LEARNING RIGID DYNAMICS WITH FACE INTERACTION GRAPH NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Simulating rigid collisions among arbitrary shapes is notoriously difficult due to complex geometry and the strong non-linearity of the interactions. While graph neural network (GNN)-based models are effective at learning to simulate complex physical dynamics, such as fluids, cloth and articulated bodies, they have been less effective and efficient on rigid-body physics, except with very simple shapes. Existing methods that model collisions through the meshes' nodes are often inaccurate because they struggle when collisions occur on faces far from nodes. Alternative approaches that represent the geometry densely with many particles are prohibitively expensive for complex shapes. Here we introduce the "Face Interaction Graph Network" (FIGNet) which extends beyond GNN-based methods, and computes interactions between mesh *faces*, rather than nodes. Compared to learned node- and particle-based methods, FIGNet is around 4x more accurate in simulating complex shape interactions, while also 8x more computationally efficient on sparse, rigid meshes. Moreover, FIGNet can learn frictional dynamics directly from real-world data, and can be more accurate than analytical solvers given modest amounts of training data. FIGNet represents a key step forward in one of the few remaining physical domains which have seen little competition from learned simulators, and offers allied fields such as robotics, graphics and mechanical design a new tool for simulation and model-based planning.

1 INTRODUCTION

Simulating rigid bodies accurately is vital in a wide variety of disciplines from robotics to graphics to mechanical design. While popular general-purpose tools like Bullet [12], Mujoco [38] and Drake [37] can generate plausible predictions, predictions that match real-world observations accurately are notoriously difficult [40, 5, 36, 14]. Numerical approximations necessary for efficiency are often inaccurate and unstable. Collision, contact and friction are challenging to model accurately, and hard to estimate parameters for. The dynamics are non-smooth and nearly discontinuous [32, 29], and influenced heavily by the fine-grained structure of colliding objects' surfaces [9]. Even slight errors in the physical model or state estimates can thus lead to large errors in objects' predicted trajectories. This underpins the well-known sim-to-real gap between results from analytical solvers and real-world experiments.

Learned simulators can potentially fill the sim-to-real gap. They can be trained to correct imperfect state estimation, and can learn physical dynamics directly from observations, potentially producing more accurate predictions than analytical solvers [4, 23]. Graph neural network (GNN)-based models, in particular, are effective at simulating liquids, sand, soft materials and simple rigids [35, 27, 25, 31, 24]. Many GNN-based models are node-based: they detect and resolve potential collisions based on whether two mesh nodes or particles are within a local neighborhood. However, collisions between objects do not only happen at nodes. For example, two cubes may collide by one corner hitting the other's face, or one edge hitting the other's edge (in fact, corner-to-corner collisions are vanishingly rare). For larger meshes, fewer collisions occur within the local neighborhood around nodes, and thus collisions may be missed. Some prior work thus restricts collisions to simple scenarios (e.g. a single object colliding with a floor [4, 32]). Alternative approaches represent the object densely with particle nodes [24, 35], but that leads to a quadratic increase in node-node collision tests, which is computationally prohibitive for nontrivial scenes.

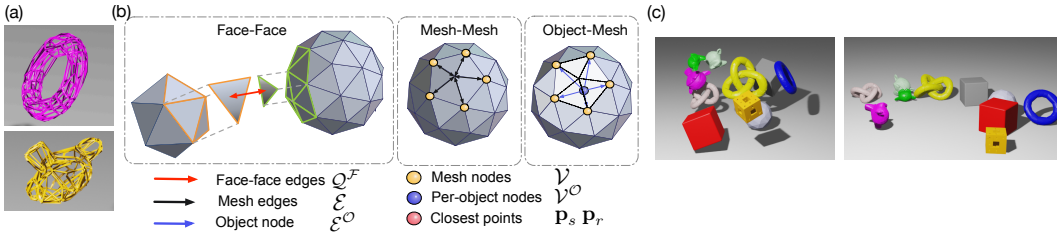


Figure 1: FIGNet model with face-to-face edges. (a) Example meshes from the Kubric MoviB dataset [17]. (b) FIGNet exploits three types of interactions: face-face collisions between objects, mesh-mesh interactions within an object, and interactions with the virtual per-object node. (c) Rollout from FIGNet trained on Kubric MoviB [17].

Here we introduce a novel mesh-based approach to collision handling—Face Interaction Graph Networks (FIGNet)—which extends message passing from graphs with directed edges between nodes, to graphs with directed hyper-edges between faces. This allows FIGNet to compute interactions between mesh faces (whose representations are informed by their associated nodes) instead of nodes directly, allowing accurate and efficient collision handling on sparse meshes without missing collisions. Relative to prior node-based models, we show that for simulated multi-rigid interaction datasets like Kubric [17], FIGNet is 8x more efficient at modeling interactions between sparse, simple rigid meshes, and around 4x more accurate in translation and rotation error for predicting the dynamics of more complex shapes with hundreds or thousands of nodes. We additionally show that FIGNet even outperforms analytical solvers for challenging real-world robotic pushing experiments [41]. To our knowledge, FIGNet is the first fully learned simulator that can accurately model collision interactions of multiple rigid objects with complex shapes.

