Supplementary Material for "Improving Generative Imagination in Object-Centric World Models"

A. Model Details

In this section, we will give a detailed description of each stage, especially those not described in detail in the main text.

For each timestep t, we will describe the generation of $\mathbf{z}_t^{\text{ctx}}, \tilde{\mathbf{z}}_t, \tilde{\mathbf{o}}_t, \bar{\mathbf{z}}_t, \bar{\mathbf{o}}_t, \mathbf{o}_t, \mathbf{x}_t$ (in that order), given the full history $\mathbf{z}_{<t}^{\text{ctx}}$, $\dot{\mathbf{z}}_{<t}$, and $\mathbf{o}_{<t}$. Generation consists of the following stages:

1. **Context**. Given context history $\mathbf{z}_{<t}^{\text{ctx}}$, we generate the new context $\mathbf{z}_t^{\text{ctx}}$.

2. **Propagation**. We compute $\{\tilde{\mathbf{z}}_t^k\}_{k=1}^K$, and then update the object attributes $\{\mathbf{o}_{t-1}^k\}_{k=1}^K$ to $\{\tilde{\mathbf{o}}_t^k\}_{k=1}^K$.

3. **Discovery**. A grid of $H \times W$ new object latents $\{\bar{\mathbf{z}}_t^{ij}, (i,j) \in \{(1,1),\dots,(H,W)\}\}$ will be sampled from some predefined prior, and then for each $(i,j) \in \{(1,1),\dots,(H,W)\}$, $\bar{\mathbf{o}}_t^{ij}$ will be obtained by passing each $\bar{\mathbf{z}}_t^{ij}$ through some deterministic function. As mentioned in the main text, discovery will only be used during inference but not generation. Here, the discovery priors are only used to regularize inference.

4. **Rendering**. Given the set of propagated objects $\tilde{\mathbf{o}}_t$ and discovered objects $\bar{\mathbf{o}}_t$, we will select a maximum number of K objects $\{\mathbf{o}^k\}_{k=1}^K$ with the highest presence value. These objects will also be propagated to the next timestep. We then render the frame $\mathbf{x}_t$ using the selected objects $\{\mathbf{o}^k\}_{k=1}^K$, which generates the foreground image $\boldsymbol{\mu}_t^{\text{fg}}$ and mask $\boldsymbol{\alpha}_t$, and the context latent $\mathbf{z}_t^{\text{ctx}}$, which generates the background image $\boldsymbol{\mu}_t^{\text{bg}}$.

Below we describe the implementation details of each stage.

A.1. Context

Generation. The prior $p_\theta(\mathbf{z}_t^{\text{ctx}}|\mathbf{z}_{<t}^{\text{ctx}})$ is implemented as follows:

$$\mathbf{h}_t^{\text{ctx}} = \text{RNN}_{\text{prior}}^{\text{ctx}}(\mathbf{z}_{t-1}^{\text{ctx}}, \mathbf{h}_{t-1}^{\text{ctx}}) \tag{1}$$

$$[\boldsymbol{\mu}_t^{\text{ctx}}, \boldsymbol{\sigma}_t^{\text{ctx}}] = \text{MLP}_{\text{prior}}^{\text{ctx}}(\mathbf{h}_t^{\text{ctx}}) \tag{2}$$

$$\mathbf{z}_t^{\text{ctx}} \sim \mathcal{N}(\boldsymbol{\mu}_t^{\text{ctx}}, \boldsymbol{\sigma}_t^{\text{ctx}}). \tag{3}$$

Inference. The posterior $q_\phi(\mathbf{z}_t^{\text{ctx}}|\mathbf{x}_t, \mathbf{z}_{<t}^{\text{ctx}})$ is implemented as follows:

$$\hat{\mathbf{h}}_t^{\text{ctx}} = \text{RNN}_{\text{post}}^{\text{ctx}}(\mathbf{z}_{t-1}^{\text{ctx}}, \hat{\mathbf{h}}_{t-1}^{\text{ctx}}) \tag{4}$$

$$\mathbf{e}_{\text{enc},t}^{\text{ctx}} = \text{Conv}_{\text{enc}}^{\text{ctx}}(\mathbf{x}_t) \tag{5}$$

$$[\boldsymbol{\mu}_t^{\text{ctx}}, \boldsymbol{\sigma}_t^{\text{ctx}}] = \text{MLP}_{\text{post}}^{\text{ctx}}([\hat{\mathbf{h}}_t^{\text{ctx}}, \mathbf{e}_{\text{enc},t}^{\text{ctx}}]) \tag{6}$$

$$\mathbf{z}_t^{\text{ctx}} \sim \mathcal{N}(\boldsymbol{\mu}_t^{\text{ctx}}, \boldsymbol{\sigma}_t^{\text{ctx}}). \tag{7}$$

A.2. Propagation

Generation. The overall procedure is described in the main text, so we only describe some network implementation details.

The self-interaction encoding $\mathbf{e}_t^{k,k}$, pairwise-interaction encoding $\mathbf{e}_t^{k,j}$, and the interaction weights $w_t^{k,j}$ are computed as follows:

$$\mathbf{e}_t^{k,k} = \text{MLP}_{\text{prior}}^{\text{self}}(\mathbf{u}_t^k) \tag{8}$$

$$\mathbf{e}_t^{k,j} = \text{MLP}_{\text{prior}}^{\text{rel}}(\mathbf{u}_t^k, \mathbf{u}_t^j) \tag{9}$$

$$w_t^{k,j} = \text{MLP}_{\text{prior}}^{\text{weight}}(\mathbf{u}_t^k, \mathbf{u}_t^j). \tag{10}$$

Given the hidden state $\mathbf{h}_t^k$ of the OS-RNN, the state latent $\tilde{\mathbf{z}}_t^{\text{state},k}$ is computed as follows:

$$[\tilde{\boldsymbol{\mu}}_t^{\text{state},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{state},k}] = \text{MLP}_{\text{prior}}^{\text{state}}(\mathbf{h}_t^k) \tag{11}$$

$$\tilde{\mathbf{z}}_t^{\text{state},k} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_t^{\text{state},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{state},k}), \tag{12}$$

and given the state latent $\mathbf{z}_t^{\text{state},k}$, the attribute latents $\mathbf{z}_t^{\text{att},k} = [\mathbf{z}_t^{\text{pres},k}, \mathbf{z}_t^{\text{depth},k}, \mathbf{z}_t^{\text{where},k}, \mathbf{z}_t^{\text{what},k}]$ are computed as follows:

$$[\tilde{\boldsymbol{\rho}}^{\text{pres},k}, \tilde{\boldsymbol{\mu}}_t^{\text{depth},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{depth},k}, \tilde{\boldsymbol{\mu}}_t^{\text{where},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{where},k}, \tilde{\boldsymbol{\mu}}_t^{\text{what},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{what},k}] = \text{MLP}_{\text{prior}}^{\text{att}}(\tilde{\mathbf{z}}_t^{\text{state},k}) \tag{13}$$

$$\tilde{\mathbf{z}}_t^{\text{pres},k} \sim \text{Bernoulli}(\tilde{\boldsymbol{\rho}}^{\text{pres},k}) \tag{14}$$

$$\tilde{\mathbf{z}}_t^{\text{depth},k} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_t^{\text{depth},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{depth},k}) \tag{15}$$

$$\tilde{\mathbf{z}}_t^{\text{where},k} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_t^{\text{where},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{where},k}) \tag{16}$$

$$\tilde{\mathbf{z}}_t^{\text{what},k} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_t^{\text{what},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{what},k}). \tag{17}$$

