cases with zero rotation, the convergence rate is second order. While the RMSE increases slightly for cases with mesh rotation, it still remains in the magnitude of $1E-2$, and the convergence rate remains near second order. These combined results



(a) Error comparison



(b) Y displacement comparison

Figure 2.18: (a) Snapshots of the cantilever with different initial rotating angles. (b) Comparison of the displacement in the y-direction at the right tip of the cantilever.

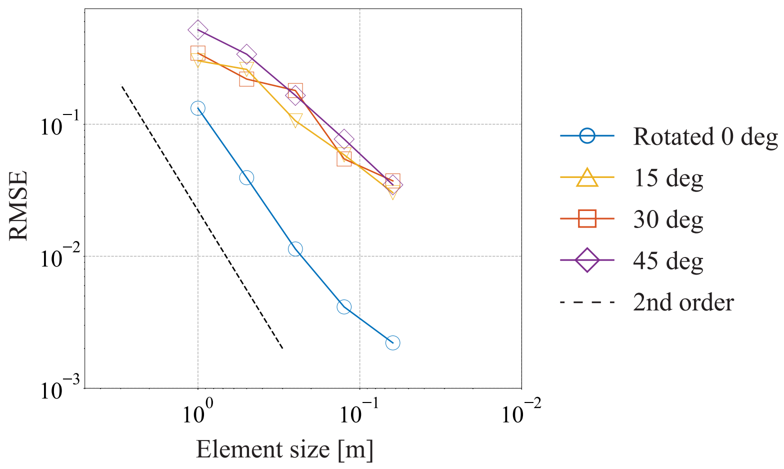


Figure 2.19: Convergence plot of the displacements of the 2D cantilever with rotations.

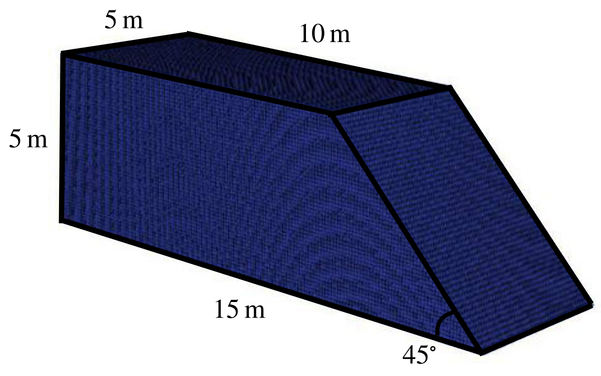


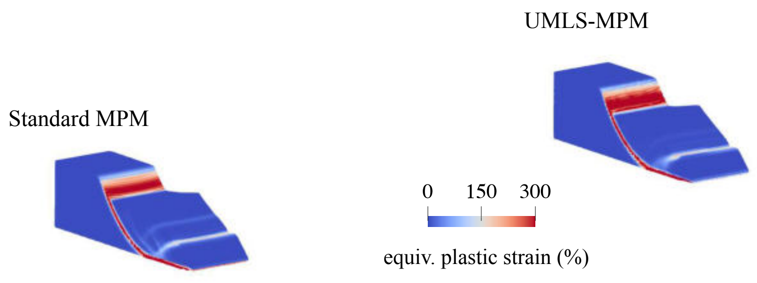
Figure 2.20: Setup of the 3D slope failure. Geometry adapted from (Zhao et al., 2023b).

demonstrate the robustness and accuracy of UMLS-MPM under mesh rotation.

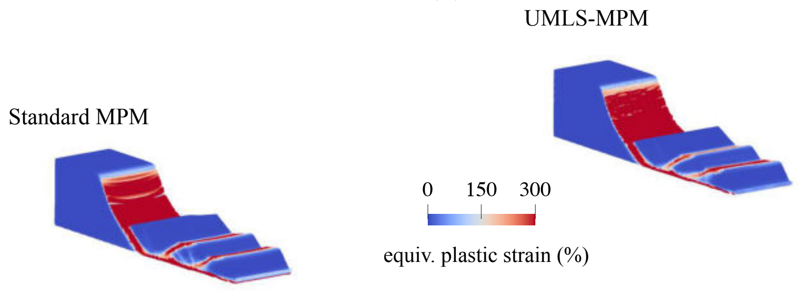
2.3.4 3D Slope Failure

Next, the performance of the proposed approach was investigated when dealing with material behavior involving plasticity. To this end, we simulated failure of a 3D slope comprosed of sensitive clay. The problem geometry was adopted from (Zhao et al., 2023b) and is illustrated in Figure 2.20. Here, the bottom boundary of the slope is fixed and the three lateral sides are supported with rollers. To model the elastoplastic behavior of the sensitive clay in an undrained condition, a combination of Hencky elasticity and J2 plasticity with softening was used. The softening behavior is governed by the following exponential form: $\kappa = (\kappa_p - \kappa_r)e^{-\eta\varepsilon_q^p} + \kappa_r$, where κ, κ_p, and κ_r denote the yield strength, the peak strength, and the residual strength, respectively, ε_q^p denotes the equivalent plastic strain, and η is a softening parameter. The specific parameters were adopted from (Zhao et al., 2023b). They are a Young's modulus of $E = 25\,\text{MPa}$, a Poisson's ratio of $\nu = 0.499$, a peak strength of $\kappa_p = 40.82\,\text{kPa}$, a residual strength of $\kappa_r = 2.45\,\text{kPa}$, and a softening parameter of $\eta = 5$. The assigned soil density is $\rho = 2.15\,\text{t/m}^3$.

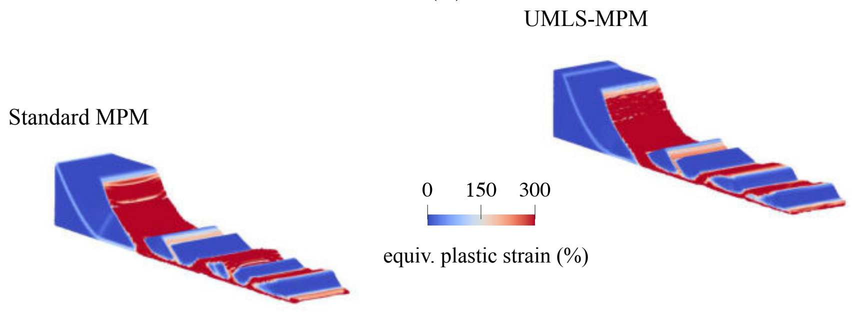
The space was discretized using Delaunay triangulation with the shortest edge length of 0.2 m. The material points were initialized with a spacing of 0.1 m in each direction, amounting to 311,250 material points in the initial slope region. Note that the spatial discretization aligns with the one used in (Zhao et al., 2023b) in terms of both the



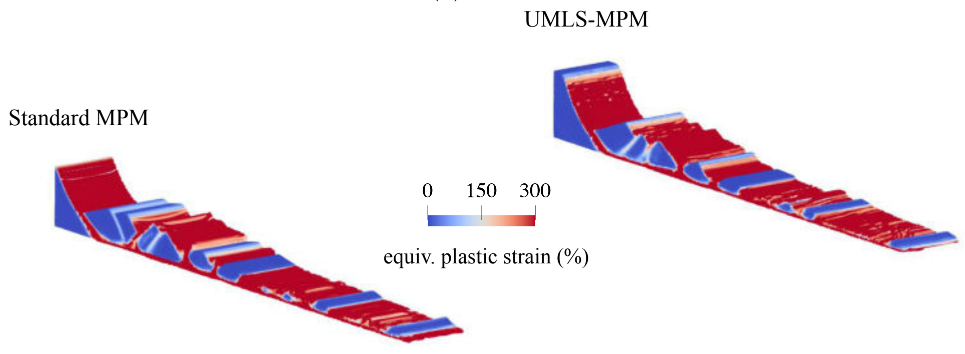
(a) $t = 1.5$ s



(b) $t = 2.5$ s

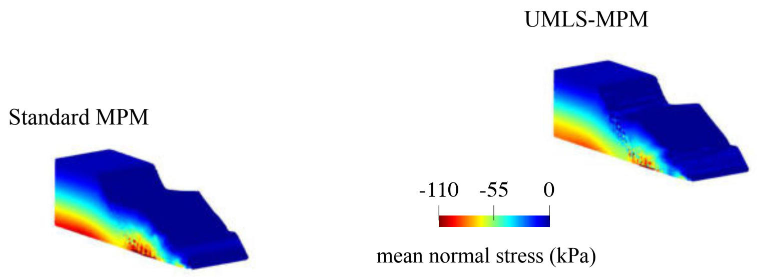


(c) $t = 3.5$ s

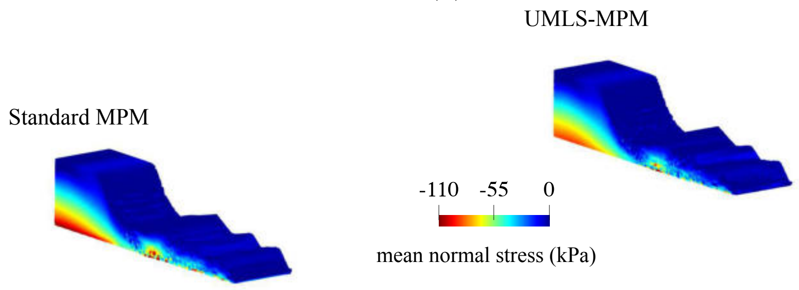


(d) $t = 5.5$ s

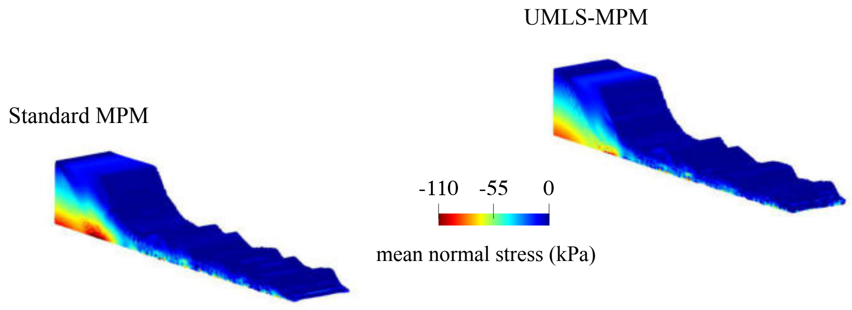
Figure 2.21: Comparison of the equivalent plastic strain between baselines. (Left) the standard MPM uses quadratic B-splines basis functions as in Zhao *et al.*(Zhao et al., 2023b), (Right) UMLS-MPM .



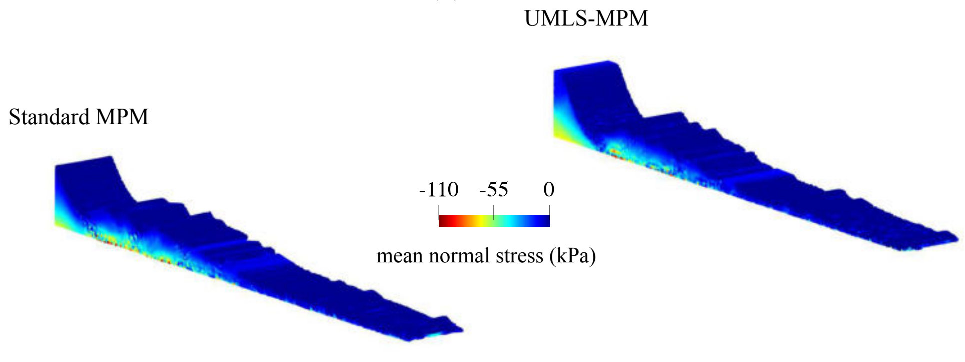
(a) $t = 1.5$ s



(b) $t = 2.5$ s



(c) $t = 3.5$ s



(d) $t = 5.5$ s

Figure 2.22: Comparison of the mean normal stress between baselines. (Left) the standard MPM uses quadratic B-splines basis functions as in Zhao *et al.*(Zhao et al., 2023b), (Right) UMLS-MPM .

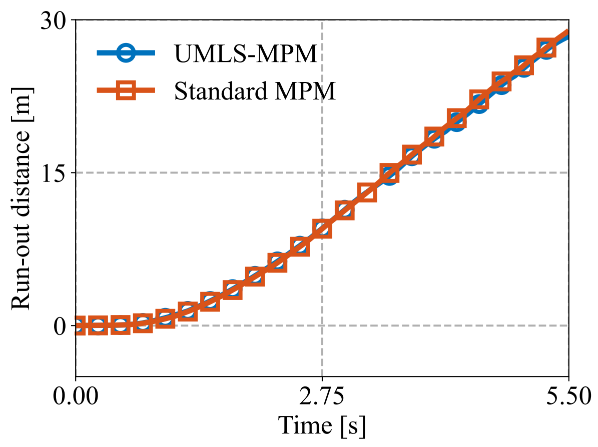


Figure 2.23: Comparison of the run-out distance of the 3D slope failure.

shortest edge length of the background element and the number of material points. Also, the $\bar{\boldsymbol{F}}$ approach proposed in (Zhao et al., 2023b) was utilized to circumvent volumetric locking that UMLS-MPM solutions encounter when simulating a large number of particles of incompressible materials. As a reference to verify the correctness of the proposed formulation, the $\bar{\boldsymbol{F}}$ solution in (Zhao et al., 2023b) was used.

Figures 2.21 and 2.22 show the snapshots of the slope simulated by the standard and UMLS-MPM, where particles are colored by the equivalent plastic strain and mean normal stress, respectively. We can see that UMLS-MPM effectively captures the retrogressive failure pattern of slopes made of sensitive clay. Also, in terms of equivalent plastic strain fields and mean normal stress fields, we observe a strong similarity between the UMLS-MPM solution and the reference solution from (Zhao et al., 2023b).

For a further quantitative comparison, Figure 2.23 presents the time evolutions of the run-out distance—a measure of the farthest movement of the sliding mass. Observe that the distances in the standard and UMLS-MPM solutions are remarkably similar. Taken together, these findings confirm that the proposed method performs similarly to the standard MPM.

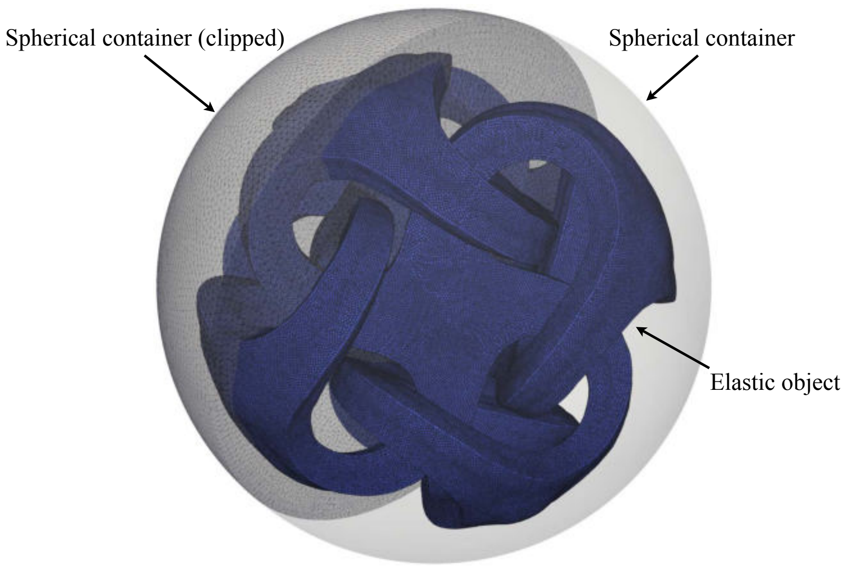


Figure 2.24: Setup of the 3D elastic object expansion.

2.3.5 3D Elastic Object Expansion in a Spherical Container

Finally, we examined the performance of UMLS-MPM in problems involving complex boundary geometry. In this problem, the standard MPM with a structured grid may be challenged to impose conforming boundary conditions. Hence, a collision between an elastic body with a spherical container was considered and simulated.

The geometry of the problem, as demonstrated in Figure 2.24, involves an elastic object in the shape of a Metatron, which is located at the center of a spherical container (with a radius of 0.5 m). The object is initially compressed isotropically (with an initial deformation gradient of $\boldsymbol{F} = 0.75\boldsymbol{I}$), storing non-zero elastic potential energy. At the onset of the simulation, the stored elastic energy is released, causing the object to expand and collide with the spherical container's boundary. To capture the elastic behavior of the object, a Neohooken elasticity was adopted with a Young's modulus of 3.3 MPa and a Poisson's ratio of $\nu = 0.49$. The elastic object was discretized using a significant number of material points (2,392,177) for high-fidelity simulation. Also, the spherical container was discretized using 2,178,129 tetrahedral elements, each with an average edge length of $h = 0.025$ m. Note that to avoid negative kernel values at boundary nodes, an extra layer of elements was added outside the original boundary, as discussed in Section 2.1.3.5.

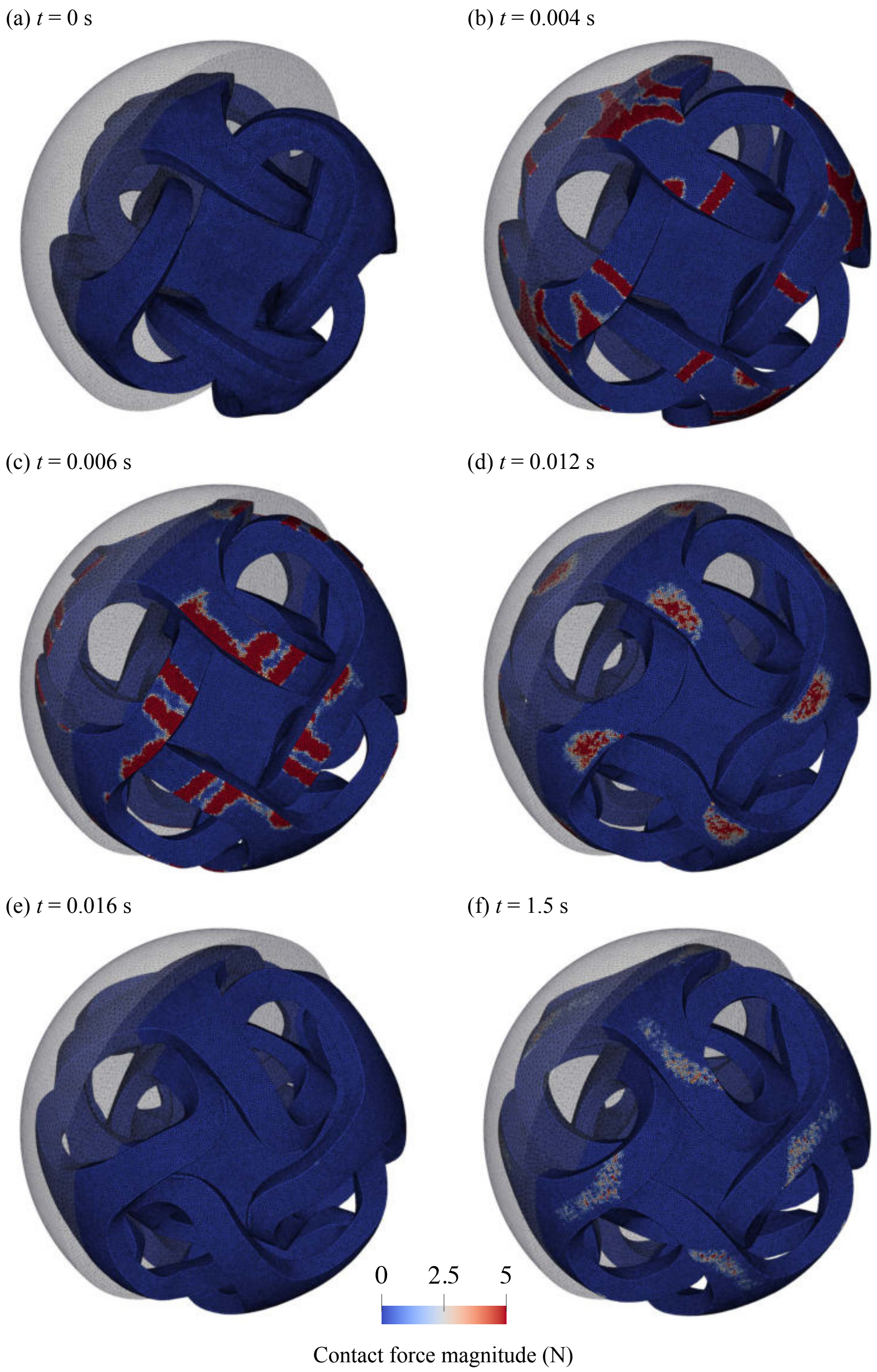


Figure 2.25: The contact force magnitude of the 3D elastic object expansion.

To consider the frictional collision between the elastic object and the container boundary, a barrier approach (Zhao et al., 2023a) was adopted, ensuring that the elastic object does not penetrate the boundary. Contact forces are applied when the distance between a material point and the boundary is below a specific value $\hat{d}$, which was chosen to be a quarter of h for sufficient accuracy. Also, a friction coefficient of $\mu = 0.5$ was introduced to stop the sliding of the elastic object in the later stages. The simulation ran with a time increment of $\Delta t = 6.16 \times 10^{-5}\,\text{s}$ until $t = 1.5\,\text{s}$.

Figure 2.25 presents six snapshots simulated with UMLS-MPM, where the particles are colored based on the magnitude of the contact force. The dynamic behavior of the object at various stages is well captured, including the initial expansion stage (a), the first collision stage (b–d), the rebounding stage (e), and the final static stage (f). Overall, the UMLS-MPM effectively handles complex geometry with a conformal discretization, which is critical for simulating a wide range of interactions between deformable objects and complex boundaries.

CHAPTER 3

Shockwave and Compressible Flow Simulation With Material Point Methods

This chapter introduces a novel approach for modeling the interaction between compressible flow, shock waves, and deformable structures, with a focus on capturing destructive dynamics. Building upon recent advancements in time-splitting compressible flow and Material Point Methods (MPMs), we present a hybrid scheme that combines Eulerian and Lagrangian/Eulerian methods for monolithic flow-structure interactions. The continuity equation is advanced using the second-order WENO scheme for improved accuracy. To effectively handle deforming boundaries with sub-cell particles, we propose a blending treatment of reflective and passable boundary conditions, drawing inspiration from porous media theory. The coupled velocity-pressure system is discretized using a new mixed-order finite element formulation that employs B-spline shape functions. Our approach seamlessly captures shock wave propagation, temperature/density-induced buoyancy effects, and topology changes in solids.

3.1 Related Work

An extensive body of mechanical literature exists on traditional partitioned methods for coupling compressible flow with solids. The most well-known categories include the arbitrary Lagrangian-Eulerian method (Banks et al., 2016), the immersed boundary method (Pasquariello et al., 2016; Wang et al., 2017; Monasse et al., 2012), the discontinuous Galerkin method (Kosík et al., 2015), the ghost fluid method (Bailoor et al., 2017), and the Lattice Boltzmann method (Li and Favier, 2017). However, many of these methods only support small deformations or vibrations, and only some of them accommodate large deformations without fracture. In the field of computer graphics, (Sewall et al., 2009) also adopted the partitioned method to weakly couple shockwaves with rigid bodies.

