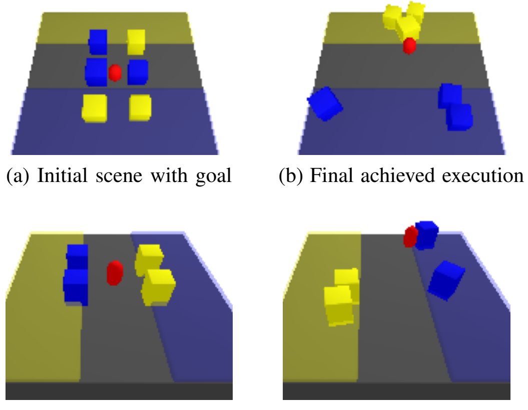
Learning Multi-Object Dynamics with Compositional Neural Radiance Fields

Danny Driess[1] Zhiao Huang[2] Yunzhu Li[3] Russ Tedrake[3] Marc Toussaint[1]
[1]Technical University of Berlin, Science of Intelligence
[2]University of California, San Diego
[3]Massachusetts Institute of Technology

Abstract*—We present a method to learn compositional predictive models from image observations based on implicit object encoders, Neural Radiance Fields (NeRFs), and graph neural networks. NeRFs have become a popular choice for representing scenes due to their strong 3D prior. However, most NeRF approaches are trained on a single scene, representing the whole scene with a global model, making generalization to novel scenes, containing different numbers of objects, challenging. Instead, we present a compositional, object-centric auto-encoder framework that maps multiple views of the scene to a *set* of latent vectors representing each object separately. The latent vectors parameterize individual NeRF models from which the scene can be reconstructed and rendered from novel viewpoints. We train a graph neural network dynamics model in the latent space to achieve compositionality for dynamics prediction. A key feature of our approach is that the learned 3D information of the scene through the NeRF model enables us to incorporate structural priors in learning the dynamics models, making long-term predictions more stable. For planning, we utilize RRTs in the learned latent space, where we can exploit our model and the implicit object encoder to make sampling the latent space informative and more efficient. In the experiments, we show that the model outperforms several baselines on a pushing task containing many objects. Video: https://dannydriess.github.io/compnerfdyn/


(a) Initial scene with goal (b) Final achieved execution

(c) Initial scene with goal (d) Final achieved execution

Fig. 1: Planning scenario with learned model. The goal is to move the blue and yellow boxes in their respective shaded areas. (a) and (c) are the initial observations, (b) and (d) show the achieved goal at the end of the planning/execution loop.

I. Introduction

Learning models from observations that predict the future state of a scene is a fundamental concept for enabling an agent to reason about necessary interventions with the environment to achieve a desired goal. A major challenge in learning such predictive models is that raw observations of our world such as image sequences are usually high-dimensional. Therefore, a common approach is to map the observation space into a lower-dimensional latent vector representation of the scene via an auto-encoder structure. Based on those latent vectors, a dynamics model can be learned that predicts the next latent state, conditioned on potential actions an agent takes. An intuition behind this approach is that if a latent vector is sufficient to reconstruct the observations of the scene, then it contains enough information about the objects in the scene to be able to learn a dynamics model on top of it as well.

While an auto-encoder structure combined with a latent dynamics model is a general approach that is applicable for a large variety of tasks, it raises multiple challenges.

On the one hand, scenes in our world are *composed* of multiple objects. Obtaining a dataset that contains combinatorics of different numbers of objects is often prohibitive. Therefore, a latent vector with a fixed size has difficulties in generalizing over different and changing numbers of objects in the scene than it has been trained on, both due to the limited capacity of a fixed-size vector and lack of diversity in the training distribution. This limits the generalization capabilities of such approaches.

On the other hand, image observations are 2D, but the 3D structure of our world is essential for many tasks to reason about the underlying physical processes governing the dynamics the model should predict. Dealing with occlusions, object permanence, and ambiguities in 2D views is challenging for 2D image representations.

Moreover, many forward predictive models in visual observation spaces suffer from instabilities in making long-term predictions, often manifested in blurry image predictions [15].

One way to address these issues is to incorporate inductive biases and structural priors in the model architectures. For instance, recently, Li et al. [38] propose to utilize learned Neural Radiance Fields (NeRFs) [42] as a decoder within an auto-encoder model for learning dynamics models in a latent space. NeRFs exhibit strong structural priors about the 3D world, leading to increased performance over 2D baselines. However, the approach of [38] represents the whole scene as a single latent vector, which we found insufficient for scenes composed of multiple and changing numbers of objects, both in terms of the representation and the dynamics model.

In the present work, we aim to overcome these challenges by incorporating inductive biases on the compositional nature of our world and its underlying 3D structure both in learning the latent representations themselves and the dynamics model. We propose a compositional, object-centric auto-encoder framework whose latent vectors are used to learn a compositional forward dynamics model in that learned latent space based on graph neural networks (GNN).

More specifically, we learn an implicit object encoder that maps image observations of the scene from multiple views to a set of latent vectors that each represent an object in the scene separately. These latent object encodings then parameterize individual NeRFs for each object. We apply compositional rendering techniques to synthesize images from multiple viewpoints, which forces the object-centric NeRF functions and the corresponding latent vectors to learn precise 3D configurations of the constituting objects.

This 3D inductive bias both in the encoder and the compositional NeRF decoder enables us to incorporate priors from the models' own predictions about objects interactions into learning the GNN dynamics model, making long-term dynamics predictions more stable.

In our evaluations, we focus on the rigid body case and consider a pushing scenario for scenes containing many objects. We show through comparisons that non-compositional auto-encoder frameworks and non-compositional dynamics models struggle with this task, while our framework generalizes well over different numbers of objects than during training and is capable of generating sharp and stable long-term predictions. Relative to more traditional multibody system identification [68], these models learn the geometry of unknown objects in addition to (implicitly) learning the inertial and frictional parameters.

Utilizing the structure of the model, we can generate informative samples for planning with a Rapidly Exploring Random Tree (RRT) in the latent space, enabling us to solve a challenging box sorting task from visual input.
To summarize, our main contributions are

- A compositional scene encoding framework that uses implicit object encoders and NeRF decoders for each object, forcing the view-invariant latent representation to learn about the 3D structure of the problem in a composable way.
- A factored dynamics model in the latent space as a graph neural network (GNN), exploiting the compositional nature of the scene representation and an adaptive adjacency matrix estimated from the model itself to yield stable long-term predictions.
- An RRT-based method to plan sequential object manipulations in the latent space, which exploits the 3D implicit object encodings and 3D predictions of the model.

We demonstrate the performance of the approach in terms of image reconstruction error, dynamics prediction error, and planning, generalizing strongly over different numbers of objects than during training. Refer to https://dannydriess.github.io/compnerfdyn/ for video results.

The rest of the paper is organized as follows. After reviewing related work in Sec. II, we briefly describe NeRFs for the purposes of this work in Sec. III. Then Sec. IV gives details about the encoder/decoder model and Sec. V is devoted to the graph neural network dynamic model in the latent space. Finally, in Sec. VI, we discuss how we can utilize RRTs in the latent space for planning.

II. Related Work

A. Learning Dynamics Models for Compositional Systems

Graph neural networks (GNNs) have shown great promise in introducing relational inductive bias [4] and modeling the dynamics of compositional systems consisting of interactions between multiple objects [3, 8, 35, 57, 19, 63]. Follow-up works extend the graph-structured neural dynamics models to large-scale dynamical systems represented using particles and meshes [43, 34, 36, 72, 58, 52]. People have also tried to model compositional systems directly from visual observation by first detecting objects using a perception module and then modeling their interactions for future prediction [81, 27, 75, 82, 55, 71, 86]. Our method differs from prior work by learning compositional scene representations grounded in 3D space directly from visual observations without any assumptions on object labels. Our novel combination of implicit object encoders and graph-based neural dynamics models reflects the structure of the underlying scene, which endows our agent with excellent generalization ability in handling complicated compositional dynamic environments.

B. NeRF for Compositional and Dynamic Scenes

The recent advances on neural implicit representations (or neural fields [78]) have demonstrated widespread success in applications like 3D shape and image synthesis [41, 51, 48, 64]. Notably, Neural Radiance Fields (NeRF) extend on previous advances and show impressive results on novel-view synthesis [42]. Initial NeRF approaches were trained on a single scene without generalization. Prior works have since proposed to modify NeRF and other neural scene representations to make them applicable to compositional environments [20, 65, 47, 84, 16], but there they typically focus on applications like object discovery and 3D scene editing without considering the dynamics of object interactions. People have also extended NeRF to enable view synthesis from a sparse set of views [83], as well as modeling dynamic scenes by learning implicitly represented flow fields or time-variant latent codes [54, 49, 13, 79, 50, 46, 70, 39, 77, 47, 33]. However, prior works for dynamic environments typically interpolate over a single time sequence and would not be able to handle scenes of different initial configurations or different input action sequences, limiting their use in downstream planning and control tasks. Li et al. [38] addressed this issue by combining NeRF with an auto-encoding framework and modeling the dynamics over the latent space. Yet they employed a one-dimensional latent vector as the scene representation, which we will show would fail at modeling compositional systems. In contrast, our method considers a graph-based scene representation to capture the structure of the underlying scene and achieves a significantly better generalization performance than [38].

C. Model-Based Planning in Robotic Manipulation

Model-based planning algorithms typically build a dynamics model of the environment and then use the model to plan the agent's behavior in order to minimize some task objectives. We can roughly categorize the methods by whether the model is constructed from first principles (i.e., physical rules) or learned from data (i.e., data-driven models). Physics-based models typically require complete information about the objects' geometry and the system's state [26, 85, 9, 69, 24], which limits their applicability in robotic manipulation tasks involving unknown object models and partially observable states. Data-driven methods, on the other hand, learn a dynamics model directly from the robot's interaction with the environment and have shown impressive results in manipulation tasks ranging from closed-loop planar pushing [5] to complicated dexterous manipulation [44]. Many of the data-driven planning frameworks learn dynamics models directly from visual observation based on representations defined at different levels of abstraction, such as pixel space [17, 14, 59, 15, 67, 10], 3D volumetric space [80], signed-distance fields [12, 66, 76], keypoint space [32, 40, 37], and low-dimensional latent space [74, 23, 22, 60]. Approaches commonly employ an image reconstruction loss [23, 22], an self-supervised time contrastive loss [62], or jointly train a forward and an inverse dynamics model [2] to make sure that the representation encodes meaningful information about the environment. Our method takes a step forward by learning graph-based latent representations from visual observations. The learned model accurately encodes the underlying 3D contents, allowing our learned model to achieve precise manipulation of compositional environments and generalize outside the training distribution, i.e. to scenes with more (and less) objects than during training.

D. Implicit Models in Robotics

Implicit models in robotics in a broader sense have been explored, for example, for grasping [6, 30, 73, 28]. Analytic signed distance functions are used in [25, 53, 12] for trajectory optimization. In [1], the 3D reconstructions of a scene via a learned NeRF model enable to plan collision free trajectories through the scene.

