3D-OES: Viewpoint-Invariant Object-Factorized Environment Simulators

Hsiao-Yu Fish Tung*, **Zhou Xian***, **Mihir Prabhudesai, Shamit Lal, Katerina Fragkiadaki**
Carnegie Mellon University
{htung, xianz1, mprabhud, shamitl, katef}@cs.cmu.edu
* These authors contributed equally to this work.

Abstract: We propose an action-conditioned dynamics model that predicts scene changes caused by object and agent interactions in a viewpoint-invariant 3D neural scene representation space, inferred from RGB-D videos. In this 3D feature space, objects do not interfere with one another and their appearance persists over time and across viewpoints. This permits our model to predict future scenes long in the future by simply "moving" 3D object features based on cumulative object motion predictions. Object motion predictions are computed by a graph neural network that operates over the object features extracted from the 3D neural scene representation. Our model's simulations can be decoded by a neural renderer into 2D image views from any desired viewpoint, which aids the interpretability of our latent 3D simulation space. We show our model generalizes well its predictions across varying number and appearances of interacting objects as well as across camera viewpoints, outperforming existing 2D and 3D dynamics models. We further demonstrate sim-to-real transfer of the learnt dynamics by applying our model trained solely in simulation to model-based control for pushing objects to desired locations under clutter on a real robotic setup.

Keywords: Robotic Manipulation, Model-Based Reinforcement Learning

1 Introduction

Humans can effortlessly imagine how a scene will change as a result of their interactions with the objects in the scene [1, 2]. What is the representation space of these imaginations? They are not pixel accurate and, interestingly, they are not affected by occlusions. Consider a teaspoon dipping inside a coffee mug. Though it will be occluded from nearly all viewpoints but the bird's eye view, we have no difficulty keeping it in our mind as present and complete. We can imagine watching it from different viewpoints, increase or decrease its size, predict whether it will fit inside the mug, or even imagine filling the mug with more spoons.

Inspired by human's capability to simulate scene changes in a viewpoint-invariant and occlusion-resistant manner, we present 3D object-factorized environment simulators (3D-OES), an action-conditioned dynamics model that predicts scene changes caused by object and agent interactions in a viewpoint-invariant 3D neural scene representation space, inferred from RGB-D videos. 3D-OES differentiably maps an RGB-D image to a 3D neural scene representation, detects objects in it, and forecasts their future 3D motions, conditioned on actions of the agent. A graph neural network operates on the extracted 3D object feature maps and the action input and predicts object 3D translations and rotations. Our model then generates future 3D scenes by simply translating and rotating object 3D feature maps, inferred from the *first time step*, according to cumulative 3D motion predictions. In this way, we avoid distribution shift in object features caused by forward model unrolling, hence minimizing error accumulation.

Our main insight is that scene dynamics are simpler to learn and represent in 3D than in 2D, for the following reasons: i) **In 3D, object appearance and object location are disentangled.** This

Videos and source code are available at https://zhouxian.github.io/3d-oes/.

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

means object appearance (what) does not vary with object locations (where). This what-where disentanglement permits generating scene variations by simply translating and rotating 3D object appearance representations. Scene generation by moving around objects is not possible in a projective 2D image space, since objects change appearance due to camera viewpoint variation, occlusions or out-of-plane object rotations [3]. It is precisely the permanence of object appearance in 3D that permits easy simulation. ii) **In 3D, inferring free space and object collisions is easy.** Given a 3D scene description in terms of object locations and 3D shapes, we can easily predict whether an object will collide with another or will be contained in another. Similar inferences would require many examples to learn directly from 2D images, and would likely have poor generalization. Yet, extracting 3D scene representations from RGB or RGB-D video streams is a challenging open problem in computer vision research [4, 5, 6, 7, 8]. We build upon the recently proposed geometry-aware recurrent neural networks (GRNNs) [6, 9] to infer 3D scene feature maps from RGB-D images in a differentiable manner, optimized end-to-end for our object dynamics prediction task.

We evaluate 3D-OES in single-step and multi-step object motion prediction for object pushing and falling, and apply it for planning to push objects to desired locations. We test its generalization while varying the number and appearance of objects in the scene, and the camera viewpoint. We compare against existing learning-based 2D image-centric or object-centric models of dynamics [10, 11] as well as graph-based dynamics learned over engineered 3D representations of object locations [12]. Our model outperforms them by a large margin. In addition, we empirically show that training the 2D baselines under varying viewpoints causes them to dramatically underfit on the training data, and be highly inaccurate in the validation set. This suggests that different architectures are necessary to handle viewpoint variations in dynamics learning and 3D-OES is one step in that direction.

In summary, the main contribution of this work is a graph neural network over 3D object feature maps extracted from convolutional end-to-end differentiable 3D neural scene representations for forecasting 3D object motion. Graph networks are widely used in 2D object motion interaction predictions [13, 11, 14, 15]. We show that by porting such relational reasoning in an 3D object-factorized space, object motion prediction can generalize across camera viewpoints, lifting a major limitation of previous works on 2D object dynamics. Moreover, future and counterfactual scenes can be easily generated by translating and rotating 3D object feature representations. In comparison to recent 3D particle graph networks [16, 17, 18], our work can operate over input RGB-D images and not ground-truth particle graphs. In comparison to recent scene-specific image-to-3D particle graph encoders [19], our image to 3D scene encoder can generalize across environments with novel objects, novel number of objects, and novel camera viewpoints, Moreover, our model presents effective sim-to-real transfer to a real-world robotic setup.

2 Related Work

The inability of systems of physics equations to capture the complexity of the world [20] has led many researchers to pursue learning-based models of dynamics, or combine those with analytic physics models to help fight the undermodeling and uncertainty of the world [21, 22]. Learning world models is both useful for model-based control [10, 23, 24] as well as a premise towards unsupervised learning of visuomotor representations [25, 26]. Several formulations have been proposed, under various names, such as world models [23], action-conditioned video prediction [27], forward models [28], neural physics [29], neural simulators, etc. A central question is the representation space in which predictions are carried out. We identify two main research threads:

(i) Methods that **predict the future in a 2D projective space**, such as future visual frames [30, 31, 32, 33], neural encodings of future frames [23, 34, 10, 25, 35], object 2D motion trajectories [36, 14, 29], 2D pixel motion fields [24, 3, 32, 37]. Models that predict motion and use it to warp pixel colors forward in time need to handle occlusions as well as changes in size and aspect ratio. Though increasing the temporal memory from where to borrow pixel colors helps [38, 3], the model needs to learn many complex relationships between pixels to handle such changes over time. Recent architectures incorporate object-centric and relational biases in order to generalize across varying number of objects present in the image, using graph neural networks [39, 13, 11, 14, 15]. Cross-object interactions are captured using edges in an entity graph, where messages are iteratively exchanged between the nodes to update each other's embedding [14, 12]. Most works that use object factorization biases either assume the objects' segmentation masks are given or are trivial to obtain from color segmentation [36, 14, 29]. These models work well under a static and fixed camera