2 Related Work

To address the weaknesses of traditional analytic simulation methods, a variety of hybrid approaches that combine machine learning with analytic physics simulators have been proposed. Learned simulation models can be used to correct analytic models for a variety of real-world domains [14, 2, 22, 15, 42, 19, 18]. Analytic equations can also be embedded into neural solvers that exactly preserve physical laws [32, 20]. While hybrid approaches can improve over analytic models in matching simulated and real object trajectories, they still struggle in cases where the analytic simulator is not a good model for the dynamics, and cannot be trivially extended to non-rigid systems.

More pure learning-centric approaches to simulation have been proposed in recent years to support general physical dynamics, and GNN-based methods are among the most effective. GNNs represent entities and their relations with graphs, and compute their interactions using flexible neural network function approximators. Such methods can capture the dynamics of fluids and deformable meshes [24, 35, 31], robotic systems [30, 34, 39], and simple rigids [33, 27, 25, 7, 4, 10].

Despite the successes of these models in many areas, the rigid dynamics settings they have been applied to generally use very simple shapes like spheres and cubes. Furthermore, most models are tested only in simulation, which may be a poor proxy for real world rigid dynamics [9, 14, 1]. It is therefore unclear whether end-to-end deep learning models, including GNNs, are generally a good fit for learning rigid-body interactions, particularly if the rigid bodies are complex, volumetric objects like those we experience in the real world.

In this work, we demonstrate how using insights from graphics for representing and resolving collisions as face-to-face interactions improves rigid body dynamics learning enough to not only very accurately capture simulated dynamics, but also outperform analytic simulators [38, 12, 26] for real-world rigid-body dynamics data. We conduct a series of ablations and experiments which showcase how face-to-face collision representations dramatically improve rigid body dynamics prediction.

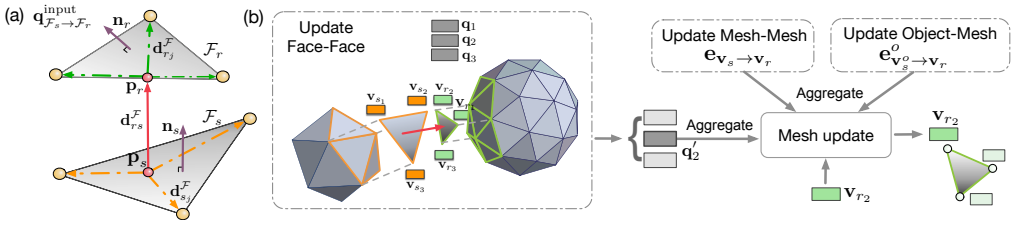


Figure 2: (a) Calculating face-face collision edge features (b) Message-passing with face-face edges. Face-face edges produce the updates for each node of the face (grey rectangles). After aggregation, the face-face update for the corresponding face node is combined with Mesh-Mesh and Object-Mesh edge representations, similarly to standard GNN message-passing. Notation is defined in text.

3 METHOD

For the purposes of rigid-body simulation, objects are represented as meshes M, consisting of the set of node positions $\{\mathbf{x}_i\}_{i=1..N}$ and a set of faces $\{\mathcal{F}\}$ that describe how nodes are connected to form objects in a mesh. Each face consists of three nodes $\mathcal{F} = \{s_j\}_{j=1,2,3}$, where s_j are the indices of the nodes in the face. We consider triangular faces but all aspects of the method generalize to faces with other numbers of nodes (quads, tetrahedra, points, segments, etc.) and their interactions.

The simulation trajectory is represented as a sequence of meshes $\mathcal{M} = (M^{t_0}, M^{t_1}, M^{t_2}, \dots)$, where each mesh represents one of more objects. It is computed by applying a sequence of object rotations and translations to a reference static mesh M^U of each object. The goal of the simulator s is to predict the next state of the system $\tilde{M}^{t+1}$ from a history of previous states $\{M^t, ..., M^{t-h+1}\}$. We use a history length of $h = 2$, following [4]. The rollout trajectory can be obtained by iteratively applying the simulator to the previous prediction; $(M^{t_0}, \tilde{M}^{t_1}, \dots, \tilde{M}^{t_k})$.

3.1 FACE INTERACTION GRAPH NETWORK

We introduce the Face Interaction Graph Network (FIGNet) (Figs. 1 and 2) as the simulator s, which is designed to extend existing general purpose simulators [35, 31] to model interactions between multiple rigid objects more efficiently. FIGNet is an evolution of the Encode-Process-Decode architecture which performs message passing on the nodes and edges of the object meshes, adding two key extensions over MeshGraphNets[31]. First, to model collisions between two objects, FIGNets generalizes message passing over directed edges to message passing over directed hyper-edges. This enables message passing on face-face edges, from a face with three sender nodes to a face with three receiver nodes. FIGNet replaces MeshGraphNets'[31] node-based "world edges" with face-face edges when faces from distinct objects are within a certain distance d_c. Second, FIGNet supports multiple node and edge types. We use this to add "object nodes" that are connected to all of the mesh nodes of a given object. These also participate in message passing (Fig. 2b) to enable long-range communication within an object[1].