A defining visual characteristic of explosions and shockwaves is their destructive impact on structures. However, the aforementioned works, due to the absence of fracture mechanics modeling, cannot accurately simulate these destructive visual effects. Additionally, the partitioned approaches of these methods often encounter numerical instabilities during significant deformations.

To enable the simulation of solid fracturing, we employ the recently advanced Material Point Method (MPM) (Sulsky et al., 1995; Stomakhin et al., 2013; Zhang et al., 2016; Jiang et al., 2016a) for the solid domain. MPM is now widely used in computer graphics and computational engineering to simulate elastoplastic materials with topology changes. Utilizing meshless Lagrangian particles and Eulerian grids, MPM naturally captures material separation and contact.

The MPM also naturally simulates mixed materials (Jiang et al., 2016a; Hu et al., 2018) without requiring explicit modeling of their coupling. Nevertheless, MPM's utilization of a single grid leads to artificial adhesion artifacts at material interfaces. Moreover, representing gas with particles is computationally and memory-intensive. As a result, we confine our modeling to the solid phase with MPM and employ an Eulerian grid-based discretization for the fluid phase. Existing work has coupled MPM sediments with Eulerian incompressible fluids (Gao et al., 2018; Baumgarten et al., 2021). (Ma et al., 2009b)'s method switches from MPM to FDM during the detonation process based on the dominance of material interaction or gas pressure evolution. However, they do not address the coupling between the two phases.

In summary, most existing approaches for coupling compressible flow with deformable solids lack simultaneous support for strong coupling, solid fracture, and efficient integration of compressible flow. This motivates our work to develop a new, computationally efficient, monolithic, and fracture-supporting strong coupling scheme for destructive effects caused by shockwaves and high-speed flows.

A closely related work is the Interface Quadrature Material Point Method (IQ-MPM) by (Fang et al., 2020). IQ-MPM employs MPM particles to represent both solid and incompressible liquid phases. Strong coupling is achieved by modeling interfacial interactions with a layer of imaginary quadrature points. Their approach is computationally efficient, requiring the solution of an SPD system per time step. We extend some of their ideas from incompressible liquids to compressible flow.

The final component of our framework concerns the performance of the compressible flow solver. We follow the work of (Kwatra et al., 2009, 2010), which successfully employed the time-splitting method in the compressible flow regime, removed the strict sound speed restriction, and built an SPD pressure-only system – a system that can be integrated into the coupling phase from (Fang et al., 2020). In recent years, the semi-implicit solver in (Kwatra et al., 2009, 2010) was further improved for stability (Grétarsson et al., 2011), mass conservation (Grétarsson and Fedkiw, 2013), and subgrid resolution (Hyde and Fedkiw, 2019). These works couple gas with sediments and simple elastic bodies, also capturing basic fracture events by incorporating pressure-induced forces into a rigid body fracture tool. In contrast, our work supports arbitrarily complex fracture events of nonlinear elastic solids under extreme deformation, along with splitting and merging dynamics of plastic materials, without additional efforts beyond having an MPM solver for modeling these processes.

3.2 Method

3.2.1 Governing Equations

In accordance with standard literature, we describe the governing equations: a conservative form of the inviscid compressible Euler equations for fluids and the conservation of mass and momentum for elastoplastic solids. Introducing density ρ, velocity $\mathbf{u}$, and pressure p, the compressible flow ($\bullet^f$) region is governed by

$$\frac{\partial \rho^f}{\partial t} + \nabla \cdot (\rho^f \mathbf{u}^f) = 0, \tag{3.1}$$

$$\frac{\partial \rho^f \mathbf{u}^f}{\partial t} + \nabla \cdot (\rho^f \mathbf{u}^f \otimes \mathbf{u}^f) + \nabla p^f = 0, \tag{3.2}$$

$$\frac{\partial \rho^f E}{\partial t} + \nabla \cdot \big((\rho^f E + p^f)\mathbf{u}^f\big) = 0, \tag{3.3}$$

and the solid region ($\bullet^s$) is governed by

$$\frac{\partial \rho^s}{\partial t} + \nabla \cdot (\rho^s \mathbf{u}^s) = 0, \tag{3.4}$$

$$\rho^s \frac{\partial \mathbf{u}^s}{\partial t} + \mathbf{u}^s \cdot \nabla \mathbf{u}^s - \nabla \cdot \sigma - \rho^s \mathbf{g} = 0. \tag{3.5}$$

At the solid-gas interface, we enforce the slip velocity condition and pressure continuity:

$$(\mathbf{u}^s - \mathbf{u}^f) \cdot \mathbf{n} = 0, \tag{3.6}$$

$$p^f - p^s = 0. \tag{3.7}$$

In (3.3), E represents the fluid's enthalpy (total energy per unit volume), related to the internal energy e as

$$E = e + \frac{1}{2}\|\mathbf{u}^f\|^2 = e + \frac{1}{2}\mathbf{u}^f \cdot \mathbf{u}^f. \tag{3.8}$$

The system is closed by the equation of state (EOS) for an ideal gas (Kwatra et al., 2010)

$$p^f = (\gamma - 1)\rho^f e, \tag{3.9}$$

where we choose $\gamma = 1.4$.

In the subsequent subsections, we partition these coupled equations into advection and non-advection components. As mentioned in the introduction, we discretize the gas with an Eulerian grid and the solid with the hybrid Lagrangian/Eulerian MPM.

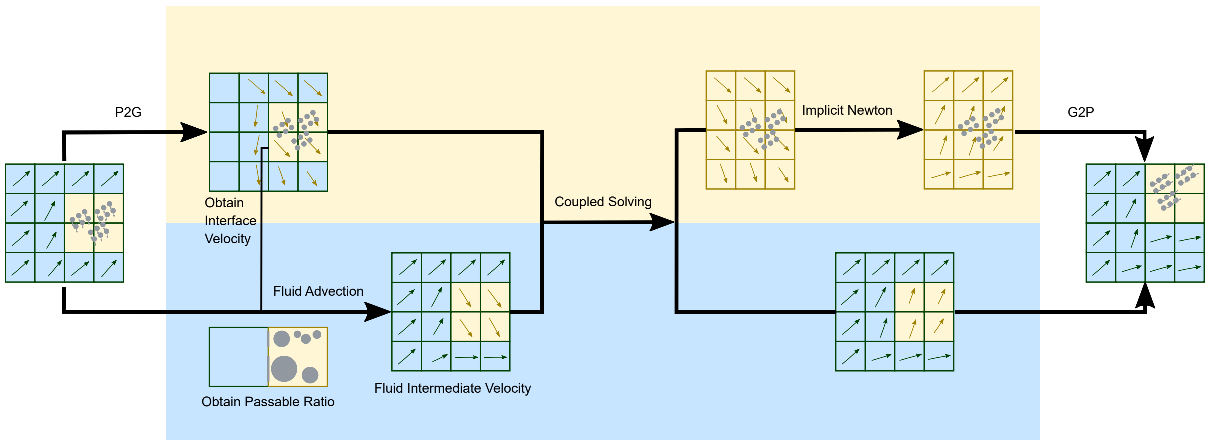


Figure 3.1: Algorithm overview for CSMPM. This schematic illustrates our method. The pipeline commences with previously advected particles transferring their physical properties to a background grid through a Particle-to-Grid (P2G) process. The grid is employed for: a) implicit integration of the solids (upper row), and b) estimating a passable ratio for more accurate flow advection (bottom row). Based on degrees-of-freedom (DOFs) on the grid, we assemble and solve a coupled velocity-pressure system to advance these quantities to the next step. Section 2.2 presents a more detailed overview of our full algorithm.

3.2.2 Pipeline Overview

We present an overview of our pipeline in Figure 3.1. Our method involves the following steps in each time step:

1. **Particle-to-Grid Transfer.** Solid particles transfer their physical properties $\mathbf{u}^s$, m^s, and J^g to the grid (Jiang et al., 2015).
2. **Moving Interface Identification.** Proper velocity and the passable ratio are computed at each cell interface between the two phases, following the technical details in Section 3.2.3.
3. **Flux-Based Advection for the Fluid.** The conserved fluid variables are updated using (3.10)–(3.12) with a second-order WENO-LLF solver.
4. **Intermediate Pressure Update.** Intermediate pressures for fluid and solid phases are computed using (3.9) and (3.17), respectively.
5. **Coupled Solve.** A coupled linear system (3.37) is constructed and solved. The solid's velocity is updated using (3.36). The fluid's conserved variables are updated using local gradient calculation, as in (Kwatra et al., 2009).

6. **MPM Grid Update.** Standard MPM-based time integration updates solid grid velocity values, incorporating elasticity and wall collisions.
7. **Grid-to-Particle Transfer.** Information from the solid grid is transferred back to solid particles. We update particles' deformation gradients and advect their positions.

3.2.3 Advection

Solid advection occurs during the grid-to-particle transfer at the end of the previous time step (Jiang et al., 2016a). At the beginning of the current step, solid particles transfer their quantities to the Eulerian grid through a standard particle-to-grid operation. Subsequently, the location and velocity of the interface between the two phases are determined.

Fluid advection involves updating the conserved fluid variables, treating them as independent variables, temporarily ignoring pressure influence. Fluid is updated to an intermediate state ($\bullet^*$) after advection. We use the finite difference method and second-order WENO-LLF flux reconstruction to solve the advection equations:

$$\rho^{f,*} = \rho^{f,n} - \Delta t \nabla \cdot (\rho^f \mathbf{u}^f), \tag{3.10}$$

$$(\rho \mathbf{u})^{f,*} = (\rho \mathbf{u})^{f,n} - \Delta t \nabla \cdot (\rho^f \mathbf{u}^f \otimes \mathbf{u}^f), \tag{3.11}$$

$$(\rho E)^{f,*} = (\rho E)^{f,n} - \Delta t \nabla \cdot \left((\rho E)^f \mathbf{u}^f\right). \tag{3.12}$$

In simulations involving extensive destruction, numerous solid fragments are scattered, resulting in intricate moving interfaces between the two phases. To handle these complexities and avoid errors, we discuss our approaches below.

Mixed Reflective and Passable Interface The particle-to-grid operation may result in cells with small solid volume fractions. Advection at the interface between such a cell and a gas cell is neither purely reflective nor purely passable. To address this, we take

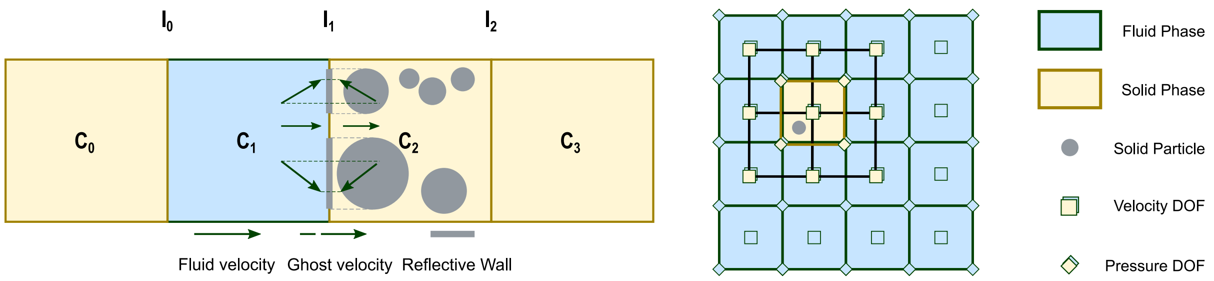


Figure 3.2: Illustration of the flux blending method and grid configuration. (Top) Illustration of the flux blending method. The overall flux at the interface is treated as a mixture of that on a passable wall and on a reflective wall. For the passable part, the ghost velocity and ghost conserved variables are extrapolated. For the reflective part, the ghost velocity is reflected, while the ghost conserved variables are extrapolated. (Bottom) Grid configuration. To avoid staggered velocity DOFs between the two phases, we adhere to B2B1 (quadratic velocity and linear pressure) shape functions for the solid phase. The grid configuration is compatible with both B2B1 and B1B0 (linear velocity and constant pressure) shape functions for the fluid phase.

three steps to blend between purely reflective and passable interfaces. First, we model each interface as a mixture of a reflective wall and a passable passage (see Figure 3 in (Hyde and Fedkiw, 2019)). The passable ratio R is determined from the cross-section of the solid's volume, treating it as a ball. Thus, we have

$$R = \min\left(2\left(\frac{V^s}{\pi}\right)^{1/2}/dx^2, 1\right) \text{ in 2D,}$$
$$R = \min\left(\pi\left(\frac{3V^s}{4\pi}\right)^{2/3}/dx^2, 1\right) \text{ in 3D,}$$

where V^s is the solid's volume (Baumgarten and Kamrin, 2019; Gao et al., 2018). Second, we extrapolate both conserved variables and velocity for the passable flux stencil and extrapolate only conserved variables, while reflecting the velocity for the reflective flux stencil (Forrer and Jeltsch, 1998; Forrer and Berger, 1999). Lastly, the two obtained fluxes are blended using the passable ratio. We illustrate this mixed interface I_1 in Figure 3.2.

Avoiding Reflection at Interfaces Except for Evaluated Flux Interface Consider an interface such as I_1 in Figure 3.2. There are scenarios where the adjacent gas cell C_1 is touching another solid cell C_0. In such cases, the ghost values of the higher-order terms in the flux stencil (those in C_0) become ambiguous. Specifically, it becomes a question of whether to treat the interface I_0 as a reflective interface. If this choice is made, the ambiguity intensifies in multi-dimensional cases due to the potential presence of other solid-gas interfaces in other dimensions (e.g., at the top face of C_0). In such scenarios, reflecting the ghost value at different interfaces leads to inconsistent results and introduces simulation artifacts. To circumvent this issue, we refrain from applying reflection at any solid-gas interfaces other than the one where we are evaluating the flux. Instead, we utilize the extrapolated value from the nearest gas cell, corresponding to a lower-order term in the stencil, for additional required higher-order terms from solid cells.

Following the advection step, we update the fluid's pressure using the equation of state (EOS) with the intermediate quantities. This approach differs from the use of semi-Lagrangian methods as employed in (Kwatra et al., 2009). The rationale behind our choice is to mitigate numerical drifting caused by non-conservative advection methods (Aanjaneya et al., 2013).

3.2.4 Post-Advection Equations

Projection is a standard technique in incompressible flow simulation (Bridson, 2015), involving the projection of intermediate post-advection velocities onto a divergence-free subspace through pressure enforcement. While the advection step in compressible flow does not explicitly involve pressure-related terms (as we use conserved variables), we treat velocity and pressure as primary states in our description of the equations they adhere to.

3.2.4.1 Fluid Mass and Momentum

Given that the mass equation lacks pressure-related terms in compressible flow, the intermediate state for ρ^f is assumed to require no further correction after advection. Thus, we have

$$\rho^{f,n+1} = \rho^{f,*}. \tag{3.13}$$

With constant density, the post-splitting momentum equation aligns with that for incompressible fluids when advancing from $(\bullet^*)$ to $(\bullet^{n+1})$:

$$\frac{\partial(\rho^f \mathbf{u}^f)}{\partial t} + \nabla p^f = \rho^{f,n+1}\frac{\partial \mathbf{u}^f}{\partial t} + \nabla p^f = 0. \tag{3.14}$$

3.2.4.2 Fluid Energy/Pressure

We convert the post-splitting energy equation into a pressure diffusion equation using the primitive form of the Euler equation and by linking pressure with internal energy through the EOS:

$$\frac{\partial p^f}{\partial t} + \rho^f c^2 \nabla \cdot \mathbf{u}^f = 0; \tag{3.15}$$

a detailed derivation is available in (Fedkiw et al., 2002).

3.2.4.3 Solid Ghost Pressure

Following (Fang et al., 2020), we enhance the hyperelastic energy density function of the solid by incorporating a quadratic volume penalization term $\Psi^g(J^g) = \frac{1}{2}\lambda^g(J^g - 1)^2$, where λ^g represents Lamé first parameter, and J^g is an independent strain measure of infinitesimal volume change ratio. In concept, this involves embedding the solid within a background ghost matrix that solely resists volume change. The overall energy density in the solid domain becomes

$$\Psi(\mathbf{F}^s, J^g) = \Psi^s(\mathbf{F}^s) + \Psi^g(J^g), \tag{3.16}$$

where $\Psi^s(\mathbf{F}^s)$ is a hyperelastic energy density function (e.g., neo-Hookean). We address the response of $\Psi^s(\mathbf{F}^s)$ during the MPM grid update step post-coupled solve. During coupling, we exclusively consider the mechanical response of $\Psi^g(J^g)$, or equivalently, the effect of its corresponding pressure (Stomakhin et al., 2014; Fang et al., 2020):

$$p^g = -\frac{\partial \Psi^g(J^g)}{\partial J^g} = -\lambda^g(J^g - 1). \tag{3.17}$$

Note that p^g is pivotal to our coupling mechanism, connecting to the solid's interface velocity and crucially ensuring the continuity of interfacial velocities. Subsequently, the influence of p^g on solid momentum adheres to

$$\rho^{s,*}\frac{\partial \mathbf{u}^s}{\partial t} + \nabla p^g = 0. \tag{3.18}$$

An evolution equation for p^g is deducible (Stomakhin et al., 2014; Gonzalez and Stuart, 2008):

$$\frac{\partial p^g}{\partial t} + \lambda^g J^g \nabla \cdot \mathbf{u}^s = 0. \tag{3.19}$$

Alternatively, we can track J^g through

$$\frac{\partial J^g}{\partial t} = (\nabla \cdot \mathbf{u}^s) J^g, \tag{3.20}$$

and update p^g utilizing (3.17). Notice the analogous form between (3.14) and (3.18), as well as between (3.15) and (3.19).

3.2.5 Summarizing Time-Discretized Equations

In summary, through operator splitting, we achieve a post-advection coupled solve for solid-fluid coupling. This coupling step occurs subsequent to solid particle-to-grid transfer and fluid advection but prior to the MPM grid update (see Figure 3.1). To be more precise, the time-discretized equations are as follows:

1. Fluid Momentum Update ($\mathbf{x} \in \Omega^f$):

$$\frac{1}{\Delta t}\rho^{f,n+1}(\mathbf{u}^{f,n+1} - \mathbf{u}^{f,*}) + \nabla p^{f,n+1} = 0; \tag{3.21}$$

2. Fluid Pressure Evolution ($\mathbf{x} \in \Omega^f$):

$$\nabla \cdot \mathbf{u}^{f,n+1} + \frac{1}{\rho^{f,n}(c^2)^n \Delta t}(p^{f,n+1} - p^{f,*}) = 0; \tag{3.22}$$

3. Solid Momentum Update ($\mathbf{x} \in \Omega^s$):

$$\frac{1}{\Delta t}\rho^{s,*}(\mathbf{u}^{s,n+1} - \mathbf{u}^{s,*}) + \nabla p^{g,n+1} = 0; \tag{3.23}$$

4. Solid Pressure Evolution ($\mathbf{x} \in \Omega^s$):

$$\nabla \cdot \mathbf{u}^{s,n+1} + \frac{1}{\lambda^g J^{g,n} \Delta t}(p^{g,n+1} - p^{g,*}) = 0. \tag{3.24}$$

In conjunction with interfacial continuity equations (3.6, 3.7) at the solid-fluid interface Γ, and Dirichlet wall boundary conditions for $\mathbf{u}^\bullet$ at $\partial\Omega^\bullet$, these equations establish a first-order system in terms of the unknown fields $\{\mathbf{u}^{f,n+1}, p^{f,n+1}, \mathbf{u}^{s,n+1}, p^{g,n+1}\}$. Further spatial discretization can be applied using the finite element method.

3.2.6 Weak Form and B-spline Based Spatial Discretization

The utilization of B-spline shape functions for finite elements has recently been demonstrated to be effective for *incompressible* flow (Fang et al., 2020; Gagniere et al., 2020). In this work, we extend the formulation to our *compressible* fluid-structure coupled system.

For ensuring continuity of elastic forces in MPM, the solid's velocity field must be $\mathcal{C}^1$ continuous (Hughes, 2012; Bardenhagen and Kober, 2004), which necessitates the use of quadratic polynomials as shape functions. The pressure, on the other hand, can be either piecewise linear or piecewise constant. (Fang et al., 2020) discussed two options for discretizing coupled incompressible fluids and solids. The first scheme employs quadratic B-splines for velocities and linear functions for pressures in both phases, resulting in a B2B1-B1-B2B1 scheme. The second scheme, aiming to reduce memory bandwidth and computational cost, employs piecewise constant pressures everywhere and piecewise linear velocities for the fluid phase, resulting in a B2B0-B0-B1B0 scheme.

In our scenario, adopting the B2B0-B0-B1B0 scheme would lead to staggered velocity nodes for the two phases, and the interface would intersect the inner volume of the fluid cell. This "cut-cell" configuration would necessitate additional complex treatments during the advection step due to our use of the flux-based solver.

To address this challenge, we adhere to the B2B1 scheme for solids and introduce offsets to the fluid coordinates, leveraging the Eulerian nature of the fluid. In most cases, the solid phase occupies only a small portion of the entire domain, making the use of B2B1 a reasonable choice without significantly impacting performance. We refer to our discretization approach as B2B1-B1/B0-B1B0 and apply it to large-scale 3D simulations, while we employ B2B1 for the fluid phase in our 1D and 2D benchmarks.