Inference. We only need to describe the implementation of $q_\phi(\tilde{\mathbf{z}}_t^{\text{state},k}|\mathbf{x}_t, \mathbf{z}_{<t}^{\text{ctx}}, \dot{\mathbf{z}}_{<t})$. First, a posterior OS-RNN will be used to update the posterior object state:

$$\hat{\mathbf{h}}_t^k = \text{RNN}_{\text{post}}^{\text{os}}([\mathbf{o}_{t-1}^k, \mathbf{z}_t^{\text{state},k}, \mathbf{e}_{t-1}^{\text{ctx},k}, \mathbf{e}_{t-1}^{\text{rel},k}], \hat{\mathbf{h}}_{t-1}^k)\,. \tag{18}$$

Here, $\mathbf{e}_{t-1}^{\text{ctx},k}$ is computed using exactly the same process and network as generation, and $\mathbf{e}_{t-1}^{\text{rel},k}$ is computed using a similar process during generation but with a separate set of posterior networks $\text{MLP}_{\text{post}}^{\text{self}}$, $\text{MLP}_{\text{post}}^{\text{rel}}$, and $\text{MLP}_{\text{post}}^{\text{weight}}$.

Then, a proposal region of the image $\mathbf{x}_t$ centered at the previous object location $\mathbf{o}_{t-1}^{xy,k}$ is extracted and encoded. The size $\mathbf{s}_t^{\text{prop}}$ (2-dimensional for (h, w)) of this proposal area is computed from $\hat{\mathbf{h}}_t^k$:

$$\mathbf{s}_t^{\text{prop}} = \mathbf{o}_{t-1}^{hw,k} + s^{\min} + (s^{\max} - s^{\min}) \cdot \sigma(\text{MLP}^{\text{prop}}(\hat{\mathbf{h}}_t^k)) \tag{19}$$

Where $s^{\min}$ and $s^{\max}$ are hyperparameters that control the minimum and maximum proposal update size. After that, the proposal is extracted and encoded:

$$\mathbf{g}_t^{\text{prop},k} = \text{ST}(\mathbf{x}_t, \mathbf{o}_{t-1}^{xy,k}, \mathbf{s}_t^{\text{prop}}) \tag{20}$$

$$\mathbf{e}_t^{\text{prop},k} = \text{Conv}^{\text{prop}}(\mathbf{g}_t^{\text{prop},k}). \tag{21}$$

Then $\hat{\mathbf{h}}_t^k$ and $\mathbf{e}_t^{\text{prop},k}$ will be used to infer $\tilde{\mathbf{z}}_t^{\text{state},k}$:

$$[\tilde{\boldsymbol{\mu}}_t^{\text{state},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{state},k}] = \text{MLP}_{\text{post}}^{\text{state}}([\hat{\mathbf{h}}_t^k, \mathbf{e}_t^{\text{prop},k}])\ , \tag{22}$$

$$\tilde{\mathbf{z}}_t^{\text{state},k} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_t^{\text{state},k}, \tilde{\boldsymbol{\sigma}}_t^{\text{state},k}). \tag{23}$$

Attribute updates. For this part we describe the details of object attribute update function f^{pres}, f^{depth}, f^{where}, and f^{what}. These functions are implemented as follows:

$$[\mathbf{g}_t^{\text{depth},k}, \mathbf{g}_t^{\text{where},k}, \mathbf{g}_t^{\text{what},k}] = \sigma(\text{MLP}^{\text{gate}}(\tilde{\mathbf{z}}_t^{\text{state},k})) \tag{24}$$

$$\tilde{\mathbf{o}}^{\text{pres},k} = \mathbf{o}_{t-1}^{\text{pres},k} \cdot \tilde{\mathbf{z}}_t^{\text{pres}} \tag{25}$$

$$\tilde{\mathbf{o}}_t^{\text{depth},k} = \mathbf{o}_t^{\text{depth},k} + c^{\text{depth}} \cdot \mathbf{g}_t^{\text{depth},k} \cdot \tilde{\mathbf{z}}_t^{\text{depth},k} \tag{26}$$

$$\tilde{\mathbf{o}}_t^{xy,k} = \mathbf{o}_t^{xy,k} + c^{xy} \cdot \mathbf{g}_t^{xy,k} \cdot \tanh(\tilde{\mathbf{z}}_t^{xy,k}) \tag{27}$$

$$\tilde{\mathbf{o}}_t^{hw,k} = \mathbf{o}_t^{hw,k} + c^{hw} \cdot \mathbf{g}_t^{hw,k} \cdot \tanh(\tilde{\mathbf{z}}_t^{hw,k}) \tag{28}$$

$$\tilde{\mathbf{o}}_t^{\text{what},k} = \mathbf{o}_t^{\text{what},k} + c^{\text{what}} \cdot \mathbf{g}_t^{\text{what},k} \cdot \tanh(\tilde{\mathbf{z}}_t^{\text{what},k}). \tag{29}$$

Note we split $\mathbf{o}^{\text{where}}$ into $\mathbf{o}^{hw}$ and $\mathbf{o}^{xy}$. Here, $c^{\text{depth}}, c^{xy}, c^{hw}, c^{\text{what}}$ are real-valued hyperparameters between 0 and 1 that control the degree of update we want. Note that for f^{depth}, f^{where}, and f^{what}, the corresponding update gates $\mathbf{g}_t^{\text{depth},k}$, $\mathbf{g}_t^{\text{where},k}$, and $\mathbf{g}_t^{\text{what},k}$ will first be computed from $\tilde{\mathbf{z}}_t^{\text{state},k}$ and used to mask the update values.