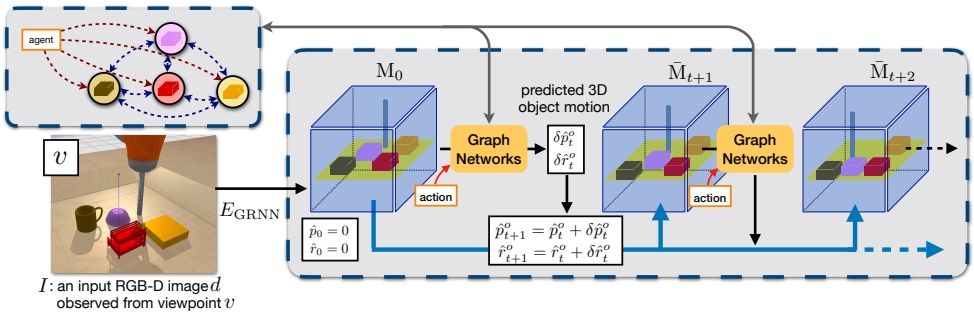

Figure 1: **3D-OES** predict 3D object motion under agent-object and object-object interactions, using a graph neural network over 3D feature maps of detected objects. Node features capture the appearance of an object node and its immediate context, and edge features capture relative 3D locations between two nodes, so the model is translational invariant. After message passing between nodes, the node and edge features are decoded to future 3D rotations and translations for each object.

across training and test conditions, but cannot effectively generalize across camera viewpoints [40] as we empirically validate.

(ii) methods that **predict the future in a 3D space** of object or particle 3D locations and poses extracted from the RGB images using human annotations [41] or assumed given [42, 43, 44]. Such explicit 3D state representations are hard in general to obtain from raw RGB input in-the-wild, outside multiview environments [41, 43]. The work of [19] attempts to extract the particle 3D locations directly from images, however the encoder proposed is specific to the scene. Different encoders are trained for different scene image to particle mappings.

Our work builds upon learnable 2D-to-3D convolutional encoders that extract 3D scene representations from images and are trained for self-supervised view prediction, along with tasks of 3D object detection and 3D motion forecasting, relevant for 3D object dynamics learning. Our image-to-3D scene encoders and object detectors generalize across object appearance and number of objects, and do not assume any ground-truth information about the object or particle locations [16, 17, 18] or image segmentation [45] during test time.

3 Object-Factorized Environment Simulators (3D-OES)

The architecture of 3D-OES is depicted in Figure 1. At each time step, our model takes as input a single or a set of RGB-D images of the scene along with the corresponding camera views to capture them, and encodes these inputs into a 3D scene feature representation by neurally mapping image and depth maps to 3D feature grids (Section 3.1). Then, it detects 3D object boxes in the inferred 3D scene representation, and crops the scene representation to obtain a set of object-centered 3D feature maps. A graph neural network over the object nodes will take as inputs object appearances and the agent actions and predict the future 3D rotation and translation for each object (Section 3.2). We will assume for now rigid objects, and we discuss in Section 5 how to extend our framework to deformable and articulated objects. Our model generates future scenes by warping object-centric 3D feature maps with the predicted cumulative 3D object motion (Section 3.2). These synthesized future 3D scene feature maps, though not directly interpretable, can be decoded to RGB images from any desired camera viewpoints via a neural renderer to aid interpretability. We use long-term simulations of 3D-OES to generate action plans for pushing objects to desired locations in cluttered environments using model predictive control (Section 3.3). We apply our model to learn dynamics of objects pushed around on a table surface and objects falling on top of others. At training time, we assume access to 3D object bounding boxes to train our 3D object detector.

3.1 Differentiable 2D-to-3D lifting with Geometry-Aware Recurrent Networks (GRNNs)

Geometry-Aware Recurrent Networks (GRNNs) introduced in [6, 9] are network architectures equipped with a differentiable unprojection (2D-to-3D) module and a 3D scene neural map as their bottleneck. They can be trained end-to-end for a downstream task, such as supervised 3D object detection or self-supervised view prediction. We will denote the 3D scene feature as $\mathbf{M} \in \mathbb{R}^{w \times h \times d \times c}$ where w, h, d, c denote width, height, depth and number of channels, respectively. Every (x, y, z) grid location in the 3D feature map $\mathbf{M}$ holds a c-dimensional feature vector that describes the se-

mantic and geometric properties of a corresponding physical location in the 3D world scene. Given an input video, GRNNs estimate the relative camera poses between frames, and transform the inferred 3D features map $\mathbf{M}_t$ to a world coordinate frame to cancel the camera egomotion, before accumulating it with 3D feature maps across time steps. In this way, information from 2D pixels that correspond to the same 3D physical point end up nearby in the 3D neural map. We use such cross-view registration in case we have access to concurrent multiple camera views for the first timestep of our simulations. Upon training, GRNNs map RGB-D images or a single RGB-D image to a *complete* 3D feature map of the scene they depict, i.e., the model learns to *imagine* the missing or occluded information from the input view. We denote this 2D-to-3D mapping as $\mathbf{M} = E_{\text{GRNN}}(I_1, ..., I_t)$, where $\mathbf{M} \in \mathbb{R}^{w \times h \times d \times c}$ and $I_t = \{d_t, v_t\}$ denotes the RGB-D image d_t and the corresponding camera pose v_t at time step t. Note that the input can be a single RGB-D view, in which case $\mathbf{M} = E_{\text{GRNN}}(I)$. For further details on GRNNs, please refer to [6, 9].

View prediction for visualizing latent 3D neural simulations We train GRNNs end-to-end for RGB view regression in videos of static scenes and moving cameras as proposed in [6], by neurally projecting the 3D scene feature maps and mapping them to 2D images. Our decoder involves a differentiable 3D-to-2D projection module that projects the 3D scene feature representation after orienting it to the query camera viewpoint. The projected features are then decoded into images through a learned decoder. In this way, the trained projection and decoding module can be used to interpret and visualize the 3D latent feature space with view-specific 2D images, given any desired camera viewpoint.

3D object detection Our model uses a 3D object detector to map the 3D scene neural map $\mathbf{M}$ to a variable number of object axis-aligned 3D boxes and corresponding 3D segmentation masks, i.e., binary 3D voxel occupancies: $\mathcal{O} = Det(\mathbf{M}), \mathcal{O} = \{\hat{b}^o = (p_x^o, p_y^o, p_z^o, w^o, h^o, d^o) \in \mathbb{R}^6, m^o \in \{0,1\}^{w^o \times h^o \times d^o}, o = 1 \cdots |\mathcal{O}|\}$, where p_x^o, p_y^o, p_z^o stands for the 3D box centroid and w^o, h^o, d^o stands for 3d box size. Its architecture is similar to Mask R-CNN [46] but uses 3D input and output instead of 2D. Given an object 3D centroid p_x^o, p_y^o, p_z^o, we crop the 3D scene feature map $\mathbf{M}$ using a corresponding fixed-size axis-aligned 3D bounding box to obtain corresponding object-centric feature maps $\mathbf{M}^o, o = 1 \cdots |\mathcal{O}|$ for all objects in the scene.

3.2 3D Object Graph Neural Networks for Motion Forecasting

Objects are the recipient of forces exercised by active agents; meanwhile, objects themselves carry momentum and cause other objects to move. How can we model cross-object dynamic relationships in a way that generalizes with varying number of objects and arbitrary chains of interactions?