Our previous work proposed to learn dynamics models [12] and more general constraints [12, 21] for manipulation planning with objects being represented as signed-distance functions or other learned implicit functions. One assumption in [12] and [21] is that signed-distance values are available for supervision during training. The present work, in contrast, directly operates on RGB images without explicit 3D shape supervision. Therefore, our proposed approach can represent color and not only shape, which allows us to solve tasks where the color of objects is relevant.

III. Background on Neural Radiance Fields

This section summarizes Neural Radiance Fields (NeRFs) for the purposes of this work. For details, we refer to the original publication [42]. The general idea of NeRF is to learn a function f that predicts, at a 3D world coordinate $x \in \mathbb{R}^3$, the (emitted) RGB color value $c(x) \in \mathbb{R}^3$ and volume density $\sigma(x) \in \mathbb{R}_{\geq 0}$.

Based on the learned $(\sigma(\cdot), c(\cdot)) = f(\cdot)$, an image from an arbitrary view and camera configuration can be rendered by determining the color $C(r) \in \mathbb{R}^3$ of each pixel along its corresponding camera ray $r(\alpha) = r(0) + \alpha d$ through

$$C(r) = \int_{\alpha_n}^{\alpha_f} T_f(r, \alpha)\sigma(r(\alpha))c(r(\alpha))\,\mathrm{d}\alpha \tag{1}$$

with

$$T_f(r, \alpha) = \exp\left(-\int_{\alpha_n}^{\alpha} \sigma(r(s))\,\mathrm{d}s\right). \tag{2}$$

Here, $r(0) \in \mathbb{R}^3$ is the camera origin, $d \in \mathbb{R}^3$ the pixel dependent direction and $\alpha_n, \alpha_f \in \mathbb{R}$ the near and far bounds within an object is expected, respectively. The function f is a fully-connected neural network and the integrals in (1) and (2) are estimated by a simple quadrature rule, see [42], which make the whole rendering process differentiable and hence trainable with stochastic gradient descent. In most NeRF formulations, f takes a view direction as an additional input, which is beneficial to reconstruct reflections and other lighting effects. For the scenario we consider in this work, we found that incorporating view directions was not necessary and therefore omitted them. Including a view direction is a straightforward extension to what we present here.

IV. Encoding Scenes with Compositional Image-Conditioned Neural Radiance Fields

A. Overview

Assume that a scene is observed by RGB images $I^i \in \mathbb{R}^{3\times h_I \times w_I}$, $i = 1, \dots, V$ from V many camera views and that the scene contains m objects $j = 1, \dots, m$. We further assume to have access to the camera projection matrices $K^i \in \mathbb{R}^{3\times 4}$ for each view and binary masks $M_j^i \in \{0,1\}^{h_I \times w_I}$ of each object j in view i.

Given those posed images and masks, the goal is to learn an encoder Ω that fuses the information of the objects observed from the multiple views into a set of latent vectors $z_{1:m}$ by querying Ω on the individual masks $M_j^{1:V}$ such that

$$z_j = \Omega\left(I^{1:V}, K^{1:V}, M_j^{1:V}\right) \in \mathbb{R}^k \tag{3}$$

represents each object j separately in order for a learned decoder D to be able to reconstruct an image

$$I = D(z_{1:m}, K) \tag{4}$$

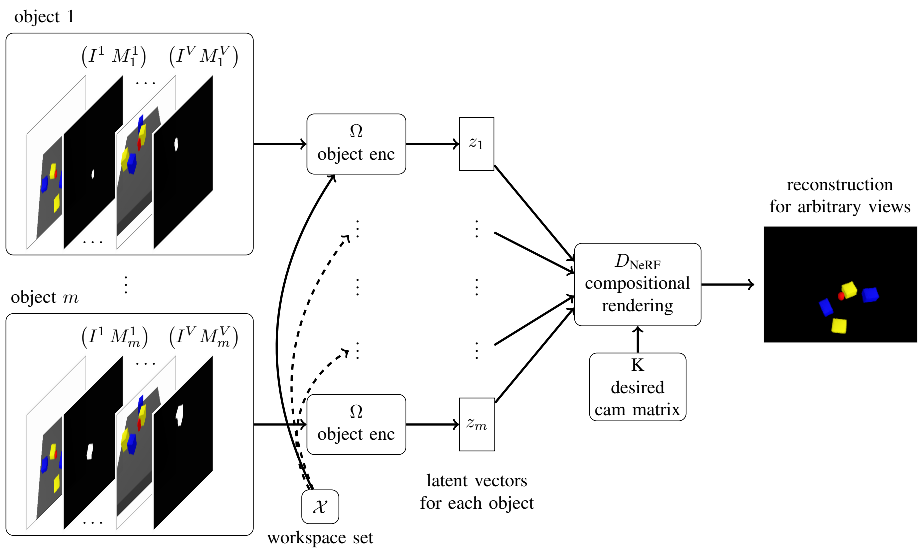


Fig. 2: Overview of the image-conditioned compositional NeRF autoencoder structure. Each object $j = 1, \ldots, m$ is observed in terms of RGB images I^i of the scene from multiple views $i = 1, \ldots, V$ with corresponding object masks M_j^i and camera matrices K^i. These inputs are encoded with the implicit object encoder Ω for each object separately, yielding latent vectors z_j for each object j (Sec. IV-B and IV-B). The weights of the object encoder Ω are shared between all objects. Further, the workspace set $\mathcal{X}$ where the implicit object feature functions $y_j(\cdot)$ within the encoder Ω are queried are the same for all objects. Refer to Sec. IV-C for the latter and Fig. 3, which shows the details of Ω. The set of latent vectors of all objects $z_{1:m}$ is rendered into an image for an arbitrary view specified by K via compositional NeRF rendering (Sec. IV-D).

for arbitrary views specified by the camera matrix K from the set of latent object representations $z_{1:m}$. See Fig. 2 for a visualization of this autoencoder structure.

Both the object encoder and the decoder will be realized as implicit functions, which we will describe in more detail in the following.

B. Representing Objects as Image-Conditioned Implicit Functions

Instead of learning Ω defined in (3) as a direct mapping from images, camera matrices and masks to the latent vectors, we first encode each object in the scene as a feature-valued *function* over 3D space, conditioned on the image observations. This allows us to incorporate multiple views of the object naturally, as well as to exploit the known behavior under 3D affine transformations, as we will discuss later in Sec. IV-F and V-C.

All object encodings are based on the same feature encoder

$$E(I^i, K^i(x)) \in \mathbb{R}^{n_o} \tag{5}$$

that outputs an n_o-dimensional feature vector from the image I^i of view i at a 3D world coordinate $x \in \mathbb{R}^3$ (see Fig. 3b). This is realized by first projecting x into camera coordinates

$$K^i(x) = \big(u^i(x), v^i(x), d^i(x)\big)^T \in \mathbb{R}^3, \tag{6}$$

where $u^i(x), v^i(x)$ are pixel coordinates in the image plane and $d^i(x) \in \mathbb{R}$ is the depth of x from the camera origin. Therefore, E is a function of the camera coordinates only and hence not a function of absolute world coordinates directly. Using bilinear interpolation, the encoder $E(I^i, K^i(x))$ first determines the RGB values of I^i at $(u^i(x), v^i(x))$ which are then passed through a dense multi-layer perceptron neural network (MLP). Parallel to this, a dense MLP encoding of $K^i(x)$ is computed. The concatenated outputs of both MLPs define the encoding feature vector $E(I^i, K^i(x))$. Similar architectures of computing such pixel features from world coordinates have been proposed, e.g. in [83, 56, 21] for the *single* object case.

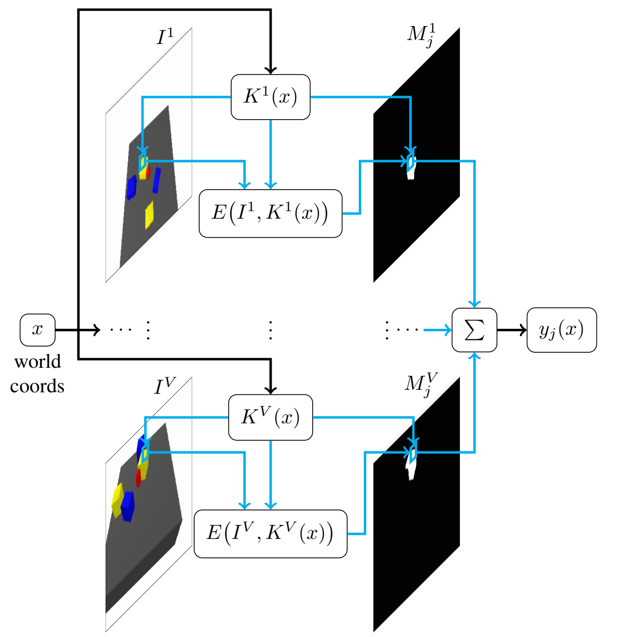
Intuitively, $E(I^i, K^i(x))$ is a feature vector computed from what can be seen of the world at x in the image I^i from viewpoint i, taking into account its location relative to the camera origin of the view i, which is important not only to enable the model to reason about the 3D geometry, but also to enable us to obtain a functional representation of a specific object j. Namely, we define the feature function for object j by summing over the individual views i,

$$y_j(x) = \frac{1}{p(x)} \sum_{i:\ K^i(x) \in M_j^i} E(I^i, K^i(x)) \in \mathbb{R}^{n_o} \tag{7}$$

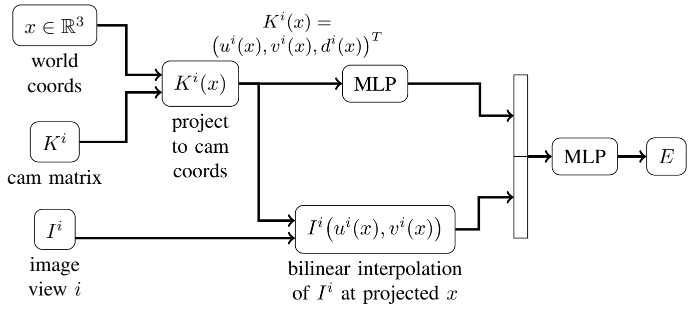
with

$$p(x) = \sum_{i:\ K^i(x) \in M_j^i} 1\ . \tag{8}$$

Importantly, for a specific x, this sum only takes those views i into account, where the object j can be seen, i.e., where the camera coordinates $K^i(x)$ of x are within the object's mask M_j^i. We additionally define that if $p(x) = 0$, then $y_j(x) = 0 \in \mathbb{R}^{n_o}$, meaning if an object is not observed from any view at x,

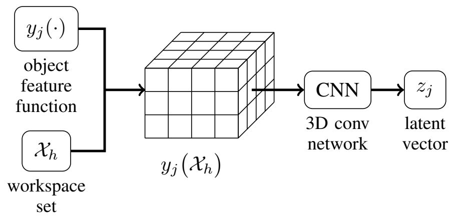


(a) Implicit object feature function y_j. The point $x \in \mathbb{R}^3$ in world coordinates where y_j is queried is projected into the camera coordinate systems of each view I^i where the (shared) feature encoder E (Fig. 3b) produces a feature of the object j at the projected x. The masks M_j^i determine whether the computed feature is relevant for aggregating the features from the different views into the final feature for object j. This aggregation is denoted with the sum symbol Σ, refer also to (7).



