Object Dynamics Modeling with Hierarchical Point Cloud-based Representations

Chanho Kim
Oregon State University
kimchanh@oregonstate.edu

Li Fuxin
Oregon State University
lif@oregonstate.edu

2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) | 979-8-3503-5300-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/CVPR52733.2024.01982

Abstract

Modeling object dynamics with a neural network is an important problem with numerous applications. Most recent work has been based on graph neural networks. However, physics happens in 3D space, where geometric information potentially plays an important role in modeling physical phenomena. In this work, we propose a novel U-net architecture based on continuous point convolution which naturally embeds information from 3D coordinates and allows for multi-scale feature representations with established downsampling and upsampling procedures. Bottleneck layers in the downsampled point clouds lead to better long-range interaction modeling. Besides, the flexibility of point convolutions allows our approach to generalize to sparsely sampled points from mesh vertices and dynamically generate features on important interaction points on mesh faces. Experimental results demonstrate that our approach significantly improves the state-of-the-art, especially in scenarios that require accurate gravity or collision reasoning.

1. Introduction

Comprehending physics constitutes a fundamental facet of common sense knowledge. Humans naturally acquire this understanding early in life, predicting the outcome of collisions and nonlinear motions without the need to understand how to solve partial differential equations. Such capabilities of fast and intuitive physical predictions would also be crucial for enabling robots to plan actions effectively. For instance, when stacking boxes, a robot needs to understand how to arrange them to prevent the boxes from collapsing. Similarly, when a robot manipulates objects on a table to achieve a goal, it must comprehend the configuration of objects resulting from its actions. While traditional physics simulators [8, 31] can be employed to grant robots such reasoning abilities, it is hard for them to generalize to planning tasks with realistic sensory inputs.

Early attempts for developing a learning-based approach for simulation aimed to learn object dynamics directly in the image pixel space [20, 23, 24, 34], which have difficulty reasoning through complex object interactions and collisions. Recently, more promising approaches have been proposed, aiming to directly learn object dynamics in 3D space. The most popular approaches in this direction have utilized Graph Neural Networks (GNNs) [3, 10]. These models represent objects using dense particles or meshes in three-dimensional space, constructing graphs with nodes corresponding to these particles or mesh vertices, and edges linking nearby particles or representing mesh edges. Message passing networks are then learned to simulate the multi-step propagation of forces over graph nodes.

Early GNNs embed all the relational information into edge weights. However, people have increasingly realized that physics happens in the three-dimensional world and that ignoring geometry is not ideal, not to mention there is also gravity, which strongly depends on the relationship in the z (vertical) direction in 3D space. Hence, later GNNs for physics and collision modeling have increasingly started to embed Euclidean space coordinates as part of the features [14, 17]. More recently, as people have started to discover the importance of long-range interaction modeling, U-Net-like structures have also emerged in the form of multi-scale GNNs, which employ different types of message passing between nodes at different levels [6, 9, 13].

Rather than forcing GNNs to embed themselves into the 3D space, an alternative would be to directly employ a network that has already been designed for unordered 3D point clouds. Such networks have undergone significant improvements in the past few years with impressive capabilities to solve complex real-world recognition tasks [25, 26, 28, 37]. They have also received many engineering updates to be able to handle hundreds of thousands of points as input simultaneously. Point-based approaches naturally employ the xyz coordinates, and their neighborhood structure based on Euclidean distances is natural for modeling collisions in the 3D world. Besides, the spatial topology can still be modified by changing the neighborhood structure, sim-

2575-7075/24/$31.00 ©2024 IEEE
DOI 10.1109/CVPR52733.2024.01982

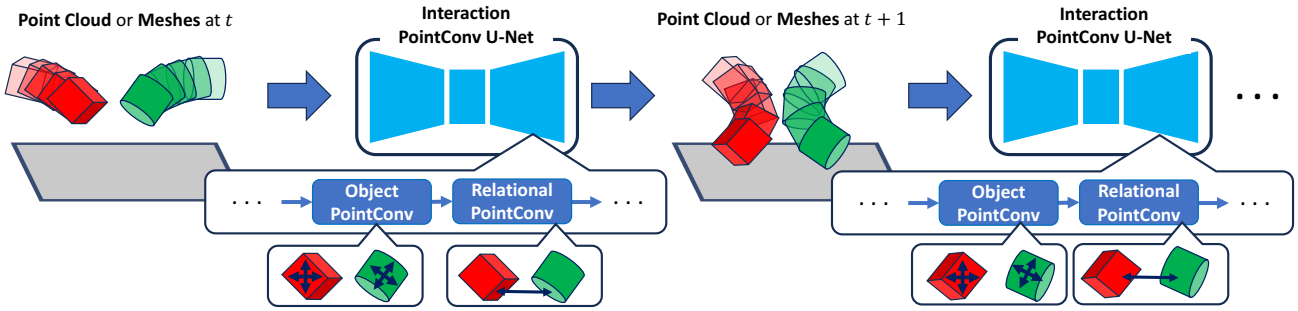

Figure 1. We propose a point-based convolutional neural network that is capable of learning object dynamics. Two different types of convolution operations, Object PointConv and Relational PointConv, are utilized alternatively to model force propagation within the same object and across different objects, respectively. A U-Net architecture encodes the point cloud into a smaller point cloud to capture long-term interactions and then decodes back to the original point cloud to make predictions. Point-based continuous convolution allows the proposed model to be compatible with both point cloud and mesh inputs with minor modifications.

ilar to that of GNNs. Some continuously defined point-based approaches, such as PointConv [35], additionally allow interpolation to generate features at any point in the 3D space even if there was no previous feature presence. This flexibility can potentially offer a better solution for hierarchical feature modeling than graph-based approaches with between-level message passing, which would require pre-computed features at each node to process messages consistently.