3.1.1 ENCODER

The encoder constructs the input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{Q}^{\mathcal{F}})$ from the mesh M^t. The vertices $\mathcal{V} = \{v_i\}$ represent the mesh nodes, $\mathcal{E} = \{e_{v_s \to v_r}\}$ are bidirectional directed edges added for all pairs of nodes that are connected in the mesh, and $\mathcal{Q}^{\mathrm{F}} = \{q_{\mathcal{F}_s \to \mathcal{F}_r}\}$ are edges between the *faces* of different objects that are close spatially. Each edge in $\mathcal{Q}^{\mathcal{F}}$ is defined between sets of nodes of the sender face $\mathcal{F}_s = (v_{s1}, v_{s2}, v_{s3})$ and the receiver face $\mathcal{F}_r = (v_{r1}, v_{r2}, v_{r3})$.

Node features and mesh edges We follow [31] for constructing the features for each mesh node v_i and mesh edge $e_{v_s \to v_r}$. The features of v_i are the finite-difference velocities estimated from the position history $\mathbf{v}_i^{\text{features}} = [\mathbf{x}_i^t - \mathbf{x}_i^{t-1}, \mathbf{x}_i^{t-h+1} - \mathbf{x}_i^{t-h}]$, where [] indicates concatenation. The

[1]For simplicity on the notation we will omit "object nodes" in the explanation of the model in the main text but a full mathematical description of the model is provided in the Appendix

features of $e_{v_s \to v_r}$ are computed as $\mathbf{e}^{\text{features}}_{v_s \to v_r} = [\mathbf{d}_{rs}, \mathbf{d}^U_{rs}]$, where $\mathbf{d}_{rs} = \mathbf{x}_s - \mathbf{x}_r$ is the relative displacement between the nodes in the current mesh M^t and $\mathbf{d}^U_{rs}$ is the relative displacement in the reference mesh M^U. Note that this representation of the graph is translation-invariant, as it does not contain the absolute positions $\mathbf{x}_i$, leading to a translation equivariant model with better generalization properties [31]. Finally, we use separate MLP encoders to embed the features of each mesh node, and mesh edge into latent representations $\mathbf{v}_i$ and $\mathbf{e}_{v_s \to v_r}$.

Face-face edge features To model face-face collisions between two rigid objects, we introduce a new type of face-face edges, $\mathcal{Q}^{\text{F}}$. A face-face edge $q_{\mathcal{F}_s \to \mathcal{F}_r}$ connects a sender face $\mathcal{F}_s = (v_{s1}, v_{s2}, v_{s3})$ to a receiver face $\mathcal{F}_r = (v_{r1}, v_{r2}, v_{r3})$. To determine which faces to connect, we use the Bounding Volume Hierarchy (BVH) algorithm [11] to find all pairs of faces from different objects that are within a certain pre-defined collision distance d_c.

To construct face-face edge features (Fig. 2a) we compute the closest points between the two faces, p_s and p_r for faces $\mathcal{F}_s$ and $\mathcal{F}_r$ respectively. Geometrically, the closest point for the sender/receiver face might be either inside of the face, on one of the face edges, or at one of the nodes. Using the closest points as local coordinate systems for each of the faces involved in the collision, we build the following translation equivariant vectors: (1) the relative displacement between the closest points $\mathbf{d}^{\mathcal{F}}_{rs} = \mathbf{p}_r - \mathbf{p}_s$ at the two faces, (2) the spanning vectors of three nodes of the sender face $\mathcal{F}_s$ relative to the closest point at that face $\mathbf{d}^{\mathcal{F}}_{s_i} = \mathbf{x}_{s_i} - \mathbf{p}_s$, (3) the spanning vectors of three nodes of the receiver face $\mathcal{F}_r$ relative to the closest point at that face $\mathbf{d}^{\mathcal{F}}_{r_i} = \mathbf{x}_{r_i} - \mathbf{p}_r$, and (4) the face normal unit-vectors of the sender and receiver faces $\mathbf{n}_s$ and $\mathbf{n}_r$. These vectors are the equivalent to the relative displacement for $\mathbf{d}_{rs} = \mathbf{x}_s - \mathbf{x}_r$ in regular edges, and provide a complete, fully-local coordinate system while keeping the model translation equivariant. In summary:

$$\mathbf{q}^{\text{features}}_{\mathcal{F}_s \to \mathcal{F}_r} = [\mathbf{d}^{\mathcal{F}}_{rs}, [\mathbf{d}^{\mathcal{F}}_{s_j}]_{j=1,2,3}, [\mathbf{d}^{\mathcal{F}}_{r_j}]_{j=1,2,3}, \mathbf{n}_s, \mathbf{n}_r] \tag{1}$$

We use an MLP followed by a reshape operation to encode **each face-face edge as three latent face-face edge vectors**, one for each receiver node[2]:

$$Q_{\mathcal{F}_s \to \mathcal{F}_r} = [\mathbf{q}_{j,\mathcal{F}_s \to \mathcal{F}_r}]_{j=1,2,3} = \text{reshape}(\text{MLP}^{\text{encoder}}_{\mathcal{Q}^{\text{F}}}(\mathbf{q}^{\text{features}}_{\mathcal{F}_s \to \mathcal{F}_r}))) \tag{2}$$

Crucially, for each face-face edge, and before computing features (2) and (3), we always sort the nodes of the sender face and the receiver face as function of the distance to their respective closest points. This achieves permutation equivariance of the entire model w.r.t. the order in which the sender and receiver nodes of each face are specified.