The weak form of our coupled system, as described in Section 3.2.5, can be derived by appropriately applying test functions $\mathbf{q}^{f,s}, r^{f,s}$:

$$\frac{1}{\Delta t}\int_{\Omega^f} \rho^{f,n+1}(\mathbf{u}^{f,n+1} - \mathbf{u}^{f,*}) \cdot \mathbf{q}^f_\alpha d\Omega^f + \int_{\Omega^f} \nabla p^{f,n+1} \cdot \mathbf{q}^f_\alpha d\Omega^f = 0, \tag{3.25}$$

$$\int_{\Omega^f} \nabla \cdot \mathbf{u}^{f,n+1} r^f d\Omega^f + \frac{1}{\rho^{f,n}(c^2)^n \Delta t}\int_{\Omega^f} (p^{f,n+1} - p^{f,*}) r^f d\Omega^f = 0, \tag{3.26}$$

$$\frac{1}{\Delta t}\int_{\Omega^s} \rho^{s,*}(\mathbf{u}^{s,n+1} - \mathbf{u}^{s,*}) \cdot \mathbf{q}^s_\alpha d\Omega^s + \int_{\Omega^s} \nabla p^{g,n+1} \cdot \mathbf{q}^s_\alpha d\Omega^s = 0, \tag{3.27}$$

$$\int_{\Omega^s} \nabla \cdot \mathbf{u}^{s,n+1} r^s d\Omega^s + \frac{1}{\lambda^g J^{g,n} \Delta t}\int_{\Omega^s} (p^{g,n+1} - p^{g,*}) r^s d\Omega^s = 0. \tag{3.28}$$

The weak form of the velocity conditions at the solid-fluid interface and the domain boundaries are:

$$\int_{\partial\Omega^f} \mathbf{u}^{f,n+1} \cdot \mathbf{n}^f r^f dA = \int_{\partial\Omega^f} \mathbf{u}_b r^f dA, \tag{3.29}$$

$$\int_{\Gamma} \mathbf{u}^{f,n+1} \cdot \mathbf{n}^i r^f dA = \int_{\Gamma} \mathbf{u}^{s,n+1} \cdot \mathbf{n}^i r^s dA, \tag{3.30}$$

$$\int_{\partial\Omega^s} \mathbf{u}^{s,n+1} \cdot \mathbf{n}^s r^s dA = \int_{\partial\Omega^s} \mathbf{u}_b r^s dA. \tag{3.31}$$

In these equations, $\mathbf{q}_\alpha$ collocates with the velocity nodes and is nonzero only on its $_\alpha$ component, Γ represents the solid-fluid interface, p denotes the pressure, $\mathbf{u}_b$ is the normal velocity at the slip boundaries of the solid or fluid domains, and r is the test function collocated with the pressure nodes. For B2B1, we have:

$$\mathbf{q}_{\alpha,i}(x, x_{\mathbf{u}}) = \delta_{\alpha,i} N^2(x - x_{\mathbf{u}}), \tag{3.32}$$

$$r(x, x_p) = N^1(x - x_p), \tag{3.33}$$

where N^k stands for the kth order B-spline kernel and δ represents the Kronecker delta. The discrete and compact forms of the above equations can be derived through the standard FEM assembling process, leading to the following system:

$$\begin{bmatrix} \frac{\mathbf{M}^f}{\Delta t} & \mathbf{G}^f & \mathbf{B}^f & \mathbf{H}^f & 0 & 0 & 0 \\ \mathbf{G}^{f,T} & -\frac{\mathbf{S}^{f,-1}}{\Delta t} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{B}^{f,T} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{H}^{f,T} & 0 & 0 & 0 & \mathbf{H}^{s,T} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{H}^s & \frac{\mathbf{M}^s}{\Delta t} & \mathbf{G}^s & \mathbf{B}^s \\ 0 & 0 & 0 & 0 & \mathbf{G}^{s,T} & -\frac{\mathbf{S}^{s,-1}}{\Delta t} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{B}^{s,T} & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{U}_\alpha^{f,n+1} \\ \mathbf{P}^{f,n+1} \\ \mathbf{Y}^{f,n+1} \\ \mathbf{h}^{n+1} \\ \mathbf{U}_\alpha^{s,n+1} \\ \mathbf{P}^{g,n+1} \\ \mathbf{Y}^{s,n+1} \end{bmatrix} = rhs, \tag{3.34}$$

where the vectors $\mathbf{h}$ and $\mathbf{Y^{f,s}}$ contain pressure DOFs at the solid-fluid interface and domain boundaries, respectively, due to the integration by parts of $\int_\Omega \nabla p^{n+1} \cdot \mathbf{q}_\alpha d\Omega$. The right-hand side is given by:

$$rhs = \left[\frac{\mathbf{M}^f}{\Delta t}\mathbf{U}_\alpha^{f,*}, -\frac{\mathbf{S}^{f,-1}}{\Delta t}\mathbf{P}^{f,*}, b^f, 0, \frac{\mathbf{M}^s}{\Delta t}\mathbf{U}_\alpha^{s,*}, -\frac{\mathbf{S}^{s,-1}}{\Delta t}\mathbf{P}^{s,*}, b^s\right]^T.$$

3.2.7 Building an SPD System

Equation (3.34), when considered as a KKT system, presents challenges in terms of solvability. We address this challenge by transforming it into a system with Symmetric Positive Definite (SPD) properties through mass lumping and the elimination of velocity degrees of freedom. While the mass matrix $\mathbf{M}^{f,s}$ and the stiffness matrix $\mathbf{S}^{f,s}$ are not inherently ill-conditioned, lumping makes them diagonal-scaling matrices, thus facilitating

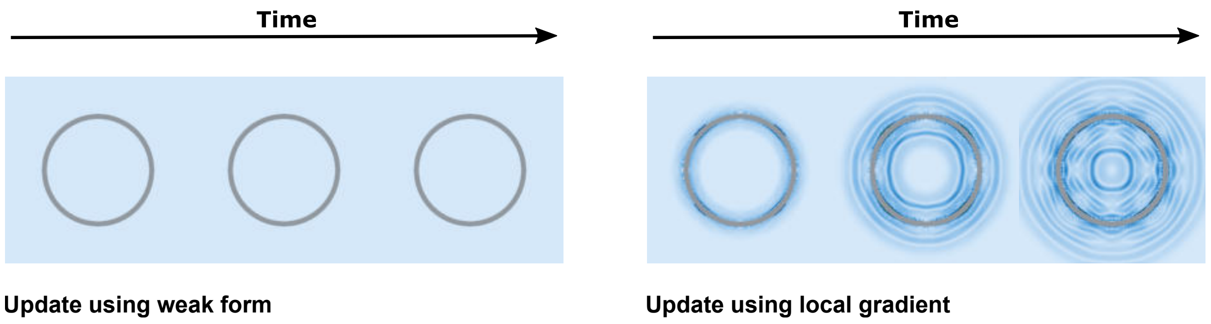


Figure 3.3: Comparison between two different methods for post-correction. A circular shell is placed in quiescent air with uniform pressure. The method from (Kwatra et al., 2009) is used to update the fluid states. For the solid, (left) adopting (3.36) yields a stable solution; (right) using the local pressure gradient leads to instability.

inversion. Velocity DOFs are eliminated through the following reorganization:

$$\mathbf{U}_\alpha^{f,n+1} = \mathbf{U}_\alpha^{f,*} - \Delta t \mathbf{M}^{f,-1}(\mathbf{G}^f \mathbf{P}^{f,n+1} + \mathbf{B}^f \mathbf{Y}^{f,n+1} + \mathbf{H}^f \mathbf{h}), \tag{3.35}$$

$$\mathbf{U}_\alpha^{s,n+1} = \mathbf{U}_\alpha^{s,*} - \Delta t \mathbf{M}^{s,-1}(\mathbf{G}^s \mathbf{P}^{s,n+1} + \mathbf{B}^s \mathbf{Y}^{s,n+1} + \mathbf{H}^s \mathbf{h}). \tag{3.36}$$

This transformation yields a pressure-only system that is both SPD and well-conditioned:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & 0 & 0 \\ \mathbf{A}_{12}^T & \mathbf{A}_{22} & \mathbf{A}_{23} & 0 & 0 \\ \mathbf{A}_{13}^T & \mathbf{A}_{23}^T & \mathbf{A}_{33} & \mathbf{A}_{34} & \mathbf{A}_{35} \\ 0 & 0 & \mathbf{A}_{34}^T & \mathbf{A}_{44} & \mathbf{A}_{45} \\ 0 & 0 & \mathbf{A}_{35}^T & \mathbf{A}_{45}^T & \mathbf{A}_{55} \end{bmatrix} \begin{bmatrix} \mathbf{P}^{f,n+1} \\ \mathbf{Y}^{f,n+1} \\ \mathbf{h}^{n+1} \\ \mathbf{Y}^{s,n+1} \\ \mathbf{P}^{s,n+1} \end{bmatrix} = rhs. \tag{3.37}$$

Here, the submatrices $\mathbf{A}_{ij}$ and the vector rhs are defined as shown previously. This pressure-only system is diagonal-dominant and can be efficiently solved using the Jacobi-preconditioned Conjugate Gradient (CG) method, typically converging in fewer than 80 iterations for our examples.

After solving the linear system, we update the solid's velocities using (3.36) and use local gradients to update the fluid's conserved variables, as described in (Kwatra et al., 2009). It is worth noting that while updating the solid's velocities with local gradients can sometimes lead to instability, adopting (3.36) tends to yield a stable solution (see

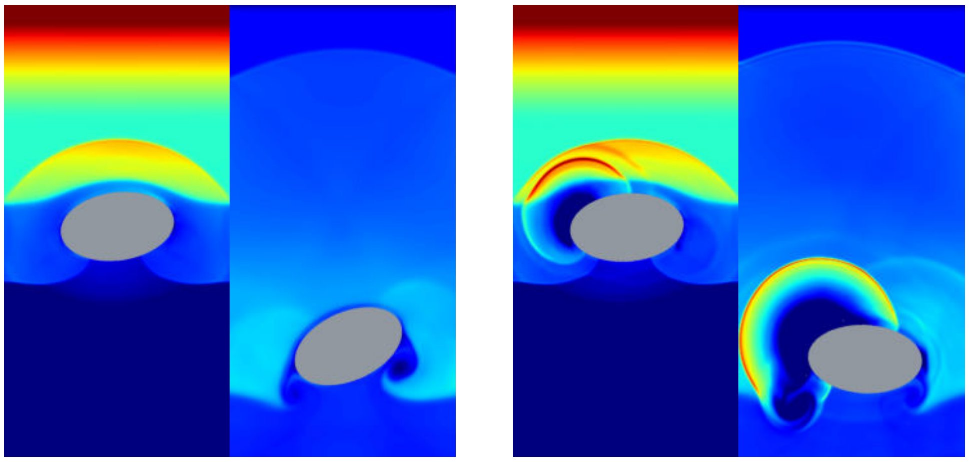


Figure 3.4: Demo: Blasting an ellipse. An ellipse with a biased mass center is blasted using two different ordering choices in our pipeline: putting the coupling step before (left) or after (right) the MPM grid update step. Placing the MPM step before the coupling step results in artificial shocks (incorrect behavior).

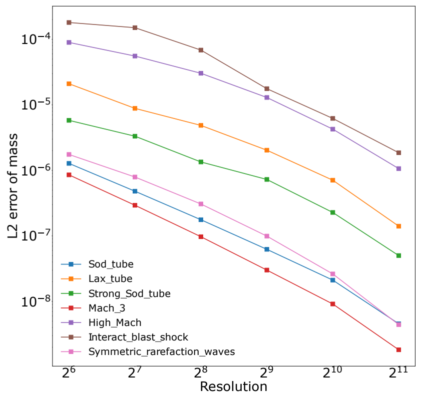
Figure 3.3 for a visual comparison).

3.2.8 MPM Step

After the coupling step, the velocities on the solid cells incorporate contributions from the fluid and need a final standard MPM grid update step to account for hyperelasticity. It is important to note that (Fang et al., 2020) adopted a different order and placed the MPM step before the coupling step. However, in our framework, placing the MPM step before the coupling step led to instabilities when strong shocks interacted with stiff materials (see Figure 3.4 for a visual comparison).

3.2.9 Inlet and Outlet Treatment

Inlet and outlet boundary conditions are modeled by extrapolating and attenuating conserved variables in ghost cells, which is similar to the approach described in (Kwatra et al., 2009). The intermediate velocities for interior cells and one layer of ghost cells are determined from their conserved variables after advection. For outlets, the intermediate transport velocity at boundary faces is interpolated and stored, while for inlets, the desired velocity values are directly stored.



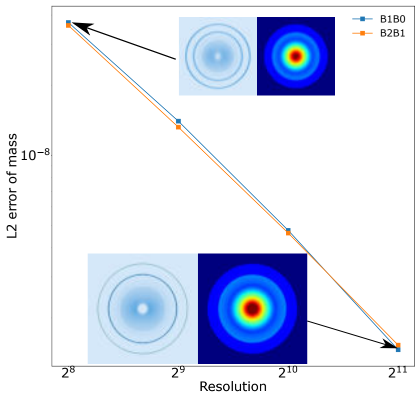


Figure 3.5: 1D and 2D numerical validation tests. (Left) L2 mass error plot of the 1D validations. (Right) L2 mass error plot of the 2D central circular shock test comparing B2B1 and B1B0 schemes.

To model inlet and outlet conditions, we:

1. Extrapolate and attenuate conserved variables in ghost cells to calculate fluxes for advection.
2. Determine intermediate velocities for interior cells and one layer of ghost cells using their conserved variables after advection. For outlets, the intermediate transport velocity at boundary faces is interpolated and stored; for inlets, the desired velocity values are directly stored.
3. Enforce velocity boundary conditions in the coupled system through contributions to boundary integrals in (3.29).

In summary, these steps allow for the accurate modeling of inlet and outlet conditions in the coupled fluid-structure simulation.

3.3 Results

3.3.1 Validations

1D validation We validate the convergence behavior of our method following the 1D cases from (Kwatra et al., 2009). High-resolution (4096 cells) results are generated using their solver, and the squared density deviation is measured as

$$e_{\Delta m} = \frac{\Sigma(\rho - \rho_b)^2}{(\Sigma\rho_b)^2}. \tag{3.38}$$

The results are plotted in Figure 3.5, demonstrating excellent convergence behavior for relatively higher speed flows, while the convergence order is 1 at lower speeds.

2D validation We employ the central circular shock test from (Kwatra et al., 2009) as a 2D validation. The results at varying resolutions (ranging from 256 to 2048) are compared against a 4096-resolution benchmark obtained by their method. This test is also used to perform an ablation study, comparing the accuracy between B2B1 and B1B0 velocity/pressure orders. The results (see Figure 3.5) indicate that both B2B1 and B1B0 schemes converge under refinement, exhibiting negligible differences in visual results. Therefore, we adopt B1B0 for our large-scale examples due to its lower computational cost.

Boundary treatment We validate our inlet and outlet boundary treatment using the classic stair flow test from computational fluid dynamics. The result in Figure 3.6 is visually plausible.

Fluid-solid interaction We validate the qualitative accuracy of our fluid-solid coupling scheme using the classic lifting cylinder test in 2D. The cylinder is sampled with uniformly distributed material points featuring a high Young's modulus. Our result is displayed in Figure 3.7. The cylinder is elevated to the top of the channel near the outlet, in agreement with the result presented in (Forrer and Berger, 1999).

3.3.2 Other Examples

We present additional examples to showcase the efficacy of our method, focusing on MPM's inherent advantage in handling large deformations and topology changes. The

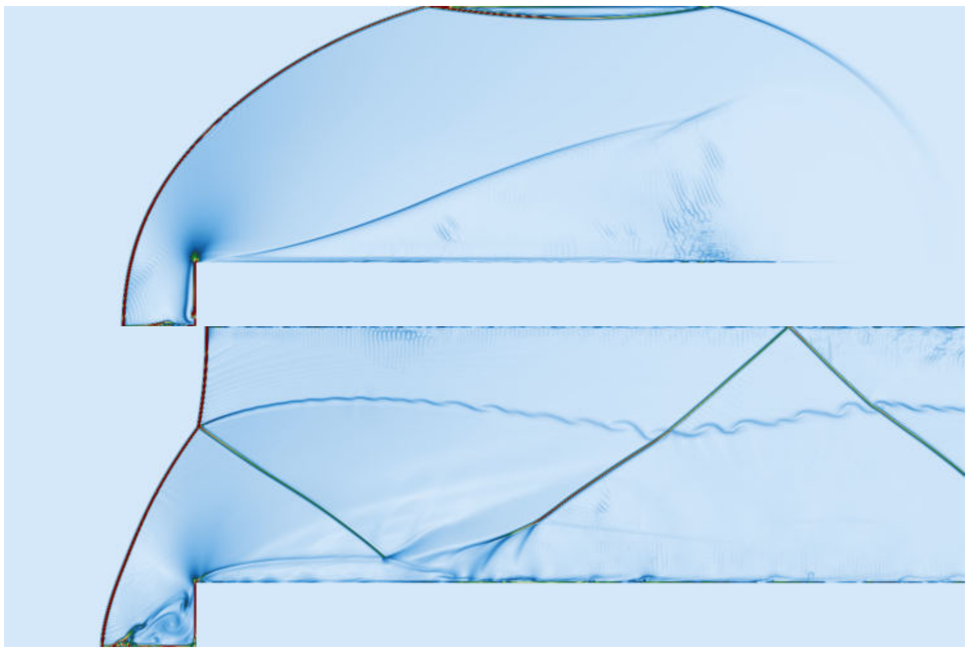


Figure 3.6: Demo: Stair flow simulation. Schlieren plot at different times during a stair flow simulation. The left boundary has a constant Mach 3 flow, the right boundary is an outlet, and the top and bottom boundaries are inviscid walls. The domain is initially filled with Mach 3 flow. The simulation accurately captures wall reflections and models inlet/outlet flow conditions correctly.

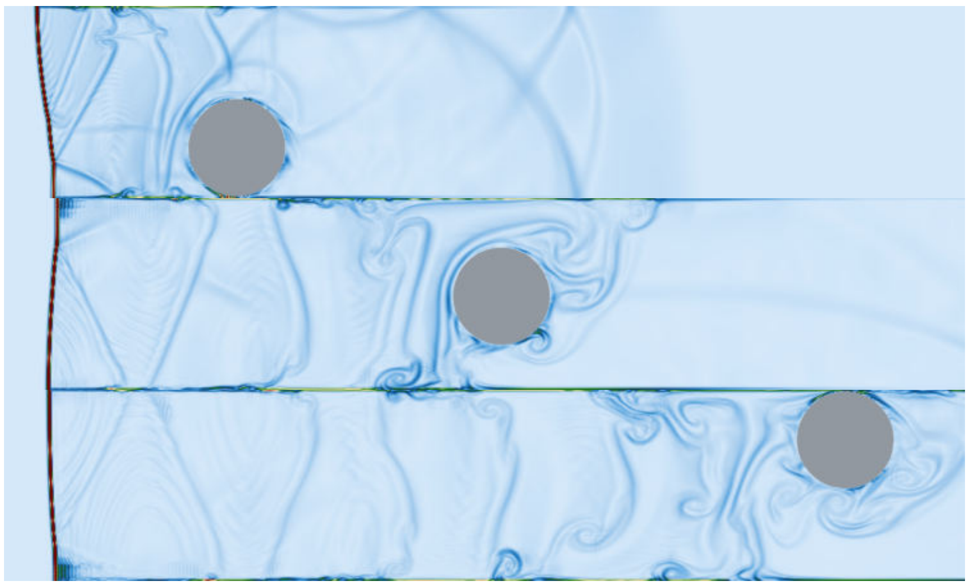


Figure 3.7: Demo: Lifting cylinder. Schlieren plot at (top to bottom) 0.18s, 0.53s, and 0.89s. The leftmost 8% of the domain features a constant Mach 3 flow; the right side of the domain serves as the outlet with a denser but slower flow; the top and bottom act as inviscid walls. Our method accurately captures the lifting of the cylinder to the top of the channel near the outlet.

Table 3.1: Simulation parameters of the CSMPM demos.

S: strong / W: weak / E: elastic / P: plastic / SE: strong elastic / WE: weak elastic / SP: strong plastic / WP: weak plastic

Example	Grid resolution	MPM particles(K)	Young's modulus	Poisson's ratio	Yield stress
(Fig. 3.8) Car driving through a land mine	128*256*256	1300	body/tire: 2e4/1.5e3	0.22	body: 600
(Fig. 3.9) Tribunny	128*64*128	600	SE/WE/iron: 1600/200/2e4	0.22	iron: 600
(Fig. 3.10) Elastic Shell Explosion 2D	1024*1024	15.8	100	0.3	N/A
(Fig. 3.11) Firing Cannon at a truck	256*128*128	450	S/W: 1e8/5e7	S/W: 0.27/0.334	S/W: 1.7e6/5.2e5
(Fig. 3.12) Firing Bullet 2D	1600*400	1.3	1e6	0.25	N/A
(Fig. 3.13) Pumpkin explosion	128*128*128	120	E/P: 1.5e3/2e4	0.25	P: 600
(Fig. 3.14) Supernova	400*80*80	900	1e6	0.25	2.6e3
(Fig. 3.15) Cylinder flow and von Kármán vortices	200*40*24	80	SE/WE/SP/WP: 1e3/2e2/1e4/5e3	0.35	SP/WP: 150/75
(Fig. 3.16) Airplane in a wind tunnel	400*60*160	98	1e8	0.25	2.6e5
(Fig. 3.17) Cactus near explosion	300*100*75	430	1e6	0.25	6e4
(Fig. 3.19) Mach Diamond and Koalas	256*128*64	174	1e3	0.22	N/A

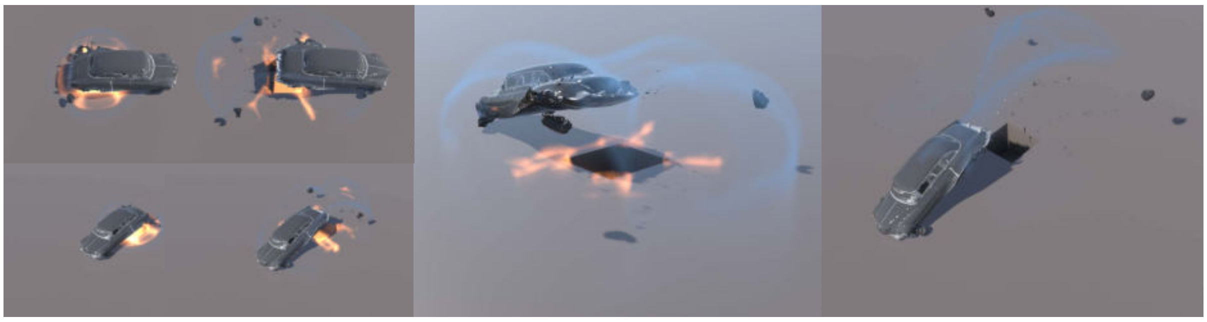


Figure 3.8: Demo: A car traverses a triggered landmine. The vehicle's distinct materials for its body and wheels result in unique interactions with the blowing exhaust. Our method naturally accounts for these effects, offering a monolithic solution to the solid-gas coupling problem.

simulation parameters are specified in Table 3.1.

Car driving through a land mine We simulate a car driving through a triggered landmine (Figure 3.8). The car's main body is made of plastic, while its wheels are constructed from lighter elastic materials. The intense explosion shatters the rear wheels and causes permanent distortion to the body.

Tribunny We simulate three bunny toys surrounding an explosion (Figure 3.9). The heavy elastic bunny and the plastic bunny sustain damage without being propelled far. Conversely, the lighter elastic bunny is fully propelled by the shockwave.