A.3. Discovery

Generation. We assume an independent prior for each object:

$$p(\bar{\mathbf{z}}_t^{ij}) = p(\bar{\mathbf{z}}_t^{\text{state},ij}) p(\bar{\mathbf{z}}_t^{\text{pres},ij}) \Big\{ p(\bar{\mathbf{z}}_t^{\text{depth},ij}) p(\bar{\mathbf{z}}_t^{\text{where},ij}) p(\bar{\mathbf{z}}_t^{\text{what},ij}) \Big\}^{\bar{\mathbf{z}}^{\text{pres},ij}}. \tag{30}$$

All of these priors are fixed Gaussian distributions with chosen mean and variance except for $p(\bar{\mathbf{z}}^{\text{pres}})$, which is a Bernoulli distribution.

Inference. We feed in the image $\mathbf{x}_t$ along with the difference between the $\mathbf{x}_t$ and the reconstructed background into an encoder to get an encoding of the current image $\mathbf{e}_t^{\text{img}}$ of shape (H, W, C):

$$\mathbf{e}_t^{\text{img}} = \text{Conv}^{\text{disc}}([\mathbf{x}_t, \mathbf{x}_t - \boldsymbol{\mu}_t^{\text{bg}}]) \tag{31}$$

To infer $\bar{\mathbf{z}}_t$, besides the current image $\mathbf{x}_t$, we also consider the propagated objects $\{\tilde{\mathbf{o}}_t^k\}_{k=1}^K$ to prevent rediscovering already propagated objects. We adopt the same mechanism in SILOT to condition discovery on propagation. Specifically, for each discovery cell $(i, j) \in \{(1, 1), \dots, (H, W)\}$, a vector $\mathbf{e}_t^{\text{cond},ij}$ will be computed as a weighted sum of all propagated objects $\{\tilde{\mathbf{z}}_t^k\}_{k=1}^K$, with the weights computed by passing the relative distance between the propagated object $\tilde{\mathbf{o}}_t^{xy,k}$ and the cell center $\mathbf{c}^{ij}$ into a Gaussian kernel:

$$\mathbf{e}_t^{\text{cond},ij} = \sum_{k=1}^K G(\tilde{\mathbf{o}}^{xy,k} - \mathbf{c}^{ij}, \sigma^{\text{cond}}) \cdot \text{MLP}^{\text{cond}}(\tilde{\mathbf{o}}_t^k) \tag{32}$$

where G is a 2-D Gaussian kernel, and σ^{cond} is a hyperparameter.

The discovered latents will then be computed conditioned on the image features and the encoding of propagated objects:

$$[\bar{\boldsymbol{\mu}}_t^{\text{state},ij}, \bar{\boldsymbol{\sigma}}_t^{\text{state},ij}, \bar{\boldsymbol{\rho}}^{\text{pres},ij}, \bar{\boldsymbol{\mu}}_t^{\text{depth},ij}, \bar{\boldsymbol{\sigma}}_t^{\text{depth},ij}, \bar{\boldsymbol{\mu}}_t^{\text{where},ij}, \bar{\boldsymbol{\sigma}}_t^{\text{where},ij}, \bar{\boldsymbol{\mu}}_t^{\text{what},ij}, \bar{\boldsymbol{\sigma}}_t^{\text{what},ij}] = \text{MLP}^{\text{disc}}([\mathbf{e}_t^{\text{img},ij}, \mathbf{e}_t^{\text{cond},ij}] \tag{33}$$

$$\bar{\mathbf{z}}_t^{\text{state},ij} \sim \mathcal{N}(\bar{\boldsymbol{\mu}}_t^{\text{state},ij}, \bar{\boldsymbol{\sigma}}_t^{\text{state},ij}) \tag{34}$$

$$\bar{\mathbf{z}}_t^{\text{pres},ij} \sim \text{Bernoulli}(\bar{\boldsymbol{\rho}}^{\text{pres},ij}) \tag{35}$$

$$\bar{\mathbf{z}}_t^{\text{depth},ij} \sim \mathcal{N}(\bar{\boldsymbol{\mu}}_t^{\text{depth},ij}, \bar{\boldsymbol{\sigma}}_t^{\text{depth},ij}) \tag{36}$$

$$\bar{\mathbf{z}}_t^{\text{where},ij} \sim \mathcal{N}(\bar{\boldsymbol{\mu}}_t^{\text{where},ij}, \bar{\boldsymbol{\sigma}}_t^{\text{where},ij}) \tag{37}$$

$$\bar{\mathbf{z}}_t^{\text{what},ij} \sim \mathcal{N}(\bar{\boldsymbol{\mu}}_t^{\text{what},ij}, \bar{\boldsymbol{\sigma}}_t^{\text{what},ij}) \tag{38}$$

Finally, we compute the object representation $\bar{\mathbf{o}}_t^{ij}$ using these latents. For $\bar{\mathbf{o}}_t^{\text{pres},ij}, \bar{\mathbf{o}}_t^{\text{depth},ij}, \bar{\mathbf{o}}_t^{\text{what},ij}$, they will just be equal to $\bar{\mathbf{z}}_t^{\text{pres},ij}, \bar{\mathbf{z}}_t^{\text{depth},ij}, \bar{\mathbf{z}}_t^{\text{what},ij}$. For $\bar{\mathbf{o}}_t^{\text{where},ij} = [\bar{\mathbf{o}}_t^{hw,ij}, \bar{\mathbf{o}}_t^{xy,ij}]$, we want $\bar{\mathbf{o}}_t^{hw,ij}$ to be in range $(0, 1)$ and $\bar{\mathbf{o}}_t^{xy,ij}$ in range $(-1, 1)$. Besides, as in SILOT, $\bar{\mathbf{z}}_t^{xy,ij}$ is relative to the cell center $\mathbf{c}^{ij}$, so we need to transform relative locations to global locations using

$$\bar{\mathbf{o}}_t^{hw,ij} = \sigma(\bar{\mathbf{z}}_t^{hw,ij}) \tag{39}$$

$$\bar{\mathbf{o}}_t^{xy,ij} = \mathbf{c}^{ij} + 2 \cdot \tanh(\bar{\mathbf{z}}_t^{xy,ij}) / [W, H] \tag{40}$$

where $\mathbf{c}^{ij} = 2 \cdot ([i, j] + 0.5) / [W, H] - 1$

A.4. Rendering

The background image $\boldsymbol{\mu}_t^{\text{bg}}$ will be decoded from the context latent $\mathbf{z}_t^{\text{ctx}}$:

$$\boldsymbol{\mu}_t^{\text{bg}} = \text{Deconv}_{\text{dec}}^{\text{ctx}}(\mathbf{z}_t^{\text{ctx}}) \tag{41}$$

