École Normale Supérieure Paris-Saclay

Master MVA

Internship
Inria Paris, Astra Team, Astra-Vision group

Learning physics of rigid bodies with Graph Neural Networks

Supervisors:
Raoul de Charette - Inria
Tuan-Hung Vu - Valeo.AI
Andrei Bursuc - Valeo.AI

Student:
Clément Weinreich

September 12, 2024

Contents

1 Introduction

A fundamental aspect of human common sense knowledge involves understanding and reasoning about how objects in a scene interact with each other through physical principles. In everyday life, we are frequently faced with problems that involve predicting what will happen next in a physical environment or to infer the underlying physical properties of complex scenes. This capability is mandatory to be able to interact with our physical world, which requires first understanding the visual stimuli and then predicting the evolution of the environment. Consider for example a billiard game, as the white ball moves, the observer can predict its future positions and how collisions may affect other balls' trajectories. Such ability relies in fact on our ability to understand the basic physics rules. The better our understanding, the more accurate we can predict the future. Such a task is difficult to mimic with machines given the numerous unobserved parameters, and the often chaotic behavior of the underlying dynamical systems. Nonetheless, humans can easily forecast a physical environment by breaking the scenario into objects and relationships and then predicting the consequences of their interactions and dynamics [1, 21]. This implies the ability to generalize and apply physical laws in various scenarios.

These issues of learning and predicting the outcome of dynamical systems in a wide range of scenarios are actively being studied in the literature [6, 40, 41, 39, 3, 16]. In fact, realistic physical simulators of complex systems are essential in many scientific and engineering disciplines such as biology, climate science, computer vision, and graphics. Traditional simulators can be very expensive to create and use. They require considerable engineering effort and often involve a trade-off between accuracy and generalization in controlled scenarios. Moreover, to set up physical simulators, it is necessary to provide all the physical parameters of the objects and environment (initial conditions, boundary conditions), which can be difficult to obtain. Using machine learning as a proxy for these simulators is a very attractive alternative. The idea is to create a learned simulator that will be trained from observed data (real or simulated) in order to reproduce the dynamics. The interpolation capabilities of deep learning models [12, 56] then allows to obtain accurate predictions when the input data is similar to the distribution of the training set. However, the complex dynamics and often very large state space pose challenges for standard learning approaches.

The scientific literature approaches the learning of physics from different angles. When objects are subject to deformation or interaction, there are two key components: the constitutive laws, which are material-specific fields that describe how the material properties respond to stress and strain (e.g. Young's Modulus, Poisson Ratio, etc.); the equations of motion that describe how the shape and position of the object change over time due to dynamic effects, usually derived from Newton's second law ($F = ma$) in physical simulators. Some studies use models that learn both constitutive laws and equations of motion purely from data through pattern recognition, without explicitly programming physical laws [41, 53]. Others incorporate established physical laws into the models as a prior and use neural networks to learn only the constitutive relationships [26, 57].

The learning of physics also has significant potential applications in computer vision. Recent advancements in image and video generation techniques have led to highly perceptually realistic results, however, they often lack physical realism. In particular, the produced images and videos appear convincing to the human eye but frequently fail when it comes to maintaining consistency with the physical properties and dynamics of the real world. This gap is particularly evident in video generation, where models struggle with temporal coherence and accurate motion representation as highlighted in VideoPhy [4]. Physically realistic scene editing has been explored in various scenarios [22, 34, 15, 11]. These methods are specifically designed to ensure that the generations are consistent with physical laws. Research has also focused on inferring physical properties (constitutive laws) of objects from visual data, such as NeRF2Physics [55]. Recently, 3D Gaussian Splatting (3DGS) [19] has made it easier to

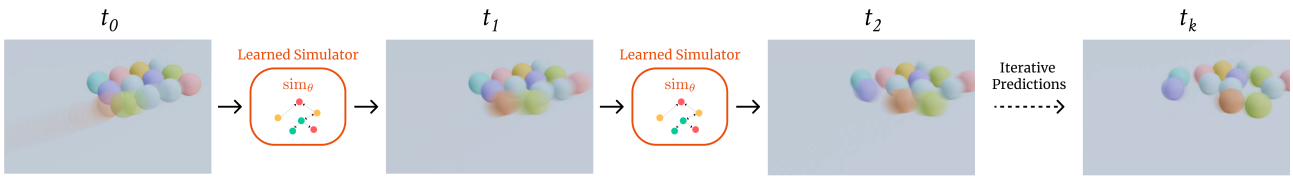

Figure 1: The internship is focused on designing a learned simulator sim_θ able to predict the next states of a rigid-body physical system, consisting of spheres interacting through collisions. This learned simulator should be able to generalize physical laws involved in rigid-body dynamics so that it can be applied universally.

manipulate and control the geometry and appearance of scenes. Opposite to NeRF which optimizes an implicit volume, 3DGS represents surface points which allows direct geometry manipulation. For example, PhysGaussian [51] manipulates Gaussians as simulatable particles of a dynamic system and enriches Gaussian kernels with physically meaningful kinematic deformation and mechanical stress attributes. Consequently, it is possible to use a learned simulator to apply physically realistic deformations to 3D scenes. Using a learned simulator, as opposed to a traditional simulator, allows for the application of complex transformations that are difficult to simulate with classical methods but have been learned from real or simulated data. Depending on the complexity of the phenomenon, there is also a significant speed advantage during inference [44]. In particular for fluid simulation and climate modeling, learned simulators have already been proven to speed up predictions without compromising performance. For instance [48, 37] shows that deep learning models can emulate the physics of climate systems much faster than traditional approaches for a similar accuracy. These factors constitute strong motivations to develop highly realistic learned simulators and allow for the application of complex deformations to 3D scenes. Achieving controllable and physically realistic augmentations of 3D scenes could significantly improve the robustness of perception systems in scenarios that are underrepresented in current datasets.

Across the literature on learned simulators, particularly in applications related to enhancing computer vision and graphics, a significant gap remains in understanding how well these models generalize fundamental physical laws. The term "generalization" in this context often refers to the extension of dynamics to different scenarios (e.g. larger spatial or temporal domain, more particles) [41, 3, 39]. **However, the primary goal of these simulators should be to generalize the underlying physical laws so that they can be applied universally, regardless of the specific scenario**. This raises an important question about how well these learned simulators can truly generalize physical laws and whether they can consistently and accurately apply them across diverse situations.

Internship contribution. For this reason, we study the generalization of physical laws from simulated data, enabling accurate application of dynamics in out-of-distribution (OOD) scenarios. The complexity of this task lies in the need to accurately model and generalize interactions governed by fundamental physical laws, which often involve nonlinear dynamics, multi-body interactions and generate chaotic behaviors. Furthermore, ensuring that these models can generalize to OOD scenarios adds another layer of complexity, as it requires to be robust to conditions they were not explicitly trained on, which is a significant challenge in the field of computational physics. To simplify the setup and facilitate the analysis, we focus on classical **rigid body physics** with **spheres** as rigid objects (see Figure 1). We consider scenarios consisting of spheres that interact with each other through collisions and interact with surfaces that are represented as planes through friction. Hence, the physical laws under consideration are gravity, kinematics, energy conservation or dissipation through friction or collisions, and normal force. We then propose a framework based on state-of-the-art methods, benchmark

the developed method's effectiveness in generalizing these laws, and propose a few promising directions toward addressing important issues. We ensure the physical realism of the learned simulators by introducing specific metrics that evaluate adherence to the considered physical laws. This project constitutes one of the first steps of a long-term vision, aiming to create more advanced learned simulators that can be applied across various real-world scenarios. This is particularly relevant in computer vision, as this would enable the application of complex deformations to 3D scenes while adhering to physical laws, something that has not yet been explored.

Internship context The internship lasted six months, from April 1st to September 30th, and took place at Inria, in the Astra team, specifically in the Astra-vision group. It was supervised by Raoul de Charette (Inria), Tuan-Hung Vu (Valeo.AI), and Andrei Bursuc (Valeo.AI). Initially, the subject was focused on learning physically realistic extreme deformations that could be applied to 3D representations such as destruction (involving a change in the topology of objects) to augment datasets. After an extensive review of the literature, we noticed that the ability to learn physically realistic transformations was a necessary step to ensure the realism of the augmentations. Hence, the subject shifted at the end of May to the study of learning physics, and more specifically rigid body physics, as it is a key component of complex deformations such as destruction.

2 Literature review

This section covers key developments related to our work in physical simulators, learned simulators, and their use in computer vision and graphics. It is organized into three parts. Section 2.1 focuses on traditional simulators, introducing widely used methods in computer graphics and physics like the Material Point Method (MPM) and Position Based Dynamics (PBD) that can both simulate complex materials and interactions between particles. Section 2.2 is dedicated to learned simulators and explores recent advances in using machine learning, especially Graph Networks (GNs) to model physical dynamics. Finally, Section 2.3 examines how these simulation techniques are being integrated with advancements in computer vision and graphics, particularly focusing on their application to physically realistic manipulation of 3D environment, and learning physical dynamics and physical properties from visual data like images and videos.

2.1 Physical simulators

When learning physics, most methods rely on ground truth data obtained from traditional physical simulators. This section is dedicated to an overview of the principles, conventions, and simulators used by the scientific community.

The Material Point Method (MPM) is a method developed in the 90s [45, 47, 46] that combines Lagrangian material particles (points) with Eulerian Cartesian grids to accurately simulate continuum materials. Widely used in computer graphics, MPM allows the simulation of various materials including elastic objects, snow, lava, sand, and viscoelastic fluids [18]. In MPM, the material continuum is discretized by a collection of Lagrangian particles, each carrying material properties such as mass, velocity, density, and force as continuous functions of position. While particles move in a Lagrangian fashion, the momentum equations are solved on a temporary regular Eulerian grid that overlays the spatial domain. At each time step, particle properties are projected onto this grid, where the equations of motion (e.g., Navier-Stokes or Cauchy momentum equation) are solved to determine grid-based velocities and stresses. Changes are then propagated back to particles using kernel interpolation to update particle positions.

Position Based Dynamics (PBD) [32] is a distinct method specifically designed for real-time simulation of deformable objects in computer graphics. PBD treats the dynamics directly manipulating particle positions to satisfy constraints without solving forces and integrating motion equations as in MPM. Objects in PBD are represented as a collection of particles, each with attributes such as position, velocity, and mass. Instead of projecting particle properties onto a grid and solving motion equations, PBD iteratively adjusts particle positions to enforce constraints such as maintaining distances, avoiding collisions, or preserving volume. While MPM remains better at simulating complex material behaviors like snow, sand, or viscoelastic fluids by capturing the interactions of particles with a grid, PBD is preferred for real-time applications where stability and computational efficiency are crucial. PBD can still accurately simulate a wide range of materials such as cloth, soft bodies, or fluids.

Rigid body dynamics is a specific area of physical dynamics where the motion and interactions of solid objects are studied, assuming that the objects do not deform under stress. In this context, the primary focus is on accurately modeling the behavior of these rigid bodies during collisions and rotations, often requiring specialized techniques to handle the discontinuous nature of contact dynamics. The simulation of interacting objects through collisions poses challenges to traditional simulators. PBD struggles with accurately capturing the dynamics of collisions due to its constraint-based approach, while MPM faces difficulties with maintaining detailed interactions during particle-grid transfers and

handling complex boundary conditions. Specific methods and software have been designed to handle this difficult task, in particular, Bullet Physics [8]. Bullet Physics is a professional open-source collision detection, rigid body, and soft body dynamics library written in C++. This library is primarily used in computer graphics for games, visual effects, and robotic simulations. Notably, it is integrated into Houdini and Blender, two of the main 3D graphics software. The library provides multiple algorithms to perform discrete and continuous collision detection including ray and convex sweep tests. It can resolve collisions and other constraints and provide the updated world transform for all the objects involved. In the literature on learned simulators (Section 2.2) for rigid body physics [2, 39, 3], the Bullet Physics engine is commonly utilized through the Kubric dataset [14], which employs Bullet to generate data involving the interaction and collision of objects with various shapes.

Simulating complex deformations that involve non-trivial changes in topology, such as fracture, shearing, or destruction by strong forces, is not targeted by previous simulators, which are instead meant to handle simple deformations and interactions. The computer graphics community tackled this task and has provided algorithms and methods that rely on physics to accurately simulate these phenomena. In particular for fracture and destruction, some methods [33, 30, 43] provide approximations of the change of topology according to the impact point and force, but do not provide the continuous transformation. Recently, More sophisticated approaches have been proposed to extend MPM with continuous damage mechanics. CD-MPM [50] employs a variational energy-based formulation for crack evolution, and AnisoMPM [49] augment the method by taking into account the inherent anisotropy of materials to model accurately the directionality of the fractures. These methods allow to obtain the continuous transformation of the material leading to fracture, at the expense of a much higher computational cost. This suggests the possibility of learning these extreme transformations in a discretized manner, aiming to apply the learned physical phenomena to other scenarios.

2.2 Learned simulators

This section shifts the focus from traditional physical simulators to machine learning-based approaches for simulating physical dynamics. We present the main developments that have led to the recent state-of-the-art frameworks relying on Graph Networks. By learning from data often obtained from the traditional simulators presented in Section 2.1, these methods can simulate different types of physics (fluids, granular, viscous, rigid bodies, etc.). The first part is focused on the Graph Network framework, which constitutes the basis of more complex approaches meant to simulate rigid bodies, detailed in the second part.

2.2.1 Graph networks to learn dynamics

Message Passing Graph Networks. The recent gain of interest in learning physics is strongly linked to the development of Graph Networks (GNs) [5], which unifies the different types of Graph Neural Networks (GNNs) first introduced in 2008 [42]. In GNs, a graph is defined as a 3-tuple $G = (V, E, \mathbf{u})$. $\{v_i \mid i \in [1 : N^v]\}$ is the set of nodes, where each v_i is a node attribute, $E = \{e_{i,j} \mid i, j \in [1 : N^v]\}$ with $e_{i,j}$ is a directed edge attribute between vertices i and j, and $\mathbf{u}$ corresponds to global attributes of the system, such as a gravitational field. The connection between vertices can be fixed or dynamic, depending on the use case. In this setting, a GN maps an input graph to an output graph that has the same structure but potentially different node, edge, or global attributes. A common architecture design is the *encode-process-decode* configuration. An input graph G^t is encoded into a latent representation by an encoder, then a GN layer (also called block) is applied M times to transform the latent graph iteratively, and finally, an output graph is decoded by a decoder. This

mapping can be optimized for some training objectives, aiming to learn a message passing [13], where latent information is propagated between nodes via the edges. Typically, the learned message passing involves three key steps: the creation of messages, the aggregation of messages received by each node, and an update step where the node's state is adjusted based on its current state and the aggregated messages. This provides a first basis to learn different types of interactions between the nodes according to the training objective.

Particle-based physics and GNs. The Graph Network architecture naturally fits the particle-based physics model used in traditional simulators such as MPM or PBD (Section 2.1). Nodes represent the physical states of particles and message-passing through edges is similar to modeling pairwise interactions between particles, where the aggregation simulates the total effects of all the interactions on a particle from its neighbors, and the update step changes the particle's states after computing the interactions.

Early methods based on GNs. The Interaction Network (IN) [6] is one of the earliest attempts to formalize how neural networks could be used to learn physical dynamics from raw observational data. They use a GNN framework to capture object interactions through a graph structure explicitly. In INs, objects are represented as nodes, and their interactions as directed edges. The IN processes this graph by predicting interaction effects using a relation-centric function f_R, and updating the object state via an object-centric function f_O. This is similar to the message-passing mechanism, where information is propagated through the graph to update node, edge, and global attributes iteratively, leading to more accurate predictions of physical dynamics. INs have been applied to various tasks, such as predicting the future positions of objects in simple physical scenarios. Concurrently to INs, Neural Physics Engine (NPE) [7] adopts a similar approach, but instead of relying on predefined relationships between objects, NPE learns the nature of these relations by focusing its attention on a neighborhood set of objects, allowing the model to infer interactions based on proximity and other learned criteria.