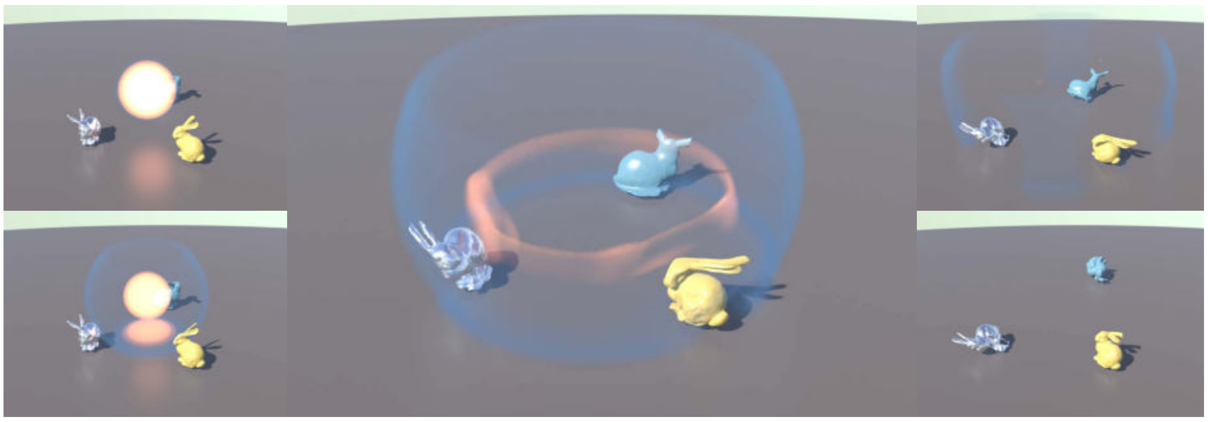


Figure 3.9: Demo: “Tribunny”; three bunny models surrounded by an explosion. The bunnies consist of iron, (yellow) rubber with high density, and (blue) rubber with low density. The iron bunny sustains permanent damage to its feet, while the heavier elastic bunny nearly regains its original shape by the end of the animation. In contrast, the lighter elastic bunny is propelled into the air, spinning during its trajectory.

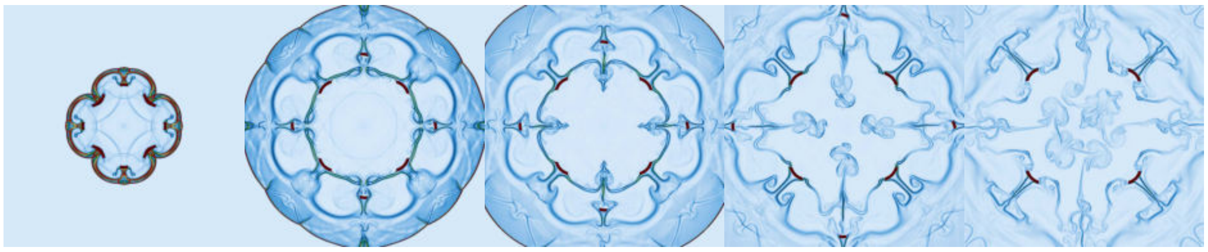


Figure 3.10: Demp: Explosive elastic shell. The left set of snapshots illustrate the primary shockwave’s expansion while the right set depicts the back-flowing exhaust gases.

Elastic shell explosion We simulate the explosion of an elastic shell (Figure 3.10). The shell is filled with highly compressed hot gas, leading to its rupture. The process results in two layers of primary shockwaves expanding outward, followed by an inward injection of gas, mixing with post-explosion exhaust. The interaction of elastic fragments with the complex flow pattern is observed.

Firing a cannon at a truck We simulate projectiles hitting toy trucks (Figure 3.11). The trucks are first damaged by the projectiles and then further deformed by shockwaves. The material parameters of each truck influence the extent of final damage.

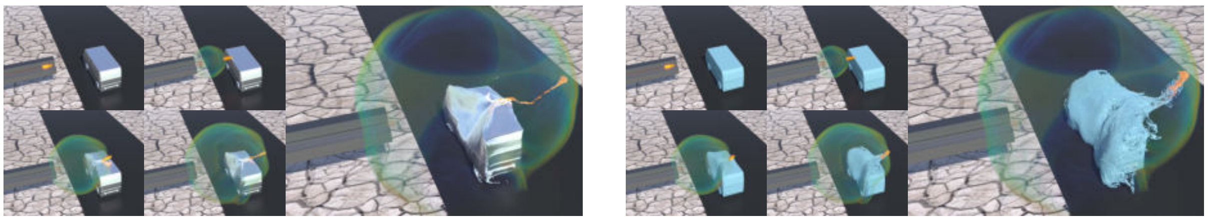


Figure 3.11: Demo: Targeting toy trucks with bullets. The trucks are constructed from distinct materials: (left, sturdy) high Young's modulus and von Mises yield stress; (right, fragile) low Young's modulus and von Mises yield stress. The robust truck experiences partial deformation due to the shock, while the weaker truck undergoes catastrophic destruction.

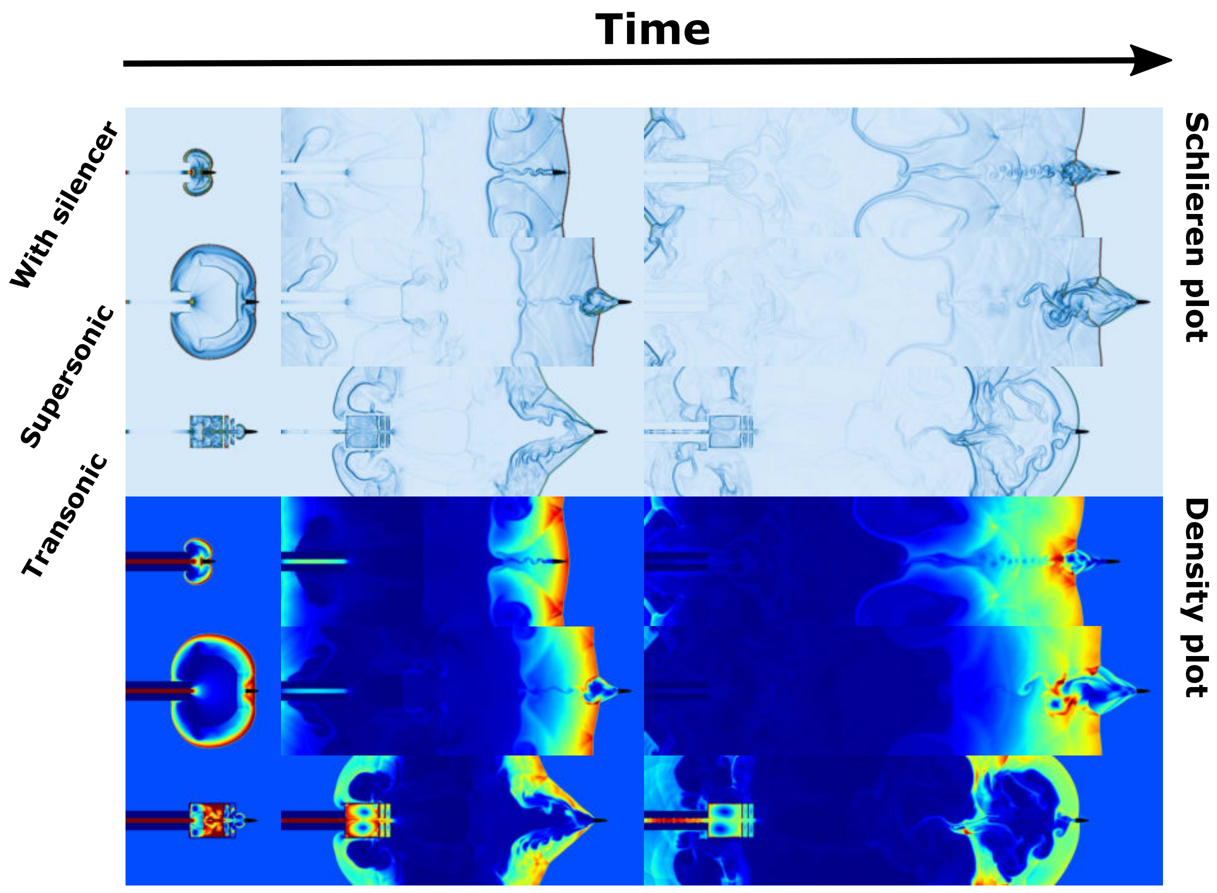


Figure 3.12: Demo: 2D Bullet simulations. Schlieren & density plots depicting the firing of heavy (left, $\rho = 1250kg/m^3$) and light (middle, $\rho = 500kg/m^3$) bullets, along with a light bullet equipped with a silencer at the chamber outlet (right, $\rho = 500kg/m^3$). Snapshots are captured at intervals of 0.028s, 0.0935s, and 0.17s, from bottom to top.

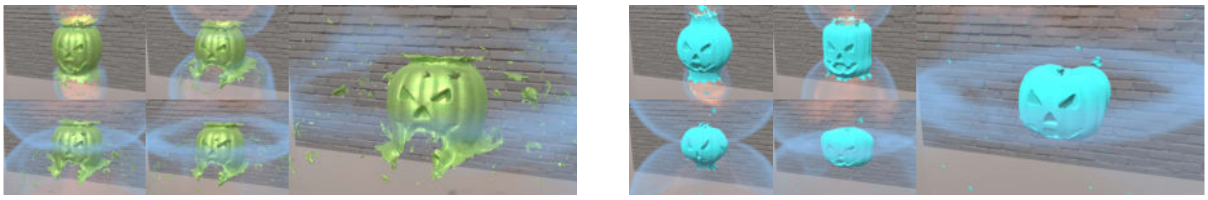


Figure 3.13: Demo: Explosion of plastic and elastic pumpkins. The high internal pressure causes the pumpkin shells to fracture, propelling their fragments away. The plastic pumpkin's deformation remains permanent; its fragments remain stretched and thin. The elastic pumpkin exhibits jiggling motion in the air, reacting to the backflow.

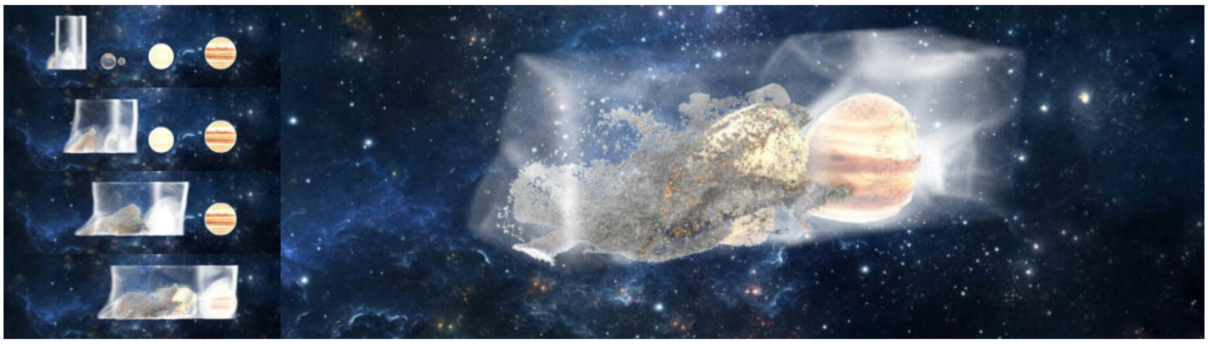


Figure 3.14: Demo: Supernova. A supernova destroys multiple planets, producing a massive amount of fragments in a mini solar system. This example contains numerous complex solid-fluid interfaces with a wide range of solid volume fractions. Our partially passable interface treatment robustly handles them.

Firing bullets We use high-temperature, high-pressure gas to propel bullets under varying conditions (Figure 3.12). A heavy bullet reaches a transonic state and breaks the wavefront towards the end of the simulation, generating uniform vortices behind its tail. The lighter bullet reaches a supersonic state, breaking the wavefront early and producing chaotic vortices. In another test, a silencer is added to the firing chamber to observe its successful suppression of the primary shock. The bullet's speed is slightly reduced, allowing it to pass the slower shockwave and create a secondary shock.

Pumpkin explosion We simulate the explosions of a plastic pumpkin and an elastic pumpkin by introducing ellipsoidal high-pressure gas inside them (Figure 3.13). Both pumpkins are extensively fragmented. The plastic pumpkin's deformation is permanent, while the elastic pumpkin vibrates vividly in the backflow.

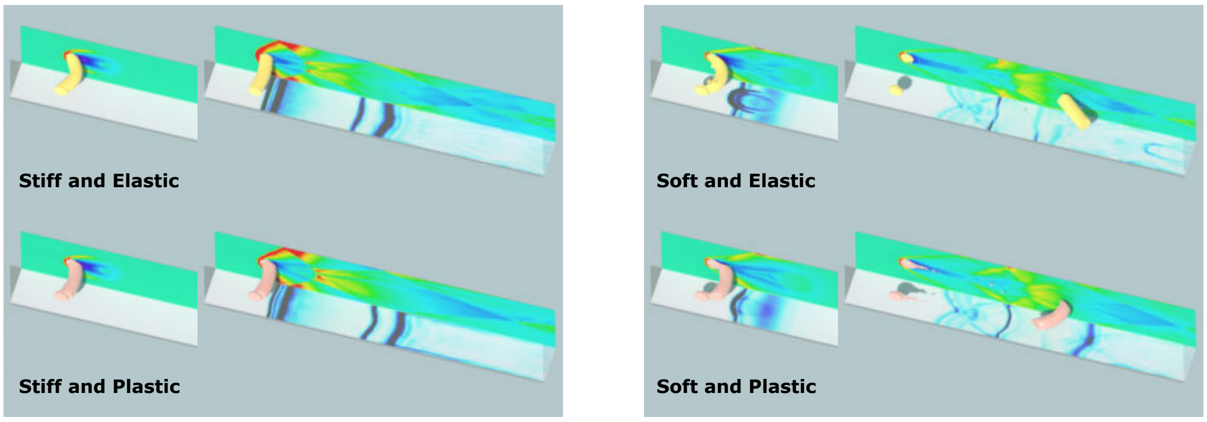


Figure 3.15: Demo: Flow through cylinders. Schlieren (bottom) and density (side) plots for Mach 3 flow through various cylinders: (top left) stiff elastic, (bottom left) stiff plastic, (top right) soft elastic, and (bottom right) soft plastic. The stiff elastic cylinder initially deforms significantly near its center and then rebounds to exhibit a more uniform deformation. The stiff plastic cylinder experiences permanent deformation near its ends. The soft cylinders quickly rupture from the walls. Elastic fragments regain their shapes, while plastic fragments remain permanently bent.

Supernova We use a planar wave to simulate a distant supernova explosion, disrupting a mini solar system (Figure 3.14). Due to the plastic nature of the solids, they are easily disintegrated.

Cylinder flow and von Kármán vortices We simulate interactions between Mach 3 flow and cylinders composed of different elastoplastic materials (Figure 3.15). Varying Young's modulus and enabling/disabling plasticity reveals intricate fluid-solid coupling behavior and distinct von Kármán vortex patterns.

Airplane in a wind tunnel We place a toy airplane in a wind tunnel and gradually increase flow speed until fractures occur (Figure 3.16). Loss of balance causes the plane to spin within the tunnel. The airplane undergoes severe deformation and fractures during dynamic interaction with supersonic flow.

Cactus near explosion We simulate the interaction between a cactus forest and a supersonic wind blow caused by a distant explosion (Figure 3.17). The cactus plants are severed near their roots as the shockwave approaches.

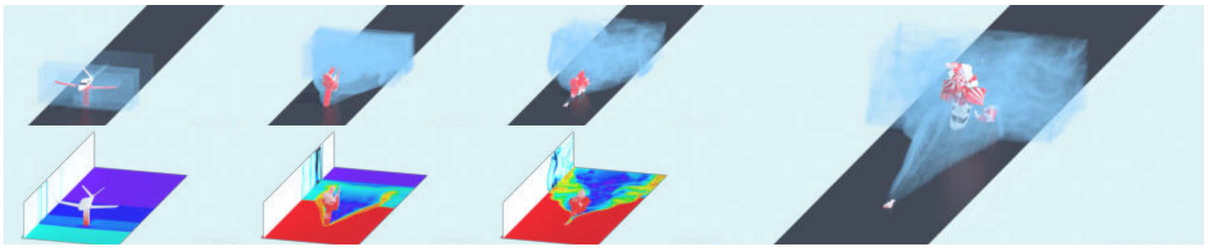


Figure 3.16: Demo: Airplane in a wind tunnel. Visualization of a slice of (bottom) flow velocity and (side) schlieren, with MPM particles color-coded by their deviatoric stress norms. The wings of the airplane experience significant lift-induced deformation, followed by deformation of the body. Subsequently, the airplane undergoes fragmentation, with a robust flow emanating from its small residual portion.

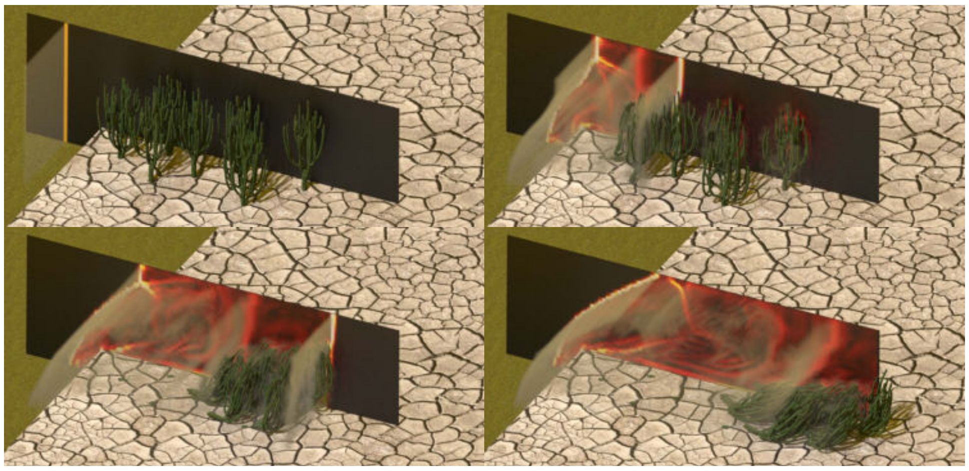


Figure 3.17: Demo: Cactus forest. The cactus plants are destroyed by high-speed airflow from an explosion. The Mach 9 wavefront is initially reflected by the stairstep between the highland and the desert. It then hits the forest and breaks the plants near their roots.

Mach Diamond and Koalas Mach diamonds form at the exhaust of a slightly over-expanded jet with an outlet pressure slightly lower than ambient. Elastic koala toys with varying densities are placed above air jets in a 3D simulation (Figure 3.19). The lightest koala is easily propelled upward, the middle one maintains balance under the jet's influence and gravity, and the heaviest koala disrupts the jet's wavefront.

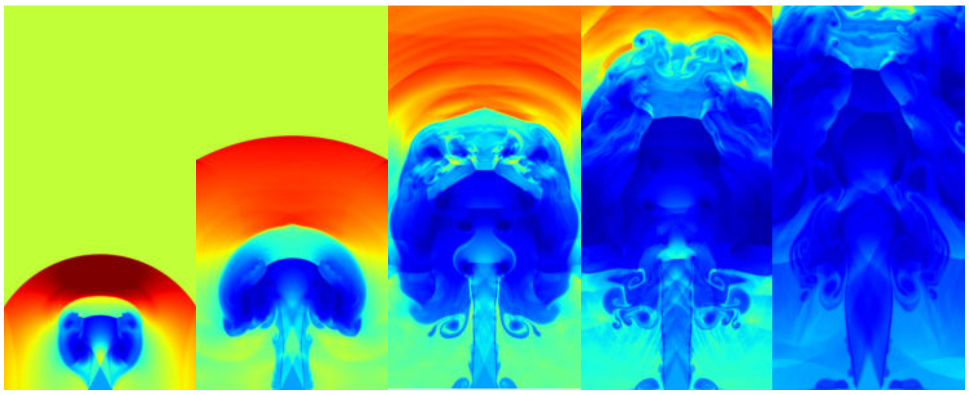


Figure 3.18: Demo: Mach diamond. Density plot of a Mach diamond in 2D at 0.22s, 0.44s, 0.85s, 1.11s, and 1.39s. The central 1/8 of the bottom serves as the inlet with Mach 3 and 50% of the ambient pressure.

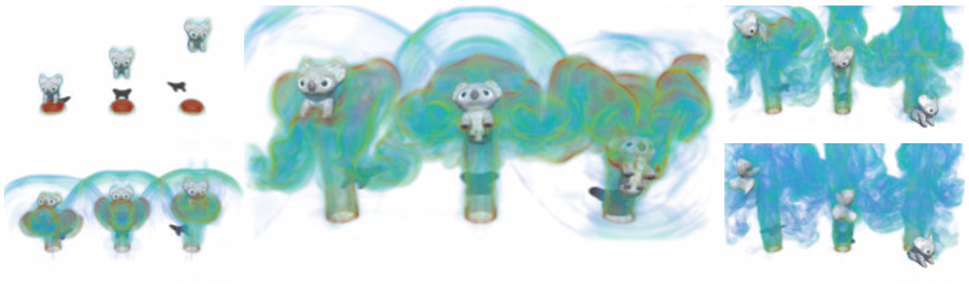


Figure 3.19: Demo: Three koala toys hit by Mach diamond jets. The inlet setting mimics the 2D version in Figure 3.18. The density ratio (from left to right) of the koalas is 1:4:9. The lightest koala is easily propelled upward; the middle one maintains balance under the effects of the pumping jet and gravity; the heaviest koala easily disrupts the jet's wavefront.

CHAPTER 4

Bi-Stride Multi-Scale Graph Neural Network for Mesh-Based Physical Simulation

Learning physical simulations on large-scale meshes using flat Graph Neural Networks (GNNs) and stacked Message Passings (MPs) presents challenges due to scaling complexity with respect to the number of nodes and issues related to over-smoothing. There has been a growing interest in introducing *multi-scale* structures to GNNs for physical simulation. However, current state-of-the-art methods are limited by their dependence on labor-intensive manual creation of coarser meshes or constructing coarser levels based on spatial proximity, which can introduce incorrect edges across geometry boundaries. Drawing inspiration from bipartite graph determination, we propose a novel pooling strategy, referred to as *bi-stride*, to address the aforementioned limitations. The bi-stride approach involves pooling nodes on alternate frontiers of the breadth-first search (BFS), eliminating the need for manual creation of coarser meshes and mitigating the issue of incorrect edges due to spatial proximity. Moreover, it facilitates a one-Message Passing scheme per level and employs non-parametrized pooling and unpooling through interpolations, reminiscent of U-Net architectures, resulting in significant reduction of computational costs. Experimental results demonstrate that our proposed framework, *BSMS-GNN*, outperforms existing methods in terms of accuracy and computational efficiency in representative physical simulations.

4.1 Related Work

GNNs for Physics-Based Simulation The application of Graph Neural Networks (GNNs) to physics-based simulation was initially explored in the context of deformable solids and fluids, both represented by particles (Sanchez-Gonzalez et al., 2018). A notable milestone in this field is the work by Pfaff et al., known as MeshGraphNets (Pfaff et al., 2020), which introduced a general framework for learning mesh-based simulations. Subsequently, various extensions of MeshGraphNets have been proposed. For instance, GNNs have been combined with Physics-Informed Neural Networks (PINNs) (Gao et al., 2022), long-term predictions have been achieved by combining Graph AutoEncoders (GAEs) with Transformers (Han et al., 2022), steady-states have been directly predicted using multi-layer readouts (Harsch and Riedelbauch, 2021), and finer-level simulation acceleration has been achieved by utilizing up-sampled coarser results inferred by GNNs (Belbute-Peres et al., 2020).