We consider a graph interaction network [14] over the graph comprised of the detected objects and the agent's end-effector. Inputs to the network are the object-centric feature maps, one per object node, the objects' velocities, the agent's action represented as a 3D translation, as well as edge features, which incorporate the relative 3D displacements between the nodes. The outputs of the network are the 3D translations $\delta\hat{p}$ and 3D relative rotations $\delta\hat{r}$ of the object nodes at the next time step. During message passing in the constructed graph, edge and node features are encoded and concatenated, and messages from neighboring nodes are aggregated via summation. Our graph network is trained supervised to minimize a standard regression loss for the next time step.

Forward unrolling with object appearance permanence To predict long term results of actions, as well as results of action sequences, the model needs to be unrolled forward in time as commonly done in related works [39, 13, 11, 14, 15]. Different from previous works though, 3D-OES can synthesize 3D neural scenes of future timesteps by warping (translating and rotating) object feature maps obtained from the first timestep—as opposed to the ones obtained from the predicted scene of the previous timestep—according to cumulative 3D motion predictions. Specifically, given predicted 3D object motions ($\delta\hat{p}_t$, $\delta\hat{r}_t$) at an unrolling step t, we estimate the **cumulative** 3D rotation and translation of the object with respect to the first timestep:

$$\hat{p}_t = \hat{p}_{t-1} + \delta\hat{p}_t, \;\; \hat{r}_t = \hat{r}_{t-1} + \delta\hat{r}_t, \;\; t = 1 \cdots T, \;\; \hat{p}_0 = 0 \;\; \hat{r}_0 = 0. \tag{1}$$

where T denotes the number of unrolling steps thus far. Then, given 3D object segmentation masks m^o and object-centric 3D feature maps $\mathbf{M}^o$ obtained by the 3D object detector from the input RGB-D image, we rotate and translate the object masks and 3D feature maps using the cumulative 3D rotation $\hat{r}_t$ and 3D translation $\hat{p}_t$ using 3D spatial transformers. We synthesize a new 3D scene feature map $\bar{\mathbf{M}}_t$ by placing each transformed object-centric 3D feature map at its predicted 3D location: $\bar{\mathbf{M}}_t = \sum_{o=1}^{|\mathcal{O}|} \text{Draw}(\text{Rotate}(m^o, \hat{r}_t^o) \odot \text{Rotate}(\mathbf{M}^o, \hat{r}_t^o), \hat{p}_t^o)$, where superscript o denotes

the object identity, $\mathrm{Rotate}(\cdot, r)$ denotes 3D rotation by angle r, $\odot$ denotes voxel-wise multiplication, and $\mathrm{Draw}(\mathbf{M}, p)$ denotes adding a feature tensor $\mathbf{M}$ at a 3D location p. This synthesized scene map is used for neural rendering to help interpret the predicted scene at t. To obtain the inputs for our graph neural network at the next time step, we can potentially crop the synthesized 3D scene map $\mathbf{M}_t$ at the predicted 3D location. However, we find that directly using object features obtained in the first time step and including accumulative relative object pose as part of the object state works better in practice.

Our graph neural motion forecaster is trained through forward unrolling. Error of each time step is back-propagated through time. More implementation details are included in Appendix Section C.1.

3.3 Model Predictive Control with 3D-OES

Action-conditioned dynamics models, such as 3D-OES, simulate the results of an agent's actions and permit successful control in zero-shot setups: achieving a specific goal in a novel scene without previous practice. We apply our model for pushing objects to desired locations in cluttered environments with model predictive control. Given an input RGB-D image I that contains multiple objects, a goal configuration is given in terms of the desired 3D location of an object $\mathbf{x}^o_{goal}$. 3D-OES infer the scene 3D feature map $\mathbf{M} = E_{\mathrm{GRNN}}(I)$ and detects the objects present in the scene. We then unroll the model forward in time using randomly sampled action sequences, as described in Section 3.2. We evaluate each action sequence based on the Euclidean distance from the goal to the predicted location $\hat{x}^o_T$ (after T time steps) for the designated object. We execute the first action of the best action sequence and repeat [47]. Our model combines 3D perception and planning using learned object dynamics in the inferred 3D scene feature map. While most previous works choose bird's eye viewpoints to minimize cross-object or robot-object occlusions [48], our control framework **can use any camera viewpoint**, thanks to its ability to map input 2.5D images to complete, viewpoint-invariant 3D scene feature maps. We empirically validate this claim in our experimental section.

4 Experiments

We evaluate our model on its prediction accuracy for single- and multi-step object motion forecasting under multi-object interactions, as well as on its performance in model predictive control for pushing objects to desired locations on a table surface in the presence of obstacles. We ablate generalization of our model under varying camera viewpoints and varying number of object and varying object appearance. Our model is trained to predict 3D object motion during robot **pushing** and **falling** in the Bullet Physics Simulator. For **pushing**, we have objects pushed by a Kuka robotic arm and record RGB-D video streams from multiple viewpoints. We create scenes using 31 different 3D object meshes, including 11 objects from the MIT Push dataset [49] and 20 objects randomly selected from *camera*, *mug*, *bowl*, and *bed* object categories of the ShapeNet dataset [50]. At training time, each scene contains at most *two* objects. We test with varying number of objects. For **falling**, we use 3D meshes of the objects introduced in [13], including a variety of shapes. We randomly select 1-3 objects and randomly place them on a table surface, and let one object fall from a height. We train our model with three camera views, and use either three or one randomly selected views as input during test time.

We compare 3D-OES against a set of baselines designed to cover representative models in the object dynamics literature: **(1)** *graph-XYZ*, a model that mimics Interaction Networks [14] and [41, 42]. It is a graph neural network in which object features are the 3D object centroid locations and their velocities, and edge features are their relative 3D locations. **(2)** *graph-XYZ-image*, a model using graph neural network over 3D object centroid locations and object-centric 2D image CNN feature embeddings, similar to [11]. The model further combines camera pose information with the node features. **(3)** Visual Foresight (*VF*) [51], a model that uses the current frame and the action of the agent to predict future 2D frames by "moving pixels" based on predicted 2D pixel flow fields. **(4)** *PlaNet* [10], a model that learns a scene-level embedding by predicting future frames and the reward given the current frame.

We compare our model against baselines *graph-XYZ* and *graph-XYZ-image* on both motion forecasting and model predictive control. Since *VF* and *PlaNet* forecast 2D pixel motion and do not predict explicit 3D object motion, we compare against them on the pushing task with model predictive control. For further details on data collection, train-test data split, and implementation of the baselines, please refer to Appendix (Section B and C.2).

Table 1: **3D object motion prediction test error during object pushing in scenes with two objects** for 1,3, and 5 timestep prediction horizon.