Extensions of Interaction Networks Since these developments, multiple contributions extended the framework of INs or NPEs to more complex scenarios. In 2018, the Graph Network framework was used to re-think INs in a more general framework [40] by adding global representations and outputs for the state of the system, as well as per-edge outputs. They introduced a recurrent GN-based model that allowed for more accurate predictions and control over physical systems. Also in 2018, the Hierarchical Relation Network (HRN) [29] introduced a hierarchical, particle-based approach to better capture complex, multi-scale interactions within physical systems. This approach enabled the simulation of scenarios involving deformable materials like plastic and elastic objects. Introduced in 2019, Dynamic Particle Interaction Networks (DPI-Nets) [23] build on these principles by extending their fixed graph structure to a dynamic graph construction scheme, adapting to evolving particle configurations in complex physical systems. Unlike earlier models, DPI-Nets could adapt their graphs in real-time as particle configurations evolved, making them particularly effective for accurately simulating fluid dynamics and deformable materials, where the relationships between particles are constantly changing. Building on the HRN model, they also add one level of hierarchy to efficiently propagate long-range influences among particles.

A foundational framework: Graph Network-based Simulators (GNS). These developments have led to the formalization of a machine learning framework termed Graph Network-based Simulators (GNS) [41] introduced in 2020. Built with the GN framework, it represents the state of a physical system with particles, expressed as nodes in a dynamically evolving graph and computes dynamics

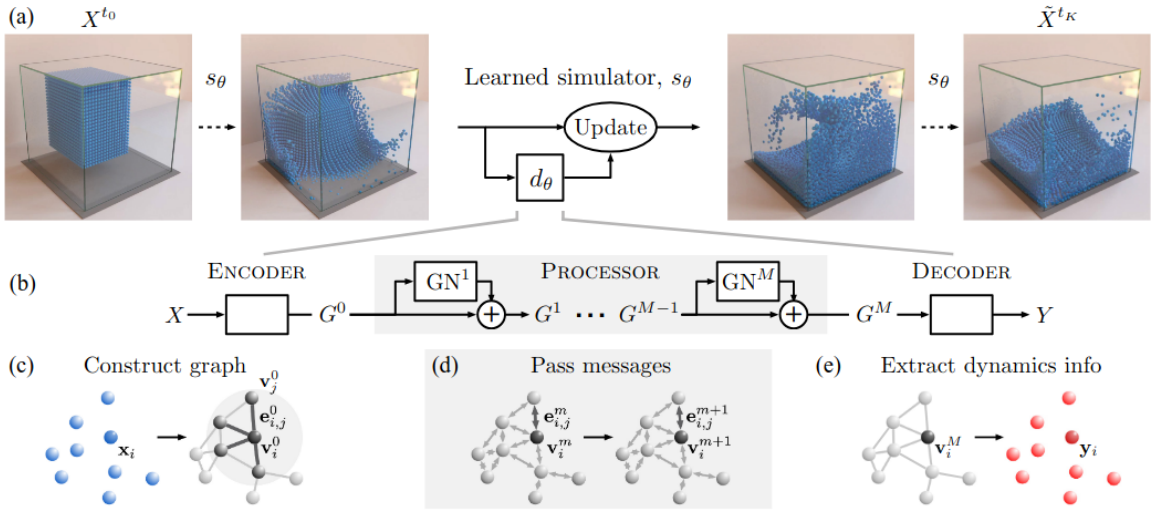

Figure 2: Main figure of [41], picturing the Graph Network-based Simulator framework. (a) Shows the prediction of future states in a fluid simulation using the learned simulator s_θ. (b) Outline the architecture of the dynamics predictor d_θ, with Encoder (c), Processor (d), and Decoder (e).

via learned message-passing. They formalize the framework with s_θ, a learned simulator that uses a dynamic predictor d_θ to compute dynamic information $Y \in \mathcal{Y}$ from the world state $X \in \mathcal{X}$ represented with particles. They consider acceleration as dynamic information and use an Euler integrator to predict the future state of the system. As shown in Figure 2, d_θ adopts an *encode-process-decode* architecture, where the encoder constructs a latent graph G^0 from X to obtain latent node and edge features. Then the processor performs M layers of learned message-passing over the latent graphs $G^0, \ldots, G^M$ to gradually update the latent representation. Finally, a decoder extracts the dynamics information Y from the final latent graph G^M. The model is supervised on one-step predictions of the dynamics information and makes autoregressive predictions during inference to produce a simulation rollout. The robustness of this approach for long-term performance is mainly due to the mitigation of the accumulation of errors with noise added to the training data. This formalism constitutes a foundational framework that can be adapted to any type of physics to be learned. In our work, we build on this framework to design a learned simulator that can learn rigid body physics with spheres.

Extension of GNS to mesh-based simulations. It was further extended in 2021 with MeshGraphNets [35] to learn mesh-based simulations of a wide range of physical systems such as aerodynamics, structural mechanics, and cloth. Within the same framework, the encoder transforms the input mesh into a graph, adding extra world-space edges by spatial proximity, allowing it to handle self-collision for nodes far in mesh space. The processor performs multiple rounds of message passing along mesh edges and world edges, updating node and edge embeddings, and the decoder extracts the dynamics information for each node which is used to obtain the mesh at the next time step. By learning local interactions, this model can generalize the dynamics to much larger spatial domains than seen during training, which is a desirable property of these learn simulators.

2.2.2 Learning rigid body dynamics

Rigid body physics and GNS. Rigid Body dynamics is a particularly interesting scenario for learned simulators. Due to the continuous nature of how deep networks are parametrized, learning discontinuous dynamics such as rigid contact or switch motion modes is challenging. In 2022, the

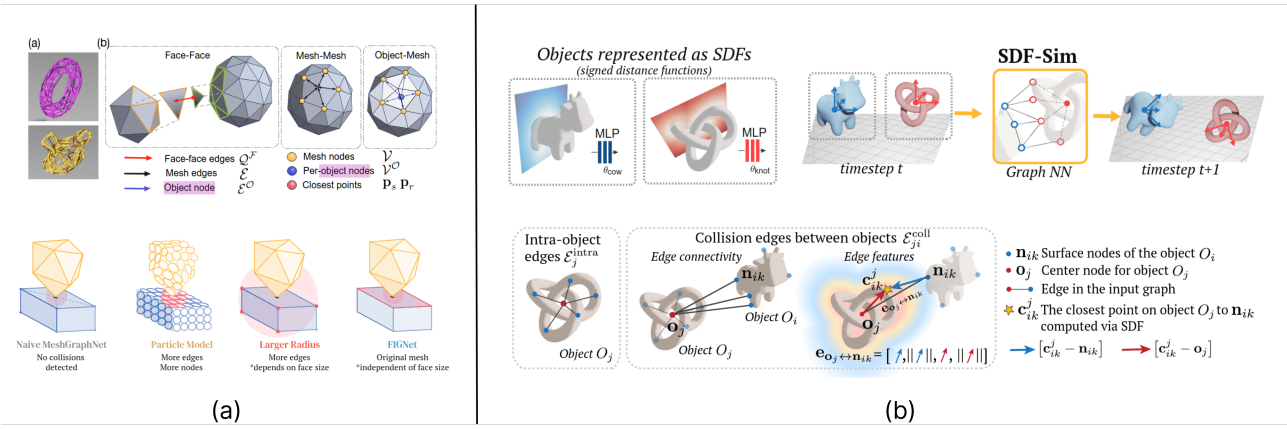

Figure 3: (a) Figures from FIGNet [3], where the top figure shows how mesh nodes and object nodes are represented, and how especially face-face edges are built on the mesh faces if a collision is detected. The bottom one compares FIGNet with other baselines, highlighting how particle methods are too expensive for rigid body dynamics, and that MeshGraphNet and its variants are not adapted to properly capture collisions on mesh with large faces. (b) Figures from SDF-Sim [39] that explain how the method relies on SDFs to build their Graph Network, and then create intra-object and collision edges using the closest point on object's surfaces.

authors of MeshGraphNets have shown that GNS can learn discontinuous, rigid contact dynamics [2] by simply adding a shape-matching step [31] during rollout. After each step, they fit a rigid transformation to the predicted node positions and re-project the nodes to the transformed mesh to enforce shape consistency.

Face Interaction Graph Networks. The same year, they also proposed an extension of MeshGraphNet called Face Interaction Graph Network (FIGNet) [3], that computes interactions between mesh faces rather than nodes. They replace the world edges of MeshGraphNets with face-face edges when faces from distinct objects are within a certain distance. One of their key observation is that in a perfect rigid, information propagates across the rigid at infinite speed. However, with M message passing steps, information can only propagate M hops. Thus, for a fine mesh, when a collision happens it may take many message-passing steps to reach all other nodes in the mesh. Hence, they add an object-level node at the center of each rigid object, which is connected to all of the mesh nodes of the object via bidirectional mesh-object edges. This allows the propagation of information through the rigid in only one message-passing step. Figure 3 (a) shows how they define face-face, mesh-mesh, and object-mesh relations, and how it compares to MeshGraphNet, to a particle model like DPI-Net, and to MeshGraphNet with a larger radius to build the graph. It shows that most methods are not suited to detect collisions on meshes with large faces. The same authors have also proposed a version that scales to real-world scenes with complex meshes, called FIGNet* [25]. The architectural change only consists of removing the mesh-mesh edges and keeping the object-mesh edges and face-face edges. Conceptually, the mesh-mesh edges enable the propagation of messages locally along an object's surface, which is not mandatory for rigid body collisions. This simple change to FIGNet unlocks the ability to train on much more complex scenes than previously possible. It is important to note that the ground truth data used to train FIGNet or FIGNet* comes from the Kubric dataset [14] that relies on the Bullet Physics engine presented in Section 2.1.

Learning rigid-body simulators over implicit shapes. Finally, introduced in May 2024, SDF-Sim [39] overcome the limitations of operating on meshes, which scale poorly to scenes with many

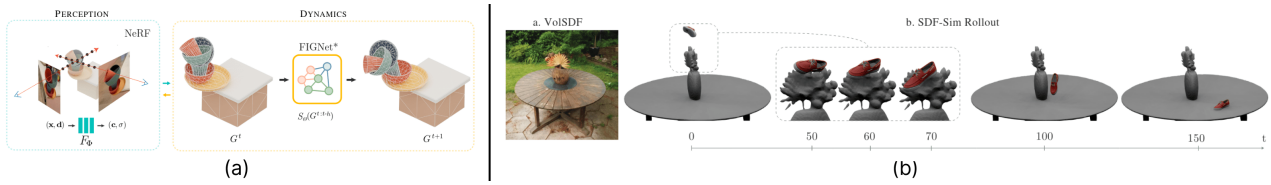

Figure 4: (a) Figure from FIGNet* [25] showing the connection between the perception pipeline and their method to simulate dynamics. (b) Figure from SDF-sim [39], showing the use of VolSDF [54] to extract an SDF from the scene, and then simulate a rollout with an object thrown on the scene.

objects or detailed shapes. As pictured in Figure 3 (b), they use learned signed distance functions (SDFs) to represent the object shapes, which allows them to find the distance and the closest point on the object surface from an arbitrary location in 3D space in constant-time. Since distance queries are the main bottleneck in traditional rigid body simulation and in GNs, this significantly speeds up the process. They keep object-level nodes, while mesh nodes are replaced by surface nodes that are sampled on the object surface. The model is supervised on one-step per-node acceleration. During inference, a rigid transformation is computed from the previous and new point locations, and shape matching [31] is used to update the object position. This last contribution constitutes the state-of-the-art in the field of learned simulators for rigid body dynamics. Although our work does not focus on generalizing to more complex shapes, SDF-Sim provides a promising solution that could be merged with our modeling (Section 3.1), since each SDF could be seen as one particle of the simulation, where its state would contain the shape information.

2.3 Application to computer vision and computer graphics

In this section, we review the applications of traditional simulators (Section 2.1) and learned simulators (2.2) to vision and graphics through the use of 3D representations. We also explore how physics can be learned from images or videos using graph networks or generative models.

Learned simulators applied to captured 3D scenes. A first relevant application to learned simulators is its use in 3D representations for video editing and generation. FIGNet* [25] made a first consequent contribution to expend the application of learned simulators to settings where only perceptual information is available. As shown in Figure 4 (a), they have built a pipeline to integrate the learned simulator with real-world scenes through NeRF [28] during inference. After extracting the meshes and obtaining the rollout trajectory, they derive a set of rigid body transformations that are utilized to edit the original NeRF. It allows a model to be pre-trained from simulation data, and then generalize to real scenes despite the noisy meshes extracted from the NeRFs. In the same optic, due to its scaling capacity SDF-sim [39] can be applied to real-world scenes after extracting SDFs from multi-view images (as shown in Figure 4 (b)). They show that they can simulate assets extracted from vision, where the SDF representation makes it able to capture intricate and complex geometry while keeping a low-memory profile. These approaches open the door to future applications including "virtualization" of real scenes, where users may be interested in editing the scene and simulate possible future outcomes while remaining physically realistic.

Simulating NeRF-encoded objects. Using NeRF, manipulating explicit geometry from reconstructed scenes is challenging due to the implicit nature of the learned volume. Still, PIE-NeRF [10] proposes a method that allows users to interact with a scene in a physically meaningful by integrating

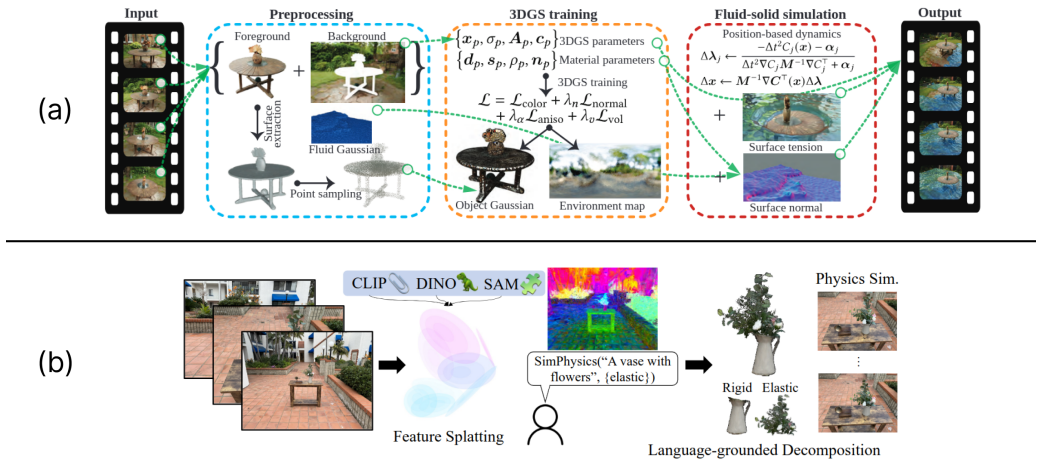

Figure 5: (a) Figure from Gaussian Splashing [9] that pictures the pipeline for training and simulation. Using pre-sampled points and a background image for an environment map, they train 3DGS with PBR material parameters and finally integrate PBD to simulate Gaussian particles in the scene. (b) Figure from Feature Splatting [36] that shows the integration of features from Vision-Languages Models to segment the Gaussians, allowing language-grounded decomposition and then simulation of the appropriate objects.

a traditional simulator with NeRFs. By adaptively sampling the density field encoded with NeRF, and using a quadratic moving least square (Q-GMLS) [27], PIE-NeRF can generate accurate elasto-dynamics. While sampling points on the density fields allows to get an approximation of the scenes's geometry, it remains computationally expensive. The use of a traditional simulator also remains computationally intensive for objects with complex geometry.