Multi-Scale GNNs Multi-Scale Graph Neural Networks (MS-GNNs) have found applications beyond physics in various graph-related tasks (Wu et al., 2020; Mesquita et al., 2020; Zhang et al., 2019). The concept of MS-GNNs was introduced by Gao et al. in the form of *GraphUNet* (Gao and Ji, 2019), which incorporates a UNet structure into GNNs along with a trainable scoring module for pooling. Additionally, a 2^{nd}-powered adjacency enhancement is utilized to preserve connectivity. In the context of physics-based simulations, several works have explored the application of MS-GNNs, including two-level and multi-level GNNs (Fortunato et al., 2022; Liu et al., 2021), often relying on manual construction of coarse meshes. Other approaches, such as MS-GNN-Grid (Lino et al., 2021, 2022a), leverage spatial proximity to generate multi-level structures. Li et al. (Li et al., 2020b) adopt a multi-level matrix factorization technique for generating kernels at coarser levels. Guillard's coarsening algorithm is employed by Lino et al. (Lino et al., 2022b) for building coarse-level meshes, primarily for 2D triangle elements. Notably, there are representative works that diverge from the traditional mesh-based representation and instead establish connections and hierarchies on point clouds. Examples include

GNS (Sanchez-Gonzalez et al., 2020), *PointNet* (Qi et al., 2017a), *PointNet++* (Qi et al., 2017b), and *GeodesicConv* (Masci et al., 2015).

4.2 Multi-Scale Building as Preprocessing

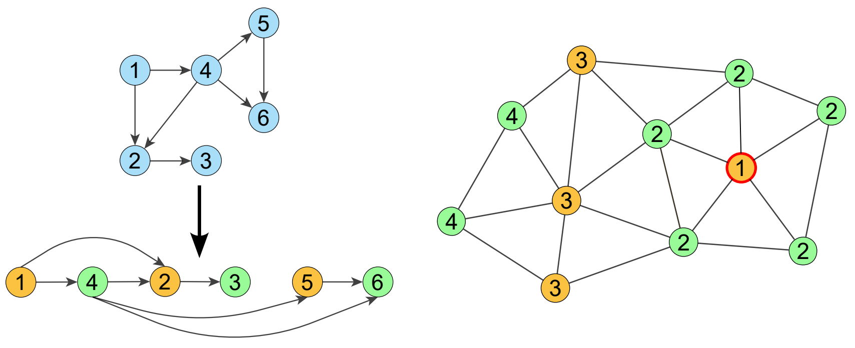
We first introduce the bi-stride pooling strategy, multi-scale building, and the corresponding data preprocessing. It is important to note that all algorithms and preprocessing steps in this section are deterministic and performed in a single pass.

4.2.1 Motivations

As depicted in Figure 1.1, pooling strategies can be categorized into two groups: those utilizing spatial proximity (Lino et al., 2021, 2022a) and those relying solely on graph information (Gao and Ji, 2019; Li et al., 2020b). When dealing with complex geometries, it is advisable to avoid spatial proximity unless necessary for simulation cases like contact or interface interactions. An example of an algorithm relying solely on graph information is GraphUNets(Gao and Ji, 2019); however, it has a significant drawback. Even with the enhancement of the adjacency matrix, as shown in Figure 1.1(a), connectivity can be lost and partitions can be introduced.

To provide a clearer illustration, we introduce the concept of K-th order adjacency enhancement, denoted as $\boldsymbol{A} \leftarrow \boldsymbol{A}^K$. In a geometric sense, $\boldsymbol{A}(i,j) = 1$ signifies the existence of an edge between nodes i and j, while $\boldsymbol{A}^K(i,j) = 1$ implies that node j is connected to node i within a maximum of K hops. We define a K-th order outlier set $\mathcal{O}_K$, where nodes in $\mathcal{O}_K$ are not connected to any pooled nodes even after K-th order adjacency enhancement ($\boldsymbol{A}^K(i,j) = 0, \forall i \in \mathcal{I}, \forall j \in \mathcal{O}_K$). We define a pooling strategy P as K-th order connection conservative (K-CC) if $\mathcal{O}_K$ is empty. Empirically, a larger K is detrimental to distinguishing node features, and the most preferred value is naturally $K = 2$.

With this rationale, (Gao and Ji, 2019) employs the smallest second-order enhancement



(a) Bi-partition of a DAG (b) Bi-stride of a mesh

Figure 4.1: Similarity between Bi-partition and Bi-stride. The number in the circle indicates the depth of the frontiers of topological sorting or BFS. The red-bounded circle (number 1) represents the starting node, i.e., the seed.

to preserve connectivity. However, there is no theoretical guarantee that a learnable pooling module is consistently 2-CC for any graph. These limitations motivate us to create a consistent 2-CC pooling strategy, as discussed in Section 4.2.2.

4.2.2 Bi-Stride Pooling and Adjacency Enhancement

Our initial inspiration for bi-stride pooling comes from the bi-partition determination algorithm (Asratian et al., 1998) in a directed acyclic graph (DAG). As shown in Figure 4.1(a), pooling on every other depth (yellow and green) after topological sorting creates a bi-partition where edges reside between two partitions, ensuring 2-CC when pooling either partition.

To extend this idea to non-bipartite meshes, we employ breadth-first search (BFS) to compute geodesic distances from a seed to all other nodes and perform stride and pool operations on every other BFS frontier (*bi-stride*), as shown in Figure 4.1(b). This pooling strategy is 2-CC by construction and conserves direct connections between pooled and unpooled nodes. Thus, we avoid relying on spatial proximity, handle complex cases like cross-boundary connections, and employ the smallest adjacency enhancement.

Seeding Heuristics We propose that seeding should be balanced to a certain degree. For training datasets, we adopt two deterministic seeding heuristics: 1) CloseCenter

Algorithm 2 CloseCenter: Seeding by minimum distance to the center of the cluster.

```
1:      Input: Unweighted, bi-directional graph $\mathcal{G} = (N, E)$; Node positions, $X$ ▷ List of seeds in each cluster, $L_s$
2:      $L_c \leftarrow$ DetermineCluster($\mathcal{G}$)
3:      $L_s \leftarrow \emptyset$
4:  for idx in $L_c$ do
5:          $\bar{X} \leftarrow \text{average}(X[\text{idx}], \text{dim} = 0)$
6:          $\Delta X \leftarrow X - \bar{X}$
7:          $D \leftarrow ||\Delta X||_2$
8:          $s \leftarrow \text{idx}[\text{argmin}(D)]$
9:          $L_s$.append($s$)
10: end for
11:     Output: $L_s$
```

Algorithm 3 MinAve: Seeding by minimum average geodesic distance to neighbors.

```
1:      Input: Unweighted, bi-directional graph $\mathcal{G} = (N, E)$    ▷ List of seeds in each cluster, $L_s$
2:      $L_c \leftarrow$ DetermineCluster($\mathcal{G}$)
3:      $L_s \leftarrow \emptyset$
4:      $D \leftarrow \{\text{BFS}(s) \text{ for } s \text{ in } N\}$
5:  for idx in $L_c$ do
6:          $D_c \leftarrow D[\text{idx}, \text{idx}]$
7:          $\bar{D}_c \leftarrow \text{average}(D_c, \text{dim} = 1)$
8:          $s \leftarrow \text{idx}[\text{argmin}(\bar{D}_c)]$
9:          $L_s$.append($s$)
10: end for
11:     Output: $L_s$
```

(Algorithm 2), which selects nodes closest to the center of a cluster for InflatingFont, and 2) MinAve (Algorithm 3), which chooses nodes with the minimum average distance for other cases. We preprocess multi-level building in a single pass. During online inference, if an unseen geometry arises, the less expensive heuristic CloseCenter can be used.

We elaborate on the two seeding heuristics mentioned above. MinAve has a time complexity of $\mathcal{O}(N^2)$ as BFS is performed for every node to find the one with the minimum average distance to neighbors. In our experiments, MinAve's quadratic cost is acceptable for all cases except InflatingFont.

For InflatingFont, where the largest mesh has around 47K nodes, the preprocessing time with MinAve becomes intolerable. In this case, we switch to CloseCenter with linear

Algorithm 4 DetermineCluster.

```
1:        Input: Unweighted, Bi-directional graph, $\mathcal{G} = (N, E)$ ▷ $R$ stands for remaining nodes that are not inside any cluster
2:        $R \leftarrow N$
3:        $L_c \leftarrow \emptyset$
4: while $R \neq \emptyset$ do
5:        $s \leftarrow R$.pop()
6:     if $|R| = 0$ then
7:            $L_c$.append($\{s\}$)
8:     else
9:            $D \leftarrow$ BFS($s$)
10:           $C \leftarrow \emptyset$
11:           $R^* \leftarrow \emptyset$
12:        for $n$ in $R$ do
13:            if $D[n] = \infty$ then
14:                $R^*$.append($n$)
15:            else
16:                $C$.append($n$)
17:            end if
18:        end for
19:           $L_c$.append($C$)
20:           $R \leftarrow R^*$
21:     end if
22: end while
23:       Output: $L_c$
```

complexity.

For both heuristics, we search for seeds in a per-cluster fashion to avoid information from other clusters that could influence the search result. For example, when determining the center of an isolated part of the input geometry, the positions of nodes from other clusters could interfere with this process. The process of determining clusters in a given graph is elaborated in Algorithm 4.

Contact Edges For problems involving contacts, such as plates (Figure 4.4(c)) and fonts (Figure 4.4(d)), the finest-level contact edges $\boldsymbol{A}^C$ are dynamically built based on spatial proximity between nodes. It is important to note that edge construction through spatial proximity is not applicable to the internal elastic mechanics, where the edge is defined solely by the mesh. Proper handling of contact edge enhancement is crucial for multi-scale GNNs, a topic that has not been previously addressed in the literature. At any level l, given adjacent matrices $\boldsymbol{A}_l$ obtained from the input mesh and the enhancement rule, as well as the contact edge $\boldsymbol{A}_l^C$ at that level, we initially perform bi-stride pooling to select nodes $\mathcal{I}$, and then enhance $\boldsymbol{A}_{l+1}$ and $\boldsymbol{A}_{l+1}^C$ using the following rule, where $[\mathcal{I}, \mathcal{I}]$ denotes striding on the matrix rows and columns:

$$\begin{aligned} \boldsymbol{A}'_{l+1} &\leftarrow \boldsymbol{A}_l \boldsymbol{A}_l, \quad \boldsymbol{A}_{l+1} \leftarrow \boldsymbol{A}'_{l+1}[\mathcal{I}, \mathcal{I}], \\ \boldsymbol{A}'^C_{l+1} &\leftarrow \boldsymbol{A}_l \boldsymbol{A}_l^C \boldsymbol{A}_l, \quad \boldsymbol{A}^C_{l+1} \leftarrow \boldsymbol{A}'^C_{l+1}[\mathcal{I}, \mathcal{I}]. \end{aligned} \tag{4.1}$$

The enhancement of contact edges can be geometrically interpreted: a contact edge (i, j) should exist if node j is reachable from node i in two hops, with at least one of those hops being a contact edge at the finer level.

Proof of Conservation of Contact Edges: With Bi-stride pooling, our pooling operation conserves all contact edges under the enhancement defined in (4.1). We assume an undirected and unweighted graph, leading to an adjacent matrix that is boolean.

Formally, for any contact edge (i, j) at level l (i.e., $\boldsymbol{A}_l^C[i, j] = 1$) and a Bi-stride pooling P that pools nodes $\mathcal{I}$, there exists a contact edge (i', j') that remains at the coarser level (i.e., $\boldsymbol{A}'^C_{l+1}[i', j'] = 1$, where $i', j' \in \mathcal{I}$), and i/i' and j/j' are connected (i.e., $\boldsymbol{A}_l[i, i'] = \boldsymbol{A}_l[j, j'] = 1$). Four scenarios involving the pooling nodes $\mathcal{I}$ and the contact edge nodes i and j can be considered, where the assertion holds in each case:

1. Both i and j are pooled, i.e., $i, j \in \mathcal{I}$. It is evident that $\boldsymbol{A}'^C_{l+1}[i', j'] = 1$ by setting $i' = i$ and $j' = j$.
2. Only i is pooled, with $i \in \mathcal{I}$ and $j \notin \mathcal{I}$. Through Bi-stride pooling, j can either be the seed at level 0 (Bi-stride can select either even or odd levels), directly connecting to all nodes at level 1, or j must have at least one direct connection from the previous level. Let j' be a neighbor of j in the adjacent level that is pooled, i.e., $\boldsymbol{A}_l[j, j'] = 1$ and $j' \in \mathcal{I}$. Then $\boldsymbol{A}_l^C \boldsymbol{A}_l[i, j'] \geq \boldsymbol{A}_l^C[i, j] * \boldsymbol{A}_l[j, j'] = 1$, and $\boldsymbol{A}_l(\boldsymbol{A}_l^C \boldsymbol{A}_l)[i, j'] \geq \boldsymbol{A}_l[i, i] * (\boldsymbol{A}_l^C \boldsymbol{A}_l)[i, j'] = 1$. Set $i' = i$, yielding $\boldsymbol{A}'^C_{l+1}[i', j'] = 1$.

3. Only j is pooled, with $i \notin \mathcal{I}$ and $j \in \mathcal{I}$. Similarly, find at least one i' such that $\boldsymbol{A}_l[i', i] = 1$ and $i' \in \mathcal{I}$. Then $\boldsymbol{A}_l\boldsymbol{A}_l^C[i', j] \geq \boldsymbol{A}_l[i', i] * \boldsymbol{A}_l^C[i, j] = 1$, and $(\boldsymbol{A}_l\boldsymbol{A}_l^C)\boldsymbol{A}_l[i', j] \geq (\boldsymbol{A}_l\boldsymbol{A}_l^C)[i', j] * \boldsymbol{A}_l[j, j] = 1$. Set $j' = j$, leading to $\boldsymbol{A'}_{l+1}^C[i', j'] = 1$.
4. Neither i nor j are pooled, i.e., $i, j \notin \mathcal{I}$. Select direct pooled neighbors for i and j, respectively, such that $\boldsymbol{A}_l[i', i] = \boldsymbol{A}_l[j, j'] = 1$, where $i', j' \in \mathcal{I}$. Then $\boldsymbol{A}_l\boldsymbol{A}_l^C[i', j] \geq \boldsymbol{A}_l[i', i] * \boldsymbol{A}_l^C[i, j] = 1$, and $(\boldsymbol{A}_l\boldsymbol{A}_l^C)\boldsymbol{A}_l[i', j'] \geq (\boldsymbol{A}_l\boldsymbol{A}_l^C)[i', j] * \boldsymbol{A}_l[j, j'] = 1$.

4.3 Bi-Stride Multi-Scale BSMS-GNN

Here, we formally introduce BSMS-GNN, a hierarchical Graph Neural Network (GNN) in which the multi-level structure is determined by the input mesh and the preprocessing discussed in Section 4.2.

4.3.1 Definitions

Figure 4.2 illustrates the overall structure of BSMS-GNN. We consider the evolution of a physics-based system discretized on a mesh, which is transformed into a bi-directional graph $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$. Here, with subscript 1, $\mathcal{V}_1$, and $\mathcal{E}_1$ denote the nodal fields and edges at the finest level (the input mesh), respectively. Specifically, for edges, we define $\mathcal{E}_1 = \{\mathcal{E}_1^1, \cdots, \mathcal{E}_1^S\}$, where $\mathcal{E}_1^1$ represents the edge set directly copied from the input mesh, and $\{\mathcal{E}_1^k|_{k=2}^S\}$ correspond to optional problem-dependent edge sets. For instance, both the DeformingPlate (Figure 4.4(c)) and InflatingFont (Figure 4.4(d)) benchmarks have a contact edge set $\mathcal{E}_1^2$ for colliding vertices. We use $\{p, q\}$ to denote stacked vectors of $\{p_i, q_i\}$ for all nodes $i \in \mathcal{V}_1$, representing the input and output nodal fields, respectively. Given an input field p^j at time t_j, one pass of BSMS-GNN returns the output field q^{j+1} at time $t_{j+1} = t_j + \Delta t$, where Δt is a fixed time step size. The output q may include more physical fields than the input p and should be capable of deriving the input for the subsequent pass. The term "rollout" refers to the iterative application of BSMS-GNN from the initial state $p^0 \to q^1 \to p^1 \to \cdots \to q^n$, producing the temporal sequence output $\{q^1, q^2, \cdots, q^n\}$ within the time range $(t_0, t_0 + n\Delta t]$, with n being the

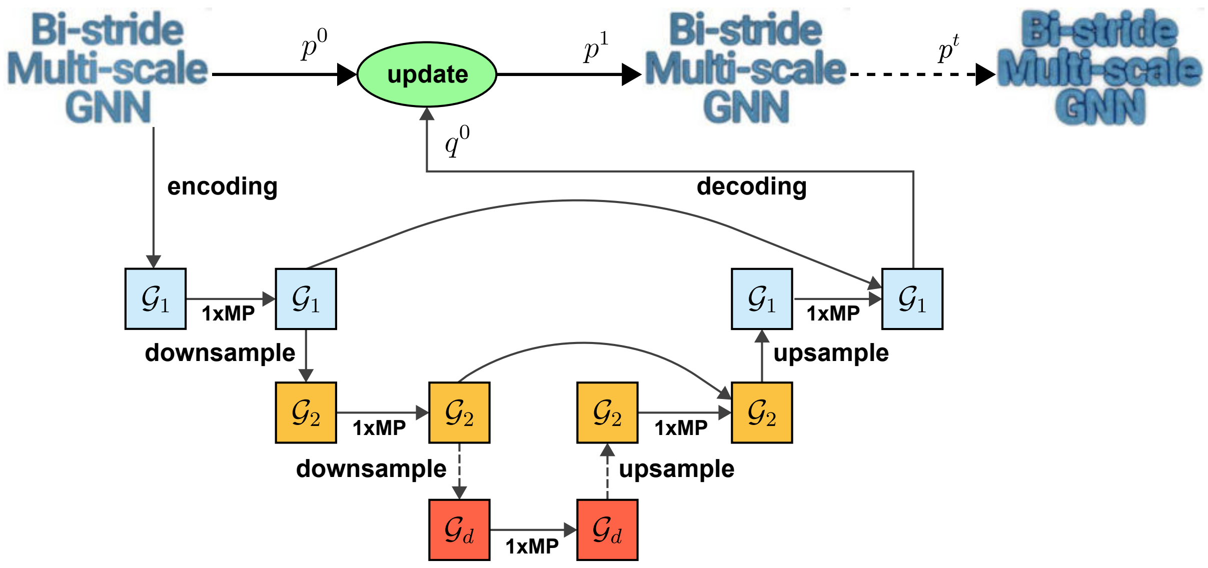


Figure 4.2: The BSMS-GNN pipeline. Trained with one-step supervision. $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_d$ represent graphs at different levels (fine to coarse). The encoder/decoder only connects the input/output fields with the latent fields at $\mathcal{G}_1$. Latent nodal fields are updated by one message passing operation at each level. Bi-stride pooling selects pooled nodes for the adjacent coarser level, and the transition occurs in a non-parameterized manner.

total number of evaluations.

Message Passing In general, we adhere to the *encode-process-decode* framework in MeshGraphNets, where the encoder and decoder are only present at the top level $\mathcal{G}_1$, mapping the nodal input p and output q to/from the latent feature $\boldsymbol{v}$, respectively. Unlike (Fortunato et al., 2022), we find that a single Message Passing (MP) operation per level is adequate for all experiments. Instead of separately encoding the edge offsets $\Delta\boldsymbol{x}_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j$, we simply append this information to the stacked sender/receiver latent vectors as input for calculating edge flow. For a problem involving S edge sets, an MP pass at level l is formulated as follows:

$$\begin{aligned} \boldsymbol{e}^s_{l,ij} &\leftarrow \mathrm{f}^s_l\left(\Delta\boldsymbol{x}_{l,ij}, \boldsymbol{v}_{l,i}, \boldsymbol{v}_{l,j}\right), \quad s = 1, \cdots, S, \\ \boldsymbol{v}'_{l,i} &\leftarrow \boldsymbol{v}_{l,i} + \mathrm{f}^V_l\left(\boldsymbol{v}_{l,i}, \sum_j \boldsymbol{e}^1_{l,ij}, \cdots, \sum_j \boldsymbol{e}^S_{l,ij}\right), \end{aligned} \tag{4.2}$$

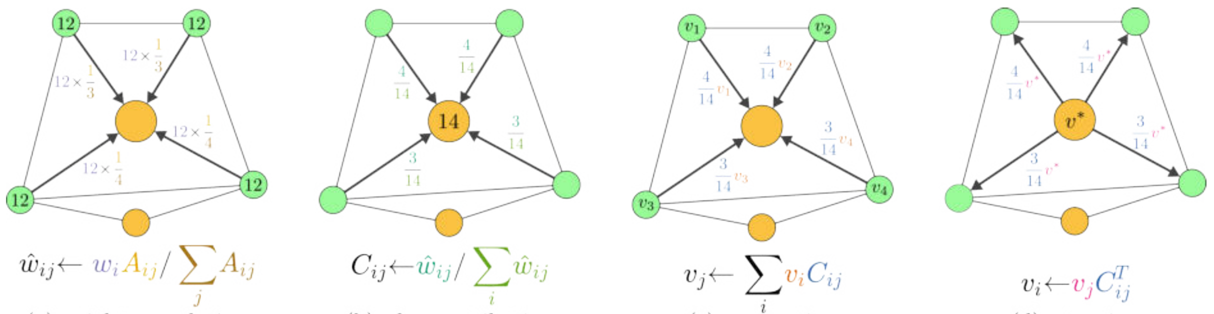

$\hat{w}_{ij} \leftarrow w_i A_{ij} / \sum_j A_{ij}$
(a) weight convolution
$C_{ij} \leftarrow \hat{w}_{ij} / \sum_i \hat{w}_{ij}$
(b) edge contribution
$v_j \leftarrow \sum_i v_i C_{ij}$
(c) aggregation
$v_i \leftarrow v_j C_{ij}^T$
(d) returning

Figure 4.3: Schematic representation of the transition between adjacent levels.

where f represents a Multi-Layer Perceptron (MLP) function, $\boldsymbol{e}$ denotes the latent information flow through an edge, and $\boldsymbol{v}$ is the latent node feature. For a detailed model architecture, please refer to Section 4.4.2.

4.3.2 Transition Between Levels

The transfer of information between two neighboring levels is facilitated using non-parametrized *downsampling* and *upsampling* modules. This approach aims to minimize the computational overhead associated with learnable transition modules between adjacent level pairs. In the context of our methodology, downsampling refers to the sequence of pooling nodes, followed by the aggregation of information from neighboring nodes onto the coarser level. Conversely, upsampling involves the sequence of unpooling, followed by the reinstatement of information from pooled nodes to their corresponding neighbors on the finer level.