Experiment Setting	Model		T=1	T=3	T=5
3 views (random, novel) + gt-bbox	*graph-XYZ* [14]	translation(mm)	4.6	32.1	66.3
		rotation(degree)	2.8	16.7	26.4
	graph-XYZ-image [11]	translation(mm)	6.0	39.3	69.7
		rotation(degree)	3.4	29.8	30.7
	Ours	translation(mm)	**3.6**	**22.5**	**43.4**
		rotation(degree)	**2.5**	**12.0**	**20.6**
1 view (random, novel) + gt-bbox	*graph-XYZ-image* [11]	translation(mm)	6.0	39.3	69.7
		rotation(degree)	3.4	29.8	30.7
	Ours	translation(mm)	**4.1**	**23.6**	**43.8**
		rotation(degree)	**3.1**	**12.2**	**20.3**
1 view (random, novel) + predicted-bbox	*graph-XYZ* [14]	translation(mm)	6.7	35.4	68.2
		rotation(degree)	3.0	20.1	30.32
	graph-XYZ-image [11]	translation(mm)	6.6	43.1	71.2
		rotation(degree)	3.6	31.8	32.4
	Ours	translation(mm)	**4.3**	**25.2**	**47.0**
		rotation(degree)	**2.7**	**12.1**	**19.7**
1 view (**fixed, same as train**) + predicted-bbox	*graph-XYZ-image* [11]	translation(mm)	5.1	29.6	54.5
		rotation(degree)	2.6	11.0	16.9

Table 2: **3D object motion prediction test error during object falling in scenes with three to four objects** for 1,3, and 5 timestep prediction horizon.

Experiment Setting	Model		T=1	T=3	T=5
1views (random, novel) + predicted-bbox	*graph-XYZ* [14]	translation(mm)	5.2	**11.7**	278.6
		rotation(degree)	**5.7**	**10.4**	43.28
	graph-XYZ-image [11]	translation(mm)	8.4	17.0	620.2
		rotation(degree)	9.2	16.6	117.9
	Ours	translation(mm)	**5.0**	13.1	**16.4**
		rotation(degree)	6.1	12.6	**18.7**

4.1 Action-Conditioned 3D Object Motion Forecasting

We evaluate the performance of our model and the baselines in single- and multi-step 3D motion forecasting for **pushing** and **falling** on novel objects in Tables 1 and 2 in terms of translation and rotation error. We evaluate the following ablations: i) using 1 or 3 camera views at the first time step, ii) using goundtruth 3D object boxes (gt-bbox) or 3D boxes predicted by our 3D detector, iii) varying camera viewpoints (random) versus keeping a single fixed camera viewpoint at train and test time. Our model outperforms the baselines both in translation and rotation prediction accuracy. When tested with object boxes predicted by the 3D object detector (Section 3.1) as opposed to ground-truth 3D boxes, our model is the least affected. *graph-XYZ-image* performs on par with or even worse than *graph-XYZ*, indicating that it does not gain from having access to additional appearance information. We hypothesize this is due to the way appearance and camera pose information are integrated in this baseline: the model simply treats camera pose information as additional input, as opposed to our model, which leverages geometry-aware representations that retain the geometric structure of the scene.

Multi-step forward unrolling The *graph-XYZ* baseline can be easily unrolled forward in time without much error accumulation since it does not use any appearance features. Still, as seen in Tables 1 and 2, our model outperforms it. *graph-XYZ* is oblivious to the appearance of the object and thus cannot effectively adapt its predictions to different object shapes.

Varying number of camera views Our model accepts a variable number of views as input, and improves when more views available; yet, it can accurately predict future motion even from a single RGB-D view. The prediction error of our single view model is only slightly higher than the model using three random views as input. As shown in Table 1, the *graph-XYZ-image* baseline performs the worst and does not improve with more views are available. We believe this is due to the geometry-

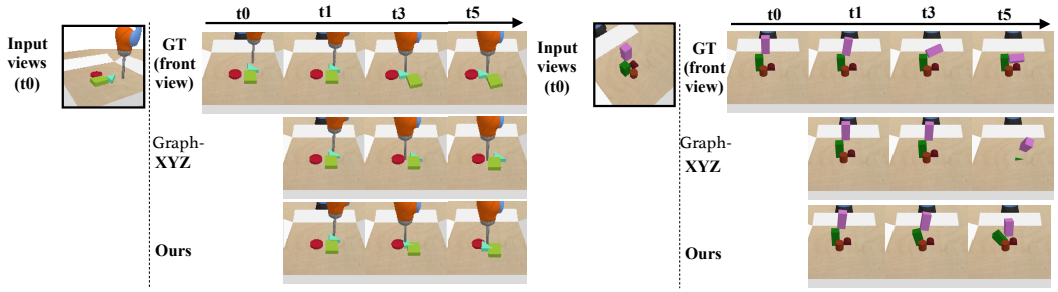

Figure 2: **Forward unrolling of our dynamics model and the *graph-XYZ* baseline.** Left: pushing. Right: falling. In the top row, we show (randomly sampled) camera views that we use as input to our model. The second row shows the ground-truth motion of the object from the front view. Rows 3, 4 show the predicted object motion from our model and the *graph-XYZ* baseline from the same front camera viewpoint. Our model better matches the ground-truth object motion than the *graph-XYZ* baseline. The latter does not capture object appearance in any way.

Table 3: Success rate for pushing objects to target locations.

graph-XYZ [14]	*graph-XYZ-image* [11]	*VF*[51]	*PlaNet*[10]	Ours	Ours-Real
0.76	0.70	0.32	0.16	**0.86**	0.78

unaware way of combining multiview information by concatenation, though the model does have access to camera poses of the input images.

Varying camera viewpoint versus fixed camera viewpoint We show in Table 1 (last 2 rows) that *graph-XYZ-image* can achieve much better performance when trained and tested on a single fixed camera viewpoint. This is a setting widely used in the recently popular learning-based visual-motor control literature [24, 52, 53, 51], which restricts the corresponding models to work only under carefully controlled environments with a fixed camera viewpoint, while ours performs competitively to these model but also handles arbitrary camera viewpoints.

Visualization of the 3D motion predictions In Figure 2, we show qualitative long term motion prediction results produced by unrolling our model forward in time (more are shown in the supplementary video). Our model generalizes to novel objects and scenes with varying number of objects, though trained only on 2 object scenes.

Neural rendering and counterfactual simulations 3D-OES not only can simulate the future state of the scene, it also provides us a way to interpret the latent 3D representation and a space to run counterfactual experiements. We visualize the latent 3D feature map by neurally projecting it from a camera viewpoint to an image through a learned neural decoder, and show the resulting images in Figure 3. We also show that our 3D representation allows us to alter the observed scene and run conterfactual simulations in multiple ways. More results are provided in Appendix Section A.

4.2 Pushing with Model Predictive Control (MPC)

We test 3D-OES on pushing objects to desired locations using MPC and report the results in Table 3. For our model and *graph-XYZ-image*, we use a single randomly sampled input view. For *VF* and *PlaNet*, we use a fixed top-down view for both training and testing as we found they only work reasonably well with a fixed viewpoint. Details of the experiment settings are included in Appendix (Section C.3). Our model outperforms all baselines by a large margin. We include videos of pushing object to desired locations in the presence of multiple obstacles in the supplementary file.