Simulating 3DGS-encoded objects. Recent advancements in neural rendering have introduced explicit radiance field representations derived from multi-view images. 3D Gaussian Splatting (3DGS) [19] utilizes anisotropic 3D Gaussians to form a non-structured representation of radiance fields. This explicit representation simplifies the manipulation of both appearance and geometry by treating the Gaussians as discrete particles. A first attempt at editing a scene represented with 3DGS in a physically realistic manner is PhysGaussian [51] introduced at the end of 2023. They employ a custom Material Point Method (MPM) simulator to enrich the 3D Gaussian kernels with physically meaningful kinematic deformation and mechanical stress attributes. A key contribution of PhysGaussian is the integration between physical simulation and visual rendering: both use the same 3D Gaussians as their discrete representation. A few months later, a similar method termed Gaussian Splashing [9] was introduced to simulate solids and fluids in 3DGS scenes using Position-Based Dynamics (PBD). They also go further by enhancing the Gaussian kernels with PBR material parameters (see Figure 5 (a)) to improve the render quality of largely deformed fluids. Their integration of PBD and 3DGS allows them to realistically reproduce surface highlights on dynamic fluids, and facilitate interactions between scene objects and fluids from new views. These methods are limited by the manual segmentation of the Gaussians that are going to be simulated. To circumvent this, Feature Splatting [36] proposes to augment static scenes with semantics attributes to enable language-grounded segmentations and physically realistic augmentations. This allows the segmentation of Gaussians based on semantics and decomposes the object according to physical properties (see Figure 5 (b)). They then rely on MPM to apply the desired dynamics as in PhysGaussian. As point-based simulations are used, these works show that a learned simulator could replace MPM or PBD to apply a desired simulation to 3D Gaussians, to ultimately edit the scene in terms of appearance and geometry.

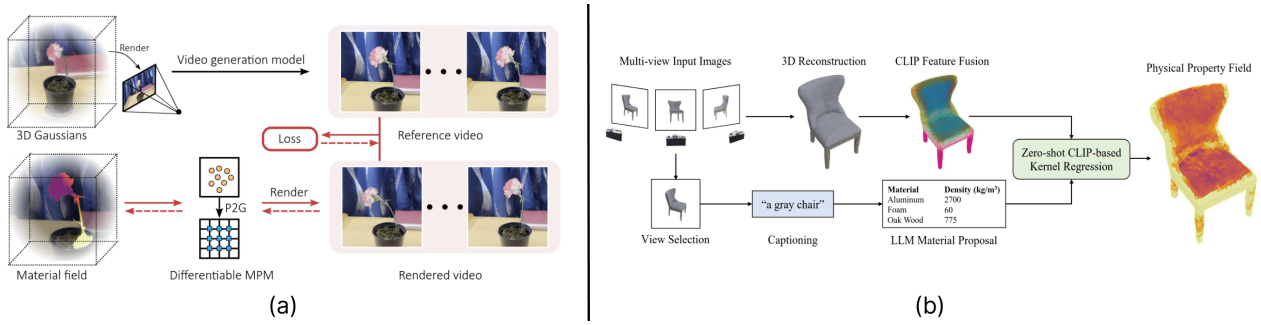

Figure 6: (a) Figure from PhysDreamer [57] to show the integration of video generation model with differentiable MPM. (b) Figure from Nerf2Physics [55] that shows the pipeline to start from multi-view input images of an object, to obtain its physical property field by combining an LLM to propose materials, and CLIP features applied point-wise on a 3D reconstruction.

Learning physics from videos. A different direction towards combining computer vision with learned simulators is to learn physics from videos. Introduced in 2023, 3D-IntPhys [52] is a framework capable of learning 3D-grounded visual intuitive physics models from videos of scenes with fluids. The method relies on NeRF to represent the dynamic scene, and a learned simulator is used to learn the point-based dynamics. The only requirement of the method is access to multiview RGB images, and instance masks acquired using color prior. The idea of 3D-IntPhys is to have a perception module that maps visual observations into an implicit volume representation, which is then subsampled to obtain a particle representation of the environment. Then, the dynamics module which is a GNS models the interaction within and between the objects and predicts the evolution of the particle set. In the same optics, 3DGS have been used to both learn appearance and physics from multi-view videos. Spring-Gauss [58] integrates a 3D Spring-Mass model to reconstruct object appearance and elastic dynamic properties. Their model allows to learn the object's appearance with its physical properties from videos where the object is subject to elastic deformation, such that new simulations under varying initial configurations can be applied to the object. This showcases the method's strength in identifying physical properties from observational data and predicting the dynamics of reconstructed digital assets.

Using generative models to learn physics from videos. Learning physics from videos can also be approached using generative models. In particular, PhysDreamer [57], DreamPhysics [17] and Physics3D [24] propose a similar idea that rely on the plausible object dynamics prior contained in video diffusion models. They use 3DGS to represent a scene with an object, and from a given viewpoint apply an image-to-video generation model to produce a reference video of that object in motion. Using differentiable MPM, they optimize a spatially-varying material field to minimize the discrepancy between the rendered video (obtained with MPM using the current material field) and the reference video (see Figure 6 (a)). This approach allows to learn the constitutive laws of the material points represented with Gaussians, which can then be used with MPM to generate any future desired dynamics. They demonstrate accurate qualitative results for elastic object dynamics. These methods rely strongly on the hypothesis that video generative models are capable of judging physical phenomena accurately in simulation environments. While this could be sound as these models are trained with real-world captured data, it has been recently shown that this assumption is not true. In VideoPhy [4], the authors evaluate the physical commonsense of video generation models and conclude that these models significantly lack the physical commonsense and semantic adherence capabilities. They build a dataset that shows that existing methods are far from being general-purpose world simulators, which questions the use of these generated videos as a reference to optimize for accurate dynamics.

Inferring physical properties using VLMs. While the previous work rely on simulators to learn the constitutive laws from images/videos, the physical properties can also be inferred using off-the-shelf Vision Language Models (VLMs) and Large Language Models (LLMs) without requiring a training step. Nerf2Physics [55] rely on VLM and LLM to estimate a physical property field from objects encoded by NeRFs (see Figure 6 (b)). They can also estimate object-level physical properties like mass, which makes them able to recreate digital twins with accurate physical responses (while only having access the the NeRF representation initially). This type of method could thus be used to infer the required physical properties of objects to be simulated with a pre-trained learned simulator.

2.4 Position of our work

This review raises interesting points about the state of the art and the potential gaps remaining in the literature.

First, we see in Section 2.1 and Section 2.2 that we can use data extracted from traditional physical simulators to train models that can generalize the associated dynamics. In particular for rigid bodies, the Bullet Physics engine [8] seems to be widely used through Kubric [14] or Blender, which uses it as a backend for rigid body dynamics computations.

Furthermore, Section 2.2 shows that general frameworks such as GNS [41] can be adapted for different types of physics. However, although these models can generalize to larger temporal or spatial domains, the question of generalizing physical laws is not addressed especially for rigid body dynamics. For instance, the SOTA method SDF-sim demonstrates results for short rollout and does not numerically assess the adherence to physical laws except for the penetration error between objects. In our work, we aim to quantitatively and qualitatively evaluate the capabilities of learned simulators to learn the underlying physical principles that must be respected. To simplify the analysis, **we focus on the physics of rigid body dynamics with sphere objects** and try not to expand the method to more complex shapes as in SDF-sim or FIGNET. Using spheres as a primitive is not limiting, as shown in HRN [29], they can be assembled hierarchically to form more complex objects. Moreover, we could get inspired by SDF-Sim [39] and incorporate SDFs to represent object shapes within our modeling of the particles (Section 3.1). In this context, each particle would encapsulate the shape information through SDF in its state vector. While these solutions are promising, generalizing to more complex rigid objects is not included in this study.

Finally, our work does not explicitly link learned simulators with computer graphics and vision yet. However, Section 2.3 shows that it is entirely feasible to infer physical parameters from 3D scenes using methods like Nerf2Physics [55]. We can then represent a scene using 3DGS to obtain an explicit representation of the scene made of simulatable particles as in PhysGaussian [51]. Using features from languages and vision as in [36], we could then select the simulatable particles. Finally, a pre-trained learned simulator could be used to apply the desired dynamics according to the inferred physical parameters, and the selected particles to be simulated. Although this application is out of the scope of this study, we highlight the plausibility of such an approach thanks to the recent literature.

3 Learning rigid body physics with GNNs

To study the capability of Graph Networks to learn physical laws, we focus on the simulation of rigid body dynamics, specifically using spheres as our primary objects. In this context, we aim to build a learned simulator that accurately models the dynamics of spheres while respecting key physical principles:

- **Gravity**: The simulator should accurately model the gravitational force acting on the rigid bodies, ensuring that objects experience a constant acceleration towards the ground.
- **Kinematics**: The simulator must integrate the equations of motion for each sphere, and be able to predict realistic future states of the system (positions, velocities, or accelerations).
- **Energy conservation/dissipation**: The GNN should correctly model the exchange of energy between the objects (dynamic or static), and ensure that the system predicts a gradual energy loss over time, representing dissipation until the system reaches a state of rest or equilibrium.
- **Normal force**: The GNN must accurately model the normal forces that occur during sphere-sphere or sphere-surface interactions. These forces prevent interpenetration and ensure that the sphere behaves realistically when in contact, such as resting on a surface without sinking or sliding unnaturally.

In this section, we present the method developed to learn rigid body physics using Graph Networks. We begin with an overview of the modeling approach, which is subdivided into two levels: the system level, where we describe the overall physical system and its dynamics, and the particle-based state level, which focuses on the features that represent the state of the system at a given time. Then, we detail the architecture of our Graph Network that incorporates strong inductive biases to learn the physical laws considered in rigid body dynamics. We outline how the physical system is encoded into a graph, and processed to learn the system's dynamics. Finally, we cover the training process explaining how the model is trained to predict future states of the system, and conclude with the inference phase, where we describe how the trained GNN is applied to make predictions based on new input data.

3.1 Modeling

3.1.1 System level

Let's first detail our general learnable simulation framework, extended from GNS [41]. We assume that $X^t \in \mathcal{X}$ is the state of the world at timestep t and that $\mathbf{X}^{t_{0:K}} = (X^{t_0}, X^{t_1}, \ldots, X^{t_K})$ is a trajectory of states over $K+1$ timesteps that results from physical dynamics (real or simulated). The state of the world X^t includes both the dynamic states of the particles and the global environment state. This global environment state consists of static surfaces such as the ground, walls, or any surface that can affect the dynamic, but remain unchanged through time. A simulator $\text{sim} : \mathcal{X} \to \mathcal{X}$ maps preceding states to causally consequent future states, and allows to obtain a simulated rollout trajectory as $\tilde{\mathbf{X}}^{t_{0:K}} = (X^{t_0}, \tilde{X}^{t_1}, \ldots, \tilde{X}^{t_K})$, which is computed iteratively by $\tilde{X}^{t_{k+1}} = \text{sim}(\tilde{X}^{t_k})$. We aim to develop a learned simulator sim_θ that computes dynamics information $Y^{t_{k+1}} \in \mathcal{Y}$ with $d_\theta : \mathcal{X} \to \mathcal{Y}$, where θ can be optimized. The dynamic information is then used to update the state of the system and obtain $\tilde{X}^{t_{k+1}}$.

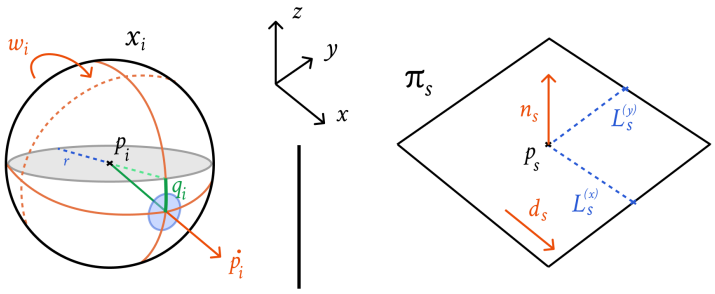

Figure 7: Each particle corresponds to a 3D sphere (left) of radius r, and is defined by its position $\mathbf{p}_i$, velocity $\dot{\mathbf{p}}_i$, angular velocity ω_i and quaternion $\mathbf{q}_i$ that encodes the rotation. The static surfaces are planes (right) defined by a center point $\mathbf{p}_s$, a normal $\mathbf{n}_s$, a reference direction $\mathbf{d}_s$, and the lengths $L_s^{(x)}, L_s^{(y)}$.

3.1.2 State level

Our approach is based on a particle-based representation of the physical system, which remains consistent with the methods developed to apply physical simulations on 3D scenes. The particles correspond to spheres of radius r and mass m (See Figure 7, left). A state of the world $X^{t_k} = (\mathbf{x}_1^{t_k}, \dots, \mathbf{x}_N^{t_k}, \Pi)$ is determined by the state of the N particles represented by $\mathbf{x}_i^{t_k}$, and the global environment state represented by the set of static surfaces $\Pi = \{\pi_1, \dots, \pi_S\}$. Each particle's state vector contains the necessary information to infer the dynamics. In our setting, this vector represents: position $\mathbf{p}_i^{t_k} \in \mathbb{R}^3$ and rotation (with a quaternion) $\mathbf{q}_i^{t_k} \in \mathbb{R}^4$, the H previous velocities $\dot{\mathbf{p}}_i^{(H)} = [\dot{\mathbf{p}}_i^{t_k}, \dot{\mathbf{p}}_i^{t_{k-1}}, \dots, \dot{\mathbf{p}}_i^{t_{k-H}}]$ and H previous angular velocities $\omega_i^{(H)} = [\omega_i^{t_k}, \omega_i^{t_{k-1}}, \dots, \omega_i^{t_{k-C}}]$, with $\dot{\mathbf{p}}_i^{t_k} \in \mathbb{R}^3$, $\omega_i^{t_k} \in \mathbb{R}^3$ and H a hyperparameter of the method. Note that for $H = 0$, we only consider the current velocity and angular velocity at time t_k. We can also optionally consider features that represent the sphere's properties (mass and radius) $f \in \mathbb{R}^d$. For state vector at time t_k we thus have $\mathbf{x}_i^{t_k} = [\mathbf{p}_i^{t_k}, \mathbf{q}_i^{t_k}, \dot{\mathbf{p}}_i^{(H)}, \omega_i^{(H)}, f_i]$. This framework also allows the addition of particle feature a_i that represents the material type (expressing a certain friction parameter for example) as in GNS [41], where they use a learned embedding of fixed size for the material type.

The global environment state here corresponds to the set of static surfaces Π, where each surface π_s is uniquely described as a bounded plane within the simulated world (See Figure 7, right). Using planes to represent surfaces offers both simplicity and flexibility, allowing complex surfaces to be modeled as compositions of multiple planes. To define a plane, we use a center point $\mathbf{p}_s \in \mathbb{R}^3$ that specifies the plane's position and a normal vector $\mathbf{n}_s \in \mathbb{R}^3$ that defines its orientation by indicating the direction perpendicular to the surface. Additionally, we use a reference direction vector $\mathbf{d}_s \in \mathbb{R}^3$ which corresponds to the unit vector along the x-axis of the plane's local coordinate system, to represent the plane's rotation around the normal. The lengths $L_s^{(x)} \in \mathbb{R}$ and $L_s^{(y)} \in \mathbb{R}$ define the plane's dimensions along the reference direction and the orthogonal direction on the plane, respectively. Together, these components fully describe static planes as $\pi_s = [\mathbf{p}_s, \mathbf{n}_s, \mathbf{d}_s, L_s^{(x)}, L_s^{(y)}]$, allowing to accurately model the interactions between particles and the surfaces in the environment.