$x \in \mathbb{R}^3$
world coords
K^i
cam matrix
I^i
image view i
$K^i(x) = \big(u^i(x), v^i(x), d^i(x)\big)^T$
$K^i(x)$
project to cam coords
MLP
$I^i\big(u^i(x), v^i(x)\big)$
bilinear interpolation of I^i at projected x
MLP
E

(b) Feature encoder E for view i of image-conditioned implicit object encoder. E first projects the query point $x \in \mathbb{R}^3$ into the camera coordinate system via K^i. This projected coordinate is, on the one hand, encoded into a feature via an MLP. On the other hand, the pixel coordinates $\big(u^i(x), v^i(x)\big)$ are used to obtain the RGB value at the projected x via bilinear interpolation. The coordinate feature and the interpolated RGB value are concatinated and further processed with another MLP, leading to the final output $E\left(I^i, K^i(x)\right) \in \mathbb{R}^{n_o}$.



(c) Volumetric object encoder Φ (Sec. IV-C). The object feature function y_j of object j is evaluated on the discretized workspace set $\mathcal{X}_h$ which produces an object feature voxel grid $y_j\left(\mathcal{X}_h\right)$ that is turned into the latent vector $z_j \in \mathbb{R}^k$ via a 3D convolutional network.

Fig. 3: Visualization of the internals of the implicit object encoder Ω as a function of the RGB image observations $I^{1:V}$, their camera matrices $K^{1:V}$, and object masks $M_j^{1:V}$ of the object j for views $i = 1, \ldots, V$, as well as the workspace set $\mathcal{X}$. (a) visualizes how the implicit object feature function aggregates the observations from different views into the feature at the query point $x \in \mathbb{R}^3$ (world coordinates) by taking into account the object masks M_j^i in the different views. (b) shows the architecture of the feature encoder E (5) inside of y_j (7). (c) is the volumetric object encoder Φ that turns y_j into the latent vector $z_j \in \mathbb{R}^k$ (Sec. IV-C).

the corresponding feature vector is zero. An advantage of this formulation is that it naturally handles occlusions in different views and fuses the observations from different camera views consistently. Fig. 3a visualizes (7).

C. Volumetric Latent Object Encoder Network

Given the implicit object descriptor function $y_j(\cdot)$ of object j, we turn it into a latent vector $z_j \in \mathbb{R}^k$ representing object j with another network Φ which we call the volumetric latent object encoder. Formally, $z_j = \Phi(y_j)$ is a function of the object *function*. As discussed in [12], learning a function of a function can be realized with neural networks by evaluating y_j on a workspace set volume as follows.

1) Workspace Set: We assume that the interactions in the scene happen within a workspace set $\mathcal{X} \subset \mathbb{R}^3$ that is large enough to contain all objects (one can think of a bounding box for the whole workspace). This workspace set is discretized as the voxel grid $\mathcal{X}_h \in \mathbb{R}^{d \times h \times w}$. For a visualization of $\mathcal{X}$ refer to Fig. 6 (red shaded box).

2) Function of Functions with 3D Convolutions: The object descriptor functions are then evaluated on $\mathcal{X}_h$ which produces an object feature voxel grid that is processed with a 3D convolutional neural network leading to the latent vector $z_j \in \mathbb{R}^k$, i.e.

$$z_j = \Phi(y_j) = \text{CNN}(y_j(\mathcal{X}_h)). \tag{9}$$

Note that the same workspace set $\mathcal{X}_h$ is used for all objects.

In summary, the object encoder

$$z_{1:m} = \Omega\left(I^{1:V}, K^{1:V}, M_{1:m}^{1:V}, \mathcal{X}_h\right) \tag{10}$$

maps posed images from multiple views, object masks and the workspace set $\mathcal{X}_h$ to the latent vectors. The resulting z_j's contain not only the appearance information of the objects, but also their spatial configurations in the scene with respect to other objects.

Note that the model does not rely on absolute coordinates. The only relevant quantities are the relative coordinates between the cameras and the workspace set center, which is translational invariant. Further, the fact that the voxel grid $\mathcal{X}_h$ has a finite resolution does not imply that the resulting object representation is limited to the same resolution, since the model predicts *feature* values at the voxel locations, which can contain more information about the object.

D. Decoder as Compositional, Conditional NeRF Model

Compared to the standard NeRF formulation (Sec. III) where one single model is used to represent the whole scene, we associate separate NeRFs with each object, meaning that the NeRF for object j

$$(\sigma_j(x), c_j(x)) = f_j(x) = f(x, z_j) \tag{11}$$

is conditioned on z_j for $j = 1, \dots, m$. σ_j is the density and c_j the color prediction for object j, respectively. To turn those $f_{1:m}$ back into a global NeRF model that can be rendered to an image, we sum the individual predicted object densities

$$\sigma(x) = \sum_{j=1}^{m} \sigma_j(x) \tag{12}$$

and obtain the colors as their density weighted combination

$$c(x) = \frac{1}{\sigma(x)} \sum_{j=1}^{m} \sigma_j(x) c_j(x). \tag{13}$$

These composition formulas have been proposed multiple times in the literature, e.g. [45, 65]. This composition forces the individual NeRFs to learn the 3D configuration of each object individually and therefore ensures that each f_j only predicts the object where it is located in the 3D space.

To summarize, the compositional NeRF-decoder D_{NeRF} takes the set of latent vectors $z_{1:m}$ for objects $j = 1, \dots, m$ and the camera matrix K for a desired view as input to render

$$I = D_{\text{NeRF}}(z_{1:m}, K). \tag{14}$$

Since we only represent the objects and not the background as NeRFs, rendering the composed NeRF will yield an image with the background subtracted.

In the experiments, we investigate the importance of the decoder being both compositional and a NeRF.

E. Training

The auto-encoder framework is trained end-to-end on an L$_2$ image reconstruction loss. Since, as mentioned, solely the objects are represented as NeRFs and not the background, we compute the union of the masks of the individual objects

$$M^i_{\text{tot}} = \bigvee_{j=1}^{m} M^i_j \tag{15}$$

and define the target image in a view as $I^i \circ M^i_{\text{tot}}$ with $\circ$ denoting the element-wise product.

A known issue of NeRF is its computational efficiency [65], since for every pixel all f_j's have to be queried on many points along the camera ray. We make two simple, but important, improvements to reduce the computational demand.

First, the near and far bounds α_n, α_f are determined individually for each camera ray such that only those points along the rays that are within the workspace set $\mathcal{X}$ are considered. This is a reasonable assumption since we assumed that the objects are in the workspace set in the first place. That way, the computational efficiency is already greatly increased by reducing the number of points where functions f_j's have to be queried.

Moreover, as the scenes we consider in our experiments (see for example Fig. 6) are composed of multiple smaller objects, when masking out the background, the majority of pixels in each view is black and therefore does not contain information about the scene, although the model is evaluated on those areas. To further decrease the number of points where the NeRFs have to be queried, we only consider those rays for a view that pass through the mask of at least one object in that view. It turned out, however, that training only on rays that go through M_{tot} leads to blurry reconstructions, since there is no loss indicating that the objects should end outside of the masks. In order to resolve this, we enlarge the combined mask M^i_{tot} of a view with a convolution operation by a few pixels. We denote this enlarged masked by $\hat{M}^i_{\text{tot}}$. Together, these techniques ensures that the model learns sharp object boundaries, while significantly reducing the number of considered rays and required NeRF evaluations. See Fig. 4 for a visualization of this procedure.

These considerations lead to the following training objective of D_{NeRF} and Ω for a view i

$$\mathcal{L}^i = \sum_{(u,v) \in \hat{M}^i_{\text{tot}}} \left\| \left(I^i \circ M^i_{\text{tot}}\right)_{uv} - D_{\text{NeRF}}\left(\Omega\left(I^{1:V}, K^{1:V}, M^{1:V}_{1:m}, \mathcal{X}_h\right), K^i\right)_{uv} \right\|_2^2. \tag{16}$$

During training, we randomly sample a view from the dataset for each mini-batch and update the parameters of D_{NeRF} and Ω using the ADAM optimizer [31].

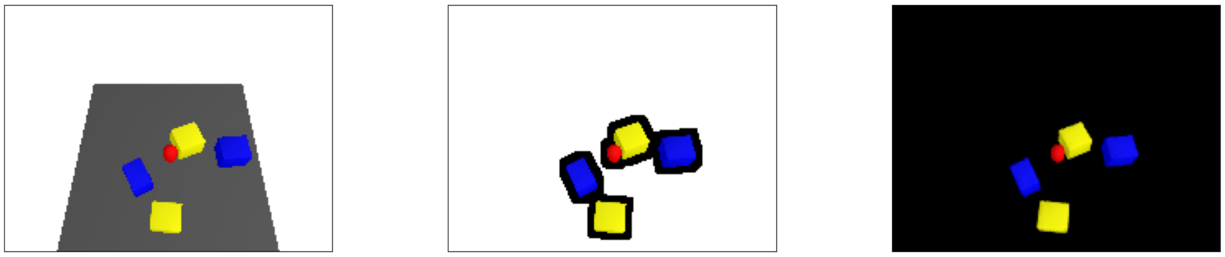
Another side effect of training on the enlarged masks $\hat{M}^i_{\text{tot}}$ only is that it improved the training stability and reconstruction qualities of the model. Indeed, when we trained the model on the whole image, depending on the weight initialization of the network, the model sometimes very quickly converged to a state where it only predicted a black image, since the majority of pixels are actually black and hence a low loss could be achieved. Training on the enlarged masks prevents this reliably.

F. Novel Scene Generation and Rigid Transformations

The compositional formulation of our model makes it trivial to add and remove objects from the scene. Furthermore, since the proposed object representation y_j is a function of a 3D coordinate, we can rigidly transform objects in the workspace by applying a rigid transformation [12]. Let $R(q) \in \mathbb{R}^{3\times3}$ and $s(q) \in \mathbb{R}^3$ be a rotation matrix and translation vector as a function of $q \in \mathbb{R}^7$ (translation + quaternion), respectively. Then,

$$y_j\left(R(q)^T(\ \cdot\ - s(q))\right) \tag{17}$$

is the object feature function transformed by q. Consequently, evaluating the transformed y_j on $\mathcal{X}_h$ with Φ from (9) leads to a



(a) Scene observation I^i

(b) Training target $I^i \circ \hat{M}^i_{\text{tot}}$ with enlarged mask. White area: no rays are considered

(c) Prediction by learned model rendered everywhere and not only at the masks

Fig. 4: Visualization of the training target for an example scene in the training dataset. Same scene as in Fig. 6.

new latent vector that represents the object j being transformed by q, which we denote with

$$\mathcal{T}(q)[z_j] = \Omega\left(I^{1:V}, K^{1:V}, M_j^{1:V}, R(q)^T\left(\mathcal{X}_h - s(q)\right)\right). \quad (18)$$

Please note the slight abuse of notation here, the term $R(q)^T\left(\mathcal{X}_h - s(q)\right)$ has to be understood elementwise for each entry in $\mathcal{X}_h$.