Sim-to-Real Transfer We train our model solely in simulation and test it on object pushing control tasks on a real Baxter platform equipped with a rod-shaped end-effector, similar to the setting in the Bullet simulation (Figure 4). We attached a Intel RealSense D435 RGB-D camera to the robot's left hand, and use only one RGB-D view as input for this experiment. The pose of the camera is different from those seen during training. Please refer to Appendix (Section C.3) for details of our real-world setup, objects selection, and 3D detector training. We report the success rate of real-world pushing in Table 3 (Ours-Real). Our model achieves similar success rates for pushing in simulation

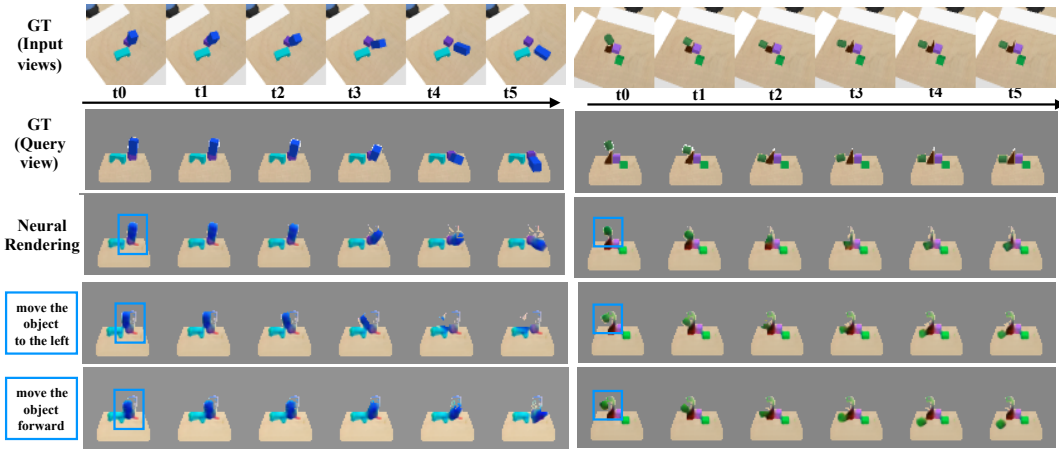

Figure 3: **Neurally rendered simulation videos of counterfactual experiments.** The first row shows the ground truth simulation video from the dataset. Only the first frame in this video is used as input to our model to produce the predicted simulations. The second row shows the ground truth simulation from a query view. The third row shows the future prediction from our model given the input image. The following rows show the simulation after manipulating an objects (in the blue box) according the instruction on the left most column.

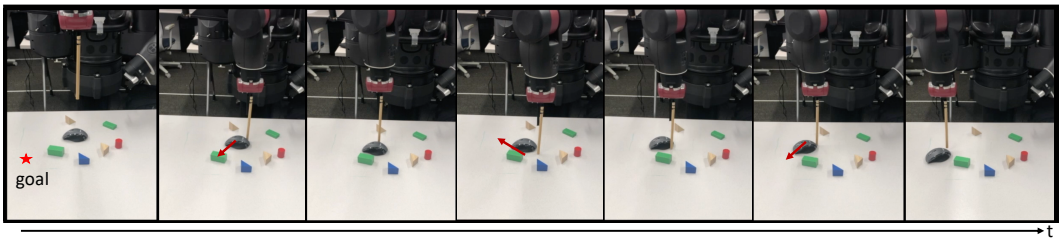

Figure 4: **Collision-free pushing on a real-world setup.** The task is to push a mouse to a target location without colliding into any obstacles. Our robot can successfully complete the task with 3 push attempts.

and in the real world. Since geometry information is shared by simulation and the real world by a large extent, and our model combines the viewpoint-invariant property of the geometry-aware representation and an object-factorized structure, it presents good sim-to-real transferrability. In Figure 4, we qualitatively show pushing objects along collision-free trajectories in complex scenes in the real-world setup. More examples are included in the supplementary video.

5 Conclusion

We have presented 3D-OES , dynamics models that predict 3D object motion in a 3D latent visual feature space inferred from 2.5D video streams. We empirically showed our model can generalize across camera viewpoints and varying number of objects better than existing 2D dynamics models or dynamics models over 3D object centroids. To the best of our knowledge, this is the first model that can predict 3D object dynamics directly from RGB-D videos and generalize across scene variations.

Our model currently has three main limitations: **(i)** It requires ground-truth 3D object locations and orientations at training time. Automatically inferring those with 3D tracking would permit our model to be trained in a self-supervised manner. **(ii)** It assumes rigid object interactions. Learning dynamics of soft bodies and fluids would require forecasting dense 3D motion fields, or considering sub-object (particle) graphs. **(iii)** It is deterministic. Handling stochasticity via stochastic models or objectives would permit to learn more complex and multimodal object motions.

Acknowledgements This paper is based upon work supported by Sony AI, DARPA Common Sense program, NSF Grant No. IIS-1849287, and an NSF CAREER award. Fish Tung is supported by Yahoo InMind fellowship and Siemens FutureMaker Fellowship.

References

[1] C. Gabbard. The role of mental simulation in embodied cognition. *Early Child Development and Care*, 183(5):643–650, 2013.

[2] J. Decety and D. H. Ingvar. Brain structures participating in mental simulation of motor behavior: a neuropsychological interpretation. *Acta psychologica*, 73 1:13–34, 1990.

[3] F. Ebert, C. Finn, A. X. Lee, and S. Levine. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017. URL http://arxiv.org/abs/1710.05268.

[4] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pages 2539–2547, 2015.

[5] L. Romaszko, C. K. Williams, P. Moreno, and P. Kohli. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 851–859, 2017.

[6] H.-Y. F. Tung, R. Cheng, and K. Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. *arXiv:1901.00003*, 2018.

[7] H. Izadinia, Q. Shan, and S. M. Seitz. IM2CAD. *CoRR*, abs/1608.05137, 2016.

[8] L. Romaszko, C. K. I. Williams, P. Moreno, and P. Kohli. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *ICCV Workshops*, Oct 2017.

[9] A. W. Harley, F. Li, S. K. Lakshmikanth, X. Zhou, H.-Y. F. Tung, and K. Fragkiadaki. Embodied view-contrastive 3d feature learning. *arXiv*, 2019.

[10] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019.

[11] Y. Ye, D. Gandhi, A. Gupta, and S. Tulsiani. Object-centric forward modeling for model predictive control. *the Conference on Robot Learning (CoRL)*, 2019.

[12] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv:1806.01242*, 2018.

[13] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu. Reasoning about physical interactions with object-oriented prediction and planning. *CoRR*, abs/1812.10972, 2018. URL http://arxiv.org/abs/1812.10972.

[14] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *CoRR*, abs/1612.00222, 2016.

[15] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems, 2018.

[16] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

[17] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. Yamins. Flexible neural representation for physics prediction. In *NIPS*, 2018.

[18] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks, 02 2020.

[19] Y. Li, T. Lin, K. Yi, D. Bear, D. L. Yamins, J. Wu, J. B. Tenenbaum, and A. Torralba. Visual grounding of learned physical models. In *International Conference on Machine Learning*, 2020.

[20] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 127–135. Curran Associates, Inc., 2015.

[21] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. A. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *CoRR*, abs/1903.11239, 2019.

[22] A. Ajay, J. Wu, N. Fazeli, M. Bauzá, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. *CoRR*, abs/1808.03246, 2018. URL http://arxiv.org/abs/1808.03246.