Given the convergence of graph- and point-based approaches, we believe that a study that applies point-based approaches in the task of learning object dynamics is timely so that future work from both fields can borrow from each other. There is prior work that uses point-based continuous convolution for learning fluid simulation [32], but the models presented there were neither deep nor large-scale. Besides, it cannot be directly applied to simulating physics in scenes with a variable number of objects.

In this work, we introduce a novel point-based continuous convolution network designed to model the collision dynamics of multiple objects composed of dense 3D points. We propose novel network designs that facilitate object-centric relation reasoning on point clouds and build a U-Net architecture that learns hierarchical feature representations of the scene at different physical scales as shown in Fig. 1. We demonstrate that our proposed model can effectively learn object dynamics from dense point clouds. Besides, we demonstrate an approach on a sparse point cloud defined on mesh vertices. Especially, we show how the interpolation capability of point convolutions enables the generation of point features at important locations on mesh surfaces, which helps the propagation of information between interacting objects.

In summary, we make the following contributions:

- We propose point cloud convolution layers specific to learning object dynamics and assemble a U-Net structure with those layers for hierarchical modeling of object dynamics.
- We extend a point cloud convolution layer for mesh collision reasoning, effectively computing collision effects between mesh faces. This is achieved by generating the features of the interaction points on the mesh faces and propagating them to the corresponding vertices.
- Experiments show that our approach significantly improves over state-of-the-art GNN methods.

2. Related Work

In this section, we focus on reviewing existing literature on learning object dynamics from 3D data directly.

2.1. GNN-based Models

GNNs have been the predominant tool when it comes to learning object dynamics in 3D space. Models have been proposed for both learning particle-based [14, 17] and mesh-based simulation [1, 19, 27] using similar GNN frameworks. These approaches rely on message passing layers to update particle or mesh features on a graph. To promote faster information propagation, these approaches often have an additional hierarchy with sparser graph nodes (corresponding to multiple cluster centers found through k-means clustering [17] or an object graph node per object [2]) connected to dense graph nodes defined in the observation space. Variants of these models have also been proposed to further improve performance through rotation-invariant features [14] or a new type of graph node that explicitly encodes face-to-face interaction of the mesh [2]. In this work, we propose a continuous convolution-based alternative to these GNN-based approaches, which enables hierarchical feature learning efficiently. We do not touch upon rotation equivariance in this work, but similar approaches to incorporate such equivariance exist in point cloud networks as well [16, 38, 39] and can be added to our work.

2.2. Continuous Convolution-based Models

A continuous convolution-based learned simulator has been proposed for fluid simulation [32]. Although these approaches showed promising results in simulating fluids interacting with an environment, these approaches cannot be applied directly to other scenarios where multiple rigid or elastic objects interact with each other, due to the lack of specific modeling of the chemical bonds within the objects. Also, the proposed model in [32] only had a few convolution layers unlike other modern networks [36] featuring many layers for processing dense point clouds. In contrast to these early convolution-based neural simulators, we propose a novel point-based convolutional network that is capable of learning deep feature representations of objects while modeling object interactions effectively.

2.3. Hierarchical Models

Prior work has also attempted to address the limitations of GNN-based simulators with a more complicated hierarchical structure for better information propagation via both high- and low-resolution scene representations, but these networks are still shallow in the sense that it focuses on adding one additional hierarchy that is constructed in a better way than the ones based on k-means clustering or simple virtual object nodes in early work [17]. A U-Net architecture [30] has also been proposed for learning physics simulation with hierarchical feature representations, but relied on the 2D convolution operator, and thus its application was limited to data represented in a 2D grid. Recently, other U-Net architectures for graph networks have been proposed for meshes [6, 9, 13] where coarser meshes can be used as downsampled levels, but these models do not possess the capability to work with dense point clouds.

In point cloud networks, hierarchical models have been proposed from early on [22] and people have studied extensively different downsampling approaches, such as random downsampling, farthest point downsampling, and grid downsampling. Grid downsampling, which selects at most one point per voxel given a specific voxel size, has been demonstrated to be both computationally efficient and to improve recognition performance [29]. This could be due to the more regular density of points after grid downsampling, in comparison with other approaches.

2.4. Other Point-based Networks

The last few years have seen a proliferation of many point-based networks being proposed, starting from PointNet [21] and PointNet++ [22]. PointNet consists of MLP layers and max-pooling layers, and PointNet++ uses max-pooling locally within each point's neighborhood. Max-pooling tends to lose information about non-max features of points, hence the performance of those approaches is suboptimal. Besides PointConv and different variants of similar continuous convolution (e.g., [5, 33]), point transformer approaches [18, 37, 40] have been proposed with slightly better prediction performance. However, these approaches still lack the capability of generalizing to points without features. Another high-performance architecture is to directly utilize the 3D convolution but make it sparse by not computing the output of the convolution if the location is not occupied with a sampled point [7, 11]. This works well on densely sampled point clouds, but the performance suffers if the sparsity is uneven and dynamic, hence not necessarily suitable for the object dynamics modeling task.