3.1.2 PROCESSOR

The procesor is a GNN generalized to directed hyper-edges that iteratively applies message passing across the mesh edges and face-face edges. We use 10 iterations of message passing with unshared learnable weights. Below we describe a single step of message passing to update the edge latents $\mathbf{e}_{v_s \to v_r}$, the face-face edge latents $Q_{\mathcal{F}_s \to \mathcal{F}_r}$, and the node latents $\mathbf{v}_i$, and produce $\mathbf{e}'_{v_s \to v_r}$, $Q'_{\mathcal{F}_s \to \mathcal{F}_r}$, and $\mathbf{v}'_i$.

The mesh edge latents $\mathbf{e}_{v_s \to v_r}$ are updated following the standard edge update approach from [8]:

$$\mathbf{e}'_{v_s \to v_r} = \text{MLP}^{\text{processor}}_{\mathcal{E}}([\mathbf{e}_{v_s \to v_r}, \mathbf{v}_s, \mathbf{v}_r]) \tag{3}$$

The approach to update each face-face edge is similar, except that each face-face edge has 3 latent vectors, 3 sender mesh nodes and 3 receivers mesh nodes; and produces 3 output vectors:

$$Q'_{\mathcal{F}_s \to \mathcal{F}_r} = [\mathbf{q}'_{j,\mathcal{F}_s \to \mathcal{F}_r}]_{j=1,2,3} = \text{reshape}(\text{MLP}^{\text{processor}}_{\mathcal{Q}^{\text{F}}}([[\mathbf{q}_{j,\mathcal{F}_s \to \mathcal{F}_r}, \mathbf{v}_{s_j}, \mathbf{v}_{r_j}]_{j=1,2,3}])) \tag{4}$$

noting that this does not update the three latent vectors independently, but rather all nine input latent vectors $[[\mathbf{q}_{j,\mathcal{F}_s \to \mathcal{F}_r}, \mathbf{v}_{s_j}, \mathbf{v}_{r_j}]_{j=1,2,3}]$ contribute to all three updated face-face edge latent vectors ${Q_{\mathcal{F}_s \to \mathcal{F}_r}}'$.

Finally, we perform the node update based on the incoming messages from face-face edges, as well as other edges between the individual nodes:

[2]The reason for this will become apparent in Eq. 5

$$\mathbf{v}_i' = \text{MLP}_{\mathcal{V}}^{\text{processor}}\Big(\big[\mathbf{v}_i, \sum_{\forall e_{v_s \to v_r}/v_r = v_i} \mathbf{e}_{v_s \to v_r}{}', \sum_{\forall q_{\mathcal{F}_s \to \mathcal{F}_r}/\mathcal{F}_r[j] = v_i} \mathbf{q}'_{j,\mathcal{F}_s \to \mathcal{F}_r}\big]\Big) \tag{5}$$

The second term corresponds to standard edge aggregation[8], and the third term sums over the set of face-face edges for which v_i is one of the receivers. Crucially, $\mathbf{q}'_{j,\mathcal{F}_s \to \mathcal{F}_r}$ selects the specific part of the face-face edge latent corresponding to v_i as a receiver, i.e. $Q'_{\mathcal{F}_s \to \mathcal{F}_r}$, selects the first, second, or third vector, depending on whether v_i, is the first, second or third vector in that receiver face.

3.1.3 Decoder and Position Updater

Following the approach of MeshGraphNets [31], the model predicts the acceleration $\mathbf{a}_i$ by applying an MLP to the final mesh node latents $\mathbf{v}_i$, which adds the bias for inertial dynamics to the model predictions. Finally, we update the node positions through a second order Euler integrator: $\mathbf{x}_i^{t+1} = \mathbf{a}_i + 2\mathbf{x}_i^t - \mathbf{x}_i^{t-1}$.

3.1.4 Implementation

Object-level nodes In a perfect rigid, information propagates across the rigid at infinite speed. However, with n message passing steps, information can only propagate n hops, and for a very fine mesh when a collision happens it may take *many* message passing steps to reach all other nodes in the mesh. To facilitate long range communication, we add a virtual object-level node $\mathbf{v}^O$ at the center of each rigid object, and connect it to all of the mesh nodes of the object via bidirectional mesh-object edges. Ultimately, these extra edges and nodes will also participate in message passing equations 3 and 5, as described in the Appendix. This design allows completely decoupling the dynamics from mesh complexity as collision events can be computed on local neighborhoods using local message passing on edges $\mathcal{E}$, and the resulting collision impulses can be aggregated and broadcasted using the virtual object node $\mathbf{v}^O$.

Loss and model training Following MeshGraphNets [31], we define the loss as the per-node mean-squared-error on the predicted accelerations $\mathbf{a}_i$, whose ground truth values are estimated from the 2 previous positions and the future target position.

We train the model using the Adam optimizer [21]. To stabilize long rollouts, we add random-walk noise to the positions at train time [35, 31, 4]. We also augment the dataset by rotating the scenes by random angles θ around the Z axis.

Generating rollouts and evaluation At test time we generate trajectories by iteratively applying the model to previous model predictions. Following Allen et al. [4], Müller et al. [28], we use shape matching to preserve the shape of the object during rollouts. We use the predicted positions of the nodes in the object mesh to compute the position of the object center, as well as the rotation quaternion with respect to the reference mesh M^U. Then we compute the new node positions by applying this rotation to the reference mesh M^U and shifting the center of the object. This is performed only during inference, not during training.