Downsampling Let $\boldsymbol{A}$ represent the unweighted, boolean adjacency matrix. We initiate a nodal weight field w, set to one on the finest level, which accumulates during the downsampling process. Given a receiver node j and its corresponding sender node i, the downsampling process can be defined as follows (Figure 4.3):

- Begin by row-normalizing, akin to a standard graph convolution, where $\hat{\boldsymbol{A}}_{ij} \leftarrow \boldsymbol{A}_{ij} / \sum_j \boldsymbol{A}_{ij}$. Then, perform a single-weight convolution, $\hat{w}_{ij} \leftarrow w_i \hat{\boldsymbol{A}}_{ij}$ (Figure 4.3(a)).

- Compute edge weights $\boldsymbol{C}_{ij} \leftarrow \hat{w}_{ij} / \sum_i \hat{w}_{ij}$, where matrix $\boldsymbol{C}$ serves as a contribution table, with $\boldsymbol{C}_{ij}$ representing the proportion of weights at receiver node j contributed by sender node i (Figure 4.3(b)).
- Conduct a convolution of the latent information using the contribution table, $\boldsymbol{v}_j \leftarrow \sum_i \boldsymbol{v}_i \boldsymbol{C}_{ij}$. This operation is analogous to equally distributing and transmitting weighted information from sender nodes, followed by a weighted average operation at the receiver nodes (Figure 4.3(c)).

Upsampling Upon unpooling, all nodes, except those that were pooled, possess zero information. A retrieval process, reminiscent of transposed convolution in U-Net architecture, aids in distinguishing information among receiver nodes. Given the contribution table $\boldsymbol{C}$, which records edge weights, a natural choice is to update node information using $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_j \boldsymbol{C}_{ij}^T$ (Figure 4.3(d)).

4.4 Experiments

4.4.1 Datasets

We adopt three representative public datasets from GraphMeshNets (Pfaff et al., 2020): 1) CylinderFlow: incompressible fluid flow around a cylinder; 2) Airfoil: compressible flow around an airfoil; and 3) DeformingPlate: deformation of an elastic plate with an actuator. Additionally, we create the InflatingFont cases using the open-source simulator (Fang et al., 2021), maintaining consistent material properties and inflation speed. The input geometries for InflatingFont consist of $1,400$ 2×2 character matrices in Chinese. Compared to existing datasets, InflatingFont exhibits more complex geometric shapes, 2 to 8 times the number of nodes, and 70 times the number of contact edges. As a result, it is highly suitable for evaluating the scalability and compatibility of various GNNs with intricate geometries. The example plots are presented in Figure 4.4. The statistics of the graph features (such as number of nodes and edges, the mesh type, the number of total time steps) are included in Table 4.1.

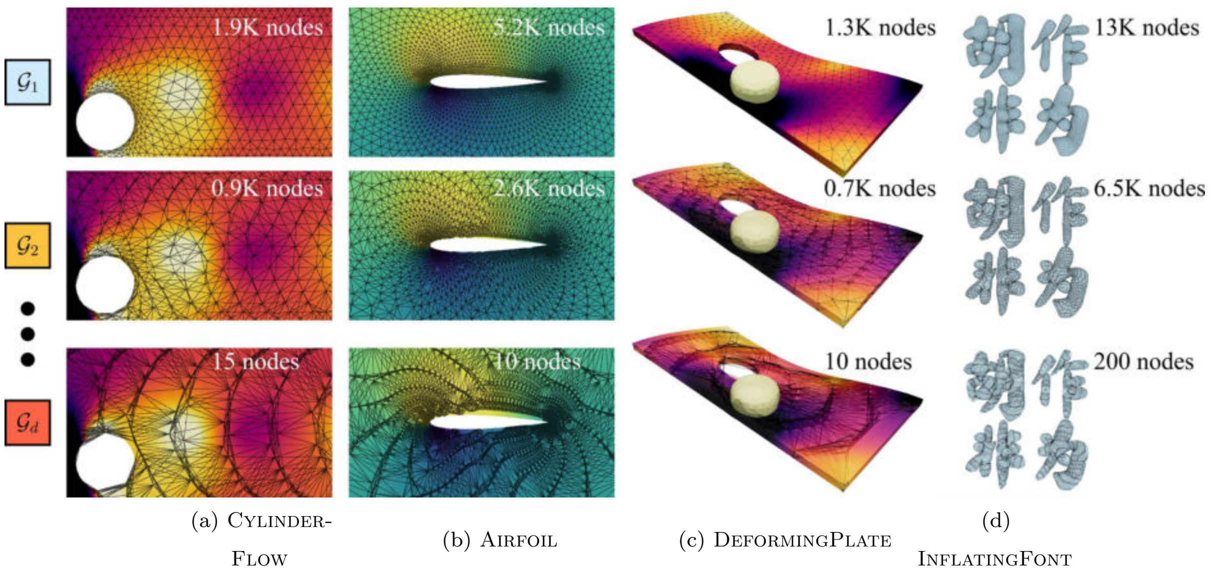


Figure 4.4: Example multi-level graphs produced by Bi-stride pooling. Our datasets contain both Eulerian and Lagrangian systems. Many meshes are highly irregular and contain significant self-contact, making the construction of coarser-level connections challenging based solely on spatial proximity. Our bi-stride strategy relies only on topological information and has been demonstrated to be robust and reliable for arbitrary geometries.

Table 4.1: Multi-level mesh configurations for benchmarks.

Case	Avg # nodes	Avg # edges	Mesh type	Seed method	# Levels	# Steps
Cylinder	1886	5424	triangle, 2D	MinAve	7	600
Airfoil	5233	15449	triangle, 2D	MinAve	9	600
Plate	1271	$4611, 94^*$	tetrahedron, 3D	MinAve	6	400
Font	13177	$39481, 6716^*$	triangle, 3D	CloseCenter	6	100

All datasets are divided into 1000 training, 200 validation, and 200 testing instances. Entries in the second column of the table labeled with superscript* in the average edge number represent contact edges:

Model Configurations We provide the input/output configurations for each experimental case below: 1) the offset inputs to prepend before the material edge processor e_{ij}^M and e_{ij}^W, and 2) nodes p_i, along with the nodal outputs q_i from the decoder. The detailed configurations for all datasets are included in Table 4.2. In these configurations, $\boldsymbol{X}$ and $\boldsymbol{x}$ represent material-space and world-space positions, $\boldsymbol{v}$ denotes velocity, ρ signifies density, P stands for absolute pressure, and the notation $\dot{a} = a_{t+1} - a_t$ indicates temporal change

Table 4.2: Model configurations for benchmarks.

Case	**Type**	**Offset inputs** e_{ij}^M	**Offset inputs** e_{ij}^W	**Inputs** p_i	**Outputs** q_i
Cylinder	Eulerian	$\boldsymbol{X}_{ij}, \lvert\boldsymbol{X}_{ij}\rvert$	NA	$\boldsymbol{v}_i, n_i$	$\dot{\boldsymbol{v}}_i$
Airfoil	Eulerian	$\boldsymbol{X}_{ij}, \lvert\boldsymbol{X}_{ij}\rvert$	NA	$\rho_i, \boldsymbol{v}_i, n_i$	$\dot{\boldsymbol{v}}_i, \dot{\rho}_i, P_i$
Plate	Lagrangian	$\boldsymbol{X}_{ij}, \lvert\boldsymbol{X}_{ij}\rvert, \boldsymbol{x}_{ij}, \lvert\boldsymbol{x}_{ij}\rvert$	$\boldsymbol{x}_{ij}, \lvert\boldsymbol{x}_{ij}\rvert$	$\dot{\boldsymbol{x}}_i, n_i$	$\dot{\boldsymbol{x}}_i$
Font	Lagrangian	$\boldsymbol{X}_{ij}, \lvert\boldsymbol{X}_{ij}\rvert, \boldsymbol{x}_{ij}, \lvert\boldsymbol{x}_{ij}\rvert$	$\boldsymbol{x}_{ij}, \lvert\boldsymbol{x}_{ij}\rvert$	n_i	$\dot{\boldsymbol{x}}_i$

for a variable a. All involved variables are normalized to have zero mean and unit variance during pre-processing.

For time integration, Cylinder, Airfoil, and Plate inherit the first-order integration from MeshGraphNets. For InflatingFont, the first-order quasi-static integration (Fang et al., 2021) is employed in the solver. Consequently, we also employ first-order integration for InflatingFont.

Baselines Across all datasets, we compare the computational complexity, training/inference time, and memory footprint of BSMS-GNN against several baselines: 1) MeshGraphNets(Pfaff et al., 2020): the single-level GNN architecture from GraphMeshNets; 2) MS-GNN-Grid(Lino et al., 2021, 2022a,b): an example of methods constructing hierarchies based on spatial proximity; and 3) GraphUNets(Gao and Ji, 2019): a representative approach employing learnable pooling modules. Implementation details are available in Section 4.4.2. We emphasize that methods such as (Liu et al., 2021; Fortunato et al., 2022) are not practical due to the requirement of manually generating coarser meshes for each trajectory instance using CAE software, involving the manual creation of approximately $20,000$ meshes across all cases.

Miscellaneous Ablation studies are conducted for our specific choice of transition method and seeding heuristics. The ablation study pertaining to the use of learnable pooling modules is not presented in isolation but is addressed through comparison with GraphUNets in comprehensive experiments (details in Section 4.4.3).

4.4.2 Implementation Details

We implemented our framework using PyTorch (Paszke et al., 2019) and PyG (PyTorch Geometric) (Fey and Lenssen, 2019). The entire model was trained by supervising the single-step L_2 loss between the ground truth and the nodal field output of the decoding module.[1]

4.4.2.1 Basic Modules and Architectures

The MLPs for the nodal encoder, processor, and nodal decoder are ReLU-activated two-hidden-layer MLPs with a hidden-layer and output size of 128, except for the nodal decoder, whose output size matches the prediction $\boldsymbol{q}$. All MLPs include a residual connection. LayerNorm is applied to normalize all MLP outputs except for the nodal decoder.

4.4.2.2 Baseline: MeshGraphNets

Our MeshGraphNets implementation uses the same MLPs as mentioned above but includes an additional module: the edge encoder. Additionally, the edge latent is updated and carried over throughout the end of multiple MPs. We use 15 time steps for message passing in all cases to maintain consistency with MeshGraphNets.

4.4.2.3 Baseline: MS-GNN-Grid

Our re-implementation of MS-GNN-Grid utilizes the same MLPs as described above, with four additional modules: the edge encoder at the finest level, aggregation modules for nodes and edges at every level for transitions, and returning modules for nodes at every level. This method also involves assigning regular grid nodes for each level. Grid nodes are assigned by defining an initial grid resolution and an inflation rate between levels. The initial grid sizes and the inflation rates between levels are recorded in Table 4.3. The

[1]Our datasets and code are publicly available at https://github.com/Eydcao/BSMS-GNN.

Table 4.3: Details of re-implementing the multi-level structure in (Lino et al., 2022a).

Case	# Levels	Initial grid dx	dx inflation	Level-wise # MPs
Cylinder	4	[5e-2, 5e-2]	2	[4, 2, 2, 4]
Airfoil	4	[4.5, 4.5]	2	[4, 2, 2, 4]
Plate	4	[4e-3, 4e-3, 4e-3]	2	[4, 2, 2, 4]
Font	4	[1.5e-2, 1.5e-2, 1e-3]	2	[4, 2, 2, 4]

Table 4.4: Training noise parameters and batch sizes.

Case	Batch size				Noise scale
	BSMS-GNN	MS-GNN-Grid	MeshGraphNets	GraphUNets	
Cylinder	32	16	16	2	velocity: 2e-2
Airfoil	8	4	8	1	velocity: 2e-2, density: 1e1
Plate	8	2	2	1	pos: 3e-3
Font	2	1	1	1	pos: [5e-3, 5e-3, 3.33e-4]

number of message passing iterations at each level follows (Lino et al., 2022a), using four at the top and bottom levels and two for the others.

4.4.2.4 Baseline: GraphUNets

Our re-implementation of GraphUNets uses the same number of levels as BSMS-GNN. We made the following modifications to the original GraphUNets: (1) We adjusted the information passing from GCN to our message passing module for consistency and translational invariance. (2) The original GraphUNetswas designed for small graphs (100 nodes) and employed dense matrix multiplications, which is not scalable for our graph size (1500 to 15000 nodes). To address this, we optimized operations such as matrix multiplication and aggregation with sparse implementations.

4.4.2.5 Noise and Batch Size

For each benchmark, we introduced Gaussian noise with a specific scale and added it to the original trajectory at the beginning of every epoch. Noise injection aimed to simulate model-generated noise, enhancing the model's ability to correct its output when given noisy inputs. We carefully tuned the batch size for each method through smaller subset experiments to achieve optimal convergence rates. The related details are recorded in

Table 4.5: Detailed measurements of baselines in BSMS-GNN. BSMS-GNN consistently produces stable and competitive global rollouts with the smallest training cost. BSMS-GNN is also lightweight and has the fastest inference time. It is also free from the large RMSE due to poor pooling on unseen geometries where the learnable pooling module of GraphUNets struggles. The second column of entries in Infer time is the speedup compared to the numerical solver. The second column of entries in Training cost is the epoch at which the converged model is achieved.

Measurements	**Case**	**Ours**	(**Lino et al., 2021**)	(**Pfaff et al., 2020**)	(**Gao and Ji, 2019**)
Training time/step [ms]	Cylinder	**10.14**	15.36	19.29	16.20
	Airfoil	**18.82**	25.26	36.72	55.08
	Plate	**15.58**	49.65	49.15	31.88
	InflatingFont	**45.96**	107.16	117.48	1,833.37
Infer time/step [ms]	Cylinder	6.75, 121x	**6.18, 133x**	14.50, 57x	24.30, 34x
	Airfoil	**8.64, 1275x**	20.40, 540x	24.20, 455x	33.60, 328x
	Plate	**14.01, 207x**	18.12, 160x	15.70, 184x	16.20, 179x
	InflatingFont	**33.33, 210x**	41.66, 168x	82.35, 85x	629.33, 11x
Training cost [hrs], Final epoch	Cylinder	**21.41, 19**	35.84, 21	64.30, 30	76.15, 39
	Airfoil	**122.33, 39**	176.82, 42	275.40, 45	206.55, 37
	Plate	**56.07, 27**	125.78, 19	176.94, 27	127.50, 30
	InflatingFont	**2.68E+01, 21**	5.66E+01, 19	6.20E+01, 19	NA
RMSE-1 [1e-2]	Cylinder	**2.04E-01**	2.20E-01	2.26E-01	8.09E-01
	Airfoil	2.88E+01	**2.68E+01**	4.35E+01	2.93E+01
	Plate	2.87E-02	2.20E-02	**1.98E-02**	2.03E-02
	InflatingFont	**1.77E-02**	1.87E-02	1.95E-02	NA
RMSE-50 [1e-2]	Cylinder	**2.42**	2.74	4.39	1.87E+01
	Airfoil	**1.10E+03**	1.22E+03	1.66E+03	1.17E+03
	Plate	3.18E-02	**2.78E-02**	2.88E-02	5.19E-02
	InflatingFont	**1.08E-01**	3.24E-01	1.78E-01	NA
RMSE-all [1e-2]	Cylinder	**8.37**	8.49	1.07E+01	1.65E+02
	Airfoil	**4.21E+03**	5.56E+03	6.95E+03	6.11E+03
	Plate	1.60E-01	**1.48E-01**	1.51E-01	5.46E-01
	InflatingFont	**2.20E-01**	3.78E-01	3.65E-01	NA

Table 4.4.

4.4.3 Results and Discussions

By evaluating BSMS-GNN and other competitors on all the baselines (Section 4.4.1), we derive the following conclusions:

- We empirically demonstrate that the learnable pooling method is not suitable for large-scale, complex geometries;
- We conduct a small-scale experiment where proximity-based pooling leads to incorrect edges and inference;
- BSMS-GNN offers significant advantages in terms of memory consumption, training time to achieve desired accuracy, and inference time;
- BSMS-GNN also achieves the highest accuracy, reducing the rollout RMSE by

Table 4.6: Memory footprint comparison between baselines. BSMS-GNN consistently reduces RAM consumption by approximately half in all cases during the training stage, and also has the smallest (except for DeformingPlate) inference RAM.

Case	Method	Training RAM (GBs) with different Batch #						Inference RAM (GBs)
		2	4	8	16	32	64	
CylinderFlow	**Ours**	**2.41**	**2.92**	**4.37**	**6.06**	**11.4**	**22.27**	**1.92**
	(Lino et al., 2021)	2.79	3.60	5.31	8.56	15.10	-	1.97
	(Pfaff et al., 2020)	3.25	4.46	6.91	11.84	21.60	-	1.94
	(Gao and Ji, 2019)	23.33	-	-	-	-	-	2.18
Airfoil	**Ours**	**3.66**	**5.46**	**8.88**	**15.70**	-	-	**2.02**
	(Lino et al., 2021)	4.18	6.25	10.65	19.25	-	-	**2.02**
	(Pfaff et al., 2020)	5.53	8.90	16.08	-	-	-	2.06
	(Gao and Ji, 2019)	-	-	-	-	-	-	2.67
DeformingPlate	**Ours**	**2.36**	**2.87**	**3.85**	**5.78**	**9.28**	**16.85**	1.95
	(Lino et al., 2021)	3.41	4.81	7.75	13.20	-	-	2.00
	(Pfaff et al., 2020)	3.10	4.29	6.59	11.49	20.80	-	**1.93**
	(Gao and Ji, 2019)	-	-	-	-	-	-	2.18
InflatingFont	**Ours**	**6.28**	**10.80**	-	-	-	-	**2.23**
	(Lino et al., 2021)	10.87	19.79	-	-	-	-	2.45
	(Pfaff et al., 2020)	12.48	23.39	-	-	-	-	2.28
	(Gao and Ji, 2019)	-	-	-	-	-	-	4.51

nearly half on InflatingFont with the largest mesh size and the most complex geometries; we also achieve accurate results with a model zero-shot on a teaser with approximately 7 times more nodes.

Detailed measurements are presented in Table. 4.5 and Table. 4.6. All experiments were conducted using a single Nvidia RTX 3090.

Disadvantages of Learnable Pooling GraphUNets shows significantly higher RMSE in both 1-step and global rollouts on all datasets, except for the Airfoil dataset, where the mesh is consistent across trajectories. To confirm that varying mesh leads to poor inference with learnable pooling, we apply the trained GraphUNets model to an unseen test trajectory of the DeformingPlate dataset. Figure 4.5 clearly reveals the unfair pooling distribution by the learnable module, which impedes information passing on coarser levels and results in poor inference. In comparison, bi-stride pooling generates uniform pooling and accurate inference.

Additionally, GraphUNets has to conduct the adjacency matrix multiplication in the forward pass, which results in a 2–40× increase in the training and inference times, particularly in larger datasets. In the largest InflatingFont, one training epoch requires nearly 50 hours to complete, making the convergence of the model infeasible. Given its

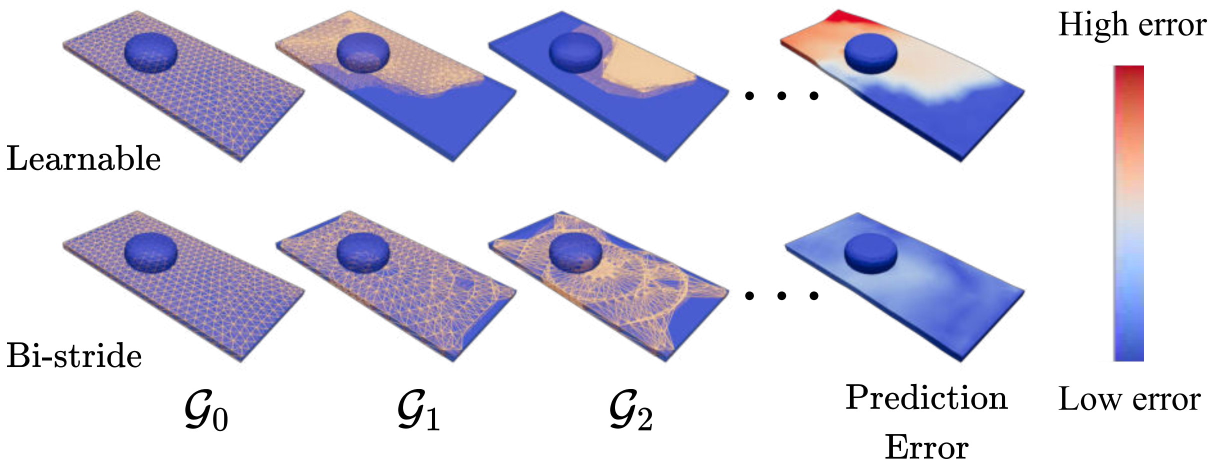


Figure 4.5: Comparison between pooling methods. Unlike bi-stride pooling, which generalizes to any input graph, a learnable module leads to unfair pooling on unseen geometries, impeding the information exchange for unselected nodes. The inferred results show larger errors than bi-stride pooling.

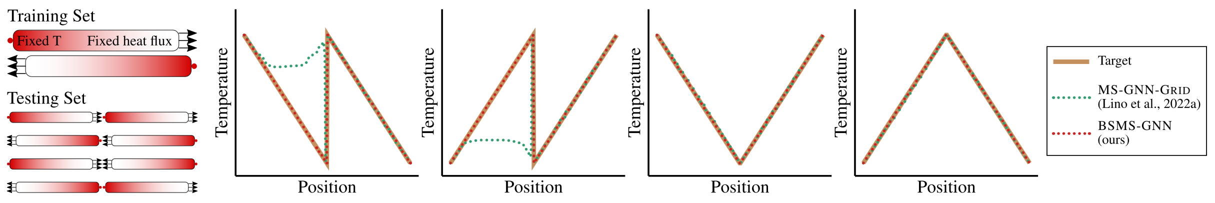


Figure 4.6: Failure cases for MS-GNN-Grid. Left: the configuration of the simplest failure case for multi-level GNNs by spatial proximity: steady-state 1-D heat transfer. Right, leading two columns: two tests showing that even if trained to convergence, the erroneous edge across the boundary can still result in the wrong inference. Right, last two columns: the erroneous edge coincidentally does not affect the results due to the symmetry of the solution, and no heat will diffuse between two nodes with the same temperature.

poor performance in both accuracy and efficiency, we conclude that GraphUNets is not suitable for simulation cases with large-scale, complex geometries. By default, we will not specifically make comparisons to GraphUNets in the following discussions.