3. Method

In this section, we introduce a new U-Net architecture that enables efficient collision dynamics modeling with a hierarchical scene representation using point-based convolutions. We begin by outlining the problem setup, describing the input and output of the model (Sec. 3.1). Next, we introduce the PointConv operator [35], which serves as a basic building block for our point-based convolutional neural network (Sec. 3.2). Following this, we present novel PointConv operators designed to model interactions within and between objects (Sec. 3.3), as well as with dense and mesh-based sampling of points. Then, we present an interaction PointConv block (Sec. 3.4) and use it to build an Interaction PointConv U-Net (Sec. 3.5). Finally, we discuss training details and loss functions (Sec. 3.6).

3.1. Problem Formulation

Given point cloud observations at time $\{t-h+1, ..., t-1, t\}$ where h is the length of the input history, our goal is to predict a future point cloud at time $t+1$. With the capability to predict the point cloud at time $t+1$, point clouds in the following frames can be obtained in an autoregressive manner. Each point $p_i(t)$ in point cloud observations has its position $p_i(t) = [x_i(t), y_i(t), z_i(t)]$ and velocity $v_i(t) = p_i(t) - p_i(t-1)$ as its states. The model takes a set of points as input where each point is associated with the velocity history $\{v_i(t-h+1), v_i(t-1), ..., v_i(t)\}$ along with the most recent position $p_i(t)$ and the outputs $v_i(t+1)$ for all points. We set the velocity history length h to 2, which essentially encodes information from 3 frames. We concatenate velocities in the input history and provide them as input features for the model. Following the prior work [14, 17], we also append the z coordinate (i.e., coordinate along the gravity axis) of $p_i(t)$ to the input so that the model knows how far an input point is from the ground plane. Besides, the entire position vector $p_i(t)$ is used to generate convolution filter weights, as described below.

3.2. PointConv

Let $p_0 \in \mathbb{R}^3$ be a point where a convolution kernel is centered and $\mathcal{N}(p_0)$ be a set of neighbor points of p_0. An ϵ-ball or k-nearest neighbors (kNN) neighborhood is often adopted when applying convolution to point clouds. However, the choice of neighborhood is flexible; for example, one can use a mesh vertex neighborhood if input points are mesh vertices. Let $x_i \in \mathbb{R}^{c_{\text{in}}}$ be the input features of a point p_i and $y_i \in \mathbb{R}^{c_{\text{out}}}$ be the output features of p_i. The naive formulation of PointConv is defined as follows:

$$y_0 = \sum_{p_i \in \mathcal{N}(p_0)} (W(p_i - p_0))^\top x_i \tag{1}$$

where p_0 represents a query point, $y_0 \in \mathbb{R}^{c_{out}}$ represents the output feature vector of point p_0 and $W(\cdot)$ represents a matrix-valued function with output dimensionality $c_{in} \times c_{out}$. A nice property of PointConv is that a query point p_0 can be any arbitrary point in the 3D space even with x_0 undefined. This has enabled a PointConv layer that can either down-sample or up-sample input points using sparser or denser query points [35, 36].

This formulation, however, is computationally expensive as the size of the tensor for the backward pass of the weight tensor W is equal to $c_{\text{out}} \times c_{\text{in}} \times k \times n$ for n input points with a k-NN neighborhood. In [35], an efficient formulation of PointConv was proposed by redefining the filter weight $W(\cdot) = W_l h(p_i - p)$, leading to the following formulation:

$$y_0 = W_l \text{vec}(\sum_{p_i \in \mathcal{N}(p_0)} h(p_i - p_0) x_i^\top) \tag{2}$$

where $h(\cdot) \in \mathbb{R}^{c_{\text{mid}}}$ outputs a c_{mid}-dimensional vector, $\text{vec}(\cdot)$ is an operator that flattens a matrix into a vector, and $W_l \in \mathbb{R}^{c_{\text{out}} \times c_{in} c_{\text{mid}}}$ represents a linear transformation. The size of the tensors becomes significantly smaller because, first, W_l is learned and shared by all input points, besides, c_{mid} can be set to as small as 4 without significantly compromising the performance [36]. This efficient PointConv formulation enables a point-based convolution network that can work with hundreds of thousands of points in dozens of layers and still fit in the memory of a single GPU.

Compared to GNNs, PointConv generates weights from relative coordinates and hence has a learned, multiplicative relationship between the coordinates and the features. This can be helpful in capturing relationships that are hard to obtain solely through concatenating positional embeddings like in GNNs [1, 17], such as the notion that effects from a point should be diminished or strengthened if the points are at a certain direction from each other. In the experiments, we show a direct ablation against graph convolution with positional embeddings, which indicates that the point convolution approach outperforms the alternative approach.

3.3. Learning Collision Dynamics with PointConv

When learning collision dynamics with neural networks, effects of physical force need to be propagated within objects and across different objects [14]. These effects are different because particles within the same object have strong chemical bonds, whereas those forces are usually negligible between different objects. Hence, it is ideal to have a special treatment for within-object interactions. We utilize two types of PointConv operators to achieve this goal. We call the convolution operator that propagates effects within objects *Object PointConv* and the other convolution operator that propagates effects across different objects *Relational PointConv*. Below, we propose different modifications to the PointConv operator (Eq. (2)) for these layers.