3.2 Architecture details

In the GNS framework [41], the way that particle-particle interactions are modeled determines the quality and generality of the simulation method (type of physics to simulate, which scenarios are

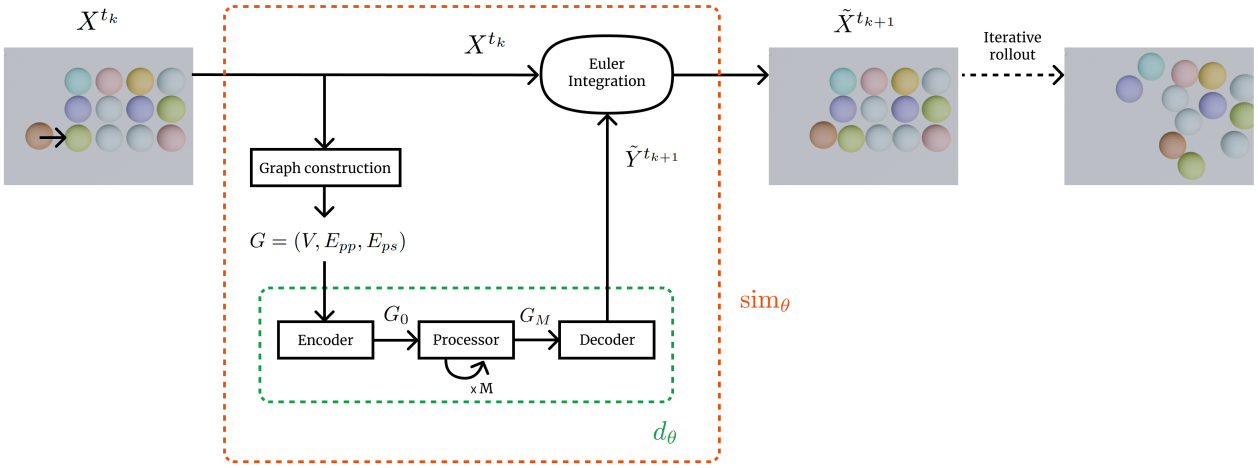

Figure 8: Starting from the world state X^{t_k}, the learned simulator sim_θ predicts the next state $\tilde{X}^{t_{k+1}}$, eventually producing a rollout through iterative predictions. The learned simulator sim_θ is made of 3 components. A Graph construction method embeds the world state into a specific graph instance, linking spheres with potential interactions. The dynamic predictor d_θ predicts the relevant dynamic information $\tilde{Y}^{t_{k+1}}$, here the linear and angular velocities. Finally, the world state at the next timestep is computed through an Euler integrator.

considered, etc.). In this case of rigid body dynamics, our learned simulator sim_θ relies on a Graph Neural Network d_θ with an *Encode-Process-Decode* architecture, which is particularly adapted as shown in the recent approaches to learn physics with graphs (Section 2.2). In this section, we present our method and architecture summarized in Figure 8, highlighting key choices to impose strong inductive biases to effectively learn rigid body physics with spheres. The idea is first to construct a meaningful graph that will be used to compute interactions between particles and with static surfaces (Section 3.2.1). Using the right features and connecting the relevant particles is crucial to accurately let the network learn the dynamics from data (Section 3.2.2). The constructed graph will then be encoded into a latent graph that is invariant to absolute spatial location (Section 3.2.3). A processor will compute interactions among nodes via multiple steps of learned message-passing with MLPs (Multi-Layer Perceptrons), to generate a sequence of recursively updated latent graphs. This step allows information to propagate by respecting constraints that enforce the adherence to physical laws (Section 3.2.4). Finally, a decoder will extract the dynamics information from the nodes of the latent graph (Section 3.2.5), which can be used for supervision, or to generate a rollout recursively through discrete integration.

3.2.1 Graph construction

The first step in predicting the dynamics at the next timestep for the world state X^{t_k} is to construct a graph using the particle states $\mathbf{x}_i^{t_k}$ and the global environment state Π. This process serves as an initial encoding step, where we map X^{t_k} into a specific (multi)graph instance $G = (V, E_{pp}, E_{ps})$ shown in Figure 9. Here:

- V is the set of nodes, where each node corresponds to a particle $\mathbf{x}_i^{t_k}$.
- E_{pp} is the set of directed edges representing particle-to-particle interactions, where each edge $(\mathbf{x}_i^{t_k}, \mathbf{x}_j^{t_k}, \mathbf{r}_{i,j}^{pp})$ captures interactions between particles $\mathbf{x}_i^{t_k}$ and $\mathbf{x}_j^{t_k}$, along with the associated edge information $\mathbf{r}_{i,j}^{pp}$.

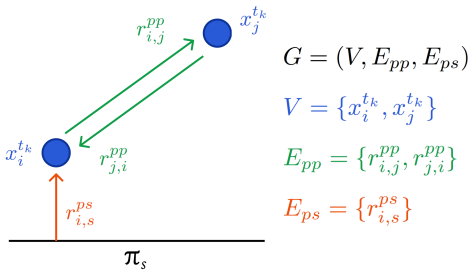

Figure 9: The graph instance $G = (V, E_{pp}, E_{ps})$ summarized in a simple 2D case with two nodes and one static surface. Particles corresponds to nodes $\mathbf{x}_i^{t_k}$ in blue, the particle-to-particle (pp) edges $\mathbf{r}_{i,j}^{pp}$ in green, and the particle-to-surface (ps) edges $\mathbf{r}_{i,s}^{ps}$ in red.

- E_{ps} is the set of directed edges representing particle-to-surface interactions, where each edge is a tuple $(\mathbf{x}_i^{t_k}, \pi_s, \mathbf{r}_{i,s}^{ps})$ that captures interactions between a particle $\mathbf{x}_i^{t_k}$ and a static surface $\pi_s \in \Pi$, along with the associated edge information $\mathbf{r}_{i,s}^{ps}$.

Multiple strategies can be adopted to create the graph. This step is crucial as at a given timestep, only the connected particles can interact and exchange information. The simplest way is to use a nearest-neighbor algorithm to connect the particles according to their proximity with a fixed connectivity radius, to identify potential interactions. For rigid body dynamics with spheres, this connectivity radius could be relatively small to only connect spheres that are going to interact with other spheres or with surfaces through collisions: $R = \epsilon$ (See Figure 10, left). The connection is made through two directed edges to capture the bidirectional interaction. In the following text, we reference this method of graph creation with ϵ-graph connectivity. However, this method is not robust, as the effectiveness of the connectivity radius is highly sensitive to the timestep size Δt in the ground truth simulated data. When the timestep size is large, particles travel greater distances between steps, while smaller timestep sizes result in shorter movements. For particles moving at high velocities, it is crucial to account for their speed and timestep size to capture potential interactions accurately. Additionally, a significant limitation of this method is its inability to account for multiple interactions that can occur within a single timestep, such as when a fast-moving particle collides with a stationary one, which in turn could impact another distant particle. Through empirical search, one can find a threshold that is suited for the scenario and the timestep size of the simulations, which is the approach employed in GNS [41]. This method gives a subset the potential interactions that could happen at the next time-step, as it is likely that interactions happening at high speed will be missed.

To address the previous limitations, a more robust graph construction method is required, and it should reliably capture the potential interactions occurring within the current timestep. A reliable approach to ensure no potential interactions are missed is first to calculate a super-set of possible interactions. This can be done by estimating the maximum distance a particle could travel within the current timestep with an explicit Euler integrator, using the maximum current velocity among all particles and the timestep duration (See Figure 10, middle). If $V_{\max} = \max_i(\|\dot{\mathbf{p}}_i^{t_k}\|_2)$ denotes the magnitude of the maximum velocity among all particles at timestep t_k, the query radius can be computed as $R = V_{\max} \cdot \Delta t$, which gives the maximum distance that spheres can travel at t_k. We call this method of creating the graph the $V_{\max}$-graph connectivity. This approach gives a super-set of potential interactions. In fact, it is not informative to connect more spheres as it already captures all the possible interactions that could happen at the following time step.

In contrast to the two previous approaches that use a fixed query radius, we also design an adaptive method to connect the nodes depending on each particle's velocities. By independently computing the

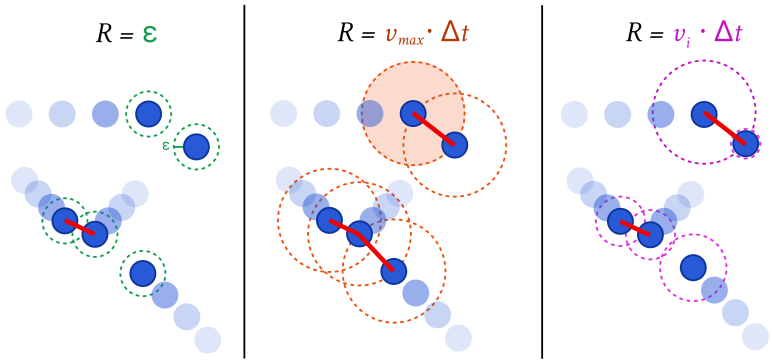

Figure 10: We design 3 methods to embed the world state into a graph. The element that differs for each method is the connectivity radius employed to connect the particles. For ϵ-graph connectivity (left), a fixed and small threshold ϵ is used for all particles to only capture collisions when they happen. The V_{max}-graph connectivity (middle) consists of using the approximate distance that the fastest sphere could travel during one timestep and use it as a connectivity radius for all spheres. Finally, the V_i-graph connectivity (right) consists of having an adapted radius to each sphere depending on its velocity.

distance that each particle could travel, we can adapt the connectivity radius accordingly (See Figure 10, right). Let $V_i = \|\dot{\mathbf{p}}_i^{t_k}\|_2$ be the magnitude of the velocity of the i^{th} particle, its connectivity radius is $R_i = V_i \cdot \Delta t$. We call this method V_i-graph connectivity. It provides a compromise between the two previous fixed connectivity radius methods, by only considering per-sphere potential interactions.

These methods allow to obtain the connected nodes (spheres) in E_{pp}. To connect spheres with surfaces in E_{ps}, the same approaches can be employed. We consider the closest point on the surface relative to the sphere and can apply either $\epsilon/V_{\text{max}}/V_i$-graph connectivity to connect the two objects. In practice, finding the closest point on the surface is easy using the plane's description detailed in Section 3.1.2. If a sphere goes in the direction of a surface, we can project its position on the local plane coordinate system and create an edge from the surface to the sphere if the distance $d = (\mathbf{p}_i^{t_k} - \mathbf{p}_s) \cdot \mathbf{n}_s$ is within the connectivity radius.

To make the model learn gravity, we additionally connect each sphere to the closest point on the ground. This connection represents the gravitational pull or force acting on the spheres, simulating the effect of gravity. This connection allows the model to account for gravitational interactions, encouraging the spheres to move in a way that reflects realistic gravitational dynamics.

3.2.2 Graph edge features

Once the graph is created, we know which particles and surfaces are going to be linked due to a potential future interaction. The information stored on these edges should reflect the interaction, and help the future network to predict the next state.

For the edge information between particles i and j, we simply store the relative displacement along with its magnitude, i.e., $r_{i,j}^{pp} = [(\mathbf{p}_i^{t_k} - \mathbf{p}_j^{t_k}), \|\mathbf{p}_i^{t_k} - \mathbf{p}_j^{t_k}\|_2] \in \mathbb{R}^4$ as in GNS [41]. Along with the sphere's states, this information is sufficient for the model to predict if a collision will happen in the next steps.

For the edge information between particle i and surface π_s, we need to ensure that the network will have enough information to predict the next velocity and angular velocity for the spheres, according to the plane orientation and the sphere direction. As the sphere direction is encoded in the velocity,

it would be sufficient to use the velocity and the normal of the plane to manually compute the next states in case of a bounce. From this assumption, we store the sphere velocity, plane normal, and the displacement between the sphere and the closest point on the surface with its magnitude. Hence, we have $r_{i,s}^{ps} = [\dot{\mathbf{p}}_i^{t_k}, \mathbf{n}_s, ((\mathbf{p}_i^{t_k} - \mathbf{p}_s) \cdot \mathbf{n}_s) \times \mathbf{n}_s, |(\mathbf{p}_i^{t_k} - \mathbf{p}_s) \cdot \mathbf{n}_s|] \in \mathbb{R}^{10}$. This constitutes the minimal information to deduce if a collision wœœill happen, and if so, how the sphere is supposed to respond to the collision. Having the relative distance to the surface is also necessary to learn normal force, and prevent the spheres from crossing it, but also to learn friction, as the model will know when the sphere is rolling on the surface. Moreover, as all spheres are constantly connected to the ground for gravitational dynamics, each sphere remains aware of the straight direction they should follow when falling.

These graph-creation methods put strong inductive biases on the model to make it learn gravitational dynamics, normal force, collisions, and friction. However, a main physical principle is still lacking: the exchange of energy during interactions in the graph should be consistent with physical laws. This will be ensured by design in the Graph Network processing steps (Section 3.2.4).

3.2.3 Graph encoder details

Now that the world state X^{t_k} is mapped to a meaningful (multi)graph, it will be encoded into a latent graph representation. The encoder is made of 3 MLPs, that separately encode node features $\mathbf{x}_i^{t_k}$, particle-to-particle edge features $\mathbf{r}_{i,j}^{pp}$, and particle-to-surface edge features $\mathbf{r}_{i,j}^{ps}$ into latent vectors of size h:

- Latent node features $\mathbf{v}_i = \varepsilon^v(\mathbf{x}_i^{t_k}) \in \mathbb{R}^h$
- Latent particle-to-particle edge features $\mathbf{e}_{i,j}^{pp} = \varepsilon^{pp}(\mathbf{r}_{i,j}^{pp}) \in \mathbb{R}^h$
- Latent particle-to-surface edge features $\mathbf{e}_{i,s}^{ps} = \varepsilon^{ps}(\mathbf{r}_{i,s}^{ps}) \in \mathbb{R}^h$

In practice, when encoding the node feature with ε^v, we mask the absolute position $\mathbf{p}_i^{t_k}$ and quaternion $\mathbf{q}_i^{t_k}$ information within $x_i^{t_k}$ such that $\mathbf{v}_i = \varepsilon^v([\dot{\mathbf{p}}_i^{(H)}, \omega_i^{(H)}, f_i])$. As shown in GNS [41], masking the position imposes an inductive bias of invariance to absolute spatial location. Hence, when encoding a state vector, the model does not have information about the current position and rotation, for the processing step and predicting the dynamics. As the underlying physical laws that we want to learn are invariant to spatial position, this encoding scheme is consistent with this invariance.

3.2.4 Graph processor details

The processor computes the interactions among nodes by M steps of learned message passing. This corresponds to using a stack of M Graph Net blocks [40], where each block contains a separate set of network parameters. Each block is applied in sequence to the output of the previous block: $G^{m+1} = GN_{\text{block}}(G^m)$; updating the node features $\mathbf{v}_i$, and particle-to-particle edge features $\mathbf{e}_{i,j}^{pp}$, while $\mathbf{e}_{i,s}^{ps}$ remains identical through the M steps. After M steps of sequential processing, we obtain a final latent graph G^M that can be decoded.

The GN block is based on learned message passing, i.e., particle messages and surface messages are constructed on the directed edges using MLPs, and then aggregated on the receiver nodes. The messages and aggregated messages are then used to update the latent edge and node features using MLPs with a residual connection.

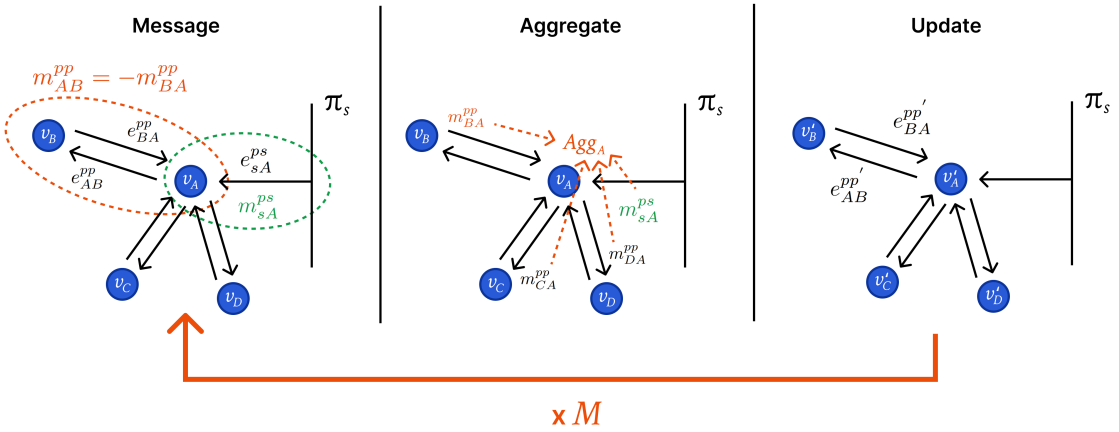

Figure 11: The graph processor consists of 3 main steps. First, messages are built on each edge using the connected vertices and the latent edge feature with an MLP (left). The received messages are then aggregated on each node $\mathbf{v}_i$ (middle), and used to update the latent node features into $\mathbf{v}_i'$ (right) through an MLP with a residual connection. Each latent particle-to-particle edge feature $\mathbf{e}_{i,j}^{pp}$ is also updated into $\mathbf{e}_{i,j}^{pp\prime}$ based on its associated message. These 3 steps are repeated M times iteratively to progressively update the graph.