For foreground, we will first select a set of K objects $\{\mathbf{o}_t^k\}_{k=1}^K$ from the set of discovered and propagated objects $\bar{\mathbf{o}}_t \cup \tilde{\mathbf{o}}_t$. To render the set of selected objects $\{\mathbf{o}_t^k\}_{k=1}^K$ into the foreground image $\boldsymbol{\mu}_t^{\text{fg}}$ and foreground mask $\boldsymbol{\alpha}_t$, a similar procedure in SILOT is used. First, individual object appearance $\hat{\mathbf{y}}_t^{\text{att},k}$ and mask $\hat{\boldsymbol{\alpha}}_t^{\text{att},k}$ are computed from $\mathbf{o}_t^{\text{what},k}$ and $\mathbf{o}_t^{\text{pres},k}$:

$$[\mathbf{y}_t^{\text{att},k}, \boldsymbol{\alpha}_t^{\text{att},k}] = \sigma(\text{Deconv}^{\text{what}}(\mathbf{o}_t^{\text{what},k})) \tag{42}$$

$$\hat{\boldsymbol{\alpha}}_t^{\text{att},k} = \boldsymbol{\alpha}_t^{\text{att},k} \cdot \mathbf{o}_t^{\text{pres},k} \tag{43}$$

$$\hat{\mathbf{y}}_t^{\text{att},k} = \hat{\boldsymbol{\alpha}}_t^{\text{att},k} \cdot \mathbf{y}^{\text{att},k}. \tag{44}$$

Here, $\hat{\mathbf{y}}_t^{\text{att},k}$ and $\hat{\boldsymbol{\alpha}}_t^{\text{att},k}$ will be of a small glimpse size (H_g, W_g). We will then transform them into full image size $(H_{\text{img}}, W_{\text{img}})$ by putting them in an empty canvas using a (inverse) Spatial Transformer:

$$\mathbf{y}_t^k = \text{ST}^{-1}(\hat{\mathbf{y}}_t^{\text{att},k}, \mathbf{o}_t^{\text{where},k}) \tag{45}$$

$$\boldsymbol{\alpha}_t^k = \text{ST}^{-1}(\hat{\boldsymbol{\alpha}}_t^{\text{att},k}, \mathbf{o}_t^{\text{where},k}) \tag{46}$$

Then $\boldsymbol{\mu}_t^{\text{fg}}$ and $\boldsymbol{\alpha}_t$ will be computed as pixel-wise weighted sums of these image-sized maps:

$$\mathbf{w}_t^k = \frac{\boldsymbol{\alpha}_t^k \cdot \sigma(\mathbf{o}_t^{\text{depth},k})}{\sum_{j=1}^K \boldsymbol{\alpha}_t^j \cdot \sigma(\mathbf{o}_t^{\text{depth},j})} \tag{47}$$

$$\boldsymbol{\mu}_t^{\text{fg}} = \sum_{k=1}^K \mathbf{w}_t^k \cdot \mathbf{y}_t^k \tag{48}$$

$$\boldsymbol{\alpha}_t = \sum_{k=1}^K \mathbf{w}_t^k \cdot \boldsymbol{\alpha}_t^k \tag{49}$$

The final rendered image will be $\boldsymbol{\mu}_t = \boldsymbol{\mu}_t^{\text{fg}} + (1 - \boldsymbol{\alpha}_t)\boldsymbol{\mu}_t^{\text{bg}}$. The likelihood $p_\theta(\mathbf{x}_t | \mathbf{z}_{\le t}^{\text{ctx}}, \dot{\mathbf{z}}_{\le t})$ is then

$$p_\theta(\mathbf{x}_t | \mathbf{z}_{\le t}^{\text{ctx}}, \dot{\mathbf{z}}_{\le t}) = \mathcal{N}(\mathbf{x}_t | \boldsymbol{\mu}_t, \sigma^2 \mathbf{I}) \tag{50}$$

where σ is a hyperparameter.

B. Architectures, Hyperparameters, and Training

B.1. Training

For all experiments, we use the Adam (Kingma & Ba, 2014) optimizer with a learning rate of 1×10^{-4} except for the maze dataset. We use a batch size of 16 for all experiments. Gradient clipping (Pascanu et al., 2013) with a maximum norm of 1.0 is applied. For both $\bar{\mathbf{z}}_k^{\text{pres},ij}$ and $\tilde{\mathbf{z}}_t^{\text{pres},k}$, we use a Gumbel-Softmax relaxation (Jang et al., 2016) with temperature τ to make sampling differentiable.

For experiments on datasets without background, we manually set $\boldsymbol{\mu}_t^{\text{bg}}$ to empty images. For the maze dataset, we turn off the gradient of the foreground module and only learn to reconstruct background for the first 500 steps. Also, we use a learning rate of 5×10^{-5} instead of 1×10^{-4}.

B.2. Architectures

All RNNs are implemented as LSTMs (Hochreiter & Schmidhuber, 1997). For all equations that describe RNN recurrence, the notation $\mathbf{h}$ includes both the hidden state and cell state used in common LSTMs. However, when $\mathbf{h}$ is used as an input to another network, we use only the hidden state. For all initial states ($\mathbf{h}_0$), we treat them as learnable parameters with unit Gaussian random initialization. For both the prior and posterior object-state RNN, inputs are first embedded with a single fully connected layer denoted by $\text{MLP}_{\text{prior}}^{\text{os}}$ and $\text{MLP}_{\text{post}}^{\text{os}}$.

For all networks that output variances of Gaussian distributions, we apply a softplus function to ensure that the variances are positive. For all networks that output the parameters of Bernoulli distributions (for $\mathbf{z}^{\text{pres}}$), we apply a sigmoid function.

Table 1 lists all networks. Here, $\text{LSTM}(a, b)$ denotes an LSTM with input size a and hidden size b. For MLPs, the Architecture column lists the hidden layer sizes, not including input and output layer. The identity of input and output variables can be found in equations where each network appears, and the dimensions of these variables will be given in Section B.3.

For all network layers except for output layers, we use the CELU (Barron, 2017) activation function. For all convolution layers except for output layers, we use group normalization (Wu & He, 2018) with 16 channels per group. Note that $\text{MLP}_{\text{prior}}^{\text{state}}$, $\text{MLP}_{\text{post}}^{\text{state}}$, $\text{MLP}_{\text{prior}}^{\text{att}}$, MLP^{gate} are implement as stride-1 convolutions to facilitate parallel computation.