3.3.1 Object PointConv

As described in Sec. 3.1, the input features for Object PointConv in the first layer can then be written as $x_i'(t) = f([v_i(t); v_i(t-1); z_i(t)])$, which concatenates the information and input it to a multi-layer perceptron (MLP) f that outputs point-wise features from input points. Note that this does not include any information encoding a surface shape around a point p_0, which is important for learning collision dynamics. Hence, we additionally concatenate the relative positional embedding with the input features. The input $x_i(t)$ around $p_0(t)$ in Eq. (1) is then defined as:

$$x_i(t) = [x_i'(t); e(p_i(t) - p_0(t))] \tag{3}$$

where $e(\cdot)$ is an MLP that takes the relative coordinates between a query point and a neighbor point as input and generates the positional embedding vector. We select the k nearest neighboring points within the same object as the neighborhood for point cloud inputs and the mesh vertex neighborhood for mesh inputs.

3.3.2 Relational PointConv for Point Cloud

For Relational PointConv, we would like to model point relationships with points from other objects; hence, we only search for neighbors that do not belong to the same object. Furthermore, we have noticed that including points from a faraway distance slows down learning since the network needs to learn to ignore those points, hence we filter the k-NN neighborhood N_{relation} to only retain those that are within a distance r. This gives us a neighborhood with $\leq k$ points which we denote as $\mathcal{N}_{\text{rel}}$. In order to account for a variable number of neighbors found in the filtered neighborhood, we divide the output of the intermediate operation by the number of neighbors as follows:

$$y_0 = W_l \text{vec}(\frac{1}{|\mathcal{N}_{\text{rel}}|} \sum_{p_i \in \mathcal{N}_{\text{rel}}(p_0)} h(p_i - p_0) x(p_i)^\top). \tag{4}$$

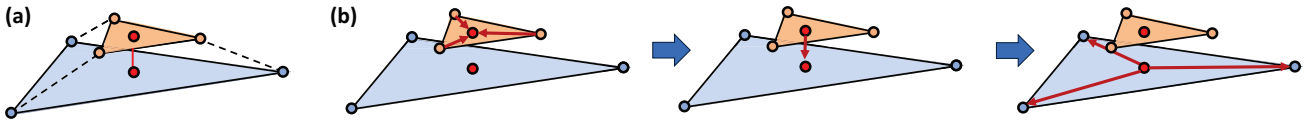

Figure 2. (a) As pointed out by [2], given two mesh faces of different objects, a neighborhood based on mesh vertices (dotted lines) may not capture proximity between two mesh faces (red solid line), depending on the location where a collision occurs. (b) We model face-to-face collision with three PointConv layers using dynamically selected interaction points (red dots) on the surfaces.

This formulation works fine for point clouds densely sampled from object surfaces because the neighborhood always includes points from other objects when two objects collide. In the next part, we introduce a variant of Relational PointConv that takes mesh vertices as input which can be sparsely sampled from object surfaces.

3.3.3 Relational PointConv for Mesh

In the case of polygonal input that can be represented with a simple mesh, densely sampling from object surfaces could be wasteful [2]. In GNNs, [2] proposed a combination of graph nodes with mesh vertex nodes and mesh face nodes, which complicates the network by requiring multiple types of interactions. Furthermore, when an object collides with another, the collision could happen at any location on a mesh face (see Fig. 2 (a)), so using a single node to represent each face may not capture these subtleties. In this paper, we propose a novel approach to mesh-based collision modeling by utilizing the interpolation capability of PointConv to compute the features of selected points on mesh faces, that are not part of the original network input.

Recall that a query point in PointConv can be any arbitrary point. As long as we have the input features of the points in its neighborhood $\mathcal{N}_{\text{rel}}(p)$, we can obtain the output features $y(p)$ of any $p \in \mathbb{R}^3$. In order to enable mesh face-to-face collision reasoning, we first compute distances between mesh faces of different objects and find pairs of points that are close between two nearby mesh faces (refer to the supplementary for more details). We call this pair *interaction points* and use it for relational reasoning as described below.

Given two nearby mesh faces, we aim to compute the effect that one face (receiver face) receives from the other face (sender face) and update the features of the receiver face's vertices accordingly. We model this process using three PointConv layers (see Fig. 2 (b)). The first PointConv layer computes the features of the interaction point p_{si} on the sender mesh face using p_{si} as a query point and mesh vertices that define the sender face as its neighborhood.

$$y_{\text{si}} = W_l \text{vec}\Big(\sum_{p_i \in \mathcal{N}_{\text{mesh-vertex}}(p_{\text{si}})} (h(p_i - p_{\text{si}}))x_i^\top \Big). \quad (5)$$

The number of points in $\mathcal{N}_{\text{mesh-vertex}}(\cdot)$ is fixed and determined by the mesh type (e.g., 3 for a triangular mesh).

The second layer then computes the features of the interaction point on the receiver face p_{ri} based on the features of interaction points on the nearby sender faces as:

$$y_{\text{ri}} = W_l \text{vec}\Big(\frac{1}{|\mathcal{N}_{\text{int}}(p_{\text{ri}})|} \sum_{p_i \in \mathcal{N}_{\text{int}}(p_{\text{ri}})} h(p_i - p_{\text{ri}})x_i^\top \Big) \quad (6)$$

where $\mathcal{N}_{\text{int}}(p_{\text{ri}})$ is the filtered k-NN neighborhood that includes only interaction points from nearby sender faces.

The last PointConv layer obtains the features of each mesh vertex p_0 by first propagating the features of interaction points on the receiver faces to the mesh vertex location via convolution and then applying average pooling. Here, the receiver faces are the ones that include the query point p_0 as one of its vertices. Then the PointConv operation can be written as:

$$y_0 = \frac{1}{|\mathcal{N}_{\text{int-surface}}(p_0)|} \sum_{p_i \in \mathcal{N}_{\text{int-surface}}(p_0)} W_l \text{vec}(h(p_i - p_0)x_i^\top) \quad (7)$$

where $\mathcal{N}_{\text{mesh-surface}}(p_0)$ includes interaction points from the mesh surfaces to which a mesh vertex p_0 belongs.