[23] D. Ha and J. Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018.

[24] C. Finn and S. Levine. Deep visual foresight for planning robot motion. *CoRR*, abs/1610.00696, 2016.

[25] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. URL http://arxiv.org/abs/1606.07419.

[26] L. Pinto, D. Gandhi, Y. Han, Y. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. *CoRR*, abs/1604.01360, 2016. URL http://arxiv.org/abs/1604.01360.

[27] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

[28] R. C. Miall and D. M. Wolpert. Forward models for physiological motor control. *Neural Netw.*, 9(8): 1265–1279, Nov. 1996. ISSN 0893-6080. doi:10.1016/S0893-6080(96)00035-4.

[29] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *CoRR*, abs/1612.00341, 2016. URL http://arxiv.org/abs/1612.00341.

[30] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015. URL http://arxiv.org/abs/1511.05440.

[31] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. *arXiv preprint arXiv:1507.08750*, 2015.

[32] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[33] F. Ebert, C. Finn, S. Dasari, A. Xie, A. X. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, abs/1812.00568, 2018. URL http://arxiv.org/abs/1812.00568.

[34] S. Chiappa, S. Racanière, D. Wierstra, and S. Mohamed. Recurrent environment simulators. *CoRR*, abs/1704.02254, 2017. URL http://arxiv.org/abs/1704.02254.

[35] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363, 2017. URL http://arxiv.org/abs/1705.05363.

[36] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. *CoRR*, abs/1511.07404, 2015.

[37] A. Byravan and D. Fox. SE3-Nets: Learning rigid body motion using deep neural networks. *CoRR*, abs/1606.02378, 2016.

[38] Z. Lai, E. Lu, and W. Xie. Mast: A memory-augmented self-supervised tracker, 2020.

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

[40] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn. Robonet: Large-scale multi-robot learning. *CoRR*, abs/1910.11215, 2019. URL http://arxiv.org/abs/1910.11215.

[41] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[42] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017.

[43] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgbSn09Ym.

[44] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems*, 2018.

[45] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017.

[46] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870.

[47] Y. Tassa, T. Erez, and W. D. Smart. Receding horizon differential dynamic programming. In *Advances in neural information processing systems*, pages 1465–1472, 2008.

[48] K. Fang, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation, 2019.

[49] K. Yu, M. Bauzá, N. Fazeli, and A. Rodriguez. More than a million ways to be pushed: A high-fidelity experimental data set of planar pushing. *CoRR*, abs/1604.04038, 2016.

[50] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[51] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[52] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell. Zero-shot visual imitation. In *ICLR*, 2018.

[53] Y. Ye, D. Gandhi, A. Gupta, and S. Tulsiani. Object-centric forward modeling for model predictive control, 2019.

[54] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL

http://arxiv.org/abs/1703.06907.

Appendix

A Additional experimental results

A.1 Neurally rendered physics simulations from multiple views

We show in Figure 5 rendered physics simulation videos using the proposed model. The latent 3D feature map of the proposed model is interpretable in the sense that we can render human-interpretable RGB images from the feature map using the learned neural image decoder. More importantly, we can render such simulation videos from any arbitrary view, and the videos captured from different views are consistent with each other.

A.2 Additional counterfactual experiments

In Figure 6, we show more results of conducting counterfactual experiments using the learned neural simulator. We can move objects to arbitrary position, and change their size by moving their features explicitly in the latent 3D feature space. Although the model has never been trained on this task, it can generate reasonable simulation results after such manipulations on the objects.

B Data collection details

Here we describe details of the data used in Section 4.

Pushing Our training data contains RGB-D video streams where the robot pushes objects which in turn can collide and push other objects on the table. We create scenes using 31 different 3D object meshes, including 11 objects from the MIT Push dataset [49] and 20 objects selected from four categories (*camera*, *mug*, *bowl* and *bed*) in the ShapeNet Dataset [50]. We split our dataset so that 24 objects are used during training. At test time, we evaluate the prediction error on the remaining 7 objects. At training time each scene contains at most two (potentially interacting) objects. At test time, we vary the number of objects from one up to five. We randomize the textures of the objects during training to improve transferability to the real world [54]. We consider a simulated Kuka robotic arm equipped with a single rod (as shown in Figure 3 of the main paper. The objects can move on a planar table surface of size $0.6m \times 0.6m$ when pushed by the arm, or by other objects. We collect training interaction trajectories by instantiating the gripper nearby a (known) 3D object segmentation mask. We sample random pushing action sequences with length of 5 timesteps, where each action is a horizontal displacement of the robot's end-effector ranging from $3cm$ to $6cm$, and each timestep is defined to be 200ms. We record objects displacement 1 sec after the push. We place cameras at 27 nominal different views including 9 different azimuth angels ranging from the left side of the agent to the right side of the agent combining with 3 different elevation angles from 20, 40, 60 degrees. All cameras are looking at the 0.1m above the center of the table, and are 1 meter away from the look-at point. At each timestep, all cameras are purturbed randomly around their nominal viewpoints, and we record all 27 views. At training time, our model consumes three randomly selected concurrent camera viewpoints as input. At test time, we use the 3D object detector to predict the 3D object segmentation mask, and our model is tested with either three or a single view as input, all randomly selected. All images are 128×128. There are 5000 pushing trajectories in the training data, and 200 pushing trajectories in the test data.

Falling We use the 3D meshes of the block objects introduced in [13], which includes cones, cylinders, rectangles, tetrahedrons, and traingles with a variety of shapes. We randomly select 1-3 objects and initialize their position by placing them on the table surface, and let one object falls freely from the air. One timestep is defined to be 40ms. All other settings are identical to the settings for pushing.