Messages construction. Messages are generated using MLPs, with separate networks for edges between particles and for edges between particles and surfaces (Figure 3.2.4, left). Specifically, the message for a particle-particle interaction is defined as

$$\mathbf{m}_{ij}^{pp} = \text{MLP}(\mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{ij}^{pp}), \text{ and } \mathbf{m}_{ji}^{pp} = -\mathbf{m}_{ij}^{pp}$$

while the message for a particle-surface interaction is given by

$$\mathbf{m}_{is}^{ps} = \text{MLP}(\mathbf{v}_i, \mathbf{e}_{is}^{ps}).$$

Here, we manually force that the message from particle i to j is the negative of the message from j to i, reflecting Newton's third law. This law states that if two bodies exert forces on each other, these forces have the same magnitude but in opposite directions. Hence, this inductive bias encodes the symmetric nature of the interactions within the latent information passed between particles.

Message aggregation. Now that each edges are associated with a message, each node aggregates its received messages:

$$\text{Agg}_i = \sum_{j \in \mathcal{N}^p(i)} \mathbf{m}_{ji}^{pp} + \sum_{s \in \mathcal{N}^s(i)} \mathbf{m}_{si}^{ps}$$

where $\mathcal{N}^p(i)$ and $\mathcal{N}^s(i)$ denotes the connected neighboring particles and surfaces respectively (Figure 3.2.4, middle). This aggregation allows the node to gather and integrate all the relevant interaction information from both neighboring particles and adjacent surfaces. This is meant to capture the combined effects of particle-to-particle and particle-to-surface interactions in the overall dynamics.

Node and edge updates. The nodes and particle-to-particle edges can now be updated with the aggregated information and messages (Figure 3.2.4, right):

$$\mathbf{v}_i' = \mathbf{v}_i + \text{MLP}(\mathbf{v}_i, \text{Agg}_i)$$
$$\mathbf{e}_{ij}^{pp\prime} = \mathbf{e}_{ij}^{pp} + \mathbf{m}_{ij}^{pp}.$$

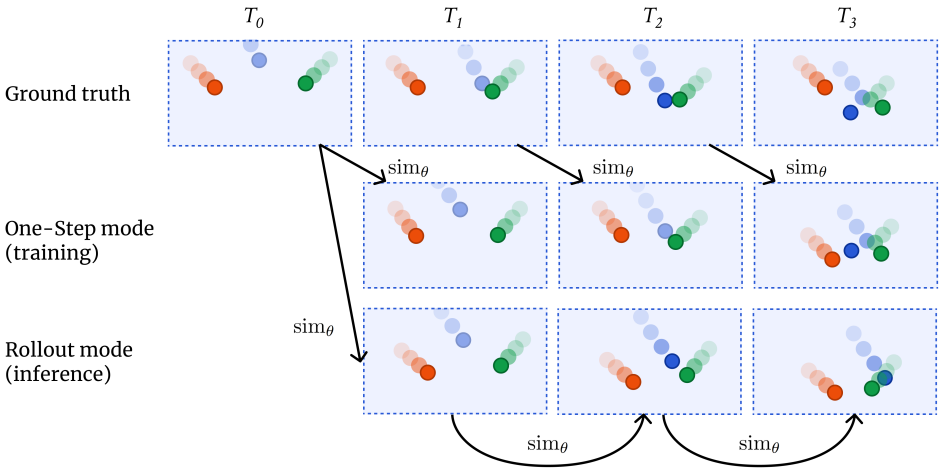

Figure 12: During training, the method relies on ground truth observed data at several time points to be supervised on one-step predictions. During inference, we generate a rollout by giving an initial state to the learned simulator, which will then iteratively predict the next states.

Note that the latent particle-to-surface edge feature $\mathbf{e}_{is}^{ps}$ is not updated. This is to ensure that the feature remains invariant and continues to represent the static, non-dynamic properties of the surface interactions. Hence, the model maintains a clear distinction between the dynamic, mutable aspects of particle interactions and the fixed, unchanging nature of the surface properties. This is consistent with the physical properties of surfaces that we aim to model, where friction is constant and does not evolve over time.

The updated node and edge features are then fed to the next GN block that will process this new graph, until arriving at the final latent graph G^M. The hyperparameter M is thus important as it determines the depth of message passing: a larger M allows for more iterations of node and edge updates through aggregation with neighbors, enabling the model to capture more complex interactions and long-range dependencies. Therefore, M must be chosen according to the complexity of the physical system and the desired runtime, as computation time scales linearly with M [41]. We explore this hyperparameter with experiments detailed in Section 4.3.3.

3.2.5 Graph decoder details

For predicting the time t_{k+1} state from the time t_k input, the decoder simply consists of an MLP δ^v that transforms the final latent node $\mathbf{v}_i$ into output features that represents velocity and angular velocity:

$$\tilde{y}_i^{t_{k+1}} = (\dot{\mathbf{p}}_i^{t_k+1}, \omega_i^{t_k+1}) = \delta^v(\mathbf{v}_i)$$

This approach directly maps the processed latent representation of each node to the physical quantities of interest, here the linear and angular velocities that can be used to compute the new positions and rotations to eventually obtain $\tilde{X}^{t_{k+1}}$.

3.3 Training

The prediction targets for supervised learning only consists of the per-particle velocities and angular velocities at the next timesteps, i.e., $Y^{t_{k+1}} = (\mathbf{y}_1^{t_{k+1}}, \ldots, \mathbf{y}_N^{t_{k+1}})$, with $\mathbf{y}_i^{t_{k+1}} = (\dot{\mathbf{p}}_i^{t_k+1}, \omega_i^{t_k+1})$. During

training, a batch of random state pairs $(X^{t_k}, X^{t_{k+1}})$ is sampled from training trajectories, and fed into the model to compute $\tilde{Y}^{t_{k+1}}$. The optimization is driven by the L_2 loss

$$\mathcal{L}_{\text{1-step}}(X^{t_k}, X^{t_{k+1}}; \theta) = \|d_\theta(X^{t_k}) - Y^{t_{k+1}}\|_2^2.$$

Training on one-step predictions (Figure 12) imposes a strong inductive bias that physical dynamics are Markovian, and should operate the same at any time during a trajectory.

As rigid body physics constitutes a complex and chaotic simulation system, small changes in the initial conditions can yield completely different dynamics. Traditional simulators can mitigate the error accumulation over long rollout, which is a desirable property for a learned simulator. As the model is trained on one-step dynamics predictions, it is limited to clean and non-corrupted data. Hence, if we forward an updated state with the model's previous prediction, it will accumulate this error and rapidly diverge over the rollout as the input will rapidly move outside the training distribution. To make the model more robust, we follow a common practice in the literature: the particle's states given as input during training are corrupted with Gaussian noise, adjusting the position, rotations, and previous linear/angular velocities. The prediction target remains unchanged, so the model learns to predict the right dynamics information with noisy input. This helps to make the training distribution closer to the one generated during rollout.

Finally, we also propose a regularization term to bias the model towards learning energy dissipation of the global system. The total positional energy of the system at time t_k consists of the sum of the potential and kinetic energies of all spheres:

$$\mathcal{E}_{\text{total}}^{t_k} = \underbrace{\sum_{i=1}^{N} \frac{1}{2} m \dot{\mathbf{p}}_i^{t_k} \cdot \dot{\mathbf{p}}_i^{t_k}}_{\mathcal{E}_{\text{kinetic}}} + \underbrace{\sum_{i=1}^{N} m g h_i}_{\mathcal{E}_{\text{potential}}}$$

where m is the mass of the spheres, g the gravity constant and h_i the height of the i^{th} sphere.

Although each sphere in the system may gain or lose energy through collisions—resulting in an open-loop dynamic at the level of individual spheres, when considering the global system as a whole, the total energy should either remain constant or decrease over time due to energy dissipation. The system is not expected to gain energy in the absence of external forces. Therefore, we aim to encourage the model to adhere to this principle of energy conservation and dissipation in the global system. After predicting $\tilde{Y}^{t_{k+1}}$ with d_θ, we use discrete integration to obtain $\tilde{X}^{t_{k+1}}$, which can then be used to compute $\mathcal{E}_{\text{total}}^{t_{k+1}}$. Hence, we have

$$\mathcal{L}_{\text{energy}} = \max(\mathcal{E}_{\text{total}}^{t_{k+1}} - \mathcal{E}_{\text{total}}^{t_k}, 0).$$

where the max ensures that only increases in energy contribute to the loss. This penalizes the model if the total energy of the system increases after an update, thus encouraging the system to either maintain or dissipate energy, aligning with physical principles of energy conservation and dissipation. Note that the same constraint can also be enforced for the rotational energy, which was not incorporated in this study. The final loss function incorporates this regularization, mitigated with a parameter λ:

$$\mathcal{L} = \mathcal{L}_{\text{1-step}} + \lambda \mathcal{L}_{\text{energy}}$$

It is important to highlight that only the positions $\mathbf{p}_i^{t_k}$ and rotations $\mathbf{q}_i^{t_k}$ are required to train the model, as velocities and angular velocities can be derived using finite differences:

$$\dot{\mathbf{p}}_i^{t_{k+1}} = \frac{\mathbf{p}_i^{t_{k+1}} - \mathbf{p}_i^{t_k}}{\Delta t}$$

$$\omega_i^{t_{k+1}} = \frac{2 \cdot \mathrm{Im}\left(\log\left(\mathbf{q}_i^{t_{k+1}} \otimes \left(\mathbf{q}_i^{t_k}\right)^{-1}\right)\right)}{\Delta t}$$

where Δt is the timestep size, $\otimes$ denotes the quaternion multiplication, q^{-1} the quaternion inverse, and the log applies the definition of the logarithm for quaternions. This prior makes the model versatile, as it could use simulation data, or tracked positions and rotations from real-world videos as input.

3.4 Inference

During inference (Figure 12), the model only needs access to an initial state of the system X^{t_k} so d_θ predicts the dynamics information $\tilde{y}_i^{t_{k+1}} = (\dot{\mathbf{p}}_i^{t_k+1}, \omega_i^{t_k+1})$. To update the state of the spheres (position and rotation), we use a first-order semi-implicit Euler integration method:

$$\begin{aligned}
\mathbf{p}_i^{t_{k+1}} &= \mathbf{p}_i^{t_k} + \Delta t \cdot \dot{\mathbf{p}}_i{}^{t_{k+1}} \\
\mathbf{q}_i^{t_{k+1}} &= \mathbf{q}_i^{t_k} + \frac{1}{2}\left(\mathbf{q}_i^{t_k} \otimes (0, \omega_{i,x}^{t_{k+1}}, \omega_{i,y}^{t_{k+1}}, \omega_{i,z}^{t_{k+1}})\right) \cdot \Delta t
\end{aligned}$$

where the integration is adapted to rotational dynamics to update the quaternion. The quaternion multiplication is applied between the current rotation, and a quaternion defined with the 3 components of the angular velocity and a real part equal to 0. To ensure a valid rotation, the quaternion is also normalized to the unit norm. This method provides a straightforward and computationally efficient way to integrate both the translational and rotational dynamics of a rigid body over time, allowing to obtain the next position and rotation of the spheres. This new state can then be fed to the model, which will iteratively predict the next states until t_K, leading to a complete rollout $\tilde{\mathbf{X}}^{t_{0:K}} = (X^{t_0}, \tilde{X}^{t_1}, \ldots, \tilde{X}^{t_K})$.

4 Experiments

In this section, we present the experiments conducted to evaluate the performance of our method and the results obtained. Section 4.1 describes the experimental setup, including the dataset generation, model configuration, and training process. In Section 4.2 we introduce the metrics used to assess both the physical realism and the predictive accuracy of the model. Section 4.3 presents the quantitative results, comparing different hyperparameters according to our metrics, and evaluating the relevance of some components of our model. Finally, Section 4.4 provides qualitative results, showcasing the model's behavior in complex scenarios and its ability to generalize beyond the training data. All qualitative results discussed are available on this Google drive.

4.1 Experimental setup

To train and evaluate the learned simulators developed within this framework, we created a pipeline that connects Blender's rigid body simulator (relying on Bullet, see Section 2.1) with Python.

Data generation. For this study, we automated the creation of 100 trajectories consisting of a random number of spheres (between 1 and 300) initialized above the ground at random heights (between 0 and 150m), that fall within a random-sized restricted area (cuboid with length/width between 0 and 100m) delimited by 4 walls for 2000 timesteps, with $\Delta t = 0.1$. This duration allows the spheres to fall, bounce, and collide with each other and with the ground (see Figure 13). These 100 trajectories are used for training, while 10 other trajectories generated following the same approach are used to make a test set. Moreover, to better evaluate the generalization of the physical laws that should be learned, 4 out-of-distribution (OOD) trajectories were created specifically to test different aspects of collisions, energy, and response to surfaces (Figure 14).

Training. We trained each model on 3 seeds for 500 epochs with a batch of 256 world state pairs, randomly selected within all trajectories, without energy regularization by default. To ensure robustness, noise sampled independently from $\mathcal{N}(0, 0.01)$ was added to the input state features. All MLPs consist of an input layer with 128 units and ReLU activation, followed by an output layer of the same size without activation, resulting in latent graph features of dimension 128. These hyperparameters were found during preliminary experiments, and are not explored in this study. We quantitatively evaluated the effect of the number of message-passing steps M, the size of the state history H, and the method of graph creation ($\epsilon/V_{\max}/V_i$-graph connectivity). Unless otherwise specified, all models are trained with $H = 2$, $M = 5$, and use the V_i-graph connectivity to create the graph. Models are trained with NVIDIA Tesla V100-16GB GPUs, taking between 8 to 15 hours depending on M (which scales linearly with the compute time).

4.2 Metrics

When evaluating a model, we want to assess how it has learned the considered physical laws detailed in Section 3, i.e., Gravity, kinematics, energy conservation/dissipation, and normal force. For this purpose, specifically designed metrics that target these laws.

To assess the respect of normal forces, we evaluate the sphere's intersections with other spheres and with surfaces. Hence, we measure:

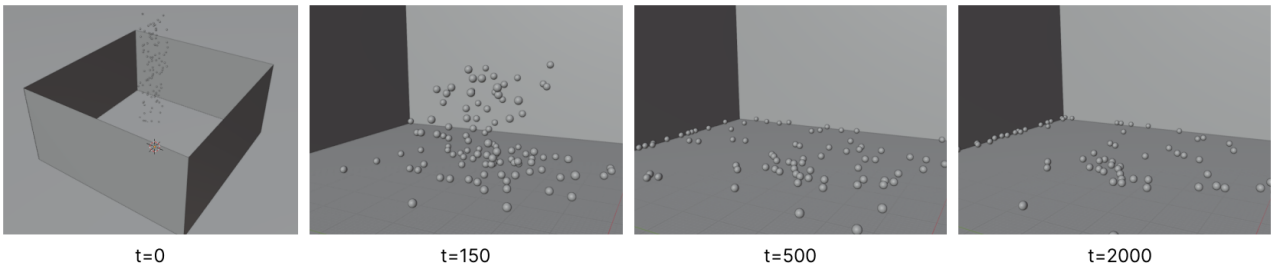

Figure 13: Sample from the training trajectories, showing that the spheres fall vertically on the ground, bounce, and collide with each other, eventually reaching a steady state at the end of the simulation (corresponding video).