In Table 1, $\text{Conv}^{\text{disc}}$ is implemented with ResNet18 (He et al., 2016) by taking the feature volume from the third block ($1/8$ of the image size) and applying a stride-1 or -2, 3×3 convolution layer depending on the grid size (H, W) (in this work $H = W$ and is either 8 or 4) to obtain $\mathbf{e}_t^{\text{img}}$. Table 2, Table 3, Table 4, Table 5, and Table 6 list other convolutional encoders and decoders that are referred to in Table 1. In these tables, Subconv denotes a sub-pixel convolution (Shi et al., 2016) implemented by a normal convolution layer plus a PyTorch PixelShuffle operation. The stride of Subconv will be used as a parameter for PixelShuffle. $\text{GN}(n)$ denotes group normalization with n groups.

B.3. Hyperparameters

Table 7 lists the hyperparameters for the 2 LAYER dataset. Hyperparameters for other experiments are similar.

Table 1. Network details

Description	Symbol	Architecture
Context prior RNN	$\text{RNN}^{\text{ctx}}_{\text{prior}}$	LSTM(128, 128)
Generate $\mathbf{z}^{\text{ctx}}_t$ from $\mathbf{h}^{\text{ctx}}_t$	$\text{MLP}^{\text{ctx}}_{\text{prior}}$	[128, 128]
Decode $\mathbf{z}^{\text{ctx}}_t$ into $\boldsymbol{\mu}^{\text{bg}}_t$	$\text{Deconv}^{\text{ctx}}$	See Table 3
Context posterior RNN	$\text{RNN}^{\text{ctx}}_{\text{post}}$	LSTM(128, 128)
Infer $\mathbf{z}^{\text{ctx}}_t$ from $[\hat{\mathbf{h}}^{\text{ctx}}, \mathbf{x}_t]$	$\text{MLP}^{\text{ctx}}_{\text{post}}$	[128, 128]
Encode $\mathbf{x}_t$ into $\mathbf{e}^{\text{ctx}}_{\text{enc}}$	Conv^{ctx}	See Table 2
Encode $\mathbf{x}_t$ into $\mathbf{e}^{\text{img}}_t$	$\text{Conv}^{\text{disc}}$	See the text
Encode $\tilde{\mathbf{o}}^k_t$ during discovery	MLP^{cond}	[128, 128]
Infer $\bar{\mathbf{z}}^{ij}_k$ from $[\mathbf{e}^{\text{img},ij}_t, \mathbf{e}^{\text{cond},ij}_t]$	MLP^{disc}	[128, 128]
Prior OS-RNN	$\text{RNN}^{\text{os}}_{\text{prior}}$	LSTM(128, 128)
OS-RNN input embedding	$\text{MLP}^{\text{os}}_{\text{prior}}$	[]
Self-interaction encoding	$\text{MLP}^{\text{self}}_{\text{prior}}$	[128, 128]
Pairwise-interaction encoding	$\text{MLP}^{\text{rel}}_{\text{prior}}$	[128, 128]
Attention weights over object pairs	$\text{MLP}^{\text{weight}}_{\text{prior}}$	[128, 128]
Attention on Environment encoder	$\text{Conv}^{\text{ctx}}_{\text{att}}$	See Table 4
Generate $\tilde{\mathbf{z}}^{\text{state},k}_t$ from $\mathbf{h}^k_t$	$\text{MLP}^{\text{state}}_{\text{prior}}$	[128, 128]
Generate $\tilde{\mathbf{z}}^{\text{att},k}_t$ from $\tilde{\mathbf{z}}^{\text{state},k}_t$	$\text{MLP}^{\text{att}}_{\text{prior}}, \text{MLP}^{\text{gate}}$	[128, 128]
Posterior OS-RNN	$\text{RNN}^{\text{os}}_{\text{post}}$	LSTM(128, 128)
OS-RNN input embedding	$\text{MLP}^{\text{os}}_{\text{post}}$	[]
Predict proposal size $\mathbf{s}^{\text{prop},k}_t$	MLP^{prop}	[128, 128]
Encode proposal into $\mathbf{e}^{\text{prop},k}_t$	$\text{Conv}^{\text{prop}}$	See Table 5
Self-interaction encoding	$\text{MLP}^{\text{self}}_{\text{post}}$	[128, 128]
Pairwise-interaction encoding	$\text{MLP}^{\text{rel}}_{\text{post}}$	[128, 128]
Attention weights over object pairs	$\text{MLP}^{\text{weight}}_{\text{post}}$	[128, 128]
Infer $\tilde{\mathbf{z}}^{\text{state},k}_t$ from $[\hat{\mathbf{h}}^k_t, \mathbf{e}^{\text{prop},k}_t]$	$\text{MLP}^{\text{state}}_{\text{post}}$	[128, 128]
Decode $\mathbf{z}^{\text{what},ij}_t$ into $\mathbf{y}^{\text{att},ij}_t, \boldsymbol{\alpha}^{\text{att},ij}_t$	$\text{Deconv}^{\text{what}}$	See Table 6

Table 2. Conv^{ctx}

Layer	Size/Ch.	Stride	Norm./Act.
Input	3		
Conv 7×7	64	2	GN(4)/CELU
Conv 3×3	128	2	GN(8)/CELU
Conv 3×3	256	2	GN(16)/CELU
Conv 3×3	512	2	GN(32)/CELU
Flatten			
Linear	128		

Table 3. $\text{Deconv}^{\text{ctx}}$

Layer	Size/Ch.	Stride	Norm./Act.
Input	128 (1d)		
Reshape	128 (3d)		
Subconv 3×3	64	2	GN(4)/CELU
Subconv 3×3	32	2	GN(2)/CELU
Subconv 3×3	16	2	GN(1)/CELU
Subconv 3×3	3	2	
Sigmoid			

Table 4. $\text{Conv}^{\text{ctx}}_{\text{att}}$

Layer	Size/Ch.	Stride	Norm./Act.
Input	3		
Conv 3×3	16	2	GN(1)/CELU
Conv 3×3	32	2	GN(2)/CELU
Conv 3×3	64	2	GN(4)/CELU
Conv 3×3	128	2	GN(8)/CELU
Flatten			
Linear	128		

Table 5. $\text{Conv}^{\text{prop}}$

Layer	Size/Ch.	Stride	Norm./Act.
Input	3		
Conv 3×3	16	2	GN(1)/CELU
Conv 3×3	32	2	GN(2)/CELU
Conv 3×3	64	2	GN(4)/CELU
Conv 3×3	128	2	GN(8)/CELU
Flatten			
Linear	128		

Table 6. $\text{Deconv}^{\text{what}}$

Layer	Size/Ch.	Stride	Norm./Act.
Input	128 (1d)		
Reshape	128 (3d)		
Subconv 3×3	64	2	GN(4)/CELU
Subconv 3×3	32	2	GN(2)/CELU
Subconv 3×3	16	2	GN(1)/CELU
Subconv 3×3	3 + 1	2	
Sigmoid			