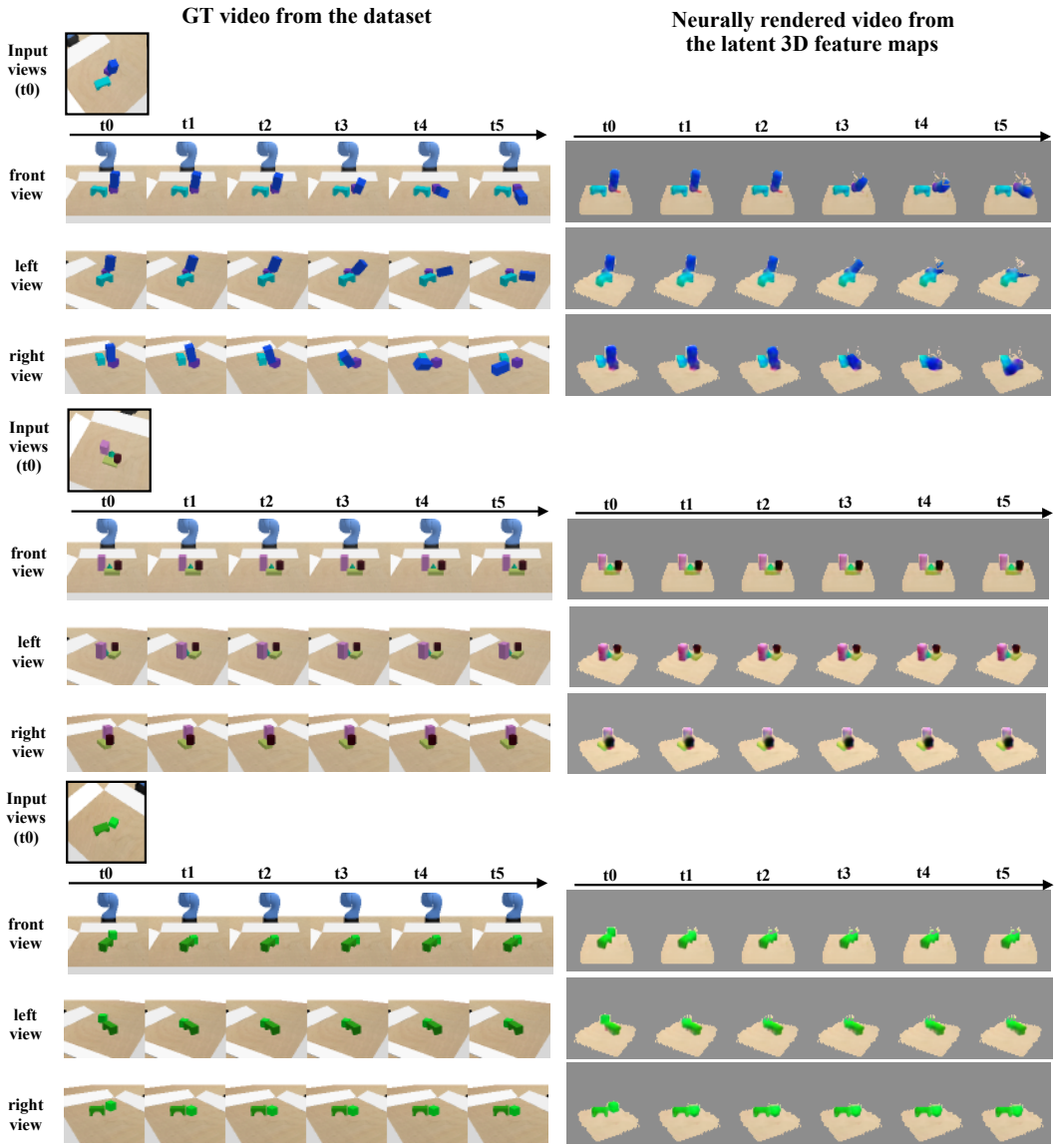

Figure 5: **Neurally rendered simulation videos from three different views** Left: groundtruth simulation videos from the dataset. The simulation is generated by the Bullet Physics Simulation. Right: neurally rendered simulation video from the proposed model. Our model forcasts the future latent feature by explicitly warping the latent 3D feature maps, and we pass these warped latent 3D feature maps through the learned 3D-to-2D image decoder to decode them into human interpretable images. We can render the images from any arbitrary views and the images are consistent across views.

C Experimental details

C.1 Implementation Details of the 3D Object Graph Neural Networks

We use ground-truth 3D bounding boxes for cropping the scene feature map $\mathbf{M}$ at training time, and 3D predicted boxes provided by our 3D detector at test time. The loss function is the summation of the L2 distance between the predicted and GT translation, and the L2 distance between the predicted and GT quaternions. The inputs to the graph networks are the cropped 3D feature maps of each objects with the size of 16×16. We first transforms the object-center 3D feature map into a feature vector with three 3D-conv layers followed by an average pooling layer and two FC layers of size

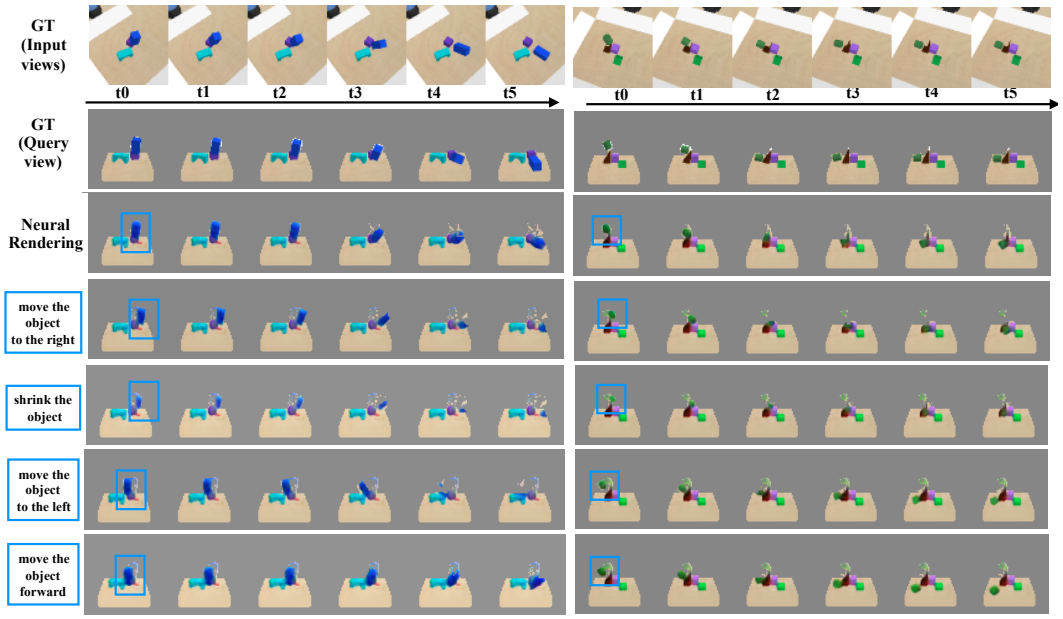

Figure 6: **Neurally rendered simulation videos of counterfactual experiments.** The first row shows the ground truth simulation video from the dataset. Only the first frame in this video is used as input to our model to produce the predicted simulations. The second row shows the ground truth simulation from a query view. Note that our model can render images from any arbitrary view. We choose this particular view for better visualization. The third row shows the future prediction from our model given the input image. The following rows show the simulation after manipulating an objects (in the blue box) according the instruction on the left most column.

32 with leaky-relu. The vectorized object features are then concatenated with the position and orientation of the objects as inputs to a standard graph network. In the graph network, both the node and edge encoders are 4-layer MLPs with layer size of 32 and leaky-relu activation. Our model is initialized with Xavier initialization and trained using the Adam optimizer for 90K steps. We train two separate models, one for pushing and one for falling.

For the 2D-to-3D image encoding using GRNNs [6], we follows the exact neural architecture as in [6], which takes as input RGB-D images and outputs 3D feature map $\mathbf{M}_t$ of size $64 \times 64 \times 64 \times 32$. Our detector also follows the architecture design of [6], which extends the 2D faster RCNN architecture to predict 3D bounding boxes from 3D features maps, as opposed to 2D boxes from 2D feature maps. The detector takes the output 3D feature map from the 2D-to-3D lifting as input to predict object bounding boxes. The detector consists of one down-sampling layer and three 3D residual blocks, each having 32 channels. We use 1 anchor box at each grid location in the 3D feature map with a size of 0.12 meters. The detector predicts an objectness score for each anchor box and selects boxes that exceeds a threshold. We set the threshold to be 0.9. We train the object detector with all the frames in the training data.