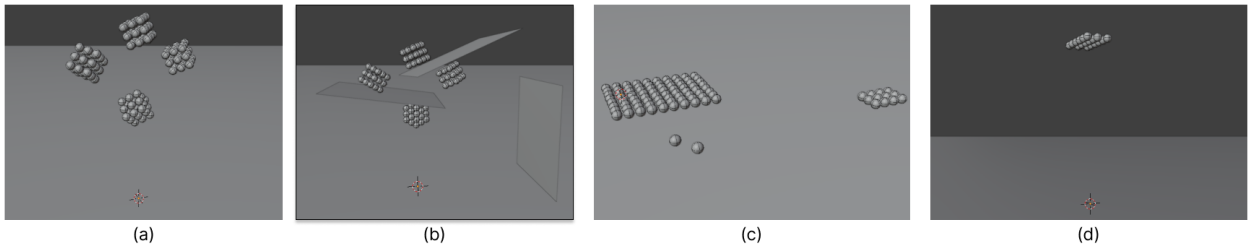

Figure 14: 4 OOD scenarios with (a) consisting of 4 cubes made of spheres that fall on the ground, the spheres are not linked, but such a pattern with very close spheres is not present in the training set. Then, (b) contains the same layout, with new surfaces including two inclined ones in the air, which was never seen during training. The trajectory shown in (c) corresponds to immobile and very close spheres that should not move through the simulation, while 2 spheres fall from a great height onto the two spheres visible in the foreground, pushing them into the immobile spheres visible on the right. Finally, (d) shows a group of spheres not touching each other, that fall from the same height and should bounce without sphere collisions. (See the corresponding videos)

- **The mean depth of penetration** which relates the average depth of intersection spheres experience, normalized by the diameter. For example, in the case of sphere's intersections, a normalized penetration depth of 0.5 means that the spheres are intersecting half of each other. For all intersections happening during the rollout, we thus average these depths of penetration.
- **The percentage of spheres with intersections** relates to the proportion of spheres that are subject to at least one intersection over the rollout.

These metrics can be independently computed for sphere-sphere intersections, sphere-ground intersections, and sphere-surface intersections. We separate the ground from other surfaces to evaluate the model's behavior when encountering different surfaces not present during training since the ground is always included.

Evaluating the respect of energy conservation/dissipation at the sphere level is challenging due to our open-loop physical system, where spheres can possibly gain or lose energy due to collisions. However, the total energy of all the spheres should either decrease or remain stable over time, which can be easily quantified. For this purpose, we compute:

- **The normalized cumulative energy gain** which tracks the total increase in energy of the system over time. This is calculated from the changes in the total energy (kinetic + potential) between frames: $\Delta\mathcal{E}^{t_k}_{\text{total}} = \mathcal{E}^{t_{k+1}}_{\text{total}} - \mathcal{E}^{t_k}_{\text{total}}$. Summing all positive energy gains and normalizing by

the initial total energy of the system gives the normalized cumulative energy gain:

$$\Delta\mathcal{E}_{>0} = \frac{1}{\mathcal{E}_{\text{total}}^{t_0}} \sum_{k=0}^{K-1} \max(0, \Delta\mathcal{E}_{\text{total}}^{t_k})$$

Finally, we can evaluate the respect of appropriate kinematics (and gravity) indirectly by comparing the predicted trajectories with the reference ones. We propose to use two metrics for this:

- **The Mean Squared Error** between the positions, that tracks the difference between corresponding sphere positions through time:

$$MSE_{\text{pos}} = \frac{1}{K} \sum_{k=0}^{K} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{p}_i^{t_k} - \tilde{\mathbf{p}}_i^{t_k}\|_2^2$$

- **The Symmetric Chamfer Distance** that considers the sphere positions as a point cloud, and measures how well the reference and inferred point clouds align. It corresponds to the minimum distance from each point in the first set to its closest point in the second set and vice-versa, all averaged over time:

$$\text{Chamfer}_{\text{total}} = \frac{1}{K} \sum_{k=0}^{K} \left(\frac{1}{N} \sum_{i=1}^{N} \min_j \left\| \mathbf{p}_i^{t_k} - \tilde{\mathbf{p}}_j^{t_k} \right\|_2^2 + \frac{1}{N} \sum_{j=1}^{N} \min_i \left\| \tilde{\mathbf{p}}_j^{t_k} - \mathbf{p}_i^{t_k} \right\|_2^2 \right)$$

These two metrics provide complementary evaluations of the predicted motion. MSE compares corresponding sphere positions directly over time, making it highly sensitive to misalignments, which can be problematic in chaotic systems. In contrast, the Symmetric Chamfer Distance is more robust, as it does not rely on a strict one-to-one correspondence between spheres. This flexibility allows it to better handle small deviations, making it more suitable to capture the overall quality of the predicted trajectories.

4.3 Quantitative results

4.3.1 Evaluation of the ground truth data

Metric	Test Set (Mean ± Std)	OOD Set (Mean ± Std)
Normalized Cumulative Energy Gain ↓	$2.11 \times 10^{-2} \pm 1.86 \times 10^{-2}$	$2.83 \times 10^{-2} \pm 5.46 \times 10^{-2}$
Normalized Mean Depth (Static) ↓	$5.91 \times 10^{-5} \pm 1.56 \times 10^{-4}$	$9.31 \times 10^{-4} \pm 1.86 \times 10^{-3}$
Normalized Mean Depth (Ground) ↓	$1.75 \times 10^{-2} \pm 5.31 \times 10^{-3}$	$5.11 \times 10^{-7} \pm 2.87 \times 10^{-7}$
Normalized Mean Depth (Sphere) ↓	$4.22 \times 10^{-3} \pm 2.17 \times 10^{-3}$	$1.65 \times 10^{-3} \pm 1.28 \times 10^{-3}$
Percentage of Intersections (Sphere-Sphere) ↓	48.41 ± 22.97	48.70 ± 49.84
Percentage of Intersections (Sphere-Ground) ↓	10.66 ± 4.98	100.0 ± 0.0
Percentage of Intersections (Sphere-Static) ↓	0.198 ± 0.322	8.80 ± 17.59

Table 1: Summary of the metrics computed on Blender's simulations for both the test set (10 trajectories) and OOD set (4 trajectories).

To first be able to compare the metrics we obtain with a reference, we provide the average metrics on the test set and the OOD set in Table 1. Surprisingly, these metrics, computed on the ground truth data, demonstrate that Blender's physical realism is not 100% accurate. In terms of inter-penetrations and energy, the system is not perfectly realistic, which raises an important point about our models that learn from this distribution of data. This discrepancy could be explained by the fact that Blender is a rendering and graphics software, that is meant to look realistic, and full adherence to physical laws is not required.

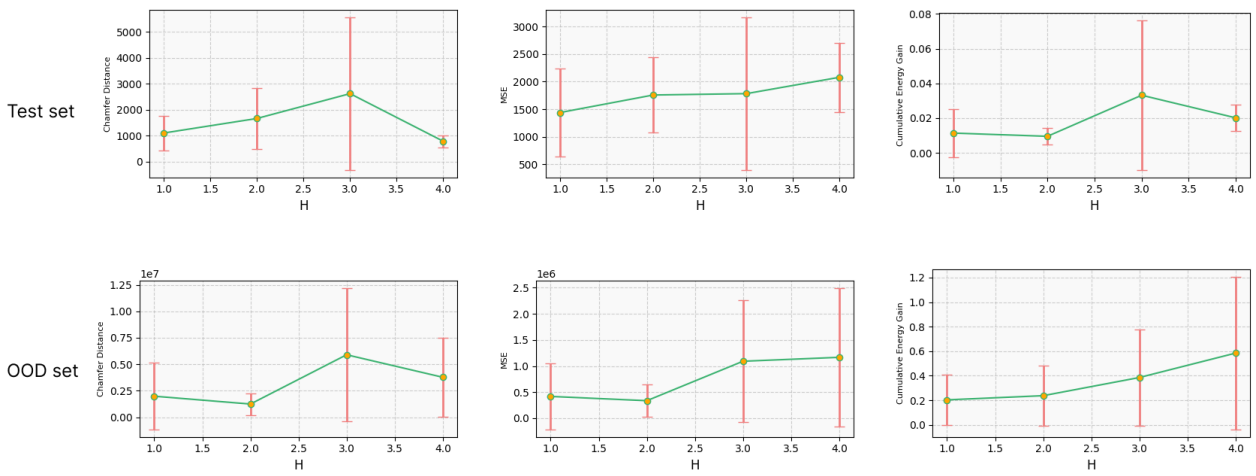

Figure 15: Evolution of the Chamfer distance, MSE, and cumulative energy gain when varying the history size H from 1 to 4 on the test set (top) and on the OOD set (bottom) with all other hyperparameters set to default. The red bars correspond to the standard deviations centered around the mean point, as each experiment is averaged over 3 seeds.

4.3.2 Impact of the state history size

We evaluated the impact of varying the history size H of previous linear and angular velocities given to the state vector of each sphere and described in section 3.1.2. Figure 15 shows the evolution of the Chamfer distance, MSE, and cumulative energy gain when varying H from 1 to 4 on the test and OOD set.

While some trends are noticeable, it's important to highlight that the standard deviation is often quite large. Due to the system's highly chaotic nature, running the same experiment with different random seeds during inference can lead to significantly different results and, consequently, varying metric values. This high variability complicates the interpretation of the metrics, requiring cautious conclusions.

In this experiment, we observe that increasing the history size H may not always benefit the model in terms of effectively learning physics. A history of size 1 or 2 seems to be reasonable to better learn the kinematics and how the energy of the system is supposed to decrease. However, when H increases, we observe a decrease in the performance of the model. This could be explained by its capacity to memorize some patterns during training that do not reflect a global physical principle to be learned. This overfitting is noticeable in the OOD set that is not sampled from the same distribution as the training set, and where we observe strong degradation of the predicted dynamic. In particular for the cumulative energy gain when $H = 4$, the system gains on average 60% of its initial energy.

4.3.3 Impact of the number of message passing steps

We evaluated the impact of varying the number of message passing steps M during the graph processing described in Section 3.2.4. Figure 16 shows the evolution of the Chamfer Distance, MSE, cumulative energy gain, and mean depth of sphere-sphere penetrations when varying the number of message passing steps M from 3 to 9 on the test and OOD sets.

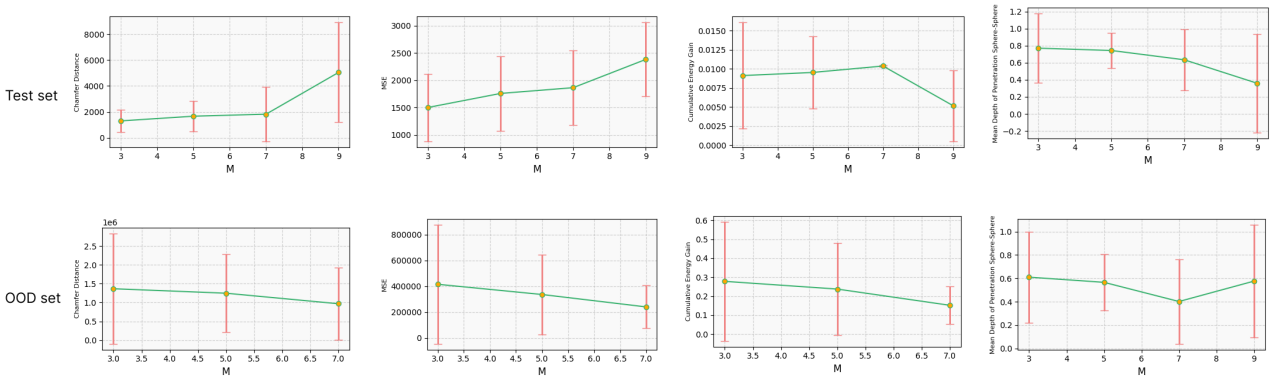

Figure 16: Evolution of the Chamfer distance, MSE, cumulative energy gain, and mean depth of sphere-sphere penetrations when varying the number of message passing steps M from 3 to 9 on the test set (top) and on the OOD set (bottom), with all other hyperparameters set to default. For evaluation on the OOD set, some of the metrics are not reported for $M = 9$ as the associate models have diverged during inference, leading to metrics of infinite values ($> 10^{32}$).

Before analyzing these results, notice that for the OOD set, the first three metrics are not reported for $M = 9$. The associated models have diverged during inference, which produced metrics of infinite values. When talking about a model that diverges during inference, it corresponds to situations where the spheres start to accelerate infinitely in one or several directions.

Rising M increases the complexity of the model as it adds more GN blocks with different parameters. The discrepancies between the test and OOD sets make these results difficult to analyze, especially with the high variability of the standard deviation. Still, we included these quantitative results as it could correspond to a case where the model was not able to learn the physical principles due to a too-high complexity when M increases. With more complexity, the model has increasing difficulties in learning the physical phenomena present during training. The model still learns elementary principles, which benefits the inference on the OOD set until a point (here $M = 9$) where the model diverges as it is likely too complex compared to the phenomenon it is supposed to learn.

4.3.4 Impact of the graph creation method

We evaluated how the $\epsilon/V_{\max}/V_i$-graph connectivity methods affect the performances of the model. Figure 17 shows the evolution of the Chamfer distance and cumulative energy gain when varying the graph connectivity method on the test and OOD sets.

Due to the high variability of the results, most of the metrics are difficult to interpret. Still, the Chamfer distance and cumulative energy gain metrics allows us to get some interesting insights. First, we observe that the $V_{\max}$ and ϵ-graph connectivity methods often diverge in OOD scenarios, leading to very high values for the two metrics. In the test set, V_i-connectivity provides the best result in terms of energy and realistic kinetics, while ϵ-connectivity often fails to predict the dynamic, probably because of the short threshold that connects only very close spheres. This method makes it not possible for the graph network to predict complex interactions.

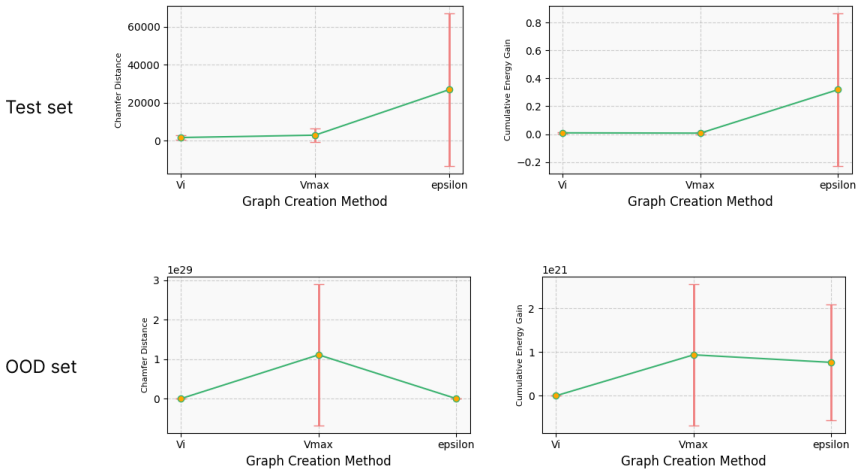

Figure 17: Evolution of the Chamfer distance and cumulative energy gain when varying the graph creation method between $\epsilon/V_{\max}/V_i$-graph connectivity on the test set (top) and on the OOD set (bottom), with all other hyperparameters set to default. Note that the MSE was not included as it provided the exact same dynamic as the Chamfer distance.