Table 7. Hyperparameters

Description	Symbol	Value
Image size	$(H_{\text{img}}, W_{\text{img}})$	(64, 64)
Glimpse size	(H_g, W_g)	(16, 16)
Discovery grid size	(H, W)	(4, 4)
Dimension of $\mathbf{z}_t^{\text{pres},k}$		1
Dimension of $\mathbf{z}_t^{\text{depth},k}$		1
Dimension of $\mathbf{z}_t^{\text{where},k}$		4
Dimension of $\mathbf{z}_t^{\text{what},k}$		64
Dimension of $\mathbf{z}_t^{\text{state},k}$		128
Dimension of $\mathbf{z}_t^{\text{ctx}}$		128
Dimension of $\mathbf{e}_t^{\text{img},ij}$		128
Dimension of $\mathbf{e}_t^{\text{cond},ij}$		128
Dimension of $\mathbf{e}_t^{\text{prop},k}$		128
Dimension of $\mathbf{e}_t^{k,k}$		128
Dimension of $\mathbf{e}_t^{k,j}$		128
Dimension of $\mathbf{e}_{\text{enc},t}^{\text{ctx}}$		128
Dimension of $\mathbf{e}_t^{\text{ctx},k}$		128
Training sequence length	T	[2:20:2]
Curriculum milestones		[10k:90k:10k]
#objects to select	K	10
Likelihood variance	σ	0.2
AOE size	s^{ctx}	0.25
Gaussian kernel sigma	σ^{cond}	0.1
Rejection IOU threshold		0.8
Discovery dropout		0.5
Auxiliary KL parameter	p	1×10^{-10}
Gumbel-softmax temperature	τ	1.0
$\bar{\mathbf{z}}_t^{\text{pres},ij}$ prior		$\text{Bern}(1 \times 10^{-10})$
$\bar{\mathbf{z}}_t^{\text{depth},ij}$ prior mean		0
$\bar{\mathbf{z}}_t^{\text{depth},ij}$ prior stdev		1
$\bar{\mathbf{z}}_t^{xy,ij}$ prior mean		0
$\bar{\mathbf{z}}_t^{xy,ij}$ prior stdev		1
$\bar{\mathbf{z}}_t^{hw,ij}$ prior mean		-1.5
$\bar{\mathbf{z}}_t^{hw,ij}$ prior stdev		0.3
$\bar{\mathbf{z}}_t^{\text{what},ij}$ prior mean		0
$\bar{\mathbf{z}}_t^{\text{what},ij}$ prior stdev		1
For updating $\tilde{\mathbf{o}}_t^{\text{depth},k}$	c^{depth}	1
For updating $\tilde{\mathbf{o}}_t^{xy,k}$	c^{xy}	0.1
For updating $\tilde{\mathbf{o}}_t^{hw,k}$	c^{hw}	0.3
For updating $\tilde{\mathbf{o}}_t^{\text{what},k}$	c^{what}	0.2
Minimum proposal size	$s^{\min}$	0.0
Maximum proposal size	$s^{\max}$	0.2

C. Dataset Details

C.1. Bouncing Balls

In all settings, the balls bounce off the walls of the frame, and no new balls are introduced in the middle of an episode. Each episode has a length of 100. We split our data into 10,000 episodes for the training set, and 200 episodes each for the validation set and test set.

In both the OCCLUSION and INTERACTION settings, there are 3 balls each with a color drawn from a set of 5 colors (blue, red, yellow, fuchsia, aqua), but for the OCCLUSION case we do not allow duplicate colors.

C.2. Random Single Ball

In this dataset, a single ball moves down the center of the frame for 9 timesteps. After 5 timesteps, the ball randomly changes direction and moves towards either the bottom left corner or the bottom right corner for the remaining 4 timesteps. We split our data into 10,000 episodes for the training set, and 100 episodes each for the validation set and test set.

C.3. Maze

The mazes are created using the `mazelib` library[1] and then removing dead ends manually. For the first frame, 3 or 4 agents of a random color drawn from 6 colors (red, lime, blue, yellow, cyan, magenta) are randomly placed in the corridors. The agents only move within the corridors and continue in a straight path until it reaches an intersection. It then randomly chooses a path, each with equal probability. Each episode has a sequence length of 99. We split our data into 10,000 episodes for the training set, and 100 episodes each for the validation set and test set.

C.4. 3D Interactions

We generate the 3D Interactions dataset using Blender (Community, 2018), with the same base scene and object properties as the CLEVR dataset (Johnson et al., 2016). In this dataset, we split our dataset into 2920 episodes for training, and 200 episodes for validation and test. Each episode has a length of 100.

We use three different objects (sphere, cylinder, cube), two different materials (rubber, metal), three different sizes, and five different colors (pink, red, blue, green, yellow) to generate the scenes. All objects move on a smooth surface without friction.

To generate the dataset, we randomly put 3 to 5 objects in the camera scene, and launch a sphere into the scene colliding with other objects. The appearance and incident angle of this initial sphere are also randomly selected.

[1] https://github.com/theJollySin/mazelib

D. Experiment Details

For all experiments that require generation, we set $\tilde{\mathbf{z}}_t^{\text{pres},k}$ to 1 for all timesteps at test time to ensure that objects do not disappear. Besides, we turn off discovery after the first timestep. For the bouncing ball experiments, during generation, we directly take the mean of each latent instead of sampling for all models since no stochasticity is involved.

D.1. Bouncing Balls

We draw random sequences of length 20 for training. During testing, for each sequence of length 100, we condition on the first 10 frames and generate the following. We use 5 random seeds to run the experiments per model per dataset. All models were trained till full convergence and the results are computed using the model checkpoints that achieve the best performances on the validation set. For quantitative results, G-SWM is trained for 160000 steps for the INTERACTION, OCCLUSION, and 2 LAYER settings, and 120000 steps for the 2 LAYER-D settings.

D.2. Random Single Ball

We use full sequences of length 9 for training. At test time, each model is provided the first 5 timesteps of the ground truth, before the ball changes direction, and predicts the final 4 timesteps.

D.3. Maze

We use sequences of length 10 for training. During testing, we provide 5 ground truth timesteps as input. For quantitative results, G-SWM, including its variants, are trained for a maximum of 500000 steps.

D.4. 3D Interactions

For this dataset, we use sequences of length 20 for training. However, since most interactions end after 30 steps, we draw training sequences only from the first 30 steps. During testing, for each test sequence of length 100, we provide the first 10 frames as input and generate the following frames.