In the first and third layers, where the features are propagated via convolution, we append the positional embedding features to the input features, as in Eq. (3). With these three layers, we successfully propagate information from the mesh vertices of one object to the mesh vertices of another object while taking into account the location where collision is likely to occur.

3.4. Interaction PointConv Block

We define an interaction PointConv block as an Object PointConv layer followed by a Relational PointConv layer. The block can be set up in a way that the number of input points can change or remain the same. When point downsampling or up-sampling is needed, we do it with sparser or denser query points in the Object PointConv layer only and maintain the spatial resolution in the Relational PointConv layer. Similar to [36], for every PointConv layer, we have a residual connection and also utilize point-wise 1×1 convolutions to decrease and increase feature dimensions before

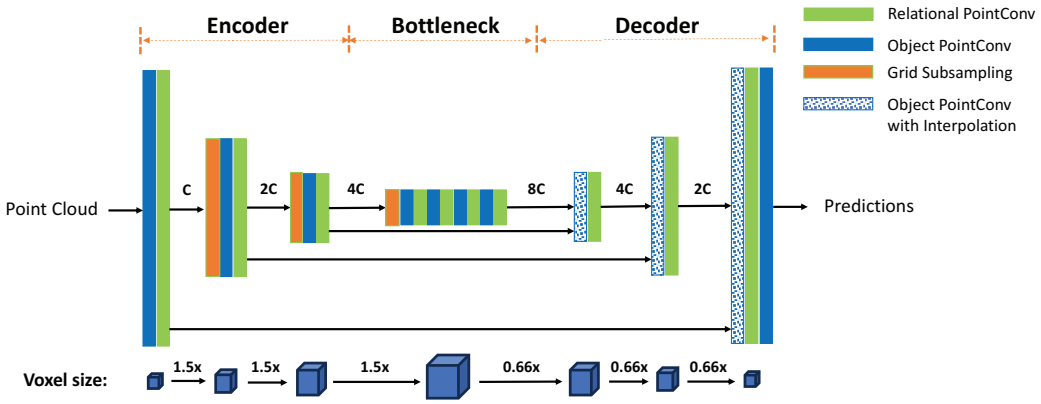

Figure 3. The proposed U-Net architecture. The input point cloud goes through Object PointConv and Relational PointConv alternatively, with successive downsampling layers in the encoding stage. In the bottleneck layers, the voxel sizes are large and the number of points is small; hence, long-range interactions are captured with several layers. Finally, in the decoder, Object PointConv with interpolation upsamples the point clouds to the point locations at the previous level. Finally, the point cloud is upsampled back to the original size, and then pointwise velocity or acceleration is predicted. We selected 32 as the base channel dimension C for our experiments in this paper.

and after a PointConv layer for the sake of reducing memory consumption.

3.5. Interaction PointConv U-Net

In order to propagate information over a long spatial range, prior work has utilized a 2-layer hierarchy, either using down-sampled points from k-means clustering as points in the upper layer [17] or having a single object node connected to all the points belonging to the object [14]. In this work, we aim to learn a hierarchical feature representation of the scene with several levels in the hierarchy with a U-Net architecture shown in Fig.3. In the encoding layers, points are downsampled successively while their locations are stored, and in the decoding layers, PointConv interpolates the features at the stored points from one level below. We follow the standard U-Net with highway connections between encoder layers and the corresponding decoder layers at the same resolution. The proposed U-Net architecture supports object interaction modeling with hierarchical object-centric scene representations from point cloud inputs.

In this U-Net architecture, long-range force propagation across multiple objects can be cheaply modeled by adding more interaction PointConv blocks in the bottleneck while still maintaining detailed geometric features of each object learned through multiple PointConv layers. In order to build a hierarchical feature representation of the scene, we utilize grid down-sampling on the original input points, following the standard practices in the existing point cloud literature [29, 36].

For mesh inputs, we used a variant of the proposed U-Net architecture, integrating relational PointConv solely at the highest resolutions. This approach circumvents the need to retain mesh faces at lower resolutions, which might introduce errors, particularly when representing complicated shapes with a limited number of vertices. For further details regarding grid down-sampling for point clouds and the U-Net architecture for learning object dynamics with meshes, please refer to the supplementary.

3.6. Training

The U-Net model is capable of making point-wise predictions after the decoder layers. Following existing work [1, 2], we created versions of either point-wise acceleration or velocity predictions. In the case of acceleration prediction, the network outputs $\hat{a}_p(t+1)$ for each point p, and the predicted position $\hat{p}(t+1)$ is then obtained by $p(t)+\hat{v}_p(t+1)$ where $\hat{v}_p(t+1) = v_p(t)+\hat{a}_p(t+1)$. In the case of velocity prediction, the network outputs $\hat{v}_p(t+1)$ directly. We supervise the model by comparing $\hat{p}(t+1)$ against the ground truth $p(t+1)$ using the Huber loss [15] as follows:

$$L(\hat{p}, p) = \frac{1}{\sum_i n_i} \sum_{i=1}^{M} \sum_{j=1}^{n_i} \text{Huber-Loss}\left(p_{ij}(t+1), p_{ij}(t) + \hat{v}_{ij}(t_i+1)\right) \quad (8)$$

where M is the number of point clouds in a mini-batch and n_i is the number of points in the ith point cloud in the mini-batch. p_{ij} represents the jth point in the ith point cloud.