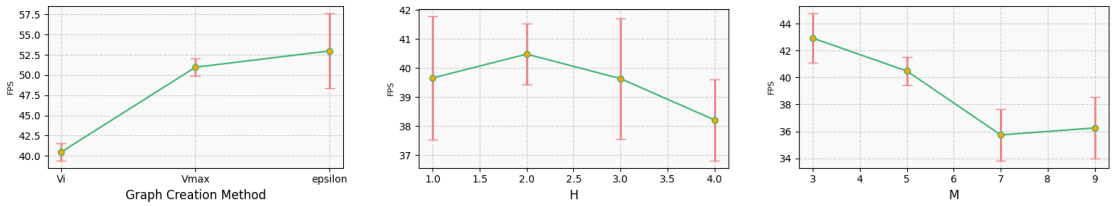

Figure 18: Evolution of the average runtime measured in FPS (Frame per Seconds) during inference on the test set when varying the graph creation method, the history size H and the number of message passing steps M.

4.3.5 Runtime comparison

We also compared the impact of M, H, and the graph creation method on the runtime measured in Frame-Per-Seconds (FPS) during inference as shown in Figure 18. For the test set with 10 trajectories, an average of 154 spheres are simulated. Using the default parameters $M = 5, H = 2$ and V_i-graph connectivity, we obtain a runtime of about 40 FPS which allows real-time execution of the learned simulator, without specific code optimization with parallelization.

During inference, the graph changes dynamically at each step, we thus need to recreate it at each iteration to pass it to the graph network afterward. A large portion of the runtime comes from this part, as the Graph Network mainly consists of elementary operations with MLPs, while the graph creation requires to compute distances efficiently between the spheres. In practice, the distances are efficiently computed using a KDTree. It is not surprising for V_i to be the slowest method as it requires querying the KDTree at each iteration due to the adaptive radius depending on the sphere's velocity. For the two other methods with a static radius, this computation is not needed, hence speeding the runtime with +10 FPS, and a slight advantage for ϵ-graph connectivity as much fewer spheres are connected due to the small query radius.

Experiment	MSE ↓	Chamfer ↓	Cum. Energy Gain ↓
$M = 1$	$4.79 \times 10^6 \pm 6.56 \times 10^6$	$1.48 \times 10^7 \pm 2.03 \times 10^7$	$5.93 \times 10^{33} \pm 1.03 \times 10^{34}$
ϵ-connectivity	$7.77 \times 10^{23} \pm 1.35 \times 10^{24}$	$2.33 \times 10^{24} \pm 4.04 \times 10^{24}$	$7.66 \times 10^{20} \pm 1.33 \times 10^{21}$
$H = 0$	$1.19 \times 10^9 \pm 1.68 \times 10^9$	$3.56 \times 10^9 \pm 5.03 \times 10^9$	$4.76 \times 10^{33} \pm 8.24 \times 10^{33}$
W/ Regularization	$7.63 \times 10^5 \pm 6.82 \times 10^5$	$3.96 \times 10^6 \pm 3.59 \times 10^6$	0.38 ± 0.26
w/o training noise	$9.55 \times 10^{21} \pm 1.65 \times 10^{22}$	$2.87 \times 10^{22} \pm 4.96 \times 10^{22}$	$2.63 \times 10^{19} \pm 4.55 \times 10^{19}$
Baseline	$\mathbf{3.36 \times 10^5 \pm 3.10 \times 10^5}$	$\mathbf{1.25 \times 10^6 \pm 1.03 \times 10^6}$	$\mathbf{0.24 \pm 0.24}$

Table 2: Comparison of MSE, Chamfer distance, and Cumulative Energy Gain obtained on the OOD set, across limit configuration to ablate and validate some design choices. Values are averaged on 3 seeds and reported as mean ± standard deviation. The red color highlights experiments where the inference has diverged. The baseline experiment uses the default hyperparameters explained in Section 4.1 (i.e.$M = 5$, $H = 2$, V_i-connectivity, no regularization, and with additive noise during training) and serves as the reference point for performance metrics.

Overall, for M and H, we observe a deterioration of the runtime when these hyperparameters increase, which is expected as H extends the size of the state vectors, and M scales linearly due to its addition of new GN blocks.

4.3.6 Ablation of the main components

We performed ablation of some key components of the method and evaluated the results on the OOD set to assess the generalization of the physical laws that have been learned. The results are displayed in Table 2.

- **$\mathbf{M = 1}$**: Using only one step of message passing, the model is too simple and ends up diverging during inference in OOD scenarios.
- **ϵ-connectivity**: Using ϵ-graph connectivity, the learned simulator creates the graph without taking into account the step size nor the current dynamic of the spheres, which makes it also diverge.
- **$\mathbf{H = 0}$**: Using no history $H = 0$ of previous linear and angular velocities, the model only knows its current state, which is not enough to predict accurate dynamics. This is true, especially for scenarios with bounces that require some context to predict the next steps.
- **W/ Regularization**: Surprisingly, adding the energy regularization term defined in Section 3.3 (with $\lambda = 0.01$, an optimal weight determined in preliminary experiments) decreases the model's ability to predict accurate dynamics, and increases the cumulative energy gain of the system. As noticed in Table 1, the cumulative energy gain in the ground truth data is not zero, indicating that the ground truth dynamics do not strictly follow the system's energy conservation or dissipation rules. Therefore, incorporating a regularization objective that enforces energy conservation or dissipation conflicts with the training objective. This contradiction may explain the slight decrease in performance observed.
- **w/o training noise**: When no noise is added to the state vectors during training, the model lacks robustness when inferring dynamics that differ from those encountered during training. As a result, it diverges in OOD scenarios.

This ablation with the previous quantitative analysis demonstrates that the baseline provides superior results in generalizing physical laws. It also shows the key elements and hyperparameters that must be carefully tuned to learn the desired dynamics. It is important to note that these hyperparameters are highly dependent on the complexity of the training data. For example, simulation data that contains

more complex interactions could benefit a higher value of M.

4.4 Qualitative results

In this section, we show and comment on some qualitative results obtained on inference with our baseline. All the videos are available discussed in the report are available on this Google drive.

4.4.1 Inference on test data

We provided two videos of inference on one trajectory of the test set. The first video is taken from a global point of view of the simulation while the second one is closer to the spheres that bounce and collide when falling.

These two videos highlight how the learned simulator can reproduce realistic interactions between the spheres and the ground. However, a key observation is that towards the end of the simulation, the spheres are still moving and never reach a steady state. Hence, the final global energy is approximately constant but is not equal to 0. This shows that the model did not properly learn the principles of energy dissipation through friction with surfaces at the sphere level.

This may be attributed to the training data, which is not fully accurate in terms of energy conservation and includes some spheres that continue to move at very low velocities by the end of the simulation. Potential directions for learning energy principles at the sphere level, and increasing the training dataset diversity are further discussed in Section 5.

4.4.2 Inference on OOD scenarios

We first explore the simulation obtained after inference on the scenario (b) of Figure 14, where spheres are packed into cubes, and fall on inclined surfaces. This scenario is interesting, as it allows us to verify if the model generalized normal force to other surfaces, as the only surfaces seen during training were 4 fixed vertical walls and the ground. We provided two videos of this inference with two different points of view, the first video gives a global picture, and the second video is closer to the inclined surfaces.

We observe from the first video that the model still suffers from issues with the energy, as they continue to roll without being stopped by friction. However, combined with the second video, also pictured in Figure 19 (top), we see that the model has surprisingly well generalized the normal force. The first cube of spheres rolls from the most inclined plane to the other plane. This suggests that the features that represent the plane described in Figure 7 are sufficient for the model to infer how the spheres are supposed to behave when in contact with planes. However, the model does not behave correctly with the second plane which is only slightly inclined, where the spheres roll in the wrong direction. Still, these results are already very promising since the model never learned from data containing these types of planes. Adding more complex surfaces in the training set could probably help the model to better learn normal force and the appropriate dynamics. In this scenario, we also focus on 2 interesting cases of collisions described in Figure 19. It shows how the model can generate realistic collisions in non-trivial situations.

We also explore the simulation obtained after inference on the scenario (c) of Figure 14, where 2 groups of spheres are supposed to remain immobile, and 2 spheres that fall from a great height will end up

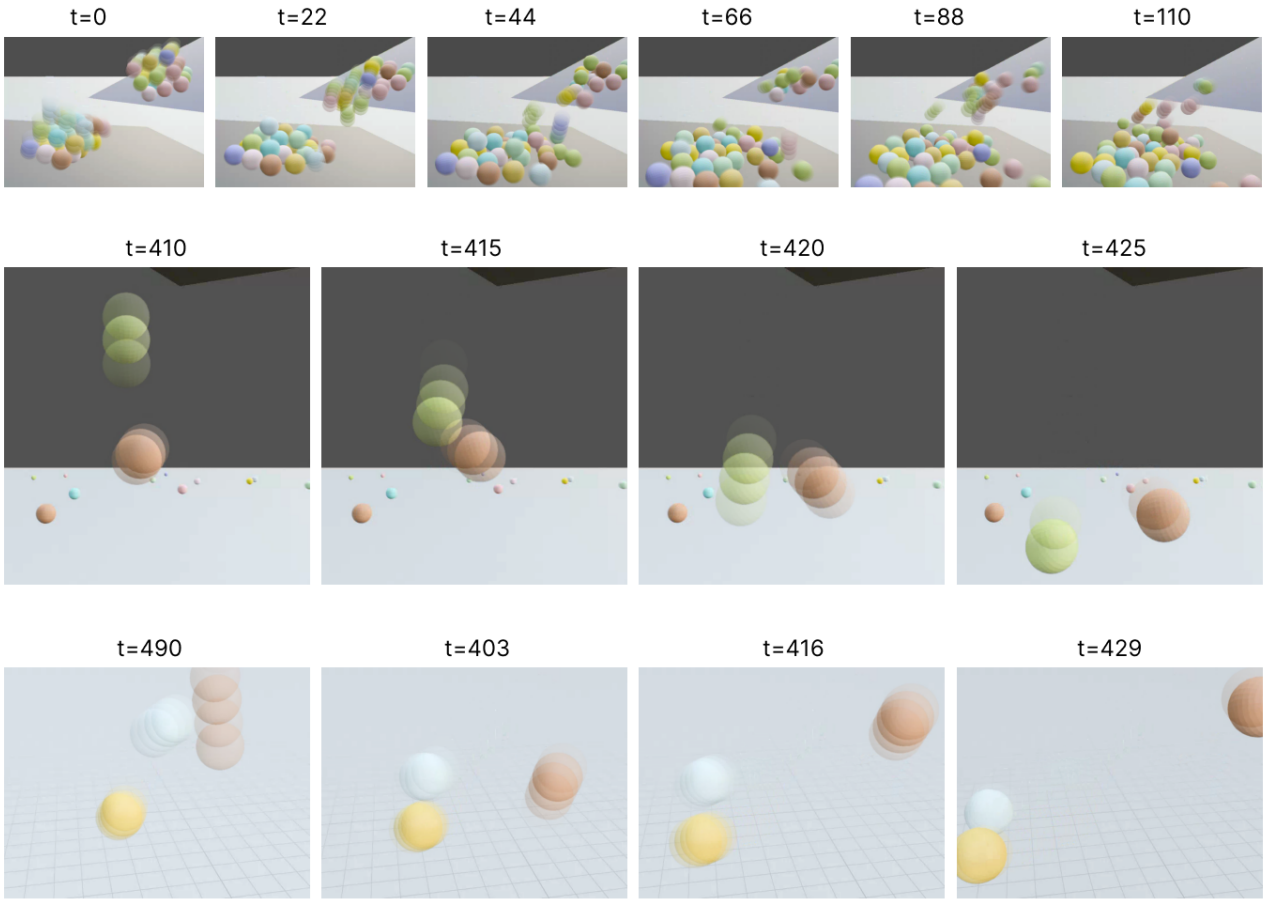

Figure 19: Inference on the OOD scenario (b) of Figure 14 pictured at multiple timesteps and focused on 3 specific parts of the simulation. The top figure shows spheres rolling on inclined surfaces, which is unseen during training (corresponding video). The middle figure shows spheres that collide in the air, where the brown sphere goes up after bouncing and the green sphere falls on it, resulting in a change of direction for both spheres (corresponding slow-motion video). The bottom figure shows an interesting collision situation, where the white sphere bounces from the ground and hits the brown sphere that is falling, changing the direction of both spheres and making the white one fall onto the yellow one that was slowly rolling (corresponding slow-motion video).

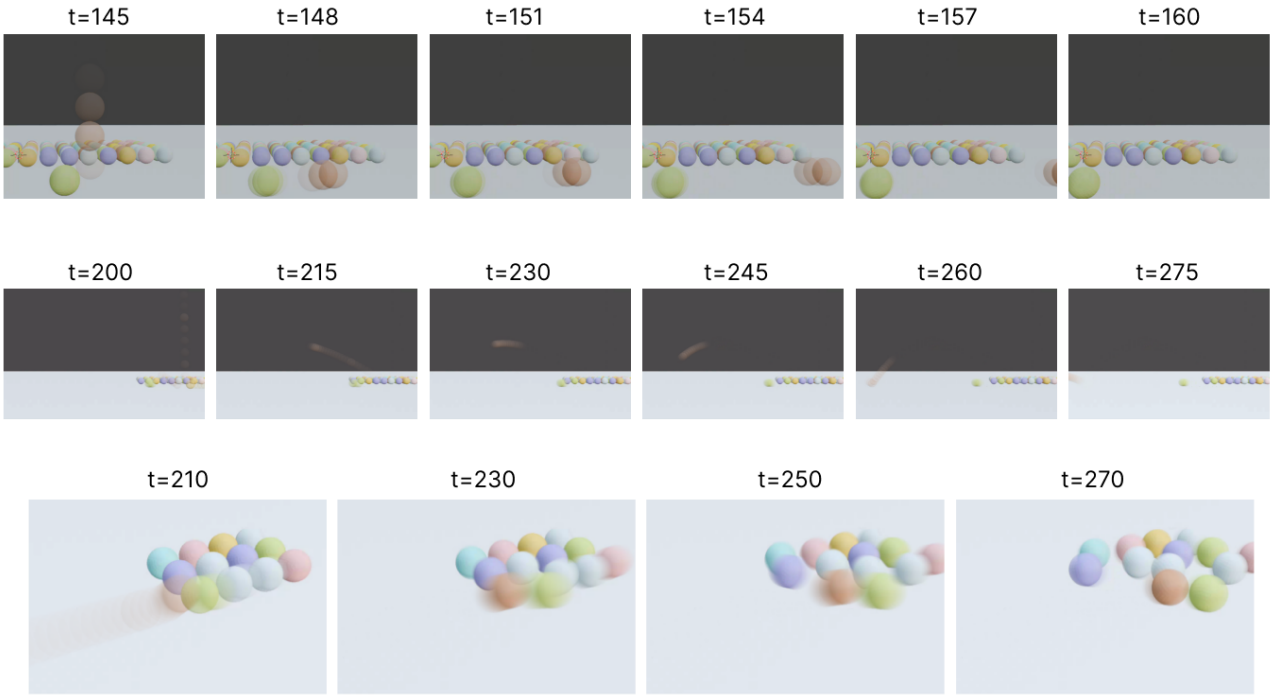

Figure 20: Inference on the OOD scenario (c) of Figure 14 pictured at multiple timesteps and focused on 3 specific parts of the simulation. The top figure shows a brown sphere that falls from a great height (113m) onto the side of an immobile green sphere, causing the green sphere to roll in the opposite direction from the brown one (corresponding video). The middle figure shows the same phenomenon that can be observed in the same video. A sphere falls from even higher (230m), achieving a high velocity which causes it to make a large bounce after being deviated by the side of another sphere. Finally, the bottom figure shows the brown sphere (of the top figure) that hit a group of immobile spheres, similarly to a pool game (corresponding video).

rolling toward one of the immobile groups. This scenario allows us to verify if the model can keep spheres in a steady state without generating imaginary interactions between them. Moreover, the two falling spheres achieve a very high speed unseen during training, which generates a collision with a high energy exchange. We provided one video that gives a global view of the inference, and a second video that focuses on the group of immobile spheres that get hit.