3.2 Baselines

Our baselines study the importance of how we represent and resolve collisions. In particular, we consider the baseline outlined in Figure 3 which detect collisions in different ways depending on whether they represent objects with particles, or meshes, and whether they represent collisions between nodes or faces.

MeshGraphNets (MGNs) [31] are similar to FIGNet in that they represent the object as a mesh, with messages propagated along the mesh surface. However, when detecting collisions between objects, they connect *nodes* that are within some fixed distance d_c. For objects that have sparse meshes (and may collide along an edge or face, see Figure 3), collisions may go undetected if the edge is sufficiently long relative to d_c. **MeshGraphNets-LargeRadius** is an attempt to fix this issue of MGNs by increasing the radius d_c to the longest edge length in the mesh to ensure collisions are detected. However, choosing such a large radius can lead to an explosion in the number of detected

collisions, which significantly slows down computation (see Results). MGNs also do not use an object-level node for efficient message-passing across the object.

DPI Reimplemented If we relax the requirement that objects are represented as meshes, another popular way to represent objects is as collections of particles [25]. Particles are typically densely sampled throughout an object's volume which allows for collision detection and efficient message propagation, but this then requires using many more nodes. We reimplement DPI [25] as a particle-based baseline by converting meshes to particle point clouds using a radius of 0.025. We additionally use the slightly different loss formulation from DPI (see Appendix), and predict velocities rather than accelerations.

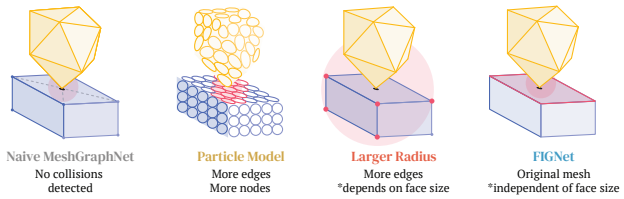


Figure 3: Schematic of how baselines treat interacting objects. If one object's vertex intersects another object's face, a small collision radius d_c will not detect a collision (Naive MeshGraphNet). A dense particle representation fixes the problem but adds more edges and more nodes. Using a larger collision radius introduces quadratically more edges. By operating on faces, FIGNet can use a small collision radius without missing collisions.

Representation of the floor In all of our datasets, the floor is the largest and simplest object. While FIGNet has the advantage of being able to represent the floor using only two large triangles, the baselines require a denser floor to ensure that collisions with the floor are detected. In practice, this means that we refine the floor mesh so that triangle edges are not longer than 1.5 units (compared to a maximum object size of 1.5). These variants are marked with $+$ throughout the results.

For more complex datasets, refining the floor mesh can lead MGN and DPI models to run out of memory during training due to excessive numbers of nodes and collision edges. When this happens, we replace the floor mesh by an implicit floor representation [35, 4] where each node includes a feature representing the distance to the floor, clamped to be within d_c. These variants are marked with * throughout the results. Note that this technique is fundamentally limited: it assumes that there is a fixed number of immovable large objects (the floor, or optionally also the walls), and therefore this model cannot be applied to scenes with varying numbers of walls or floor planes either during training or inference.

4 RESULTS

Here we investigate FIGNet's ability to model both simulated rigid body scenes as well as real world rigid interactions. We focus on investigating three key aspects of FIGNet: its efficiency, accuracy, and generalization. Across simple and complex simulated scenes, FIGNet outperforms node-based baselines in computational efficiency (by up to 8x) and in rollout accuracy (up to 4-8x). FIGNet also generalizes remarkably well across object and floor geometries far outside its training distribution. Finally, we demonstrate that FIGNet approximates real world planar pushing data better than carefully tuned analytic simulators like PyBullet [12], MuJoCo [38], and hand-crafted analytic pushing models [26]. Throughout the results, we compare to the representative model variations outlined in Figure 3. For rollout videos, please refer to the website: sites.google.com/view/fig-net.

Efficient modeling for simple meshes As alluded to previously, even relatively simple meshes can be challenging to model for node-based techniques. To accurately model the interaction between faces or edges of meshes, node-based models must increase their computational budget by either remeshing an object so the distance between nodes is smaller, or by increasing their collision distance d_c to ensure that possible collisions are detected. This is particularly problematic when objects have simple meshes which mostly consist of large faces, like cubes and cylinders.

The Kubric MoviA dataset [17] perfectly encompasses this scenario. In this dataset, 3-10 rigid objects (cubes, spheres or cylinders) are tossed onto a floor. The distance between nodes *within* an object's geometry can range from 0.25 to 1.4. To compare models, we measure the root mean squared error (RMSE) between the predicted and ground truth trajectories for translation and rotation after 50 rollout steps.