C.2 Implementation Details for Baselines

Here we describe the baselines discussed in Section 4 in detail.

1. *graph-XYZ*, a model that uses the 3D object centroid (X, Y, Z) as object state, and incorporate cross-object interactions for forecasting 3D translation using graph convolutions over a object graph, similar to [41, 42]. Since the canonical pose of an object is undefined, object orientation is not included in the object state. This model neglects object shape and appearance. The graph networks used in all baselines follow the exact design as the one we use in our model (4-layer MLPs for both the node and edge encoder). The only difference is that its inputs do not contain any object appearance features.

2. *graph-XYZ-image*, a model that uses the 3D object centroid (X, Y, Z) and object-centric 2D image feature embeddings for forecasting 3D translation. This baseline model extracts 2D CNN features from each image, concatenates the features with the camera viewpoint, and transforms the combined features into an object appearance feature vector. The feature vector is concatenated with the 3D object centroid and fed into a graph network (identical to the one used in *graph-XYZ*) to predict future object 3D translation. When taking multiple views as inputs, the model takes the average of the appearance feature vectors across views.

3. Visual Foresight (*VF*) [51], a model that uses the current frame and the action of the agent to predict future 2D frames by "moving pixels" based on predicted 2D pixel flow fields. It is based on the publicly available code of [51] that uses such frame predictive model to infer an action trajectory that brings an object pixel to the desired (2D) location in the image space.

4. *PlaNet*[10], a model that learns a scene-level embedding by predicting future frames and the reward given the current frame. *PlaNet* only deals with single-goal tasks and does not apply to our multi-goal pushing task. We extend it to our setting by appending the goal state to the observation. In practice, we augment the latent state vector produced from its state encoder's first fully connected layer with a randomly selected goal, and provide the model with reward computed correspondingly. The reward at each timestep is the computed as the negative of the distance-to-goal.

Note that both *VF* and *PlaNet* are self-supervised models that do not require ground-truth object states during training. However, we believe that since such supervision is readily accessible in simulation, we should leverage them to push the performance of the learned dynamics model. Self-supervised models are more favored when trained directly in the real world, where strong supervisions are not available, but as we showed in our experiments, our model trained solely in simulation can transfer reasonably well to the real world without any fine-tuning. As a result, we believe including the comparison with such self-supervised baselines is arguably fair and reasonable.

C.3 Details for Pushing with MPC

Here we described details of pushing with MPC discussed in section 4.3.

C.3.1 Pushing in simulation

Pushing without obstacle We test the performance of our model with MPC to push objects to desired locations. We run 50 experiments in the Bullet simulator. For each testing sample, we place either 1 or 2 objects in the $0.6m \times 0.6m$ workspace randomly, and sample a random goal for each object. The maximum distance of the goal to the initial position for each object is capped at $0.25m$. For our model and *graph-XYZ-image*, we use a single randomly sampled view. For *VF* and *PlaNet*, we use a fixed top-down view for both training and testing. We set the maximum number of steps for each action sequence to be 10, and evaluate 30 random action sequences before taking an action. We use planning horizon of 1 since greedy action selection suffices for this task. The results are reported in Table 3 in the main paper. Note that we also train and test variants of VF and PlaNet to take observations from varying camera viewpoints, together with camera pose information. However, they both fail completely on this task.

Collision-free pushing In order to test our model's multi-step prediction performance, we evaluate our model on pushing in scenes with randomly sampled obstacles, and the robot is required to push an object to desired goal without colliding into any obstacle. For quantitative evaluation, we randomly place an object of interest and a goal position in the planar workspace. One obstacle object is placed between them with a small perturbation, so that there exists no straight collision-free path to reach the goal. The distance from the object to its goal is uniformly sampled from the range $[0.24m, 0.40m]$. Similarly, we run 50 examples, and use only one randomly selected camera view as input to our model. We evaluate 60 randomly sampled action sequences with length of 25 steps, and use a planning horizon of 10 steps. We achieve a success rate of **0.68** for this task.

Since randomly placing *multiple* obstacles in the scene for quantitative evaluation while ensuring existence of collision-free path is non-trivial, we show qualitative planning results for such complex scenes in the supplementary video.

For both with- and without- obstacle pushing, it is considered a successful pushing sequence if all objects end up within 4cm (about half of the average object size) from the target positions on average.

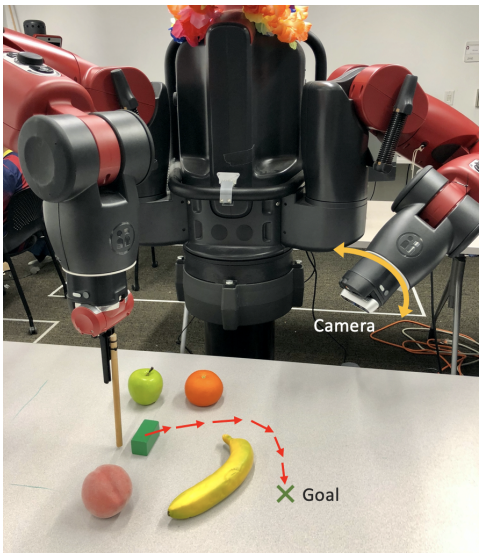

Figure 7: Real-world setup with Baxter

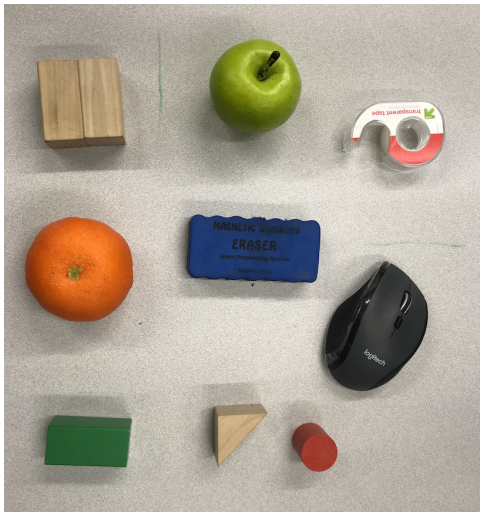

Figure 8: Objects for real-world experiments

C.3.2 Sim-to-real transfer for pushing in the real world

We use a Baxter robot equipped with a rod-shaped end-effector attached to its right hand, similar to the setting in the Bullet simulation. One Intel RealSense D435 RGB-D camera is attached to the robot's left hand, and we use only one view for our experiment, as shown in Figure 7. Please refer to the supplementary video for more qualitative results.

Due to reachability considerations, we down-scaled the size of the planar workspace by twice from the one in simulation, resulting a workspace of $0.3m \times 0.3m$. For a fair comparison, we also down-scaled with the same factor the object-to-goal distance, length of horizontal movement per action step, and size of the tolerance for determining success/failure. We pick 20 objects with size of 5 to $10cm$, which are commonly seen in a office setting, including fruits, wooden blocks, and stationery, and evaluate 5 pushing samples for each of them. Some of objects selected are shown in Figure 8.

For object detection in the real-world, we train our 3D detector using simulated data, and fine-tune it using a small set of real data (100 images capturing 25 distinct object configurations) collected using 4 cameras. The ground truth bounding-boxes and segmentation masks are obtained via background subtraction.