3.7. Inference

We estimate a single rigid-body transformation from point-wise predictions for each rigid object to preserve the object shape throughout prediction rollouts, using the same pose

fitting algorithm as in prior work [4, 14, 17]. For non-rigid objects, we directly use the pointwise velocity predictions.

4. Experiments

We conducted our experiments using the Physion [4] and Kubric [12] datasets. The Physion benchmark provides training, validation, and testing data for seven scenarios, such as dominoes and support, involving rigid-body objects colliding as well as one scenario with non-rigid-body objects like drapes. The publicly available Kubric dataset offers training and validation data across various difficulty levels. For our experiments using Kubric, we utilized Movi-A and Movi-C videos. We excluded Movi-B as the public Movi-B data lacked object scale information necessary for generating input data for our models. In this paper, the results presented for the Physion dataset were generated using the testing data, while the results for the Kubric dataset were generated using the validation data.

As for the evaluation metrics, Physion provides the object contact prediction task, where an agent is asked to predict whether or not two query objects will contact each other based on the initial observation of the scene. Following the evaluation protocols in [4, 14], we measured the minimum distance between two objects over time while the prediction of the dynamics was made. We predict that two objects will touch if the distance comes below a threshold. This touch prediction is compared against the ground truth label. We used the contact prediction accuracy to compare against our own baselines and other recent GNN-based approaches [14, 17, 27] for Physion. For Kubric, we measured the Euclidean distance between ground truth point trajectories and predicted point trajectories at the end of prediction rollouts.

4.1. Benchmark Results

In this section, we compare the performance of the proposed model with other recent GNN-based approaches using the Physion benchmark. As for the Kubric benchmark, FIGNet [2] provided evaluation results, but they generated their own Kubric sequences instead of using public Kubric sequences. As we do not have access to these Kubric sequences and the code to train FIGNet on our dataset, we could not compare against the results presented in [2]. Thus, we only present comparisons against other published work on Physion, and for Kubric, we compare against our own baselines, which include running GNNs instead of PointConv within the same U-Net architecture.

The comparison to other GNN-based simulators is presented in Table 1. Both ours (vel) and ours (acc) are better than the state-of-the-art SGNN in 4 out of 7 scenarios involving rigid body objects, and ours (vel) is significantly better than SGNN in the non-rigid Drape scenario by more than 15%. The only scenario in which SGNN performs better is when compared to ours (vel) in the Link scenario. We suspect this is due to our velocity-based model being more sensitive to some label noises in the Link training data where multiple connected rigid parts are often labeled as separate rigid objects. For the other scenarios, the differences were not statistically significant. On average, ours (vel) is better than SGNN by 1.9%, and ours (acc) is better than SGNN by 5.0% in rigid scenes. Point-based approaches are especially better than graph-based approaches in the Support, Drop, Collide, and Drape scenarios, which require accurate reasoning about gravity and object collisions. Our significantly better performance on the Drape scenario further showcases the capability of our framework to learn non-rigid-body object dynamics. In the case of the Drape scenario, we did not predict pointwise acceleration since the deformable motion made the acceleration unstable. Unlike SGNN/DPI, which used dense particles provided by the Physion dataset across all 8 scenarios, we used dense particles for Drape only. For our results of other scenarios with no deformable object (i.e., except for Drape in Table 1). we used point clouds (consisting of only surface points) extracted from meshes as input. For further details about point cloud processing, please refer to the supplementary.

Results on the Kubric dataset are shown in Table 3. Our approach utilizing PointConv demonstrates superior performance compared to U-Net with GNNs in acceleration prediction. Besides, we show that sampling densely on the surface yields better results than sampling sparsely on the mesh vertices (i.e., dense point cloud inputs versus mesh inputs). This is understandable as dense sampling provides much more information on the surfaces than sparse sampling. However, when we are predicting acceleration, the gap between the mesh vertices and dense samples is reduced, similar to the findings of [1]. Besides, we can see significant improvements when we enable collision reasoning with interaction points from the faces in the case of Movi-A where many object shapes can be accurately represented with a limited number of vertices. Explicit face-to-face collision reasoning does not appear to enhance performance in Movi-C. We speculate that in scenarios like Movi-C, where input meshes have dense vertices because of their complicated shapes, the proximity between distinct objects can still be effectively captured by measuring the distances between mesh vertices. This results in PointConv U-Net learning collision dynamics as effectively as when explicit face-to-face collision reasoning is enabled.