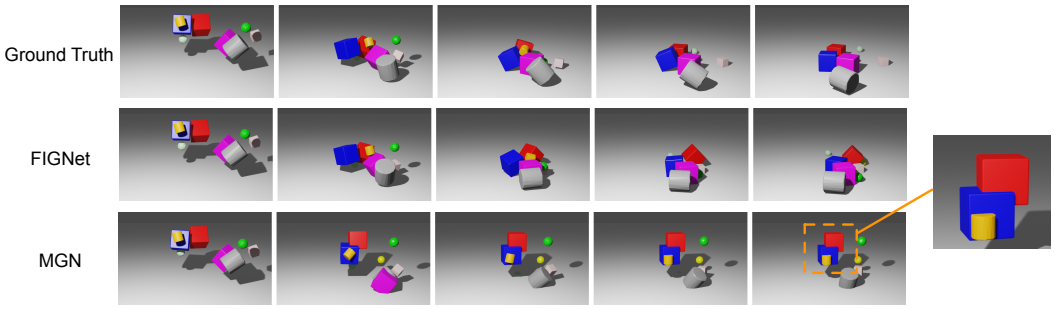
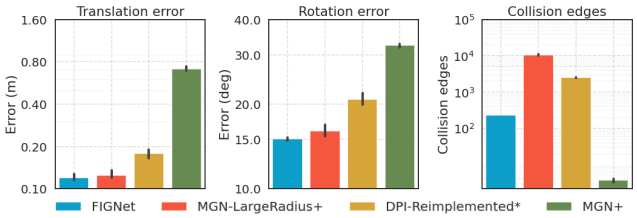


Figure 4: Rollout examples on Movi-A dataset. The node-based MeshGraphNet model struggles to resolve the collision between the sides of the sparse cube meshes, where the nodes are only in the corners of the cube.

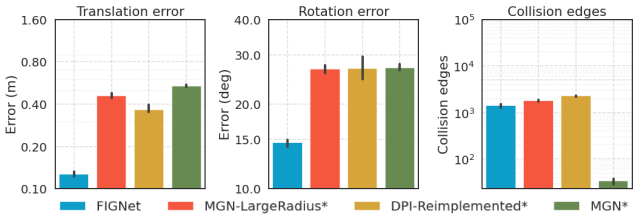
As expected, because the object geometries are sparse, using the node-based MGN with the same collision radius as FIGNet (d_c=0.1) exhibits poor prediction performance, with objects often interpenetrating one another (Figure 4) leading to poor translation and rotation errors relative to ground truth (Figure 5a). Increasing the collision radius to 1.5 solves these problems, but at great computational cost with an explosion in the number of collision edges that must now be considered (Figure 5a). This is especially true for DPI-Nets (d_c=0.5), the particle-based method, which can no longer represent the floor as sets of particles without running out of memory during training. By comparison, FIGNet predictions are plausible with low translation and rotation errors (Figure 4), while not increasing the number of collision edges as dramatically, and therefore maintaining computational efficiency.

Modeling unprecedentedly complex scenes FIGNet's representation of object geometries as faces enables more accurate predictions for more complex meshes without sacrificing efficiency. The Kubric MoviB dataset [17] contains much more complex shapes, including teapots, gears, and torus knots, with a few hundred up to just over one thousand vertices per object.

Despite the incredible complexity of this dataset, FIGNet produces exceptionally good rollouts that closely track the ground truth dynamics (Figure G.3). FIGNet has 4x lower translation RMSE, and 2x lower rotation rotation RMSE compared to other models (Figure 5b). Note that the meshes in MoviB are complex enough that all baselines run out of memory if applied to MoviB directly with a densely sampled floor. Therefore, all baselines use the implicit floor distance feature, which greatly reduces the number of overall edges (since there will be no detected collisions with the floor). Even with this adjustment, the node-based baselines (DPI-Reimplemented* (d_c=0.25) and MGN-LargeRadius (d_c=0.3)) still generally require more collision edges than FIGNet (Figure 5b). Moreover, DPI-Reimplemented* requires dense particle representations of the objects (on average 4000 nodes per simulation compared to the 1500 nodes in the original mesh), increasing the model runtime and memory requirements.



(a) Kubric MoviA



(b) Kubric MoviB

Figure 5: Performance for FIGNet and several baseline methods: root mean squared-error on translation and rotation after 50 timesteps, and number of collision edges for (a) MoviA and (b) MoviB datasets. FIGNet has significantly better translation and rotation error, while also creating fewer collision edges overall. Models with * use an implicit floor representation, while models with + use a subdivided floor.

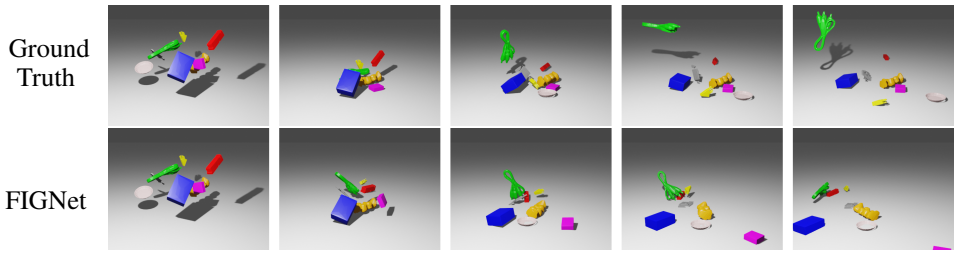


Figure 6: Rollout example on Kubric MoviC dataset [17] from a FIGNet model trained on the MoviB dataset. FIGNet is able to generalize to the complex geometries of the MoviC objects which it has never seen during training.

The performance of FIGNet here is partly driven by the object-level nodes that FIGNet uses to more efficiently propagate information through object meshes, but not entirely (see ablations in Figure E.1 and Figure E.1). This partly explains why DPI-Reimplemented* performs better than MGNs in this dataset, as it also uses object-level virtual nodes.