Failure Cases for Spatial Proximity To illustrate the adversarial impact of wrongly constructed edges by spatial proximity, we design a simple 1-D steady-state heat transfer simulation on sticks (Figure 4.6 left), on which BSMS-GNN and MS-GNN-Grid are trained and evaluated. The training set consists of two mirrored instances, where one end of the stick is fixed at a certain temperature and the other end has a fixed heat flux,

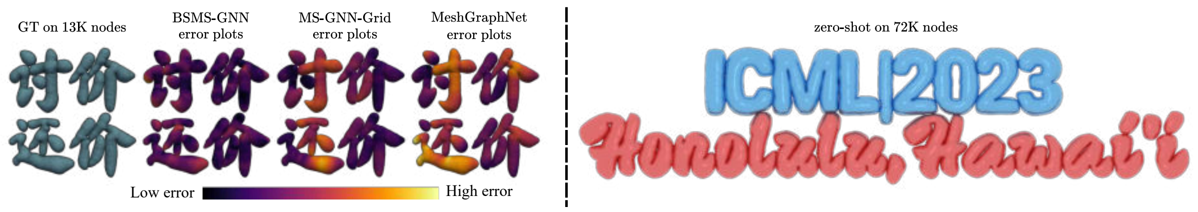
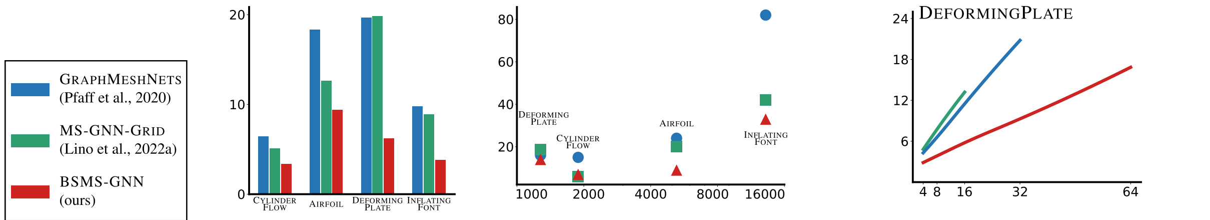


Figure 4.7: Zero-shot ability of BSMS-GNN. Left: Comparing the rollout RMSE between BSMS-GNN and existing SOTAs on the InflatingFont benchmark. Our framework reaches the highest accuracy, demonstrating a stronger capability for complex, large geometries with massive contacts. Right: Our trained model can zero-shot infer on never-seen fonts, with about 7× more nodes in the mesh size while maintaining the same accuracy.

resulting in a linear temperature distribution. In the test set, we simply align two sticks in a head-to-tail configuration with a tiny space between them to prevent heat diffusion across the boundary. MS-GNN-Grid, utilizing spatial proximity, incorrectly constructs an edge between the two sticks. As a result, in half of the test cases, MS-GNN-Grid shows wrong results at the boundary due to the erroneous edge (Figure 4.6 right, leading two columns). Although only in simple cases, one can alleviate this issue by marking the two sticks as separate clusters and making inferences independently; a similar fix is unfeasible for connected, complex geometries. For instance, in a long, thin U-shaped tunnel, two nodes located on the parallel sides of the "U"S are spatially close but geodesically distant, and hence should not be connected by an edge.

Accuracy and Generalization In terms of accuracy, our method exhibits the smallest rollout Root Mean Squared Error (RMSE) across all cases except for the DeformingPlate dataset, where all three competitors achieve similar results. We attribute this outcome to the relatively small mesh size of DeformingPlate, which does not pose a disadvantage to the flat architecture. In the case of the largest and most complex dataset, InflatingFont, our method achieves the highest accuracy, reducing the rollout RMSE by 40% compared to the competitors (Figure 4.7, Left). Furthermore, our model showcases the capability for zero-shot inference on larger meshes than those present in the training set. It consistently generates accurate global rollouts, even when the characters are in a different language.



(a) training time per epoch [hr]

(b) inference time per step [ms] vs. # nodes

(c) memory footprint [GB] vs. batch size

Figure 4.8: Performance comparison between baselines. GraphUNets is not included due to poor performance. Full results are plotted in Table. 4.5 and Table. 4.6.

Performance Advantages of BSMS-GNN Our method features a lightweight architecture characterized by a reduced number of Message Passing (MP) units at each level and the absence of learnable transition modules. This leads to a significant reduction in memory usage during training (Figure 4.8(c) and Table 4.6). Specifically, BSMS-GNN consumes only 43% $\sim$ 87% of the memory used by MS-GNN-Grid, 48% $\sim$ 53% of that used by MeshGraphNets, and merely 10% of the memory used by GraphUNets. During inference, our method also boasts the lowest memory consumption, except for the DeformingPlate dataset, where memory usage is slightly higher (20MB) than MeshGraphNets.

The reduction in memory footprint contributes to enhanced training efficiency by accommodating larger batch sizes, increased random sampling, and fewer CPU-GPU data transfers. Consequently, BSMS-GNN exhibits the fastest unit training speed per epoch among all competitors (Figure 4.8(a) and Table 4.5), requiring only 26% $\sim$ 58% of the unit training time of MS-GNN-Grid and MeshGraphNets.

Regarding inference time, BSMS-GNN performs comparably to MS-GNN-Grid on datasets with smaller mesh sizes (CylinderFlow and DeformingPlate), both surpassing the performance of MeshGraphNets. However, as the mesh size increases, BSMS-GNN outperforms MS-GNN-Grid, demonstrating superior scalability. For datasets with larger mesh sizes (Airfoil and InflatingFont), our method exhibits a 1.5× and 1.9× improvement over MS-GNN-Grid and MeshGraphNets, respectively (Figure 4.8(b) and Table 4.5).

Regarding the total training cost to achieve a desired global rollout RMSE, we observe that all methods reach the target with a similar number of epochs. This is due to the fact that all methods are trained to withstand rollout noise by encountering different random noises in each epoch. Sufficient exposure to noise patterns (proportional to the epoch number) is essential for accurate rollouts. Consequently, the total wall time is roughly proportional to the unit training time, given similar epoch numbers, making our method the most efficient.

4.4.4 Ablation Study

4.4.4.1 Transition Method

While exploring non-parametric transition solutions, we initially began with no transition since our method is directly adapted from GraphUNets(Gao and Ji, 2019). The no-transition strategy yields sufficiently low 1-step RMSE and visually accurate rollouts for InflatingFont. However, in the global rollouts of CylinderFlow and Airfoil cases, we observed stripe patterns (Figure 4.9 (c), column **None**), with stripes aligning with edges at coarser levels (Figure 4.9 (d)). This error seems to result from the unpooled nodes having zero information prior to message passing (MP), making them indistinguishable from processor modules and exacerbating differences between pooled and unpooled nodes during rollouts.

The no-transition strategy is analogous to no interpolation during the super-resolution phase of CNN+UNet. Consequently, we attempted a single step of graph convolution (without activation) to emulate interpolation in regular grids. However, this approach overly smooths features (Figure 4.9 (e), column **Graph Conv**), and information propagation is smeared out except near the generator (in this case, near the cylinder).

We believe the over-smoothing issue arises due to the mesh's irregularity. Unlike CNNs, where fine nodes are centrally located in coarser grids, irregular meshes exhibit varying topology and element sizes. Element sizes are typically smaller near the interface for higher simulation precision. Consequently, unweighted graph convolution can smear

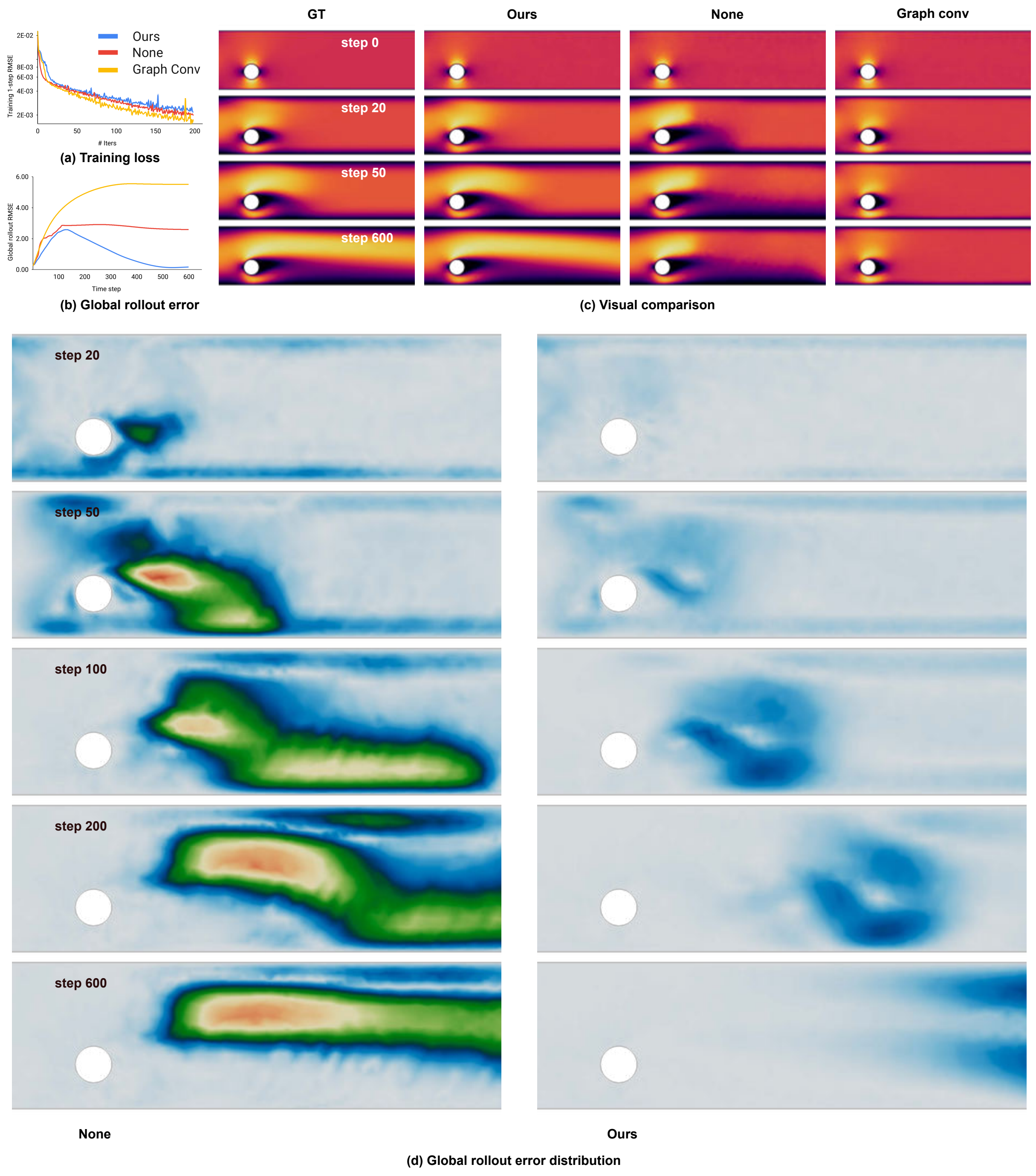


Figure 4.9: Rollout error plots for different transition methods. (a) All three transition methods can achieve the target training RMSE within 200 iterations. (b) However, our weighted graph aggregation+returning approach exhibits stronger resistance to noise during rollouts. (c) Visual comparisons show that no transition produces mosaic-like patterns, while the graph convolution transition smears out information and ceases propagation downstream. (d) The global rollout error distribution of no transition (**Left**) reveals mosaic-like pattern edges resembling the simulation mesh; the error of our transition (**Right**) travels with the generated vortices downstream and leaves the domain after step 200, explaining the RMSE drop in (b).

Table 4.7: Detailed measurements of different transition methods. Our method and **Pos-Kernel** are the only two non-parametric transitions offering lightweight and reliable rollouts compared to the resourcE-intensive **Learnable** transition.

Measurement	**Ours**	**None**	**Graph-Conv**	**Pos-Kernel**	**Learnable**
Training time/step [ms]	10.14	**9.30**	10.07	10.06	17.75
Infer time/step [ms]	6.75	**5.70**	6.46	6.90	11.28
Training RAM [GBs]	**11.041**	**11.041**	**11.041**	**11.041**	18.033
Infer RAM [GBs]	**1.923**	**1.923**	**1.923**	**1.923**	1.931
RMSE-1 [1E-2]	2.85E-01	**1.49E-01**	3.41E-01	6.38E-01	4.70E-01
RMSE-50 [1E-2]	1.43E+01	2.05E+02	2.40E+02	1.77E+01	**1.35E+01**
RMSE-all [1E-2]	1.68E+01	2.59E+02	5.51E+02	2.01E+01	**1.57E+01**

finer information near the cylinder and its adjacent neighbors during return. To address this irregularity, we propose the solution detailed in Section 4.3.2, utilizing nodal weights during aggregation and recording contribution shares for subsequent return. Our transition method consistently performs well across all experimental cases and yields the lowest RMSE for global rollouts (Figure 4.9 (b)).

Comparison to Alternative Transition Methods Additionally, we compare our transition method to two alternatives from prior works: (1) computing edge weights (kernels) for information flow using inverse length (node position offset), denoted as **Pos-Kernel** (Liu et al., 2021); and (2) level-wise learnable transition modules implemented through additional MP, denoted as **Learnable** (Fortunato et al., 2022).

The detailed measurements of training costs and inference performance and metrics are presented in Table 4.7. Besides the high RMSE values for **None** and **Graph-Conv** shown in Figure 4.9, we observe that: (1) training/inference time and RAM consumption for all non-parametric transitions (including **None**) are similar, indicating the lightweight nature of our transition method. (2) **Learnable** transition achieves slightly higher accuracy but at approximately 70% greater training/inference time and RAM costs. As mentioned in Section 4.4.3, higher training RAM can hinder batch size and increase CPU-GPU data communication frequency, further slowing training with larger scales. (3) **Pos-Kernel** results in slightly higher RMSE compared to our method, making it a competitive alternative for production.

Table 4.8: Sensitivity analysis of seeding heuristics on model performance. The "Random" heuristic refers to choosing a random seed for every cluster, while "Inverse w.r.t. Train" refers to choosing the inverse seeding heuristic compared to that used during the training phase. For example, if the MinAve heuristic was used during training, the CloseCenter heuristic was chosen during the testing phase.

Seeding @ Test	**Ratio in RMSE**	**Cylinder**	**Airfoil**	**Plate**	**Font**
Random	1-step	2.06	14.04	2.95	1.15
	50-step	1.06	2.16	5.21	1.04
	Rollout	1.01	2.04	1.53	1.04
Inverse w.r.t. Train	1-step	1.96	12.77	3.14	1.15
	50-step	1.07	1.76	7.52	1.02
	Rollout	1.13	2.21	1.46	1.06

4.4.4.2 Seeding Heuristics

In this section, we investigate the impact of employing different seeding heuristics during the training and testing phases on the sensitivity of a converged model. We intentionally modify the heuristics used for each benchmark and assess the resulting RMSEs. The results reveal that inconsistencies in seeding heuristics lead to increased roll-outs compared to the outcomes obtained when using a consistent seeding approach, as presented in Table 4.8. Specifically, the roll-outs range from 1.01× to 2.04× for random seeding and 1.13× to 2.21× for inverse seeding, relative to the consistent seeding. However, the RMSEs remain within the same magnitude, indicating that our method is not highly sensitive to initial seeding.

4.4.5 Scaling Analysis on Font Datasets

In our investigation of different resolutions (5K, 15K, 30K, and 45K) for the INFLATING-FONT dataset, we find that both BSMS-GNN and MS-GNN-GRID exhibit near-linear scaling behavior. Notably, our method (BSMS-GNN) demonstrates superior efficiency compared to MS-GNN-GRID, as depicted in Figure 4.10.

To assess scalability, we create scaled-down and scaled-up versions of the INFLATING-FONT dataset, each with varying average node counts for the initial geometry. We then apply consistent simulation settings across these datasets. To ensure stable convergence

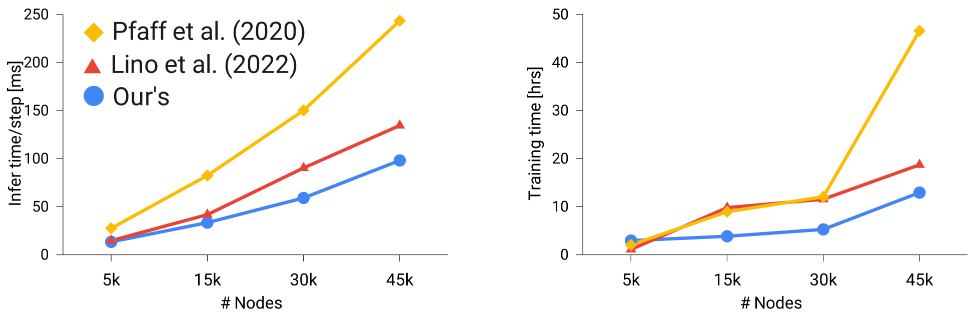


Figure 4.10: Scaling analysis for different baselines. As the size of the font dataset (INFLATINGFONT) increases, our proposed method (BSMS-GNN) consistently outperforms both MESHGRAPHNETS and MS-GNN-GRID in terms of efficiency and scalability.

Table 4.9: Adjustments for parameters in scaling analysis.

# Nodes	d_1	d_2	**Initial Grid Resolution**	**Target RMSE**	**Noise Injection**
5k	4	2	[6e-2 6e-2 4e-3]	1.73e-4	[8.5e-3, 8.5e-3, 5.7e-4]
15k	6	4	[1.5e-2 1.5e-2 1e-3]	1e-4	[5e-3, 5e-3, 3.33e-4]
30k	7	5	[7.5e-3 7.5e-3 5e-4]	1e-4	[3.5e-3, 3.5e-3, 2.4e-4]
45k	7	5	[7.5e-3 7.5e-3 5e-4]	1e-4	[2.9e-3, 2.9e-3, 1.9e-4]

in the low-resolution scenarios, we relax the termination criteria by increasing the target Root Mean Square Error (RMSE) relative to the average edge length, thereby preventing convergence failures. Similarly, noise injection is adjusted proportionally to the average edge length.

Furthermore, as the number of nodes decreases, fewer levels are required to achieve equivalent bottom resolutions. Corresponding adjustments are made to the levels of our model (d_1) and those of MS-GNN-GRID (d_2). The specific adjustments are presented in Table 4.9.

CHAPTER 5

Conclusions and Future Work

This thesis has advanced physics-based simulation techniques from both conventional and machine learning perspectives. The unifying objective was to address the complexities encountered when applying scientific computing and simulation to real-world industrial-level geometries. In this context, each work within this thesis contributes uniquely towards enhancing availability, accuracy, and computational efficiency.

Chapter 2 introduced the Unstructured Moving Least Square Material Point Method (UMLS-MPM), which enables MPM to be applied to any 2D and 3D mesh tessellations. The method guarantees no cell-crossing errors and demonstrates a high-order convergence rate empirically, which significantly enhances the adaptability and accuracy of the MPM on unstructured domains.

In Chapter 3, our Coupling Shockwaves with Material Point Method (CSMPM) introduced a hybrid Eulerian and Lagrangian/Eulerian method for the coupled simulation of complex flow-structure interactions. This approach offers a robust solution and the ability to capture shock wave propagation and solid fracturing within arbitrary geometries, thus significantly improving robustness and efficiency.

Chapter 4 developed our Bi-Stride Multi-Scale Graph Neural Network (BSMS-GNN) method, a significant advancement in building multi-level graph neural networks for mesh-based physical simulations. By employing a novel pooling strategy, BSMS-GNN efficiently constructs multi-level meshes while preserving connectivity. It also reduces computational overhead and prevents the introduction of erroneous cross-interface edges, thereby maintaining high prediction accuracy.

5.1 Future Work

The research conducted presents several promising avenues for future investigation:

- For the UMLS-MPM, developing algorithms that dynamically adjust the sample weight function according to a sizing field within the mesh could address the challenges of non-uniform kernel effects and enhance the method's applicability to broader simulation contexts. Automating the splitting and merging of particles could adjust the resolution appropriately across different domain size regions.
- Coupling the UMLS-MPM with Finite Volume Methods (FVMs) for fluid phases is a natural expansion, as both can adapt to unstructured meshes and irregular domain geometries.
- For the CSMPM, future research could explore integrating cut-cell methods or adaptive mesh refinement techniques to further enhance the spatial resolution and accuracy of interface capture.
- Extending the CSMPM to include simulations with incompressible fluids, granular media, and other complex materials could broaden the applicability of these simulation techniques across various domains.
- For the BSMS-GNN method, further exploration into batched and distributed training methods could enhance the scalability of BSMS-GNNs. Integrating these networks with advanced architectures such as Transformers could also improve the stability and accuracy of simulations over long sequences.
- Recent advances in graph neural networks and conventional neural networks have begun to incorporate mesh and numerical scheme information, achieving much higher accuracy. The kernel construction in the UMLS-MPM can be a potential novel paradigm for learning hybrid simulations, akin to embedding both kernel and mesh information into the network's channels.