Composing scenes via rigid transformations applied to the input of the individual NeRFs f has been considered before, e.g. in [45]. However, transforming a NeRF by applying the rigid transformation to its x input only leads to changes in the rendered visual space, i.e. it has, in particular, no influence on the latent vectors of the objects. Since we want the latent vectors to represent not only the appearance of an individual object, but the geometric information of the object within the scene which is crucial for our downstream dynamics prediction task, just transforming the NeRF models is not sufficient.

V. Latent Dynamics Model with Graph Neural Networks

Due to the compositional nature of the scenarios we consider, we require a dynamics model that maintains the capabilities of our auto-encoder to generalize over changing numbers of objects, for which graph neural networks (GNNs) are a natural choice.

A. Propagation Network

The general idea behind learning dynamics models with GNNs is to associate each object in the scene with a node in a graph, which, in our case, means that each node in the graph is a latent vector z_j. Edges between the nodes indicate if objects interact, e.g. by exchanging forces due to contact. As argued in [35], applying a simple GNN to the problem of dynamics prediction is problematic, since interactions caused at one node can influence not only the neighboring nodes, but higher-order neighbors. For example, if three objects touch, the effects of applying a force at the first object have to propagate. The scenarios we consider in the experiments contain multiple objects such that more than two objects can interact in one time step. To take this into account, we use a message passing architecture inspired by [35].

Let z_i and z_j be the latent vectors of objects i and j. An edge encoder network F_e determines a feature

$$e_{ij} = F_e(z_i, z_j) \in \mathbb{R}^{n_e} \quad (19)$$

describing the interaction between the objects i, j. An adjacency matrix $A \in \{0,1\}^{m \times m}$ has entry $A_{ij} = 1$ if object i is influenced by object j. Assume the state of all latent vectors $z^t_{1:m}$ at time t is known. The node propagator network F_z recursively is queried L many times to propagate the state $z^t_{1:j}$ to the next time step $t+1$ as follows:

$$l = 1, \dots, L \ : \quad {}^{l}z_i^{t+1} = F_z\Big(z_i^t, \sum_{j \,:\, {}^{l-1}A^t_{ij}=1} {}^{l-1}e^t_{ij}\Big) \quad (20)$$

with ${}^{0}z_i^{t+1} = z_i^t$, ${}^{l}e^t_{ij} = F_e\left({}^{l}z_i^{t+1}, {}^{l}z_j^{t+1}\right)$ for $l = 0, \dots, L-1$ and the final new predicted state $z_i^{t+1} = {}^{L}z_i^{t+1}$.

B. Adjacency Matrix from Learned Model

The adjacency matrix A in the GNN dynamics model (20) plays an important role in indicating which objects interact. While a dense adjacency matrix, i.e. a graph where each node is connected to every other node implying that each object interacts with all other objects in the scene, would in principle work as the network could figure out from the latent representations itself which objects interact, we found that the long-horizon prediction performance is greatly increased if A is more selective in reflecting which objects actually interact (refer to the experiments in Sec. VII-D). This is especially relevant for compositional scenes as considered in this work where there are many objects, but which often do not interact with each other in every timestep.

A central question is how the adjacency matrix can be obtained from the observations of the scene without manually specifying it. Due to our model having strong 3D priors, we can exploit the density prediction σ_j as defined in (11) for each object to determine the adjacency matrix from the models' own predictions during training and planning. In order to do so, for a threshold $\kappa \geq 0$, the collision integral

$$S_{ij} = \int_{\mathcal{X}} [\sigma(x, z_i) > \kappa][\sigma(x, z_j) > \kappa] \, \mathrm{d}x \quad (21)$$

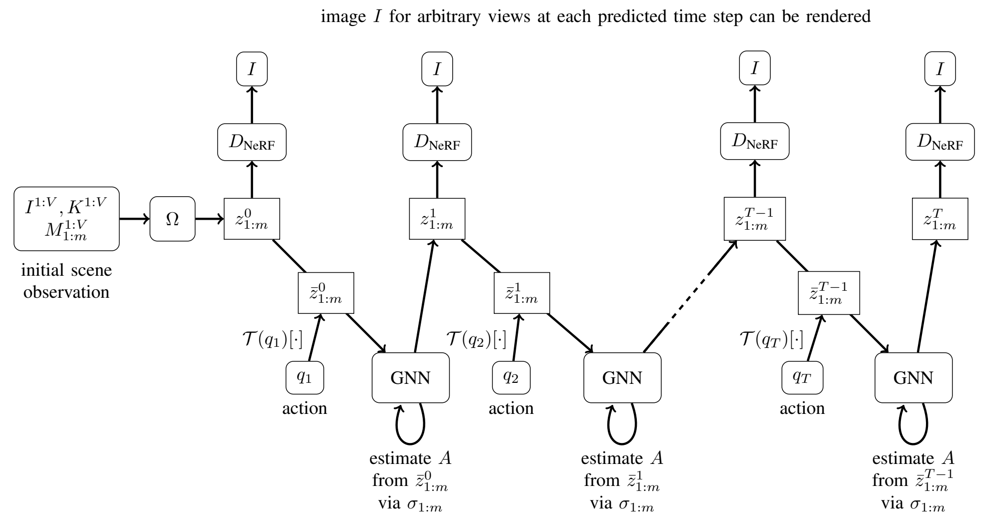


image I for arbitrary views at each predicted time step can be rendered
I
D_{NeRF}
$I^{1:V}, K^{1:V}$ $M^{1:V}_{1:m}$
initial scene observation
Ω
$z^0_{1:m}$
$\bar{z}^0_{1:m}$
$\mathcal{T}(q_1)[\cdot]$
q_1
action
GNN
estimate A from $\bar{z}^0_{1:m}$ via $\sigma_{1:m}$
$z^1_{1:m}$
$\bar{z}^1_{1:m}$
$\mathcal{T}(q_2)[\cdot]$
q_2
action
estimate A from $\bar{z}^1_{1:m}$ via $\sigma_{1:m}$
$z^{T-1}_{1:m}$
$\bar{z}^{T-1}_{1:m}$
$\mathcal{T}(q_T)[\cdot]$
q_T
action
estimate A from $\bar{z}^{T-1}_{1:m}$ via $\sigma_{1:m}$
$z^T_{1:m}$

Fig. 5: Visualization of the forward prediction Algorithm 1. After the initial scene observation is encoded into the latent vectors $z^0_{1:m}$, the GNN dynamic model is used to forward predict the evolution of the latent vectors in time for actions q_t from the initial observation only. These actions rigidly transform the articulated objects (in the experiments the pusher), leading to $\bar{z}^t_{1:m}$ which is the input to the GNN to produce $z^{t+1}_{1:m}$. The adjacency matrix for the GNN dynamics model is estimated from the density prediction $\sigma(\cdot, z_j)$ for all objects from the predicted z_j at each time step, see Sec. V-B. The predicted latent vectors $z^t_{1:m}$ at time t can be used to render images from arbitrary views or to reconstruct the scene.

over the density predictions of the learned NeRF model for objects i and j indicates if the two objects overlap or not. A similar integral as in (21) has been proposed in [12] to estimate collisions from signed-distance functions. Based on this integral, we define the entries of the adjacency matrix between objects i and j as

$$A_{ij} = \begin{cases} 1 & S_{ij} > 0 \\ 0 & \text{else} \end{cases}, \tag{22}$$

which implies that only those objects that are or are close to being in contact potentially interact. Estimating A this way takes the actual geometry of the objects in the scene into account. In relation to the node propagation network (20), this means that the adjacency matrix at step l of the propagation becomes a function of the node encodings itself, i.e.

$${}^{l}A^t_{ij} = {}^{l}A^t_{ij}\left({}^{l}z_i{}^{t+1}, {}^{l}z_j{}^{t+1}\right). \tag{23}$$

For training the GNN, however, changing the adjacency matrix during prediction is not differentiable. Therefore, we compute the adjacency matrices from the model such that they are constant within one time-step as follows. We first compute an occupancy grid

$$S_j = [\sigma(\mathcal{X}_h, z_j) > \kappa] \in \{0,1\}^{d\times h\times w} \tag{24}$$

for each object j over the discretized workspace set $\mathcal{X}_h$ and then apply a 3D convolution operation on S_j with a kernel consisting of only ones to expand the occupancy grid. The now constant within one time-step t entries ${}^{l}A^t_{ij} = A^t_{ij}$ for all $l = 0\ldots, L-1$ are then determined by checking if there is a voxel cell where both enlarged S_i and S_j have value one. The size of the convolution kernel is chosen large enough such that the adjacency matrix is not going to change within one timestep. This allows for a trade-off between sufficient sparsity of A while ensuring that all objects that potentially interact have corresponding entries in A.

In the experiments in Sec. VII-D, we investigate the influence on the prediction performance for multiple different ways of predicting/using the adjacency matrix.

C. Actions

So far, the way we have formulated the graph neural network dynamics model in Sec. V-A does not contain a notion of actions. Instead, we interpret an action as a modification to a node in the graph and train the GNN to predict the state of the nodes at the next time step as a result to this modification. This allows us to not explicitly distinguish between controlled and uncontrolled/passive objects.

In order to realize modifications to a node and hence to incorporate actions in the first place, we exploit the fact that our object encoder is an implicit function of 3D world coordinates. Assume that the object j is articulated by a known rigid transformation $q \in \mathbb{R}^7$, which is the action. As described in Sec. IV-F, via (18) we can transform the object's latent vector z^t_j into the transformed $\bar{z}^t_j = z^{t+1}_j = \mathcal{T}(q)[z_j]$, which is kept constant during the propagation step of (20), i.e. controlled nodes are excluded from the dynamics prediction, as their evolution is known through $\bar{z}^t_j$.

D. Quasi-Static Dynamics

If we assume quasi-static dynamics, meaning that the next system state only depends on the current latent state $z_{1:m}$ without history and immediate actions (refer to the discussion

Algorithm 1 Forward Prediction Model

```
1:  Input: Initial observation of the scene in terms of $I^{1:V}$, $K^{1:V}$, $M^{1:V}_{1:m}$, action sequence $q_{1:T}$, index $a$ of the articulated object
2:  $z^1_{1:m} = \Omega\left(I^{1:V}, K^{1:V}, M^{1:V}_{1:m}\right)$
3:  for all $t = 1, \dots, T$ do
4:      ${}^{0}z_{1:m}{}^{t+1} = z^t_{1:m}$
5:      ${}^{0}z_{a}{}^{t+1} = \bar{z}^t_a = \mathcal{T}(q_t)[z^t_a]$
6:      for all $l = 1, \dots, L$ do
7:          $\forall_{i,j,\ i\neq a} :\ {}^{l}A^t_{ij} = {}^{l}A^t_{ij}\left({}^{l}z_i{}^{t+1}, {}^{l}z_j{}^{t+1}\right)$
8:          $\forall_{i,j\ i\neq a} :\ {}^{l}e^t_{ij} = F_e\left({}^{l}z_i{}^{t+1}, {}^{l}z_j{}^{t+1}\right)$
9:          $\forall_{i\neq a}\ {}^{l}z_i{}^{t+1} = F_z\left(z^t_i, \sum_{j\ :\ {}^{l-1}A^t_{ij}=1} {}^{l-1}e^t_{ij}\right)$
10:     end for
11:     $z_i^{t+1} = {}^{L}z^t_i$
12:     $z_i^{t+1} = z^t_i \quad \text{if} \quad \forall_{j=1,\dots,m,\ j\neq i}\ :\ A_{ij} = 0$
13:     Use $z^{t+1}_{1:m}$, e.g. render it from arbitrary views
14: end for
```

in the last paragraph Sec. V-C about the notion of actions in this work), we can further increase the long-term stability by utilizing the adjacency matrices estimated by the learned model. When an object is not involved in any interactions with other objects, then, under quasi-static assumptions, it does not change between time steps, i.e. its latent vector stays constant, which means we can set

$$z_i^{t+1} = z_i^t \quad \text{if} \quad \forall_{j=1,\dots,m,\ j\neq i}\ :\ A_{ij} = 0. \tag{25}$$

The condition $\forall_{j\neq i}\ :\ A_{ij} = 0$ means that the node associated with z_i in the graph has no incoming edges. As we will show in the experiments (Sec. VII-D), this can greatly increase the stability for long-term open-loop model predictions as it will prevent drift in objects that do not take part in any interaction with other objects.