Generalization to different object and floor geometries Given FIGNet's strong performance on MoviA and MoviB, we next sought to understand how well FIGNet generalizes across different object shapes. We take the FIGNet models pre-trained on the MoviA and MoviB datasets and apply them to the MoviB and MoviC respectively. In both the MoviA-to-MoviB and MoviB-to-MoviC generalization settings, FIGNet has 2-4x lower translation and rotation error than the baselines (Figure G.1a, Figure G.1b) and produces realistic-looking rollouts (Figure 6, Figure G.4). This is particularly remarkable in the MoviA-to-MoviB setting, because the model has only observed 3 different object shapes during training (cubes, cylinders and spheres). Furthermore, the MoviB translation error of FIGNet trained on MoviA (Figure G.1a) is *better* than any baseline models *trained directly on the MoviB dataset* (Figure 5b), once again demonstrating the generalization ability of face-face collisions. In contrast, DPI-Reimplemented* and both versions of MGNs struggle to generalize to scenes with new objects (Figure G.1a and Figure G.1b).

Modeling real world dynamics better than analytic simulators In the preceeding sections, we saw that FIGNet significantly outperforms alternative learned models for complex, simulated datasets. However, the real world is more complicated than simulation environments – even simple dynamics like planar pushing under friction are hard to model because of sub-scale variations in object surfaces. Here we investigate how FIGNet compares to analytic simulators in modeling real-world pushing dynamics.

For these experiments, we use the MIT Pushing Dataset ([41]; Figure 7a), which consists of 6000 real-world pushes for 11 different objects along 4 different surfaces (see subsection B.2 for details). The objects are tracked for the full push. We sample different training dataset sizes to examine FIGNet's sample efficiency and test on a set of 1000 trajectories across objects. To mitigate the effect of outlier trajectories, we report the median translation and rotation error between the ground truth and predicted trajectories after 0.5 seconds, which corresponds to 125 timesteps. To provide an interpretable unit for the metrics, we scale the rotation and translation error by the error that would

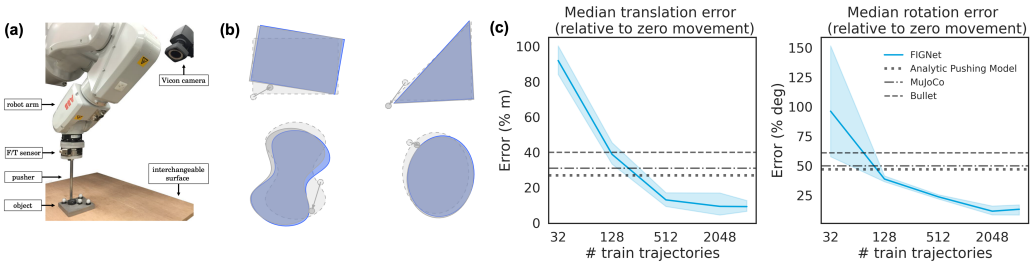


Figure 7: MIT Pushing dataset results. (a) A demonstration of the setup from Yu et al. [41]. (b) Example rollouts from FIGNet (blue) vs. ground truth (gray). The dashed outline indicates the initial object position. (c) Performance on MIT Pushing dataset for FIGNet as compared to analytic simulators MuJoCo [38], PyBullet [12], and the planar pushing analytic model developed by [26] and implemented for this dataset by Kloss et al. [23].

be predicted by a model that always outputs zeroes. 100% error therefore corresponds to performing no better than a model which always predicts 0.

We compare FIGNet to two widely used analytic rigid body simulators, MuJoCo [38] and PyBullet [12], as well as the planar pushing specific model from Lynch [26]. For tuning system parameters of the analytic simulators, we use Bayesian Optimization on the training dataset with all available friction parameters, as well as the "timestep" for MuJoCo and PyBullet. Futhermore, while MuJoCo and PyBullet trajectories are evaluated based on the tracked object positions directly (and are therefore prone to perception errors), the planar pushing analytic model uses the metadata for each pushing trajectory (giving information about the pushing direction, velocity, etc. without perception noise). This allows us to separately examine whether FIGNet improves on analytic simulators because it can compensate for perception errors (where it would perform better than PyBullet and MuJoCo only) or because it can model pushing under friction more effectively (where it would perform better than all analytic models). We evaluate the analytic models on the same test dataset as FIGNet.

Remarkably, FIGNet outperforms all analytic simulators trained on only 128 trajectories of real world data (Figure 7c). Unlike MuJoCo and PyBullet, FIGNet is robust to errors in perception which occasionally reveal the pusher moving an object even when not in contact, or when the pusher penetrates the object. However, because FIGNet also improves on the analytic simulator [26], this means that it is also simply modeling pushing under friction more accurately than can be analytically modeled. Its particular improvement in rotation error relative to the analytic models suggests that it can model torsional friction more accurately. This is exciting as it opens new avenues for considering when learned models may be preferable to more commonly used analytic simulators.

5 DISCUSSION

We introduced FIGNet, which models collisions using multi-indexed face-to-face interactions, and showed it can capture the complex dynamics of multiple rigid bodies colliding with one another. Compared to prior node-based GNN learned simulators, FIGNet can simulate much more complex rigid body systems accurately and efficiently. It shows strong generalization properties both across object shapes, and even to different floor geometries. On real-world data, FIGNet outperforms analytic simulators in predicting the motion of pushed objects, even with relatively little data, presenting an exciting avenue for future development of learning-based simulators that model real world behavior better than current analytical options, such as MuJoCo [38] or PyBullet [12].