Aanjaneya, M., Patkar, S., and Fedkiw, R. (2013). A monolithic mass tracking formulation for bubbles in incompressible flow. *J Comp Phys*, 247:17–61. 55

Alfonsi, G. (2009). Reynolds-averaged Navier-Stokes equations for turbulence modeling. *Applied Mechanics Reviews*, 62. 1

Andersen, S. and Andersen, L. (2010). Analysis of spatial interpolation in the material-point method. *Computers & Structures*, 88(7-8):506–518. 29

Anderson, J. D. and Wendt, J. (1995). *Computational Fluid Dynamics*, volume 206. Springer. 1

Asratian, A. S., Denley, T. M., and Häggkvist, R. (1998). *Bipartite Graphs and Their Applications*, volume 131. Cambridge University Press. 76

Bailoor, S., Annangi, A., Seo, J. H., and Bhardwaj, R. (2017). Fluid–structure interaction solver for compressible flows with applications to blast loading on thin elastic structures. *Applied Mathematical Modelling*, 52:470–492. 48

Banks, J. W., Henshaw, W. D., Kapila, A. K., and Schwendeman, D. W. (2016). An added-mass partition algorithm for fluid-structure interactions of compressible fluids and nonlinear solids. *Journal of Computational Physics*, 305:1037–1064. 48

Bardenhagen, S. G. and Kober, E. M. (2004). The generalized interpolation material point method. *Computer Modeling in Engineering and Sciences*, 5(6):477–496. 3, 16, 58

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*. 7

Batty, C., Bertails, F., and Bridson, R. (2007). A fast variational framework for accurate solid-fluid coupling. *ACM Trans Graph (TOG)*, 26(3):100–es. 5

Baumgarten, A. and Kamrin, K. (2019). A general fluid-sediment mixture model and constitutive theory validated in many flow regimes. *Journal of Fluid Mechanics*, 861:721–764. 54

Baumgarten, A. S., Couchman, B. L., and Kamrin, K. (2021). A coupled finite volume and material point method for two-phase simulation of liquid–sediment and gas–sediment flows. *Computer Methods in Applied Mechanics and Engineering*, 384:113940. 49

Belbute-Peres, F. D. A., Economon, T., and Kolter, Z. (2020). Combining differentiable PDE solvers and graph neural networks for fluid flow prediction. In *International Conference on Machine Learning*, pages 2402–2411. PMLR. 7, 74

Beuth, L., Wikeckowski, Z., and Vermeer, P. (2011). Solution of quasi-static large-strain problems by the material point method. *International Journal for Numerical and Analytical Methods in Geomechanics*, 35(13):1451–1465. 3

Bonet, J. and Wood, R. D. (2008). *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press. 12

Brackbill, J., Kothe, D., and Ruppel, H. (1988). FLIP: A low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25–38. 2, 4, 16

Bridson, R. (2015). *Fluid Simulation for Computer Graphics*. CRC press. 1, 4, 5, 6, 55

Cao, Y., Chai, M., Li, M., and Jiang, C. (2023a). Efficient learning of mesh-based physical simulation with bi-stride multi-scale graph neural network. In *International Conference on Machine Learning*. 9

Cao, Y., Chen, Y., Li, M., Yang, Y., Zhang, X., Aanjaneya, M., and Jiang, C. (2022). An efficient B-spline Lagrangian/Eulerian method for compressible flow, shock waves, and fracturing solids. *ACM Transactions on Graphics (TOG)*, 41(5):1–13. 6, 9

Cao, Y., Gao, X., and Li, R. (2019). A liquid plug moving in an annular pipe-heat transfer analysis. *International Journal of Heat and Mass Transfer*, 139:1065–1076. 6

Cao, Y. and Li, R. (2018). A liquid plug moving in an annular pipe—flow analysis. *Physics of Fluids*, 30(9):093605. 6

Cao, Y., Zhao, Y., Li, M., Yang, Y., Choo, J., Terzopoulos, D., and Jiang, C. (2023b). Material point methods on unstructured tessellations: A stable kernel approach with continuous gradient reconstruction. *arXiv preprint arXiv:2312.10338*. 8

Charlton, T., Coombs, W., and Augarde, C. (2017). iGIMP: An implicit generalised interpolation material point method for large deformations. *Computers & Structures*, 190:108–125. 3

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. (2020). Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445. 7

Chorin, A. J. (1967). The numerical solution of the Navier-Stokes equations for an incompressible fluid. *Bulletin of the American Mathematical Society*, 73(6):928–931. 4

De Borst, R., Crisfield, M. A., Remmers, J. J., and Verhoosel, C. V. (2012). *Nonlinear Finite Element Analysis of Solids and Structures*. John Wiley & Sons. 1

Deconinck, H., Roe, P. L., and Struijs, R. (1993). A multidimensional generalization of Roe's flux difference splitter for the Euler equations. *Computers & Fluids*, 22(2-3):215–222. 5

Eymard, R., Gallouët, T., and Herbin, R. (2000). Finite volume methods. *Handbook of Numerical Analysis*, 7:713–1018. 1

Fang, Y., Li, M., Jiang, C., and Kaufman, D. M. (2021). Guaranteed globally injective 3D deformation processing. *ACM Trans. Graph.*, 40(4):75–1. 1, 83, 85

Fang, Y., Qu, Z., Li, M., Zhang, X., Zhu, Y., Aanjaneya, M., and Jiang, C. (2020). IQ-MPM: an interface quadrature material point method for non-sticky strongly two-way coupled nonlinear solids and fluids. *ACM Trans Graph (TOG)*, 39(4):51–1. 50, 56, 57, 58, 62

Fedkiw, R., Liu, X., and Osher, S. (2002). A general technique for eliminating spurious oscillations in conservative schemes for multiphase and multispecies Euler equations. *International J. of Nonlinear Sci. and Num. Sim.*, 3(2):99–106. 56

Feldman, B. E., O'brien, J. F., and Arikan, O. (2003). Animating suspended particle explosions. In *ACM SIGGRAPH 2003 Papers*, pages 708–715. ACM. 5

Fern, J., Rohe, A., Soga, K., and Alonso, E. (2019). *The Material Point Method for Geotechnical Engineering: A Practical Guide.* CRC Press. 3

Fey, M. (1998). Multidimensional upwinding. Part II. decomposition of the Euler equations into advection equations. *Journal of Computational Physics*, 143(1):181–199. 5

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds.* 86

Forrer, H. and Berger, M. (1999). Flow simulations on Cartesian grids involving complex moving geometries. In *Hyperbolic Problems: Theory, Numerics, Applications*, pages 315–324. Springer. 5, 54, 64

Forrer, H. and Jeltsch, R. (1998). A higher-order boundary treatment for Cartesian-grid methods. *J Comp Phys*, 140(2):259–277. 5, 54

Fortunato, M., Pfaff, T., Wirnsberger, P., Pritzel, A., and Battaglia, P. (2022). Multi-scale meshgraphnets. In *ICML 2022 2nd AI for Science Workshop.* 7, 8, 74, 81, 85, 95

Fotiadis, S., Pignatelli, E., Valencia, M. L., Cantwell, C., Storkey, A., and Bharath, A. A. (2020). Comparing recurrent and convolutional neural networks for predicting wave propagation. *arXiv preprint arXiv:2002.08981.* 6

Fu, C., Guo, Q., Gast, T., Jiang, C., and Teran, J. (2017). A polynomial particle-in-cell method. *ACM Trans Graph (TOG)*, 36(6):1–12. 4

Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*, pages 267–285. Springer. 6

Fung, Y., Tong, P., and Chen, X. (2001). *Classical and Computational Solid Mechanics.* World Scientific. 1

Gagniere, S., Hyde, D., Marquez-Razon, A., Jiang, C., Ge, Z., Han, X., Guo, Q., and Teran, J. (2020). A hybrid Lagrangian/Eulerian collocated velocity advection and projection method for fluid simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2020, Vol. 39.* 58

Gan, Y., Sun, Z., Chen, Z., Zhang, X., and Liu, Y. (2018). Enhancement of the material point method using B-spline basis functions. *International Journal for Numerical Methods in Engineering*, 113(3):411–431. 4

Gao, H. and Ji, S. (2019). Graph U-nets. In *International Conference on Machine Learning*, pages 2083–2092. PMLR. 7, 74, 75, 85, 88, 89, 93

Gao, H., Sun, L., and Wang, J. (2021). PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *Journal of Computational Physics*, 428:110079. 2, 6

Gao, H., Zahr, M. J., and Wang, J. (2022). Physics-informed graph neural galerkin networks: A unified framework for solving PDE-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502. 2, 7, 74

Gao, M., Pradhana, A., Han, X., Guo, Q., Kot, G., Sifakis, E., and Jiang, C. (2018). Animating fluid sediment mixture in particle-laden flows. *ACM Trans Graph (TOG)*, 37(4):1–11. 49, 54

Gao, M., Tampubolon, A. P., Jiang, C., and Sifakis, E. (2017). An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Transactions on Graphics (TOG)*, 36(6):1–12. 1

Gaume, J., Gast, T., Teran, J., Herwijnen, A., and Jiang, C. (2018). Dynamic anticrack propagation in snow. *Nature Communications*, 9(1):1–10. 3

Gaume, J., Herwijnen, A., Gast, T., Teran, J., and Jiang, C. (2019). Investigating the release and flow of snow avalanches at the slope-scale using a unified model based on the material point method. *Cold Regions Science and Technology*, 168:102847. 3

Germano, M., Piomelli, U., Moin, P., and Cabot, W. H. (1991). A dynamic subgrid-scale eddy viscosity model. *Physics of Fluids A: Fluid Dynamics*, 3(7):1760–1765. 1

Gonzalez, O. and Stuart, A. M. (2008). *A First Course in Continuum Mechanics.* Cambridge University Press. 57

Grétarsson, J. T. and Fedkiw, R. (2013). Fully conservative leak-proof treatment of thin solid structures immersed in compressible fluids. *J Comp Phys*, 245:160–204. 50

Grétarsson, J. T., Kwatra, N., and Fedkiw, R. (2011). Numerically stable fluid-structure interactions between compressible flow and solid structures. *J Comp Phys*, 230(8):3062–3084. 50

Grzeszczuk, R., Terzopoulos, D., and Hinton, G. (1998). 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. 6

Guilkey, J., Harman, T., and Banerjee, B. (2007). An Eulerian-Lagrangian approach for simulating explosions of energetic devices. *Computers & Structures*, 85(11-14):660–674. 3

Guilkey, J. E., Hoying, J. B., and Weiss, J. A. (2006). Computational modeling of multicellular constructs with the material point method. *Journal of Biomechanics*, 39(11):2074–2086. 3

Guillard, H. (1993). *Node-nested multi-grid method with Delaunay coarsening*. PhD thesis, INRIA. 8

Guo, X., Li, W., and Iorio, F. (2016). Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 481–490. 6

Han, X., Gao, H., Pffaf, T., Wang, J., and Liu, L. (2022). Predicting physics in mesh-reduced space with temporal attention. *arXiv preprint arXiv:2201.09113*. 74

Harlow, F. H. (1962). The particle-in-cell method for numerical solution of problems in fluid dynamics. Technical report, Los Alamos Scientific Lab., N. Mex. 2

Harsch, L. and Riedelbauch, S. (2021). Direct prediction of steady-state flow fields in meshed domain with graph networks. *arXiv preprint arXiv:2105.02575*. 7, 74

Homel, M. A., Brannon, R. M., and Guilkey, J. E. (2014). Simulation of shaped-charge jet penetration into drained and undrained sandstone using the material point method with new approaches for constitutive modeling. *CIMNE, Barcelona*, pages 676–687. 3

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., and Jiang, C. (2018). A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Trans. Graph.*, 37(4). 4, 17, 18, 49

Hughes, T. J. (2012). *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Courier Corporation. 1, 58

Hyde, D. A. and Fedkiw, R. (2019). A unified approach to monolithic solid-fluid coupling of sub-grid and more resolved solids. *Journal of Computational Physics*, 390:490–526. 50, 54

Ionescu, I., Guilkey, J. E., Berzins, M., Kirby, R. M., and Weiss, J. A. (2006). Simulation of soft tissue failure using the material point method. *Journal of Biomechanical Engineering*, 128:917–924. 3

Jassim, I., Stolle, D., and Vermeer, P. (2013). Two-phase dynamic analysis by material point method. *International Journal for Numerical and Analytical Methods in Geomechanics*, 37(15):2502–2522. 3

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

Jiang, C., Schroeder, C., Selle, A., Teran, J., and Stomakhin, A. (2015). The affine particle-in-cell method. *ACM Trans Graph (TOG)*, 34(4):1–10. 4, 31, 32, 33, 52

Jiang, C., Schroeder, C., and Teran, J. (2017b). An angular momentum conserving affine-particle-in-cell method. *J Comp Phys*, 338:137–164. 4, 31

Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., and Selle, A. (2016a). The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Course*, pages 24:1–24:52. 1, 49, 53

Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., and Selle, A. (2016b). The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, pages 1–52. Association for Computing Machinery. 6, 36

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440. 2, 6

Kawada, G. and Kanai, T. (2011). Procedural fluid modeling of explosion phenomena based on physical properties. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 167–176. 5

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B. (2019). Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38(2):59–70. 6

Kim, B., Liu, Y., Llamas, I., and Rossignac, J. R. (2005). Flowfixer: Using BFECC for fluid simulation. Technical report, Georgia Institute of Technology. 4

Klár, G., Gast, T., Pradhana, A., Fu, C., Schroeder, C., Jiang, C., and Teran, J. (2016). Drucker-Prager elastoplasticity for sand animation. *ACM Transactions on Graphics (TOG)*, 35(4):1–12. 1, 3

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. (2021). Machine learning-accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118. 2

Kosík, A., Feistauer, M., Hadrava, M., and Horáček, J. (2015). Numerical simulation of the interaction between a nonlinear elastic structure and compressible flow by the discontinuous Galerkin method. *Applied Mathematics and Computation*, 267:382–396. 48

Koster, P., Tielen, R., Wobbes, E., and Möller, M. (2021). Extension of B-spline material point method for unstructured triangular grids using powell-sabin splines. *Computational Particle Mechanics*, 8(2):273–288. 4

Kwatra, N., Gretarsson, J., and Fedkiw, R. (2010). Practical animation of compressible flow for shockwaves and related phenomena. In *Symposium on Computer Animation*, pages 207–215. 9, 50, 51

Kwatra, N., Su, J., Grétarsson, J. T., and Fedkiw, R. (2009). A method for avoiding the acoustic time step restriction in compressible flow. *J Comp Phys*, 228(11):4146–4161. 50, 52, 55, 61, 62, 63, 64

Levin, D. (1998). The approximation power of moving least-squares. *Mathematics of Computation*, 67(224):1517–1531. 21, 31

Li, M., Ferguson, Z., Schneider, T., Langlois, T. R., Zorin, D., Panozzo, D., Jiang, C., and Kaufman, D. M. (2020a). Incremental potential contact: Intersection- and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4):49. 1, 6

Li, X., Cao, Y., Li, M., Yang, Y., Schroeder, C., and Jiang, C. (2022a). Plasticitynet: Learning to simulate metal, sand, and snow for optimization time integration. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*. 1

Li, X., Li, M., and Jiang, C. (2022b). Energetically consistent inelasticity for optimization time integration. *ACM Transactions on Graphics (TOG)*, 41(4):1–16. 1

Li, Z. and Favier, J. (2017). A non-staggered coupling of finite element and lattice boltzmann methods via an immersed boundary scheme for fluid-structure interaction. *Computers & Fluids*, 143:90–102. 48

Li, Z., Huang, D. Z., Liu, B., and Anandkumar, A. (2022c). Fourier neural operator with learned deformations for PDEs on general geometries. *arXiv preprint arXiv:2207.05209*. 7

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stuart, A., Bhattacharya, K., and Anandkumar, A. (2020b). Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766. 7, 8, 74, 75

Lino, M., Cantwell, C., Bharath, A. A., and Fotiadis, S. (2021). Simulating continuum mechanics with multi-scale graph neural networks. *arXiv preprint arXiv:2106.04900*. 7, 8, 74, 75, 85, 88, 89

Lino, M., Fotiadis, S., Bharath, A. A., and Cantwell, C. (2022a). Towards fast simulation of environmental fluid mechanics with multi-scale graph neural networks. *arXiv preprint arXiv:2205.02637*. 7, 8, 74, 75, 85, 87

Lino, M., Fotiadis, S., Bharath, A. A., and Cantwell, C. D. (2022b). Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics. *Physics of Fluids*, 34(8):087110. 7, 8, 74, 85

Liu, W., Yagoubi, M., and Schoenauer, M. (2021). Multi-resolution graph neural networks for PDE approximation. In *International Conference on Artificial Neural Networks*, pages 151–163. Springer. 7, 8, 74, 85, 95

Liu, X., Osher, S., and Chan, T. (1994). Weighted essentially non-oscillatory schemes. *J Comp Phys*, 115(1):200–212. 6

Ma, S., Zhang, X., Lian, Y., and Zhou, X. (2009a). Simulation of high explosive explosion using adaptive material point method. *Computer Modeling in Engineering and Sciences (CMES)*, 39(2):101. 3

Ma, S., Zhang, X., Lian, Y., and Zhou, X. (2009b). Simulation of high explosive explosion using adaptive material point method. *Computer Modeling in Engineering & Sciences*, 39:101–124. 49

Masci, J., Boscaini, D., Bronstein, M., and Vandergheynst, P. (2015). Geodesic convolutional neural networks on Riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 37–45. 75

Mesquita, D., Souza, A., and Kaski, S. (2020). Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33:2220–2231. 74

Moin, P. and Mahesh, K. (1998). Direct numerical simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics*, 30(1):539–578. 1

Monasse, L., Daru, V., Mariotti, C., Piperno, S., and Tenaud, C. (2012). A conservative coupling algorithm between a compressible flow and a rigid body using an embedded boundary method. *Journal of Computational Physics*, 231(7):2977–2994. 5, 48

Obiols-Sales, O., Vishnu, A., Malaya, N., and Chandramowliswharan, A. (2020). CFDNet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12. 6

Osher, S. and Fedkiw, R. P. (2001). Level set methods: An overview and some recent results. *Journal of Computational physics*, 169(2):463–502. 1

Pasquariello, V., Hammerl, G., Örley, F., Hickel, S., Danowski, C., Popp, A., Wall, W. A., and Adams, N. A. (2016). A cut-cell finite volume – finite element coupling approach for fluid–structure interaction in compressible flow. *Journal of Computational Physics*, 307:670–695. 48

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in*

Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc. 86

Peskin, C. S. (2002). The immersed boundary method. *Acta Numerica*, 11:479–517. 1

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. (2020). Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*. 2, 7, 74, 83, 85, 88, 89

Piomelli, U. (1999). Large-eddy simulation: Achievements and challenges. *Progress in Aerospace Sciences*, 35(4):335–362. 1

Pretti, G., Coombs, W. M., Augarde, C. E., Sims, B., Puigvert, M. M., and Gutiérrez, J. A. R. (2023). A conservation law consistent updated lagrangian material point method for dynamic analysis. *Journal of Computational Physics*, 485:112075. 3

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660. 75

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 30. 75

Qu, Z., Zhang, X., Gao, M., Jiang, C., and Chen, B. (2019). Efficient and conservative fluids using bidirectional mapping. *ACM Trans Graph (TOG)*, 38(4):1–12. 4

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). 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. 2, 6

Reddy, J. N. and Gartling, D. K. (2010). *The Finite Element Method in Heat Transfer and Fluid Dynamics*. CRC press. 1

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. (2020). Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR. 2, 7, 75

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. (2018). Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR. 7, 74

Setaluri, R., Aanjaneya, M., Bauer, S., and Sifakis, E. (2014). SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph.*, 33(6). 5

Sewall, J., Galoppo, N., Tsankov, G., and Lin, M. (2009). Visual simulation of shockwaves. *Graphical Models*, 71(4):126–138. 49

Shih, T., Liou, W. W., Shabbir, A., Yang, Z., and Zhu, J. (1995). A new $k - \varepsilon$ eddy viscosity model for high Reynolds number turbulent flows. *Computers & Fluids*, 24(3):227–238. 1

Shu, C. and Osher, S. (1988). Efficient implementation of essentially non-oscillatory shock-capturing schemes. *J Comp Phys*, 77(2):439–471. 6

Simo, J. C. and Hughes, T. J. (2006). *Computational Inelasticity*, volume 7. Springer Science & Business Media. 16

Spiegel, E. A. and Veronis, G. (1960). On the Boussinesq approximation for a compressible fluid. *The Astrophysical Journal*, 131:442. 5

Stam, J. (1999). Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 121–128. 4

Steffen, M., Kirby, R. M., and Berzins, M. (2008). Analysis and reduction of quadrature errors in the material point method (MPM). *International Journal for Numerical Methods in Engineering*, 76(6):922–948. 4, 16

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A. (2013). A material point method for snow simulation. *ACM Trans Graph (TOG)*, 32(4):1–10. 3, 49

Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J., and Selle, A. (2014). Augmented MPM for phase-change and varied materials. *ACM Trans Graph (TOG)*, 33(4):1–11. 57

Sulsky, D., Zhou, S., and Schreyer, H. L. (1995). Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications*, 87(1-2):236–252. 2, 49

Sun, L., Gao, H., Pan, S., and Wang, J. (2020). Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732. 2, 6

Taira, K. and Colonius, T. (2007). The immersed boundary method: A projection approach. *Journal of Computational Physics*, 225(2):2118–2137. 1

Takeshita, D., Ota, S., Tamura, M., Fujimoto, T., Muraoka, K., and Chiba, N. (2003). Particle-based visual simulation of explosive flames. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pages 482–486. IEEE. 5

Tampubolon, A. P., Gast, T., Klár, G., Fu, C., Teran, J., Jiang, C., and Museth, K. (2017). Multi-species simulation of porous sand and water mixtures. *ACM Transactions on Graphics (TOG)*, 36(4):105. 3

Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K. (2017). Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR. 6

Tran, Q., Wobbes, E., Sołowski, W. T., Möller, M., and Vuik, C. (2019). Moving least squares reconstruction for B-spline material point method. In *International Conference on the Material Point Method for Modelling Soil-Water-Structure Interaction*, pages mpm2019–07. 4

Vaucorbeil, A. and Nguyen, V. P. (2021). Modelling contacts with a total lagrangian material point method. *Computer Methods in Applied Mechanics and Engineering*, 373:113503. 3

Vaucorbeil, A., Nguyen, V. P., and Hutchinson, C. R. (2020). A total-Lagrangian material point method for solid mechanics problems involving large deformations. *Computer Methods in Applied Mechanics and Engineering*, 360:112783. 3

Vaucorbeil, A., Nguyen, V. P., Hutchinson, C. R., and Barnett, M. R. (2022a). Total Lagrangian material point method simulation of the scratching of high purity coppers. *International Journal of Solids and Structures*, 239:111432. 3

Vaucorbeil, A., Nguyen, V. P., and Mandal, T. K. (2022b). Mesh objective simulations of large strain ductile fracture: A new nonlocal Johnson-Cook damage formulation for the total Lagrangian material point method. *Computer Methods in Applied Mechanics and Engineering*, 389:114388. 3

Wang, L., Coombs, W. M., Augarde, C. E., Cortis, M., Brown, M. J., Brennan, A. J., Knappett, J. A., Davidson, C., Richards, D., White, D. J., et al. (2021). An efficient and locking-free material point method for three-dimensional analysis with simplex elements. *International Journal for Numerical Methods in Engineering*, 122(15):3876–3899. 3

Wang, L., Currao, G. M., Han, F., Neely, A. J., Young, J., and Tian, F. (2017). An immersed boundary method for fluid–structure interaction with compressible multiphase flows. *Journal of Computational Physics*, 346:131–151. 48

Wikeckowski, Z. (2004). The material point method in large strain engineering problems. *Computer Methods in Applied Mechanics and Engineering*, 193(39-41):4417–4438. 3

Wilson, P., Wüchner, R., and Fernando, D. (2021). Distillation of the material point method cell crossing error leading to a novel quadrature-based C0 remedy. *International Journal for Numerical Methods in Engineering*, 122(6):1513–1537. 35, 36, 38

Wu, K., Truong, N., Yuksel, C., and Hoetzlein, R. (2018). Fast fluid simulations with sparse volumes on the GPU. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2018)*, 37(2):157–167. 5

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24. 74

Zhang, D. Z., Ma, X., and Giguere, P. T. (2011). Material point method enhanced by modified gradient of shape function. *Journal of Computational Physics*, 230(16):6379–6398. 4

Zhang, X., Chen, Z., and Liu, Y. (2016). *The Material Point Method: A Continuum-Based Particle Method for Extreme Loading Cases*. Academic Press. 12, 49

Zhang, Z., Bu, J., Ester, M., Zhang, J., Yao, C., Yu, Z., and Wang, C. (2019). Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*. 74

Zhao, Y., Choo, J., Jiang, Y., and Li, L. (2023a). Coupled material point and level set methods for simulating soils interacting with rigid objects with complex geometry. *Computers and Geotechnics*, 163:105708. 47

Zhao, Y., Jiang, C., and Choo, J. (2023b). Circumventing volumetric locking in explicit material point methods: A simple, efficient, and general approach. *International Journal for Numerical Methods in Engineering*, 124(23):5334–5355. 41, 42, 43, 44