For additional ablation experiments on Physion, we make a comparison between the message-passing algorithm designed for GNNs and the PointConv operator within the same U-Net architecture to showcase the efficacy of point-based continuous convolution networks compared to GNNs in learning object dynamics. In Table 2, we compare a PointConv-based simulator against a GNN-based simulator,

	Dominoes	Contain	Link	Support	Drop	Collide	Roll	Drape	Average
GNS [27]	78.6 ± 0.9	71.6 ± 1.6	66.7 ± 1.5	68.2 ± 1.6	65.3 ± 1.1	86.1 ± 0.5	81.3 ± 1.8	58.8 ± 1.0	74.0 (72.1)
DPI [17]	82.3 ± 1.3	72.3 ± 1.8	63.7 ± 2.2	64.8 ± 2.0	70.7 ± 0.8	84.4 ± 0.7	82.3 ± 0.6	53.3 ± 0.9	74.4 (71.7)
SGNN [14]	$\mathbf{89.1} \pm 1.5$	$\mathbf{78.1} \pm 1.5$	$\mathbf{73.3} \pm 1.1$	71.2 ± 0.9	74.3 ± 1.0	85.3 ± 1.1	84.2 ± 0.6	60.6 ± 0.5	79.4 (77.0)
Ours (vel)	87.3 ± 2.0	$\mathbf{82.4} \pm 2.8$	55.1 ± 3.5	76.2 ± 2.1	$\mathbf{89.8} \pm 0.3$	$\mathbf{92.0} \pm 1.1$	$\mathbf{86.5} \pm 0.3$	$\mathbf{75.8} \pm 1.9$	**81.3** (**80.6**)
Ours (acc)	$\mathbf{90.2} \pm 1.3$	75.1 ± 2.1	$\mathbf{75.3} \pm 1.9$	$\mathbf{83.6} \pm 3.5$	88.0 ± 0.9	$\mathbf{90.9} \pm 0.8$	$\mathbf{87.5} \pm 0.3$	-	**84.4** (-)

Table 1. Physion Benchmark. We used the performance numbers reported in [14] for other approaches in this table. Following [14], we also averaged our results across 3 runs. T-tests are used to determine statistical significance between pairs of approaches. The average numbers in the parentheses are the ones calculated including the Drape scenario.

	Dominoes	Contain	Link	Support	Drop	Collide	Roll	Average
GNN U-Net (vel)	87.3 ± 2.5	69.3 ± 0.9	$\mathbf{74.4} \pm 2.0$	62.7 ± 3.8	85.5 ± 1.4	92.0 ± 0.6	86.7 ± 0.0	79.7
PointConv U-Net (vel)	87.3 ± 2.0	$\mathbf{82.4} \pm 2.8$	55.1 ± 3.5	$\mathbf{76.2} \pm 2.1$	$\mathbf{89.8} \pm 0.3$	92.0 ± 1.1	86.5 ± 0.3	**81.3**
GNN U-Net (acc)	88.7 ± 1.5	65.1 ± 1.7	71.8 ± 1.9	59.1 ± 1.1	84.9 ± 0.8	88.4 ± 2.2	87.3 ± 0.3	77.9
PointConv U-Net (acc)	90.2 ± 1.3	$\mathbf{75.1} \pm 2.1$	75.3 ± 1.9	$\mathbf{83.6} \pm 3.5$	$\mathbf{88.0} \pm 0.9$	90.9 ± 0.8	87.5 ± 0.3	**84.4**

Table 2. Ablation for GNN-based and PointConv-based learned simulators on the Physion dataset with the contact prediction accuracy (%)

	input	face	Movi-A	Movi-C
GNN U-Net (vel)	PC	-	2.17 ± 0.01	2.58 ± 0.03
Ours (vel)	PC	-	2.15 ± 0.12	2.52 ± 0.02
Ours (vel)	M	no	2.95 ± 0.08	2.70 ± 0.05
Ours (vel)	M	yes	$\mathbf{2.47} \pm 0.04$	2.63 ± 0.05
GNN U-Net (acc)	PC	-	2.20 ± 0.04	2.55 ± 0.08
Ours (acc)	PC	-	$\mathbf{1.87} \pm 0.01$	$\mathbf{2.18} \pm 0.01$
Ours (acc)	M	no	2.49 ± 0.02	2.23 ± 0.01
Ours (acc)	M	yes	$\mathbf{1.98} \pm 0.03$	2.23 ± 0.02

Table 3. Results on the Kubric dataset with the Euclidean distance as an error metric (lower is better). PC stands for point clouds, and M stands for meshes. The third column (face) indicates whether or not explicit face reasoning is enabled.

which is currently the most popular type of model in learning object dynamics. In order to make a fair comparison, we implemented the same message-passing steps used in [17] within our U-Net architecture. Hence, the only difference would be the difference between GNN and PointConv layers. One can see that PointConv U-Net significantly outperforms GNN U-Net in the Contain and Support scenarios and also outperforms GNN U-Net in the Drop scenario. These are all scenarios that involve gravity, which shows that the multiplicative relationship between coordinates and features helps PointConv to better learn the geometric information in 3D space.

Additionally, we include ablation results on different numbers of layers used in U-Net and on the interaction PointConv block, which creates the separation between object and relational PointConv within U-Net, in the supplementary.

5. Conclusion

In this paper, we propose an approach based on a U-Net structure with continuous point-based convolutions for modeling object dynamics. We extend PointConv to Object PointConv and Relational PointConv, which learn within- and between-object effects, respectively. Additionally, we propose an approach to propagate information between nearby meshes with vertex features by selecting interaction points on mesh faces dynamically and using PointConv to interpolate features on those interaction points. Experimental results demonstrate that our approach outperforms state-of-the-art graph neural network approaches, particularly on tasks involving reasoning about gravity and collisions. We hope this work can help bring the community of graph neural networks and point cloud neural networks together so that both can adopt best practices derived from the other side.

Acknowledgements

This work was partially supported by the USDA AFRI award 2019-67019-29462, DARPA award N66001-19-2-4035 and NSF award 2321851.