E. Training

We first train the compositional NeRF auto-encoder framework on training data, which gives us a dataset of trajectories of latent vectors. The GNN dynamics model is then trained on the one-step mean squared error between $z^{t+1}_{1:m}$ and $z^t_{1:m}$ of samples of such trajectories using the ADAM optimizer.

F. Forward Prediction Algorithm

Algorithm 1 summarizes the forward prediction procedure. The algorithm is visualized in Fig. 5. Starting from a single initial observation of the scene in terms of the images $I^{1:V}$ from V many views and the objects masks $M^{1:V}_{1:m}$ of the m many objects, Algorithm 1 predicts the latent vectors $z^t_{1:m}$ at times $t = 1, \dots, T$ for all objects in the scene given a desired action sequence $q_{1:T}$ of rigid transformations applied to object a. At every point t in time, the scene can be rendered from arbitrary view points from the predicted $z^t_{1:m}$. Note that the masks of the objects are required only for the initial observation, i.e. no mask prediction has to be performed as compared to [80].

In line 2, the initial object encodings from the scene observation are computed. Line 5 applies the action to the object with index a. Lines 6-10 then perform the prediction step of the GNN in the latent space using message passing. Crucially, in line 7, the adjacency matrix is estimated from the current predictions during message passing (Sec. V-B). Note that here the original collision integral (21), computed on the grid $\mathcal{X}_h$, can be used without enlarging the intermediate occupancy grids, since, if during the message passing step objects interact that previously did not, it will be captured, as A is estimated in every step of the message passing part. This leads to further increased prediction stability, as we will show in the experiments. Finally, in line 12, objects that have not interacted with other objects as predicted by the adjacency matrix are kept at their previous latent state (quasi-static assumption from Sec. V-D)

VI. Planning

In this section, we describe our planning and control algorithm to manipulate objects in scenes using the learned scene encoding and dynamics model to achieve a desired goal. The main part of the planning algorithm is an RRT in the latent space. Such latent space RRTs have been considered, for example, in [29]. One central question here is how one can sample in the latent space effectively, since a uniform random sample in the latent space not necessarily is a valid (and/or uniform) sample in the original space. In [29], they assume to have access to a set of valid latent vectors from which they can sample. In contrast, we can produce valid samples in the latent space directly by exploiting the properties of our model.

On a high level, our model iteratively perceives multi-view images of the scene, finds a plan using a Latent-Space RRT (LS-RRT) based on the forward predictions of the model over a long horizon, and then executes the found plan with Model-Predictive Control (MPC) [7] for a shorter horizon. We describe the algorithm here with the pushing scenario as considered in the experiments (see Sec. VII-A) in mind.

A. Planning

Algorithm 2 summarizes the LS-RRT algorithm. We grow a tree in the latent space, starting at the latent vector $z^0 = z^0_{1:m}$ that represents the current state of the environment, encoded by the implicit object encoder Ω from the current visual observation of the scene. In standard RRTs, a target is uniformly sampled in the configuration space to steer the growth of the tree towards a Voronoi bias. To introduce a particular goal-targeted sampling bias and as we do not have an inverse model or steering function, we modify the standard approach as follows:

In a latent space RRT, sampling uniformly in the latent space does neither guarantee that the samples are from the latent space manifold nor that they explore the original space. Therefore, we sample random targets $g \sim \mathcal{G}_{\text{sampler}}$ not in the latent space directly, but only in the space of center of mass configurations of all objects, which is of dimension $2m$ in the experiments (objects and pusher). In this way, we can design

Algorithm 2 Latent-Space RRT

```
1: Input: Initial latent vector $z^0$, action space $\{q\}$, metric $d$, sparse goal cost function $\mathcal{C}_g$.
2: Initialize tree $\mathbb{T}$ with $z^0$ as the root.
3: while time remains do
4:     Sample a target $g \sim \mathcal{G}_{\text{sampler}}$
5:     Find nearest $z_{\text{prop}} = \arg\min_{z \in \mathbb{T}} d(z, g)$
6:     Sample action $q$ randomly
7:     Execute action $q$ in learned forward model with Algorithm 1 to get new latent vector $z_{\text{next}}$
8:     Add $z_{\text{next}}$ into the tree $\mathbb{T}$, record action $q$ and its parent $z_{\text{prop}}$
9:     if goal $\mathcal{C}_g(z_{\text{next}})$ fulfilled then
10:        return the action sequence from $z^0$ to $z_{\text{next}}$
11:    end if
12: end while
```

a sampling distribution $\mathcal{G}_{\text{sampler}}$ biased to target configurations that have low costs, i.e., more objects within the goal region, or targets in which the articulated object (the pusher in the experiments) is close to one of the objects, inducing a bias for interaction. This sampling distribution and cost evaluation $\mathcal{C}_g$ is possible because we can apply rigid transformations to the objects through our object encoder being an implicit function, since, for a sampled random target, we have to move the objects to this target to check the cost on the transformed configuration. Further, the metric d to select the expanded node is the L_2-norm in the full configurations between g and the centers-of-masses computed from z. Using the predictions of the NeRF model, we can estimate (under homogeneous density assumption) the center of mass of an object with latent vector z_j as

$$x_j^{\text{com}}(z_j) = \frac{\int_{\mathcal{X}} x \cdot [\sigma(x, z_j) > \kappa] \, \mathrm{d}x}{\int_{\mathcal{X}} [\sigma(x, z_j) > \kappa] \, \mathrm{d}x}, \tag{26}$$

i.e. $d(z, g) = \|x_{1:m}^{\text{com}}(z) - g\|_2$. Note that the sampling distribution, cost function evaluation and metric calculation are done solely based on predictions of the model.

Finally, as we do not have an inverse dynamics model or other kinds of steering function, we expand the tree using a random action q, similar to plain control trees. However, our goal-targeted node selection ensures that the tree expands effectively.

B. Model-based Control

Although our model achieves impressive performance over a long horizon, the accumulated prediction errors may still lead to a failure when executing the plans open-loop. We therefore apply an MPC scheme, which in each cycle feeds the current visual observation into the model, samples and select actions that match the plan (in terms of the center-of-mass metric) within a short horizon predicted by the learned dynamics model. If there is a significant mismatch between the plan and current observation, the LS-RRT is used again to find a new long-term plan starting from the current observation.

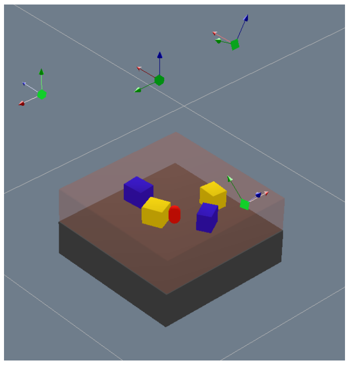


Fig. 6: Example scene. The red shaded box visualizes the workspace set volume $\mathcal{X}$. The green coordinate systems denote the camera origins and view directions. The red cylinder is the pusher that is actuated.

VII. Experiments

In the experiments, we focus on a pushing task in scenarios with multiple objects on a table, see, e.g. Fig. 6 or 4 for such scenes. For a quantitative analysis and comparison to multiple baselines, we investigate the forward prediction error of the model both in the image space (Fig.10) and, for baselines that use a compositional NeRF decoder, the error in predicting the center of mass of the objects (Fig. 9) over long-horizons. In all plots of Fig. 9 and Fig. 10, the blue curve corresponds to our proposed framework as summarized in Algorithm 1. Sec. VII-J presents planning and execution results for a challenging box sorting task.

Please refer to https://dannydriess.github.io/compnerfdyn/ for videos showing the reconstructions of the model, novel scene generation, forward predictions of the model and planning/execution results for the box sorting task.

A. Setup

We consider a rigid-body scenario with multiple objects on a table, see Fig. 6 for an example. In all cases, the red cylinder is the pusher that is articulated in order to push the other objects around.

This scenario is challenging due to multiple reasons. First, it is composed of many objects, which implies not only a broad scene distribution, but especially also that many objects can interact. The mechanics of such multi-body pushing is non-trivial, since, for instance, contact can be established and broken between the objects at multiple phases of the motion. Contact between multiple objects at the same time can occur. Furthermore, we do not assume that the red pusher starts in contact with an object. Hence, if a task implies that an object should be pushed, long-term predictions inherently have to be made in order to establish contact, before any object movement is registered.

The workspace is an area of 40 cm $\times$ 40 cm $\times$ 10 cm and we choose $\mathcal{X}_h \in \mathbb{R}^{10\times40\times40}$, i.e. a resolution of 1 cm. All scenes in the training data contain 4 box-shaped objects of randomly sampled sizes, positions and orientations

(5 dimensional parameter space for each of the 4 objects) and one cylinder-shaped object with randomly sampled position. To generate the training data, we randomly sample one of the 4 objects and then move the red pusher towards the center of this chosen object (with Gaussian noise added to the direction vector in each time step) until either the pusher leaves the workspace, in which case a new target object is chosen, or an object is pushed outside the workspace, in which case the data collection for this scene is terminated and a new scene is sampled. In total, the training dataset contains 5752 scenes with an average sequence length of 17. We generate 3 test datasets for evaluating the reconstruction and prediction performance which contain 2, 4, and 8 objects, respectively, plus the pusher. There are 312 scenes for each test dataset, generated with a different random seed than the training data. As visualized in Fig. 6 by the green coordinate systems, we choose 4 camera views for each scene.

B. Network Architectures

The dimension of the latent vectors z_j is $k = 64$ for each object. All hidden activation functions are ReLUs.

The MLP that encodes the projected coordinate (see Fig.3b) of the implicit object feature encoder E has one layer with output dimension 32. The other MLP in E has 2 hidden layers with 128 units each and an output dimension of $n_o = 64$.

The volumetric feature encoder Φ (Sec. IV-C and Fig. 3c) consists of three 3D convolutional layers with kernel size 3 and channel size 128, each. Layers 2 and 3 have strides of 2. After the convolutional layers, the output is flattened and processed with 3 dense layers with 300 hidden units each.