E. Additional Results

SILOT. We also test SILOT (Crawford & Pineau, 2020) on the four bouncing ball datasets and the results are shown in Figure 1. Being a very similar model to SCALOR, it can handle frequent occlusions and is scalable, but cannot handle the ball collisions well in the INTERACTION, 2 LAYER, and 2 LAYER-D settings, despite having a simple distance-based interaction module.

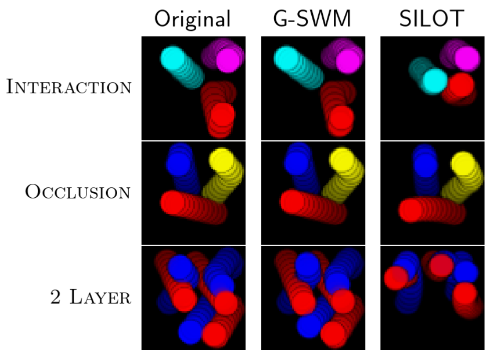

Figure 1. Generated frames of SILOT on the bouncing ball datasets.

Tracking Performance. Table 8 shows the tracking performance for the bouncing ball datasets. For tracking, we report the Multi-Object Tracking Accuracy (MOTA) (Milan et al., 2016), with an IoU threshold of 0.5.

Additional Visualizations. Figure 2 and Figure 3 show visualizations of G-SWM on the two 2 LAYER datasets. Figure 4 and Figure 5 show additional results on the Maze and 3D datasets respectively.

References

Barron, J. T. Continuously differentiable exponential linear units. *arXiv preprint arXiv:1704.07483*, 2017.

Community, B. O. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL `http://www.blender.org`.

Crawford, E. and Pineau, J. Exploiting spatial invariance for scalable unsupervised object tracking. In *AAAI*, pp. 3684–3692. AAAI Press, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Johnson, J. E., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1988–1997, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Milan, A., Leal-Taixé, L., Reid, I. D., Roth, S., and Schindler, K. Mot16: A benchmark for multi-object tracking. *ArXiv*, abs/1603.00831, 2016.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pp. 1310–1318. JMLR.org, 2013.

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1874–1883, 2016.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Table 8. Tracking performance on the bouncing ball datasets.

	G-SWM	SCALOR	STOVE
INTERACTION	0.9870 ± 0.0032	0.9688 ± 0.0101	$\mathbf{0.9979 \pm 0.0005}$
OCCLUSION	$\mathbf{0.9919 \pm 0.0013}$	0.9447 ± 0.0119	0.9618 ± 0.0023
2 LAYER	$\mathbf{0.9967 \pm 0.0041}$	0.9686 ± 0.0102	–
2 LAYER-D	$\mathbf{0.9756 \pm 0.0066}$	0.9501 ± 0.0087	–