The beginning of this simulation shows very promising results for generalizing gravity and kinematics in different conditions (higher speed and height). However, we still observe in the video that after 10 seconds, the immobile spheres start to move even though they are not supposed to, and all the spheres end up moving in the same direction. This shows that the model is not yet able to disentangle the links in the graph between the spheres interacting with each other and those that should not. Still, the first 300 frames provide interesting and realistic results. Figure 20 shows 3 cases with close-ups of the interactions generated by the two spheres that fall.

These qualitative results highlight that despite a very simple training set, the model can generalize some physical laws to more complex scenarios with unseen dynamics.

5 Discussion and Perspectives

Summary of key results. In this work, we studied how learned simulators could be used to generalize physical laws in the context of rigid body dynamics, enabling accurate application of dynamics in out-of-distribution (OOD) scenarios. Using spheres as a primitive rigid body object, we designed a framework to learn physical dynamics from simulation data. Our method is based on Graph Networks and specifically incorporates strong inductive biases to generalize the physical laws we consider:

- The graph creation method connects spheres with other spheres or surfaces only when a potential collision is likely to happen. Our method V_i-graph connectivity adopts a dynamic approach by connecting spheres according to their estimated next positions. Moreover, the model is informed about gravity through a constant edge with the ground.
- The learned simulator is invariant to absolute spatial location as it only works with relative distances. This is consistent with physical laws that are invariant to spatial location.
- Newton's third law is incorporated by design in the message-passing steps, encoding the symmetric nature of physical interactions between rigid bodies within the latent information passed between particles.
- The model is trained on one-step predictions, enforcing it to behave like physical dynamics in a Markovian manner, meaning it should operate consistently at any point during the trajectory, as physical dynamics are inherently Markovian.

In contrast with other works dealing with learned simulators and described in Section 2.2, we emphasize the study of our method through the scope of respect of physical laws. We evaluated the physical realism of the learned simulators by introducing specific metrics that evaluate the adherence to the considered physical principles. Though challenging due to the chaotic nature of rigid body physics, we provided an analysis of the main components of the method through several experiments and an ablation study, proving that our baseline provides promising results towards generalizing physical laws. Finally, we provided qualitative results of the inferred simulations in OOD scenarios, showcasing the model's behavior in unseen and more complex scenarios, as well as its ability to generalize beyond the training data.

Limitations. Although our model displays promising quantitative and qualitative results, we observed that it is not yet able to properly learn energy dissipation through friction, as the spheres continue to move infinitely during a simulation, reaching a stable but non-zero total energy. This limitation could be partly due to the training data from Blender which is not perfectly accurate in terms of physical realism, especially for the object's penetrations and energy conservation/dissipation. For the evaluation in terms of physical realism, we focused our analysis on positional information, and did not consider the rotational energy or the accuracy of the rotations. In fact, we qualitatively noticed that Blender's sphere rotations were not accurate. While it would be feasible to introduce new metrics measuring rotation accuracy, for simplicity, we have decided to ignore this part in this study. Moreover, the analysis of the method is also limited by the chaotic behavior of rigid body physics during inference: the same experiment with different random seeds can provide drastically different results. Identifying the main source of variability in the model would be key to obtaining more reliable quantitative results. Finally, to compare our method with other learned simulator approaches, it will be necessary to either replicate existing work or adapt our method to the settings previously evaluated in the literature. We face the challenge that many of the state-of-the-art methods discussed in Section 2.2 are developed by the same industrial team and their source code is not publicly available. For instance, we contacted the authors of FIGNET [3], a model specifically designed for learning rigid body physics, but they were unable to share their code. As an alternative, we could adapt the publicly

available GNS code [41] and evaluate its performance on our rigid body physics data to compare its adherence to physical laws with our approach.

Perspectives. Although Section 4 covers different hyperparameters and provides insights into the model's performances, we would like to fully assess the effect of the introduced inductive biases on learning physics. A complete ablation study would be necessary, for example, we did not yet evaluate the effect of removing the prior about Newton's third law during message passing. This is something that can be addressed on the short term through new experiments. In order to generalize to more complex scenarios, we also plan to increase the complexity of our dataset, in particular by incorporating more diverse types of dynamics and types of surfaces. Moreover, we could consider using a more sophisticated physical simulator to enhance data quality, while also allowing for the effective application and evaluation of our energy regularization term. We highlighted in Section 4.4 that energy principles of conservation/dissipation were not learned at the sphere level. While this may be partially due to the current dataset, perhaps it could be addressed through a more principled training strategy aimed at learning physical laws gradually. Our current training approach tries to make our Graph Network learn all physical laws at once, but we could consider having a curriculum learning strategy to learn physical rules one by one by increasing the complexity gradually. For example, starting with one sphere that falls to learn gravity and normal forces, then continuing with a sphere that rolls to learn energy dissipation through friction, progressively combining both behaviors, and finally, incorporating more spheres to learn collisions. We hypothesize that the energy signal is stronger when learning with fewer spheres, which could help to learn basic physical principles from a small number of spheres, that could then be extrapolated in more complex scenarios. Finally, to expand our framework to non-sphere rigid objects, a promising direction is to integrate the shape information encoded by an SDF, inside the state vectors of each particle. Inspired by SDF-Sim [39], this approach could enable interactions with complex objects and surfaces.

Application of this work to computer vision. This project is part of a longer-term goal of applying learned physical deformations to 3D scenes. We aim to design a simple pipeline that links videos containing spheres interacting with our learned simulator framework, to be able to infer dynamics from the initial state of a video and compare with the real dynamics. Using a model like SAM (Segment Anything Model) [20] or the very recent SAM-2 follow-up model [38], we can segment spheres in each frame to have positions tracked through time, which is the only required information to run our method and infer future dynamics. With such a pipeline we could also learn from real-world videos, hence reducing the gap between real and simulated data when performing inference on real-world scenarios.

References

[1] Kelsey R Allen, Kevin A Smith, and Joshua B Tenenbaum. "Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning". In: *Proceedings of the National Academy of Sciences* 117.47 (2020), pp. 29302–29310.

[2] Kelsey R Allen et al. "Graph network simulators can learn discontinuous, rigid contact dynamics". In: *Conference on Robot Learning*. PMLR. 2023, pp. 1157–1167.

[3] Kelsey R. Allen et al. *Learning rigid dynamics with face interaction graph networks*. 2022. arXiv: 2212.03574 [cs.LG]. URL: https://arxiv.org/abs/2212.03574.

[4] Hritik Bansal et al. "VideoPhy: Evaluating Physical Commonsense for Video Generation". In: *arXiv preprint arXiv:2406.03520* (2024).

[5] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[6] Peter W. Battaglia et al. *Interaction Networks for Learning about Objects, Relations and Physics*. 2016. arXiv: 1612.00222 [cs.AI]. URL: https://arxiv.org/abs/1612.00222.

[7] Michael B. Chang et al. *A Compositional Object-Based Approach to Learning Physical Dynamics*. 2017. arXiv: 1612.00341 [cs.AI]. URL: https://arxiv.org/abs/1612.00341.

[8] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016.

[9] Yutao Feng et al. *Gaussian Splashing: Dynamic Fluid Synthesis with Gaussian Splatting*. 2024. arXiv: 2401.15318 [cs.GR].

[10] Yutao Feng et al. "Pie-nerf: Physics-based interactive elastodynamics with nerf". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 4450–4461.

[11] Jian Gao et al. *Relightable 3D Gaussian: Real-time Point Cloud Relighting with BRDF Decomposition and Ray Tracing*. 2023. arXiv: 2311.16043 [cs.CV].

[12] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural networks and the bias/variance dilemma". In: *Neural computation* 4.1 (1992), pp. 1–58.

[13] Justin Gilmer et al. "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 1263–1272. URL: https://proceedings.mlr.press/v70/gilmer17a.html.

[14] Klaus Greff et al. *Kubric: A scalable dataset generator*. 2022. arXiv: 2203.03570 [cs.CV]. URL: https://arxiv.org/abs/2203.03570.

[15] Ayaan Haque et al. *Instruct-NeRF2NeRF: Editing 3D Scenes with Instructions*. 2023. arXiv: 2303.12789 [cs.CV].

[16] Siyu He et al. "Learning to predict the cosmological structure formation". In: *Proceedings of the National Academy of Sciences* 116.28 (June 2019), pp. 13825–13832. ISSN: 1091-6490. DOI: 10.1073/pnas.1821458116. URL: http://dx.doi.org/10.1073/pnas.1821458116.

[17] Tianyu Huang et al. "DreamPhysics: Learning Physical Properties of Dynamic 3D Gaussians with Video Diffusion Priors". In: *arXiv preprint arXiv:2406.01476* (2024).

[18] Chenfanfu Jiang et al. "The material point method for simulating continuum materials". en. In: *ACM SIGGRAPH 2016 Courses*. Anaheim California: ACM, July 2016, pp. 1–52. ISBN: 978-1-4503-4289-6. DOI: 10.1145/2897826.2927348. URL: https://dl.acm.org/doi/10.1145/2897826.2927348 (visited on 08/22/2024).

[19] Bernhard Kerbl et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics* 42.4 (2023).

[20] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. URL: https://arxiv.org/abs/2304.02643.

[21] Daniel C Krawczyk. "The cognition and neuroscience of relational reasoning". In: *Brain research* 1428 (2012), pp. 13–23.

[22] Yuan Li et al. "Climatenerf: Extreme weather synthesis in neural radiance field". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 3227–3238.

[23] Yunzhu Li et al. "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids". In: *arXiv preprint arXiv:1810.01566* (2018).

[24] Fangfu Liu et al. "Physics3D: Learning Physical Properties of 3D Gaussians via Video Diffusion". In: *arXiv preprint arXiv:2406.04338* (2024).

[25] Tatiana Lopez-Guevara et al. "Scaling Face Interaction Graph Networks to Real World Scenes". In: *arXiv preprint arXiv:2401.11985* (2024).

[26] Pingchuan Ma et al. "Learning neural constitutive laws from motion observations for generalizable pde dynamics". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 23279–23300.

[27] Sebastian Martin et al. "Unified simulation of elastic rods, shells, and solids". In: *ACM Transactions on Graphics (TOG)* 29.4 (2010), pp. 1–10.

[28] Ben Mildenhall et al. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *Communications of the ACM* 65.1 (2021), pp. 99–106.

[29] Damian Mrowca et al. *Flexible Neural Representation for Physics Prediction*. 2018. arXiv: 1806.08047 [cs.AI]. URL: https://arxiv.org/abs/1806.08047.

[30] Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. "Real time dynamic fracture with volumetric approximate convex decompositions". In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–10.

[31] Matthias Müller et al. "Meshless deformations based on shape matching". In: *ACM transactions on graphics (TOG)* 24.3 (2005), pp. 471–478.

[32] Matthias Müller et al. "Position based dynamics". In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118.

[33] Matthias Müller et al. "Real-time simulation of deformation and fracture of stiff materials". In: *Computer Animation and Simulation 2001: Proceedings of the Eurographics Workshop in Manchester, UK, September 2–3, 2001*. Springer. 2001, pp. 113–124.

[34] Thu Nguyen-Phuoc, Feng Liu, and Lei Xiao. *SNeRF: Stylized Neural Implicit Representations for 3D Scenes*. 2022. arXiv: 2207.02363 [cs.CV].

[35] Tobias Pfaff et al. "Learning mesh-based simulation with graph networks". In: *arXiv preprint arXiv:2010.03409* (2020).

[36] Ri-Zhao Qiu et al. *Feature Splatting: Language-Driven Physics-Based Scene Synthesis and Editing*. 2024. arXiv: 2404.01223 [cs.CV]. URL: https://arxiv.org/abs/2404.01223.

[37] Stephan Rasp, Michael S Pritchard, and Pierre Gentine. "Deep learning to represent subgrid processes in climate models". In: *Proceedings of the national academy of sciences* 115.39 (2018), pp. 9684–9689.

[38] Nikhila Ravi et al. "Sam 2: Segment anything in images and videos". In: *arXiv preprint arXiv:2408.00714* (2024).

[39] Yulia Rubanova et al. *Learning rigid-body simulators over implicit shapes for large-scale scenes and vision*. 2024. arXiv: 2405.14045 [cs.LG]. URL: https://arxiv.org/abs/2405.14045.

[40] Alvaro Sanchez-Gonzalez et al. "Graph networks as learnable physics engines for inference and control". In: *International conference on machine learning*. PMLR. 2018, pp. 4470–4479.

[41] Alvaro Sanchez-Gonzalez et al. "Learning to simulate complex physics with graph networks". In: *International conference on machine learning*. PMLR. 2020, pp. 8459–8468.

[42] Franco Scarselli et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[43] Silvia Sellán et al. "Breaking good: Fracture modes for realtime destruction". In: *ACM Transactions on Graphics* 42.1 (2023), pp. 1–12.

[44] Kim Stachenfeld et al. "Learned simulators for turbulence". In: *International conference on learning representations*. 2021.

[45] Deborah Sulsky, Zhen Chen, and Howard L Schreyer. "A particle method for history-dependent materials". In: *Computer methods in applied mechanics and engineering* 118.1-2 (1994), pp. 179–196.

[46] Deborah Sulsky and Howard L Schreyer. "Axisymmetric form of the material point method with applications to upsetting and Taylor impact problems". In: *Computer Methods in Applied Mechanics and Engineering* 139.1-4 (1996), pp. 409–429.

[47] Deborah Sulsky, Shi-Jian Zhou, and Howard L Schreyer. "Application of a particle-in-cell method to solid mechanics". In: *Computer physics communications* 87.1-2 (1995), pp. 236–252.

[48] Jonathan A Weyn, Dale R Durran, and Rich Caruana. "Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere". In: *Journal of Advances in Modeling Earth Systems* 12.9 (2020), e2020MS002109.

[49] Joshuah Wolper et al. "AnisoMPM: animating anisotropic damage mechanics". In: *ACM Trans. Graph.* 39.4 (Aug. 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392428. URL: https://doi.org/10.1145/3386569.3392428.

[50] Joshuah Wolper et al. "CD-MPM: continuum damage material point methods for dynamic fracture animation". In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: 10.1145/3306346.3322949. URL: https://doi.org/10.1145/3306346.3322949.

[51] Tianyi Xie et al. "Physgaussian: Physics-integrated 3d gaussians for generative dynamics". In: *arXiv preprint arXiv:2311.12198* (2023).

[52] Haotian Xue et al. "3D-IntPhys: towards more generalized 3D-grounded visual intuitive physics under challenging scenes". In: *Advances in Neural Information Processing Systems* 36 (2024).

[53] Ce Yang et al. *Learning to Simulate Unseen Physical Systems with Graph Neural Networks*. 2022. arXiv: 2201.11976 [cs.LG]. URL: https://arxiv.org/abs/2201.11976.

[54] Lior Yariv et al. *Volume Rendering of Neural Implicit Surfaces*. 2021. arXiv: 2106.12052 [cs.CV]. URL: https://arxiv.org/abs/2106.12052.

[55] Albert J. Zhai et al. *Physical Property Understanding from Language-Embedded Feature Fields*. 2024. arXiv: 2404.04242 [cs.CV]. URL: https://arxiv.org/abs/2404.04242.

[56] Chiyuan Zhang et al. "Understanding deep learning (still) requires rethinking generalization". In: *Communications of the ACM* 64.3 (2021), pp. 107–115.

[57] Tianyuan Zhang et al. *PhysDreamer: Physics-Based Interaction with 3D Objects via Video Generation*. 2024. arXiv: 2404.13026 [cs.CV]. URL: https://arxiv.org/abs/2404.13026.

[58] Licheng Zhong et al. "Reconstruction and Simulation of Elastic Objects with Spring-Mass 3D Gaussians". In: *arXiv preprint arXiv:2403.09434* (2024).