The NeRF network f first lifts the 3D input to 64 dimensions with an MLP, where it is concatinated with the latent vector z. This is followed by 3 hidden layers with 300 units each. For the density output σ, we use a softplus activation and a sigmoid for the color outputs c.

Both the edge encoder F_e and the node propagator network F_z have 3 hidden layers with 256 units each.

C. Reconstruction/Prediction Performance and Generalization to Different Numbers of Objects

Fig. 8a shows the predictions of the model forward unrolled in time for an action sequence of the red pusher, i.e. applying Algorithm 1 to an initial scene observation. Despite the movements in this scene leading to multiple object interactions, even after 38 time steps, the rendered predictions from the model are still sharp and reflect the underlying dynamics. By utilizing the estimated adjacency matrix, there is little drift in the objects, leading to long-term prediction stability.

Due to its compositional nature, our model generalizes to scenes that contain more or less objects than in the training set. For example, in Fig. 7, eight objects plus the pusher are observed and reconstructed with high quality, although during training the model has seen only and exactly 4 objects. This holds true not only for the observed views (Fig. 7a), but also for novel ones (Fig. 7b).

Our proposed method outperforms all baselines in the reconstruction and prediction error, see following sections for a more detailed, quantitative discussion and Figs. 9, 10, and 11.

D. Importance of Estimating the Adjacency Matrix

In Sec. V-B, we have proposed how the adjacency matrix of the GNN can be estimated from the density predictions of the learned NeRFs and that under quasi-static assumptions this estimated adjacency matrix can further be exploited to increase the long-term stability of the predictions, cf. Sec. V-D. Here we investigate the consequences of utilizing the adjacency matrix this way by comparing the full Algorithm 1 to:

1) Not exploiting quasi-static assumption: In this case, line 12 of Algorithm 1 is not used, i.e. the latent vectors of all objects, even when they do not interact with other objects as estimated through the model, are updated using the model forward predictions.

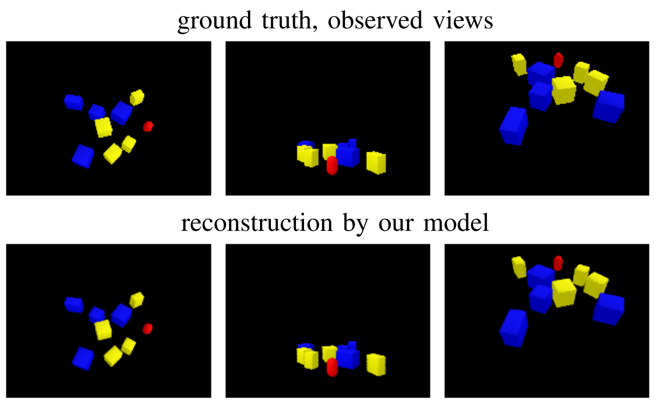
2) Adjacency matrix estimation not during message passing: Here, we estimate the adjacency matrix only at the beginning of the message passing step, i.e. before line 6 in Algorithm 1. In order to ensure that it can still capture all object interactions that might occur during the message passing step, we enlarge the determined occupancy grids exactly the same way as for training, see the discussion in Sec.V-B. The effects of this are that objects that are close to each other but do not interact still have entries in A indicating that they interact, which means slight errors in the predictions accumulate and lead to drift, although the object would not move in reality.

3) Dense adjacency matrix: We further consider a dense adjacency matrix, i.e. where the network has to figure out from the latent vectors themselves if objects interact. Preventing drift in this case is considerably harder.

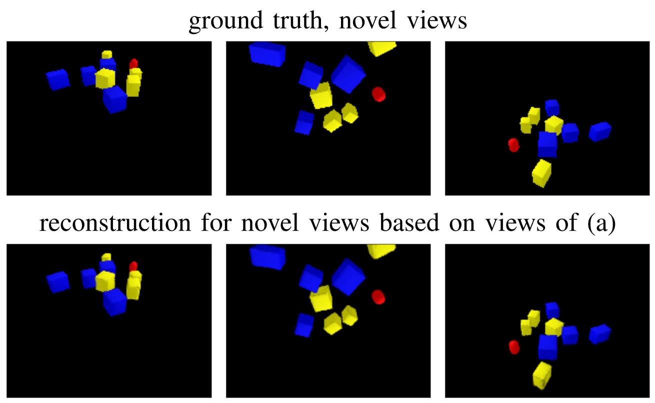
In all these comparisons, the rest of the method remains the same, i.e. same object encoder, same GNN, same compositional NeRF decoder.

In Fig. 9 one can see the mean error of the model predicting the center of mass, computed from its density predictions of the NeRFs for each object according to (26), over the number of steps predicted into the future on the test dataset for different numbers of objects in the scene.

As one can see in Fig. 9a, for the two object case, the choices of how the adjacency matrix is used, as long as it is not a dense one, are not significant. For the 4 (Fig. 9b) and 8 (Fig. 9c) object case, however, our proposed utilization of the adjacency matrix, i.e. estimating it during propagation steps and using it to exploit the quasi-static assumption, leads to a significant increase in performance. This can be explained by the fact that utilizing the adjacency matrix as we propose leads to significantly less drift. Especially with the dense adjacency matrix, the predictions are very unstable for all, the 2, 4, and 8 object case. Fig. 8b shows this qualitatively. In the 8 object case, the predictions with the dense A are basically useless

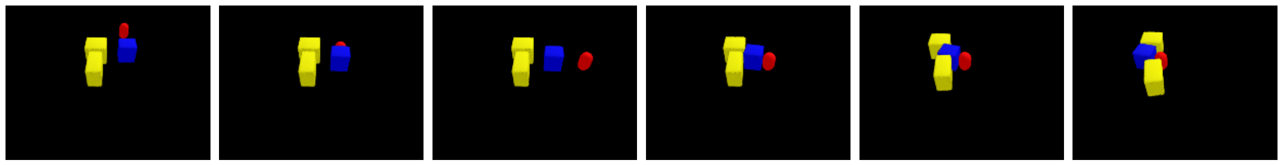


(a) Reconstructions on observed views

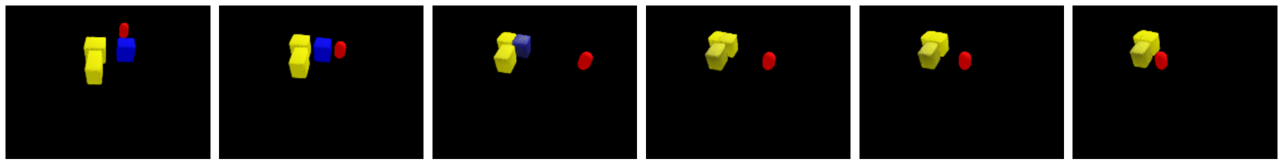

ground truth, novel views
reconstruction for novel views based on views of (a)

(b) Novel view synthesis from latent vectors computed from the views in (a)

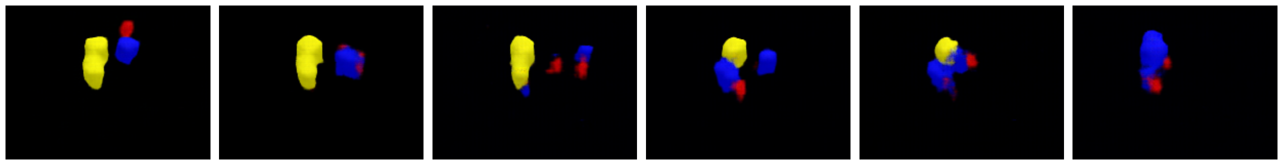
Fig. 7: Generalization to more objects (8 objects plus 1 pusher) than during training. The scenes in the training dataset contain exactly 4 objects and one pusher. The top row in (a) corresponds to the observed scene from different views; the bottom row is the reconstruction by our method. The bottom row in (b) is the reconstruction of novel views (top row ground truth) with the latent vectors computed from the views in the top row of (a).



(a) Predictions with our method

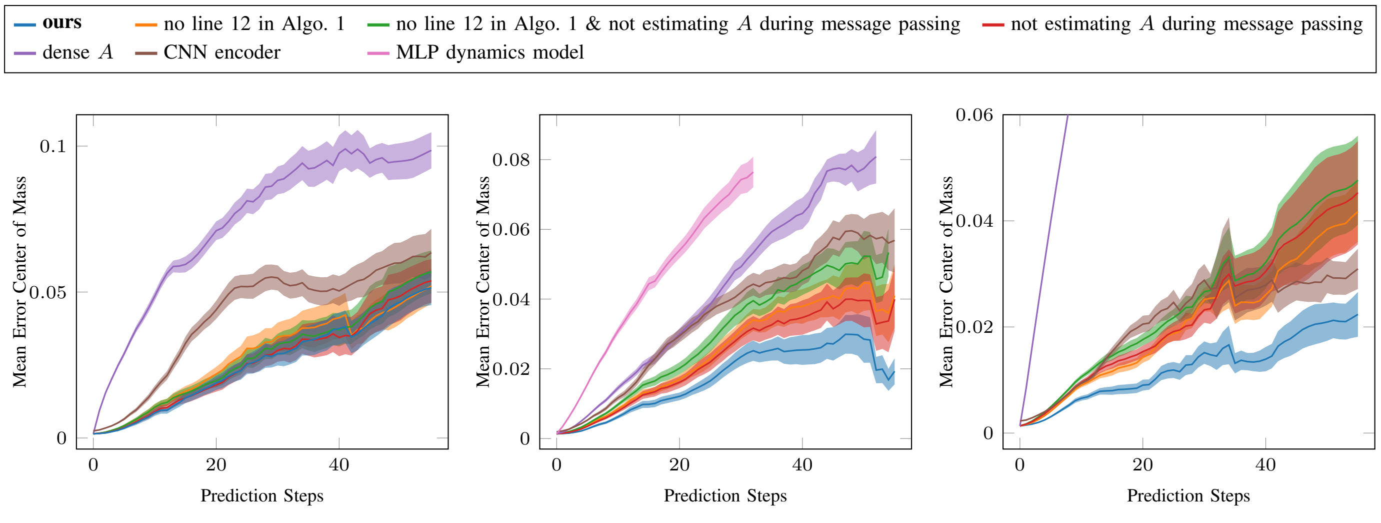


(b) Predictions with dense adjacency matrix baseline Sec. VII-D3



(c) Predictions with CNN decoder baseline Sec. VII-G

Fig. 8: Forward predictions of the model when applying an action sequence to the red pusher after observing the scene only initially. The column on the right corresponds to the prediction after 38 steps. As one can see, with our proposed method in (a), the predictions are very sharp, even after 38 steps, while the dense adjacency matrix version (Sec. VII-D3) in (b) leads to drifting objects until the predictions are not useful anymore. The CNN decoder baseline Sec. VII-G is even worse, such that after only a few steps the predictions are of little use. Note that multiple object interactions happen in this scene.



(a) 2 objects in the scene (b) 4 objects in the scene (as during training) (c) 8 objects in the scene

Fig. 9: Performance comparison in terms of mean center of mass prediction error of the objects over the number of prediction steps into the future on test data containing different numbers of objects. The center of mass is estimated from the density prediction of the NeRF corresponding to each object according to (26). One prediction step corresponds to a 2 cm movement of the pusher, i.e. for 50 steps the pusher has moved 1 m. For the MLP dynamics model in (b), it at some point did not predict all objects anymore, which meant no center of mass could be calculated further. Our method outperforms all baselines.

after only a few time-steps, showing that it has overfitted to the number of 4 objects as in the training data.