References

[1] Kelsey R Allen, Tatiana Lopez Guevara, Yulia Rubanova, Kim Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Graph network simulators can learn discontinuous, rigid contact dynamics. In *Proceedings of The 6th Conference on Robot Learning*, pages 1157–1167. PMLR, 2023. 2, 4, 6, 7

[2] Kelsey R Allen, Yulia Rubanova, Tatiana Lopez-Guevara, William F Whitney, Alvaro Sanchez-Gonzalez, Peter

Battaglia, and Tobias Pfaff. Learning rigid dynamics with face interaction graph networks. In *The Eleventh International Conference on Learning Representations*, 2023. 2, 5, 6, 7

[3] Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018. 1

[4] Daniel Bear, Elias Wang, Damian Mrowca, Felix Binder, Hsiao-Yu Tung, Pramod RT, Cameron Holdaway, Sirui Tao, Kevin Smith, Fan-Yun Sun, Fei-Fei Li, Nancy Kanwisher, Josh Tenenbaum, Dan Yamins, and Judith Fan. Physion: Evaluating physical prediction from vision in humans and machines. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran, 2021. 7

[5] Alexandre Boulch. Convpoint: Continuous convolutions for point cloud processing. *Computers & Graphics*, 88:24–34, 2020. 3

[6] Yadi Cao, Menglei Chai, Minchen Li, and Chenfanfu Jiang. Bi-stride multi-scale graph neural network for mesh-based physical simulation. *arXiv preprint arXiv:2210.02573*, 2022. 1, 3

[7] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3075–3084, 2019. 3

[8] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, New York, NY, USA, 2015. Association for Computing Machinery. 1

[9] Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. Multiscale meshgraphnets. In *ICML 2022 2nd AI for Science Workshop*, 2022. 1, 3

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, page 1263–1272. JMLR.org, 2017. 1

[11] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9224–9232, 2018. 3

[12] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: a scalable dataset generator. 2022. 7

[13] Artur Grigorev, Michael J Black, and Otmar Hilliges. Hood: Hierarchical graphs for generalized modelling of clothing dynamics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16965–16974, 2023. 1, 3

[14] Jiaqi Han, Wenbing Huang, Hengbo Ma, Jiachen Li, Josh Tenenbaum, and Chuang Gan. Learning physical dynamics with subequivariant graph neural networks. *Advances in Neural Information Processing Systems*, 35:26256–26268, 2022. 1, 2, 3, 4, 6, 7, 8

[15] Peter J. Huber. *Robust Estimation of a Location Parameter*, pages 492–518. Springer New York, New York, NY, 1992. 6

[16] Xingyi Li, Wenxuan Wu, Xiaoli Z Fern, and Li Fuxin. Improving the robustness of point convolution on k-nearest neighbor neighborhoods with a viewpoint-invariant coordinate transform. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1287–1297, 2023. 2

[17] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2018. 1, 2, 3, 4, 6, 7, 8

[18] Chunghyun Park, Yoonwoo Jeong, Minsu Cho, and Jaesik Park. Fast point transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16949–16958, 2022. 3

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

[20] Luis Piloto, Ari Weinstein, Peter Battaglia, and Matthew Botvinick. Intuitive physics learning in a deep-learning model inspired by developmental psychology. *Nature Human Behaviour*, 6:1–11, 2022. 1

[21] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 3

[22] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 3

[23] Haozhi Qi, Xiaolong Wang, Deepak Pathak, Yi Ma, and Jitendra Malik. Learning long-term visual dynamics with region proposal interaction networks. In *ICLR*, 2021. 1

[24] Ronan Riochet, Mario Ynocente Castro, Mathieu Bernard, Adam Lerer, Rob Fergus, Véronique Izard, and Emmanuel Dupoux. Intphys 2019: A benchmark for visual intuitive physics understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5016–5025, 2022. 1

[25] Danila Rukhovich, Anna Vorontsova, and Anton Konushin. Fcaf3d: Fully convolutional anchor-free 3d object detection.

In *European Conference on Computer Vision*, pages 477–493. Springer, 2022. 1

[26] Danila Rukhovich, Anna Vorontsova, and Anton Konushin. Tr3d: Towards real-time indoor 3d object detection. *arXiv preprint arXiv:2302.02858*, 2023. 1

[27] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020. 2, 7, 8

[28] Jonas Schult, Francis Engelmann, Alexander Hermans, Or Litany, Siyu Tang, and Bastian Leibe. Mask3D: Mask Transformer for 3D Semantic Instance Segmentation. In *Internationl Conference on Robotics and Automation*, 2023. 1

[29] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019. 3, 6

[30] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58 (1):25–36, 2020. 3

[31] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. 1

[32] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2020. 2, 3

[33] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2589–2597, 2018. 3

[34] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. 1

[35] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 9621–9630, 2019. 2, 3, 4

[36] Wenxuan Wu, Li Fuxin, and Qi Shan. Pointconvformer: Revenge of the point-based convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21802–21813, 2023. 3, 4, 5, 6

[37] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. *Advances in Neural Information Processing Systems*, 35:33330–33342, 2022. 1, 3

[38] Yinshuang Xu, Jiahui Lei, Edgar Dobriban, and Kostas Daniilidis. Unified fourier-based kernel and nonlinearity design for equivariant networks on homogeneous spaces. In *International Conference on Machine Learning*, pages 24596–24614. PMLR, 2022. 2

[39] Zhiyuan Zhang, Binh-Son Hua, David W Rosen, and Sai-Kit Yeung. Rotation invariant convolutions for 3d point clouds deep learning. In *2019 International conference on 3d vision (3DV)*, pages 204–213. IEEE, 2019. 2

[40] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021. 3