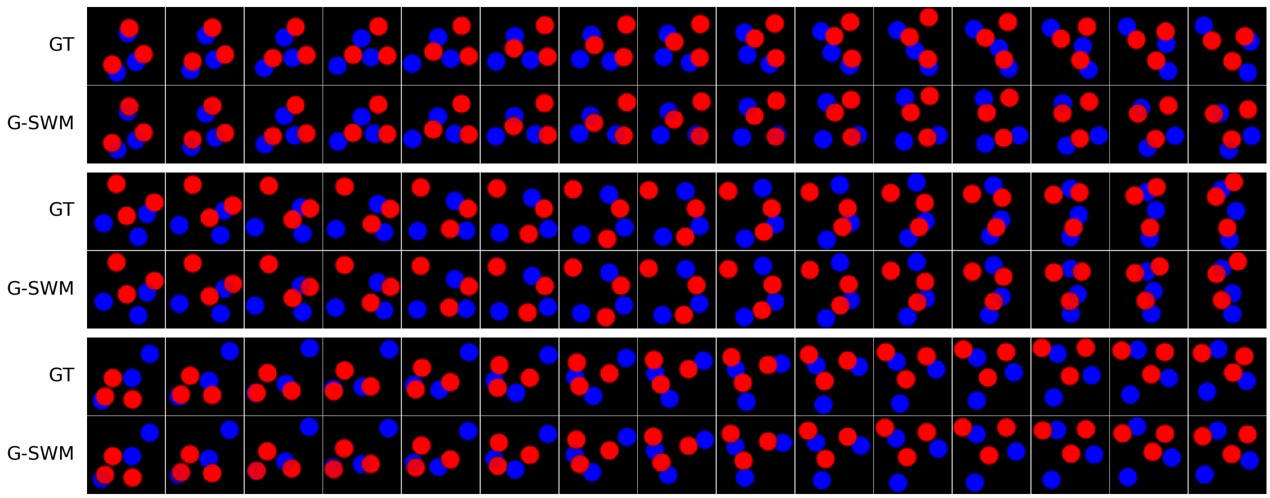

Figure 2. G-SWM on the 2 LAYER dataset

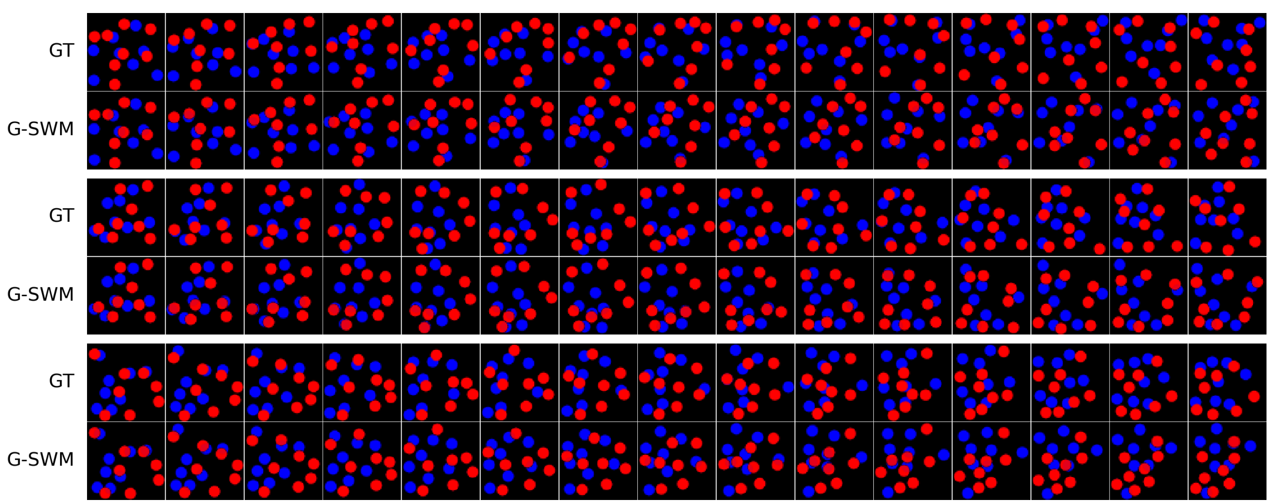

Figure 3. G-SWM results on the 2 LAYER-D DATASET

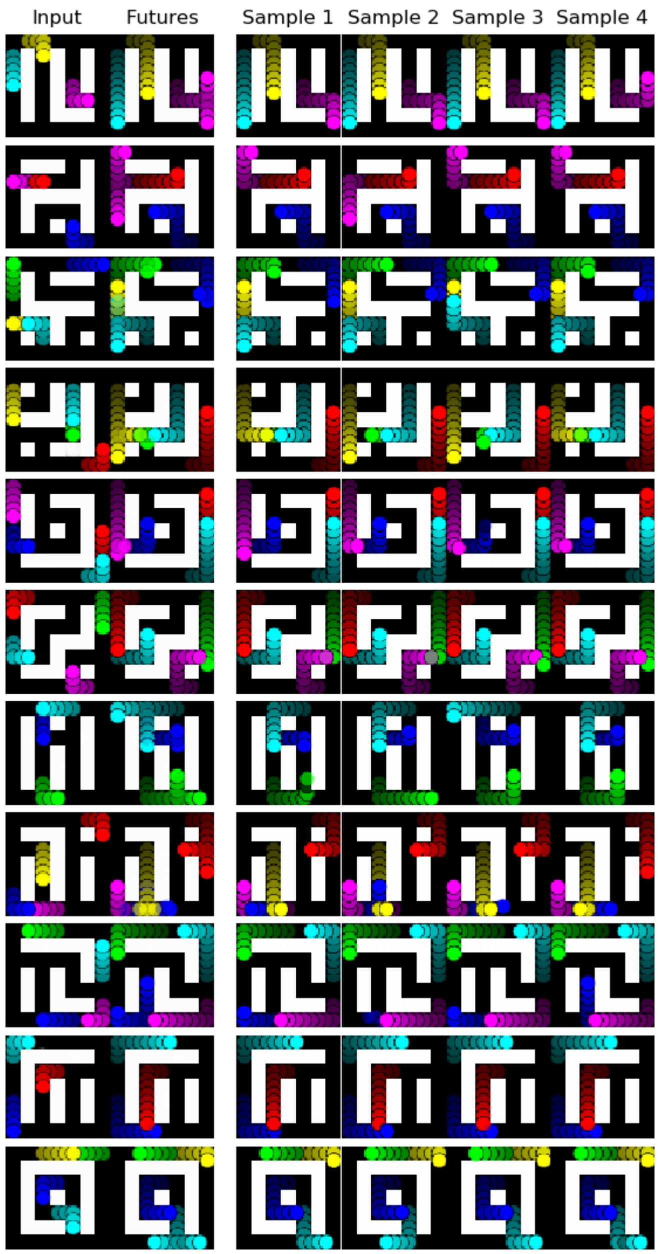

Figure 4. Maze

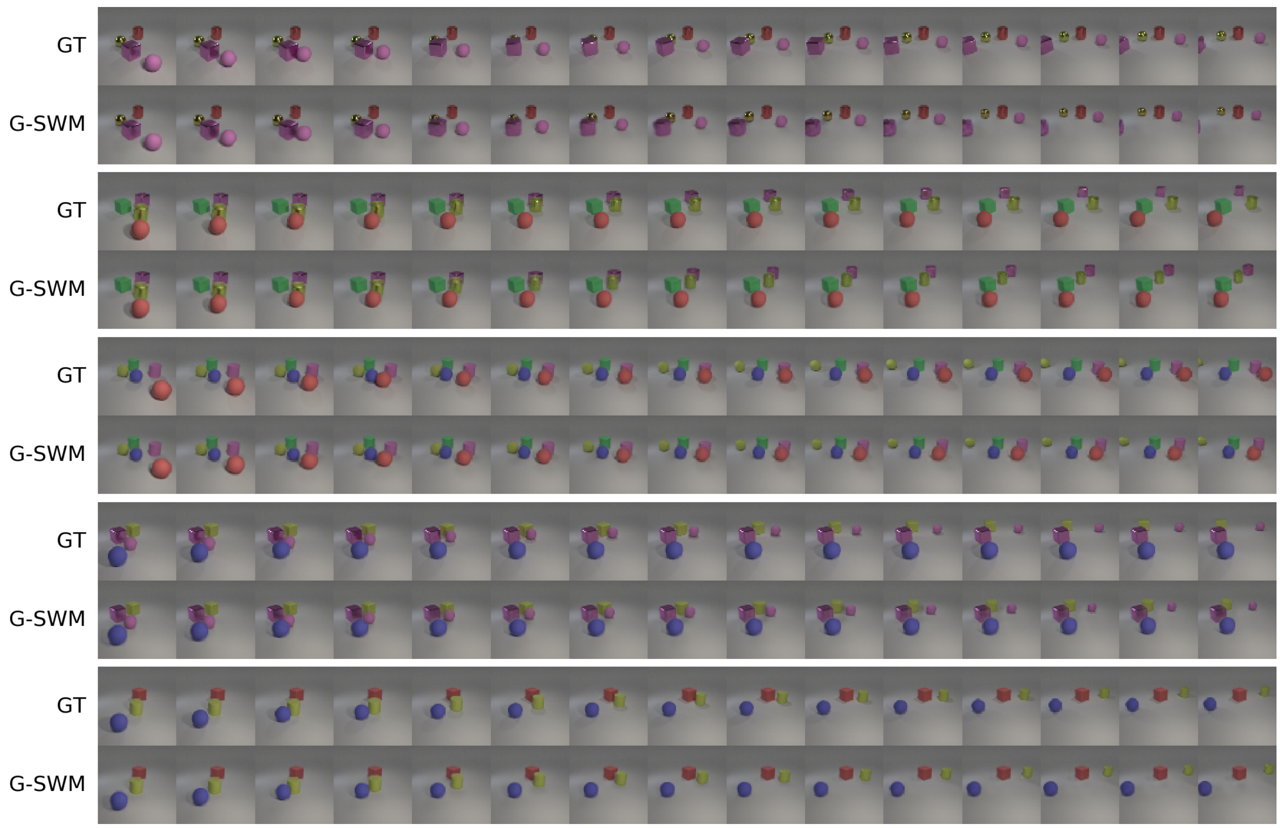

Figure 5. G-SWM results on the 3D-Interactions dataset