E. Importance of GNN – Comparison to Dense Dynamic Models

Replacing the GNN with a fully connected MLP that predicts the whole set of latent vectors $z_{1:m}^{t+1} = F_{\text{MLP}}(z_{1:m}^{t})$ from the previous $z_{1:m}^{t}$ leads to even worse performance than with a dense adjacency matrix within a GNN regarding the center of mass forward prediction capabilities as shown in Fig. 9b. Obviously, this model cannot generalize to more objects than during training due to its fixed size input.

F. Advantages of Implicit Object Encoder – Comparison to CNN Encoder

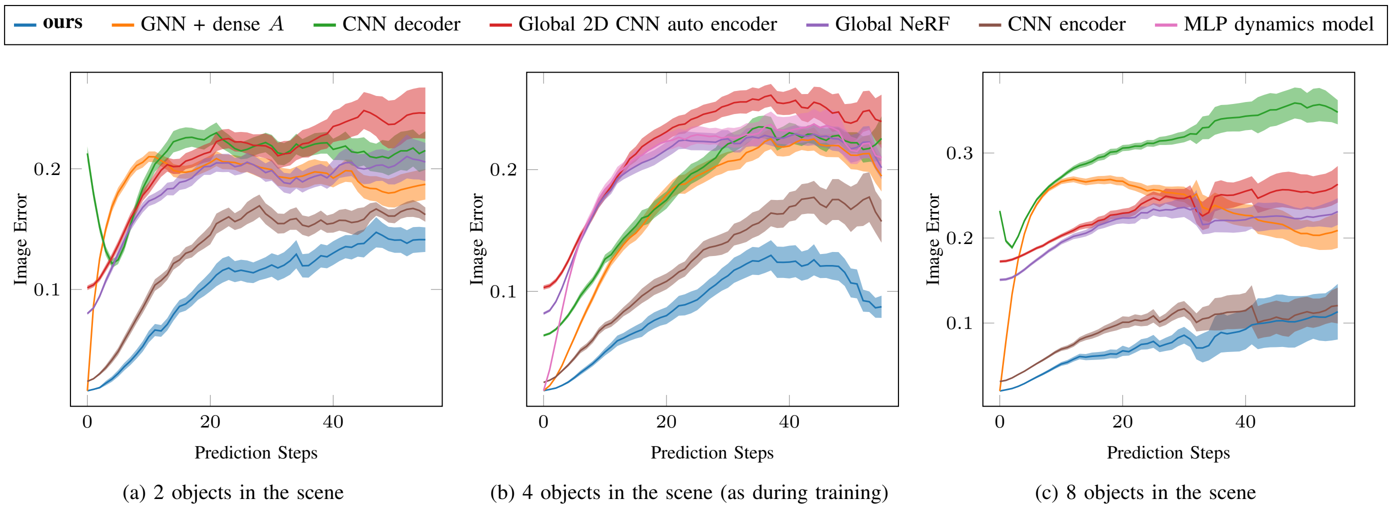
Here we exchange the implicit object encoder with a 2D CNN object encoder. The resulting auto-encoder framework is very similar to the architecture of [65]. More specifically, we encode each masked image observation with a 2D CNN to produce a feature vector. The feature vectors from the different views are aggregated into the final latent vectors for each object. We use the encoder architecture from [38], but adjust it to the compositional multi-object case by incorporating object masks. Since this encoder is not an implicit function of $\mathcal{X}$, we cannot modify the latent vectors by applying rigid transformations and hence need to encode the actions differently. In order to do so, we train a separate MLP network that predicts the latent vector of the pusher resulting from applying an action to it. This gives the modified z_a for line 5 in algorithm 1 for the CNN encoder baseline. The rest of the architecture, i.e. the GNN, the compositional NeRF decoder, estimating the adjacency matrix during message passing from the model, etc., stays the same.

As can be seen in Fig. 9a and Fig. 10, replacing the proposed implicit object encoder with a CNN encoder, the performance is better compared to the other baselines, but still clearly worse than with the proposed method.

G. Comparison to 2D Baselines – Importance of NeRF as Decoder

In this section, we replace the NeRF decoder with a 2D image decoder based on a transposed convolutional network to investigate the importance NeRFs. We use the same architecture for the decoder, which we call D_{CNN}, as the CNN baseline in [38]. This decoder takes as input one single latent vector and the camera matrix, i.e. $I = D_{\text{CNN}}(z, K)$. In order to make it compositional, we aggregate the set of latent vectors $z_{1:m}$ from Ω with a mean operation and then pass the aggregated feature through an MLP to produce the single z (with 4 times the dimension of z_j) for D_{CNN}. The rest of the architecture, i.e. the implicit object encoder in particular and the GNN, stays the same. Since there is no clear way to estimate the adjacency matrix from D_{CNN}, we use a dense adjacency matrix for the GNN.

As one can see in Fig. 10, the long-term prediction performance of the CNN decoder is significantly worse than with a compositional NeRF model as the decoder, especially when asking for numbers of objects that differ from the training distribution. Qualitatively, one can see in Fig. 8c that not only the initial reconstruction is much less sharp compared to the NeRF-based models, but especially also that even after only a few time-steps, the predictions with the CNN decoder are of



(a) 2 objects in the scene (b) 4 objects in the scene (as during training) (c) 8 objects in the scene

Fig. 10: Performance comparison in terms of mean absolute image error between the reconstructed image from the predicted latent vectors over the number of time steps into the future and the corresponding ground truth image observations, averaged over the test dataset for scenarios containing 2, 4, and 8 objects (plus the pusher). One prediction step corresponds to 2 cm movement, i.e. for 50 steps the pusher has moved 1 m.

little use.

H. Comparison to Global Scene Representation Baselines

Finally, we compare to two other baselines where the scene is represented globally with one single latent vector per time-step. The dynamics model for these baselines is an MLP $z^{t+1} = F_{\text{MLP}}(z^t, q)$ that takes the action q as an additional input. Therefore, the model also implicitly has to determine which object is the pusher where the action is applied to.

1) Global Latent Variable Model with Global NeRF Decoder: This baseline is very similar to [38], i.e. we use a CNN encoder, but for the whole scene, producing one latent vector which conditions a single NeRF that tries to reconstruct the whole scene.

2) Global Latent Variable Model with 2D CNN Auto-Encoder: The 2D CNN auto-encoder baseline uses both a 2D CNN encoder and 2D CNN decoder as well as a single latent vector representing the whole scene. Such frameworks have been used many times in the literature, e.g. [74, 23, 22, 60]. We use the same CNN decoder D_{CNN} from Sec. VII-G that now takes the single latent vector directly as input. As the encoder, we use the one from [38].

Fig. 10 shows that both global baselines struggle with our scenarios as they contain multiple objects that interact.

I. Summary of Performance Comparisons

In the previous sections, we have shown that our method outperforms all other baselines both in terms of pure reconstruction error (as can be seen in Fig. 10 by the error after zero prediction steps, i.e. in the initial configuration of each test scene) *and* its ability to perform long term predictions for action sequences forward unrolled on the model's own predictions. Estimating the adjacency matrix from the model itself is important for long-term stability as it prevents objects from drifting away. Too large drift makes future predictions for a pushing task meaningless.

Since the reconstruction error of our proposed method from observations without any dynamics is already better than the baselines, the question arises if the increased prediction performance compared to the baseline is just an artifact of the lower reconstruction error. To investigate this, we show in Fig. 11 the error in the image space between the reconstruction when having access to the observations at each time step and the reconstruction from the predicted latent vectors into the future after observing the scene only once at the beginning. This shows the increase in error relative to the reconstruction process. The results indicate that not solely the reconstruction process itself is the reason for the better performance, but that the structural choices of the proposed framework also enable to learn the underlying dynamics more precisely.

J. Planning and Execution Results on Box Sorting Task

To demonstrate the effectiveness of the learned model, we utilize it to solve a challenging box sorting task, where the red pusher needs to push the blue and yellow boxes into their corresponding goal regions as shown in Fig. 1. This task is in part inspired by the object sorting task in [18]. The cost function $\mathcal{C}_g$ in Algorithm 2 determines how many objects are outside of their goal region, which is computed for each object j from their corresponding density σ_j and color c_j predictions of the model itself. The goal is fulfilled if all objects are in their respective goal regions.

This object-sorting task is challenging for multiple reasons. First, the dynamics of pushing is non-trivial [26, 85, 11, 61]. In our particular case, many objects potentially interact, which further complicates the setup. Pushing one object could undo an object that is already at the goal, hence a greedy strategy of just pushing the objects straight to the goal region would

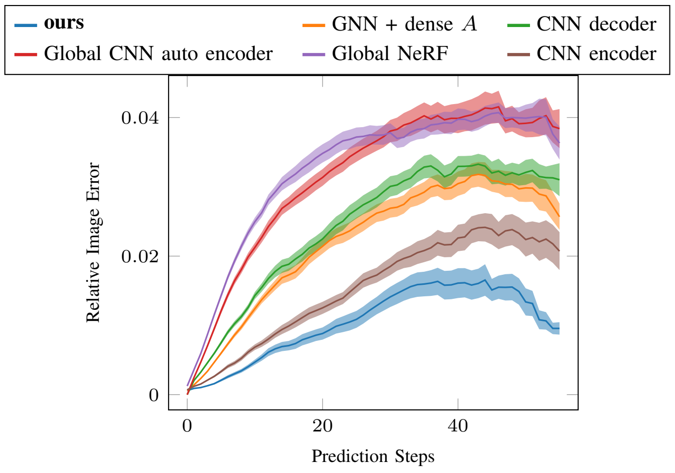


Fig. 11: Relative prediction error, i.e. error in image space between reconstruction when having access to observations at each time step and the reconstruction from the predicted latent vectors into the future after observing the scene only in the beginning.

fail. In addition, movements, i.e. actions, of the pusher do usually not immediately lead to a change in the cost function, since contact with the object from a suitable side has to be established, for which it is often necessary for the pusher to move around objects [12, 61]. Therefore, applying planning methods that are too local like a cross-entropy method would fail for this scenario. Our prediction model combined with the LS-RRT algorithm solves these tasks efficiently, just from image observations of the scene, see Fig. 1 and the video. In the first row of Tab. I, we show the total size of the exploration trees for solving the tasks for scenes that contain 1 to 6 objects.

As a baseline comparison, we consider planning with a GNN that uses a fully connected adjacency matrix (Sec. VII-D3) to understand the importance of a precise dynamics model. From the second row of Tab. I, one can see that planning with a dense A either fails to find a solution as the pusher may not be able to move the object correctly, or generates a plan that is tough to follow in the simulation environment. The reason for this is that with a dense A, as shown in Fig. 8b, the model induces too much drift of objects that do not interact, which makes planning for pushing scenarios extremely difficult. Objects can also drift closer or away again to/from the goal by model errors.