Our face-to-face approach is not limited to triangular faces, nor rigid body interactions. Extensions to quadric, tetrahedral, etc. faces can use the same message-passing innovations we introduced here. Similarly, while this work's focus was on rigid body dynamics, in principle FIGNet naturally extends to multi-material interactions. Soft bodies or fluids can be represented by collections of particles, and their interactions with faces can be modeled similarly to FIGNet. Exciting directions for future work could investigate whether different kinds of face interactions could even be learned from different datasets, and then composed to handle few-shot dynamics generalization.

In terms of limitations, FIGNet, like other GNN-based simulators, is trained with a deterministic loss function. In practice, this works well for the complex rigid systems studied here, however for highly stochastic or chaotic systems this may not be as effective. The datasets studied here include complex geometry, however the extent to which FIGNet will scale to extremely complex meshes (millions of faces) is unclear, and other approaches, e.g., implicit models, may be more suitable. While FIGNet works better than analytic simulators for real world data, it still requires access to state information, and future work should investigate how it can be coupled with perception to enable learning from raw pixel observations.

FIGNet represents an important advance in the ability to model complex, rigid dynamics. By modeling mesh geometries directly, FIGNet avoids the pitfalls of previous node-based learned simulators. FIGNet opens new avenues for learned simulators to be used as potential alternatives to existing analytic methods even for very complex scenes, with valuable implications for robotics, mechanical design, graphics, and beyond.

REFERENCES

[1] Brian Acosta, William Yang, and Michael Posa. Validating robotics simulators on real-world impacts. *IEEE Robotics and Automation Letters*, 2022.

[2] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[3] Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310, 2020. doi: 10.1073/pnas.1912341117. URL https://www.pnas.org/doi/abs/10.1073/pnas.1912341117.

[4] 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 *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=rbIzq-I84i_.

[5] M Anitescu and F A Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14(3):231–247, 1997.

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[7] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems 29*, pp. 4502–4510. Curran Associates, Inc., 2016.

[8] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[9] Maria Bauza and Alberto Rodriguez. A probabilistic data-driven model for planar pushing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3008–3015, 2017.

[10] Daniel M Bear, Elias Wang, Damian Mrowca, Felix J Binder, Hsiau-Yu Fish Tung, RT Pramod, Cameron Holdaway, Sirui Tao, Kevin Smith, Fan-Yun Sun, et al. Physion: Evaluating physical prediction from vision in humans and machines. *arXiv preprint arXiv:2106.08261*, 2021.

[11] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, oct 1976. ISSN 0001-0782. doi: 10.1145/360349.360354. URL https://doi.org/10.1145/360349.360354.

[12] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, pp. 7, 2015.

[13] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. *arXiv preprint arXiv:2204.11918*, 2022.

[14] Nima Fazeli, Elliott Donlon, Evan Drumwright, and Alberto Rodriguez. Empirical evaluation of common contact models for planar impact. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3418–3425. IEEE, 2017.

[15] Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pp. 817–828. PMLR, 2018.

[16] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1487–1495, 2017.

[17] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3749–3761, June 2022.

[18] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9474–9481. IEEE, 2021.

[19] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[20] Yifeng Jiang, Jiazheng Sun, and C Karen Liu. Data-augmented contact model for rigid body simulation. In *Learning for Dynamics and Control Conference*, pp. 378–390. PMLR, 2022.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Alina Kloss, Stefan Schaal, and Jeannette Bohg. Combining learned and analytical models for predicting action effects. *CoRR*, abs/1710.04102, 2017.

[23] Alina Kloss, Stefan Schaal, and Jeannette Bohg. Combining learned and analytical models for predicting action effects from sensory data. *The International Journal of Robotics Research*, 41(8):778–797, 2022.

[24] 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*, 2019.

[25] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 1205–1211. IEEE, 2019.

[26] Kevin M Lynch. The mechanics of fine manipulation by pushing. In *ICRA*, pp. 2269–2276. Citeseer, 1992.

[27] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. *Advances in neural information processing systems*, 31, 2018.

[28] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM transactions on graphics (TOG)*, 24(3):471–478, 2005.

[29] Mihir Parmar, Mathew Halm, and Michael Posa. Fundamental challenges in deep learning for stiff contact dynamics. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5181–5188. IEEE, 2021.

[30] Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. Learning to control self-assembling morphologies: A study of generalization via modularity. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

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

[32] Samuel Pfrommer, Mathew Halm, and Michael Posa. Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations. *Conference on Robot Learning (CoRL)*, 2020.

[33] Yulia Rubanova, Alvaro Sanchez-Gonzalez, Tobias Pfaff, and Peter W. Battaglia. Constraint-based graph network simulator. *CoRR*, abs/2112.09161, 2021.

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

[35] 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*, pp. 8459–8468. PMLR, 2020.

[36] D Stewart and JC J.C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with Coulomb friction. *International Journal for Numerical Methods in Engineering*, 39(15): 2673–2691, 1996.

[37] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL `https://drake.mit.edu`.

[38] 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*, pp. 5026–5033. IEEE, 2012.

[39] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[40] Pierre-Brice Wieber, Russ Tedrake, and Scott Kuindersma. Modeling and control of legged robots. In *Springer handbook of robotics*, pp. 1203–1234. Springer, 2016.

[41] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 30–37. IEEE, 2016.

[42] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.