Finally, we compare the LS-RRT with a naive control tree algorithm, where we still use our full model, but only sample random nodes in the tree to extend. As shown in the last row of Tab. I, though the planner can find a path to push a single object, it fails to solve tasks containing more objects within the timeout. This demonstrates the benefits of our object encoder being an implicit function and being able to relate information in the 3D world in terms of center of mass predictions via the learned NeRFs to the latent vectors, both of which make planning much more efficient compared to naive control trees.

Number of Objects	1	2	4	6
RRT with full model	256	2341	23819	85022
RRT with dense A	NS	FE	FE	NS
Control tree with full model	24019	NS	NS	NS

TABLE I: Number of samples to find a solution with the latent space RRT planning algorithm for scenarios containing 1, 2, 4, and 6 objects. NS means no solution found within 10^5 samples. FE means failed to execute the found plan for 10 times.

VIII. Conclusion

Purely visual dynamics models are of high interest to both, the computer vision and robotics community, as they avoid making explicit shape and scene model assumptions and imply end-to-end perception. However, to support manipulation planning and reasoning we need models that generalize strongly over objects and provide stable long-term predictions. In this paper we proposed a system that introduces 3D structural and compositional priors at various levels, namely compositional NeRFs, 3D implicit object encoders, and compositional GNNs dynamics with an adaptive adjacency matrix. Together our system exhibits significantly stronger long-term prediction performance compared to multiple baselines without these priors or without compositional scene representations, and supports using a latent space RRT planner to solve a challenging box rearrangement task.

We have shown generalization over different numbers of objects, notably up to two times more than during training.

We have considered rigid body dynamics in this work only. However, we believe that the insights can be extended to deformable scenarios as well, although we anticipate that other planning methods or extensions to RRT might become necessary.

Acknowledgements

This research has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2002/1 "Science of Intelligence" – project number 390523135. Danny Driess thanks the International Max-Planck Research School for Intelligent Systems (IMPRS-IS) for the support. The authors thank Valentin Hartmann for discussions regarding RRTs.

References

[1] Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 2022. doi: 10.1109/LRA.2022.3150497.

[2] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *arXiv preprint arXiv:1606.07419*, 2016.

[3] Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks

for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

[4] 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.

[5] Maria Bauza, Francois R Hogan, and Alberto Rodriguez. A data-efficient approach to precise and controlled pushing. In *Conference on Robot Learning*, pages 336–345. PMLR, 2018.

[6] Michel Breyer, Jen Jen Chung, Lionel Ott, Siegwart Roland, and Nieto Juan. Volumetric grasping network: Real-time 6 dof grasp detection in clutter. In *Conference on Robot Learning*, 2020.

[7] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

[8] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[9] Danny Driess, Jung-Su Ha, and Marc Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. *arXiv preprint arXiv:2006.05398*, 2020.

[10] Danny Driess, Jung-Su Ha, Russ Tedrake, and Marc Toussaint. Learning geometric reasoning and control for long-horizon tasks from visual input. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[11] Danny Driess, Jung-Su Ha, and Marc Toussaint. Learning to solve sequential physical reasoning problems from a scene image. *The International Journal of Robotics Research (IJRR)*, 2021. doi: 10.1177/02783649211056967.

[12] Danny Driess, Jung-Su Ha, Marc Toussaint, and Russ Tedrake. Learning models as functionals of signed-distance fields for manipulation planning. In *Conference on Robot Learning (CoRL)*, 2021.

[13] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14324–14334, 2021.

[14] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *CoRL*, pages 344–356, 2017.

[15] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[16] Cathrin Elich, Martin R Oswald, Marc Pollefeys, and Joerg Stueckler. Weakly supervised learning of multi-object 3d scene decompositions using deep shape priors. *arXiv preprint arXiv:2010.04030*, 2020.

[17] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *arXiv preprint arXiv:1605.07157*, 2016.

[18] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Proceedings of the 5th Conference on Robot Learning*. PMLR, 2022.

[19] Niklas Funk, Georgia Chalvatzaki, Boris Belousov, and Jan Peters. Learn2Assemble with Structured Representations and Search for Robotic Architectural Construction. In *Proceedings of the 5th Conference on Robot Learning*. PMLR, 2022.

[20] Michelle Guo, Alireza Fathi, Jiajun Wu, and Thomas Funkhouser. Object-centric neural scene rendering. *arXiv preprint arXiv:2012.08503*, 2020.

[21] Jung-Su Ha, Danny Driess, and Marc Toussaint. Learning neural implicit functions as object representations for robotic manipulation. *arXiv preprint arXiv:2112.04812*, 2021.

[22] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[23] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[24] Valentin Noah Hartmann, Andreas Orthey, Danny Driess, Ozgur S Oguz, and Marc Toussaint. Long-horizon multi-robot rearrangement planning for construction assembly. *arXiv preprint arXiv:2106.02489*, 2021.

[25] Kris Hauser. Semi-infinite programming for trajectory optimization with non-convex obstacles. *The International Journal of Robotics Research*, 2018.

[26] François Robert Hogan and Alberto Rodriguez. Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. *arXiv preprint arXiv:1611.08268*, 2016.

[27] Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. *arXiv preprint arXiv:1806.04166*, 2018.

[28] Jeffrey Ichnowski, Yahav Avigal, Justin Kerr, and Ken Goldberg. Dex-nerf: Using a neural radiance field to grasp transparent objects. *arXiv preprint arXiv:2110.14217*, 2021.

[29] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019.

[30] Zhenyu Jiang, Yifeng Zhu, Maxwell Svetlik, Kuan Fang, and Yuke Zhu. Synergies between affordance and geometry: 6-dof grasp detection via implicit representations. *arXiv preprint arXiv:2104.01542*, 2021.

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

[32] Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. *Advances in neural information processing systems*, 32:10724–10734, 2019.

[33] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, and Zhaoyang Lv. Neural 3d video synthesis. *arXiv preprint arXiv:2103.02597*, 2021.

[34] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.

[35] 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)*, pages 1205–1211. IEEE, 2019.

[36] Yunzhu Li, Toru Lin, Kexin Yi, Daniel Bear, Daniel Yamins, Jiajun Wu, Joshua Tenenbaum, and Antonio Torralba. Visual grounding of learned physical models. In *International conference on machine learning*, pages 5927–5936. PMLR, 2020.

[37] Yunzhu Li, Antonio Torralba, Anima Anandkumar, Dieter Fox, and Animesh Garg. Causal discovery in physical systems from videos. *Advances in Neural Information Processing Systems*, 33, 2020.

[38] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022.

[39] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6498–6508, 2021.

[40] Lucas Manuelli, Yunzhu Li, Pete Florence, and Russ Tedrake. Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning. *arXiv preprint arXiv:2009.05085*, 2020.

[41] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[42] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421, 2020.

[43] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. *arXiv preprint arXiv:1806.08047*, 2018.

[44] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[45] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[46] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5379–5389, 2019.

[47] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2856–2865, 2021.

[48] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[49] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021.

[50] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021.

[51] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020.

[52] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

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

[54] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*,

pages 10318–10327, 2021.

[55] Haozhi Qi, Xiaolong Wang, Deepak Pathak, Yi Ma, and Jitendra Malik. Learning long-term visual dynamics with region proposal interaction networks. *arXiv preprint arXiv:2008.02265*, 2020.

[56] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2304–2314, 2019.

[57] 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*, pages 4470–4479. PMLR, 2018.

[58] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[59] Connor Schenck and Dieter Fox. Perceiving and reasoning about liquids using fully convolutional networks. *The International Journal of Robotics Research*, 37(4-5): 452–471, 2018.

[60] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[61] Ingmar Schubert, Danny Driess, Ozgur S. Oguz, and Marc Toussaint. Learning to execute: Efficient learning of universal plan-conditioned policies in robotics. In *NeurIPS 2021 - Neural Information Processing Systems*, 2021.

[62] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

[63] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks. *arXiv preprint arXiv:2009.05613*, 2020.

[64] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *arXiv preprint arXiv:1906.01618*, 2019.

[65] Karl Stelzner, Kristian Kersting, and Adam R Kosiorek. Decomposing 3d scenes into objects via unsupervised volume segmentation. *arXiv preprint arXiv:2104.01148*, 2021.

[66] Michael Strecke and Joerg Stueckler. DiffSDFSim: Differentiable rigid-body dynamics with implicit shapes. In *International Conference on 3D Vision (3DV)*, December 2021.

[67] HJ Suh and Russ Tedrake. The surprising effectiveness of linear models for visual foresight in object pile manipulation. *arXiv preprint arXiv:2002.09093*, 2020.

[68] Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832)*. 2022. URL http://underactuated.mit.edu.

[69] Marc Toussaint, Jung-Su Ha, and Danny Driess. Describing physics for physical reasoning: Force-based sequential manipulation planning. *IEEE Robotics and Automation Letters*, 2020.

[70] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12959–12970, 2021.

[71] Hsiao-Yu Fish Tung, Zhou Xian, Mihir Prabhudesai, Shamit Lal, and Katerina Fragkiadaki. 3d-oes: Viewpoint-invariant object-factorized environment simulators. *arXiv preprint arXiv:2011.06464*, 2020.

[72] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2019.

[73] Mark Van der Merwe, Qingkai Lu, Balakumar Sundaralingam, Martin Matak, and Tucker Hermans. Learning continuous 3d reconstructions for geometrically aware grasping. In *Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[74] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *arXiv preprint arXiv:1506.07365*, 2015.

[75] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. *Advances in neural information processing systems*, 30:4539–4547, 2017.

[76] Youngsun Wi, Pete Florence, Andy Zeng, and Nima Fazeli. Virdo: Visio-tactile implicit representations of deformable objects. *arXiv preprint arXiv:2202.00868*, 2022.

[77] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9421–9431, 2021.

[78] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar.

Neural fields in visual computing and beyond. *arXiv preprint arXiv:2111.11426*, 2021.

[79] Hongyi Xu, Thiemo Alldieck, and Cristian Sminchisescu. H-nerf: Neural radiance fields for rendering and temporal reconstruction of humans in motion. *Advances in Neural Information Processing Systems*, 34, 2021.

[80] Zhenjia Xu, Zhanpeng He, Jiajun Wu, and Shuran Song. Learning 3d dynamic scene representations for robot manipulation. *arXiv preprint arXiv:2011.01968*, 2020.

[81] Yufei Ye, Maneesh Singh, Abhinav Gupta, and Shubham Tulsiani. Compositional video prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10353–10362, 2019.

[82] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B Tenenbaum. Clevrer: Collision events for video representation and reasoning. *arXiv preprint arXiv:1910.01442*, 2019.

[83] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021.

[84] Hong-Xing Yu, Leonidas J Guibas, and Jiajun Wu. Unsupervised discovery of object radiance fields. *arXiv preprint arXiv:2107.07905*, 2021.

[85] Jiaji Zhou, Yifan Hou, and Matthew T Mason. Pushing revisited: Differential flatness, trajectory planning, and stabilization. *The International Journal of Robotics Research*, 38(12-13):1477–1489, 2019.

[86] Guangxiang Zhu, Zhiao Huang, and Chongjie Zhang. Object-oriented dynamics predictor. In *Advances in Neural Information Processing Systems*, 2018.