

# DRAWER: Digital Reconstruction and Articulation With Environment Realism

## Supplementary Material

### A. Additional Experiment Results

#### A.1. Interactable 3D reconstruction

In this section, we will show more results of our interactable 3D reconstructions. **For more video results, please refer to our website in the supplementary material.**

**DRAWER in more scene types:** DRAWER is not limited to kitchens; instead, it generalizes across various scenes. Fig. 7 demonstrates the performance of DRAWER when applied to office, bedroom, and bathroom.

**Articulation motion simulation:** Here we show more qualitative results on the simulated motions of ours and KlingAI<sup>1</sup> in Fig. 3 and 2D dense pixel trajectories comparison between ours and the GT trajectories in Fig. 4. In Fig. 3 we can observe that KlingAI fails to generate accurate articulation simulations despite using manual segmentation masks and motions as input. DRAWER could generate realistic and correct articulation results based on the underlying 3D geometry. In Fig. 4, we show the comparisons between our predicted articulation motion trajectories and the GT trajectories. The opening speed of our predicted trajectories is aligned with the GT trajectories. The dense blue points are the GT 2D dense pixel trajectories. The dense red points are the predicted 2D dense pixel trajectories with our approach. We can observe that our predicted trajectories match with GT trajectories very well, which proves the accuracy of our articulation estimation.

**Articulation estimation of objects:** Here we show more qualitative results on the estimation of articulation objects. From Fig. 5, we can observe that our approach is more robust and achieves higher accuracy. By grounding predictions in the underlying 3D, we further improve the precision of estimated rotation axes.

**Articulation estimation of complete scenes:** We provide more qualitative comparison results of the full pipeline evaluation. We compare DRAWER with recent methods URDFormer [1] and Digital Cousin [2] on multiple captured scenes. We show the interactable 3D reconstruction in Fig. 1. To avoid clutter in visualization, we randomly select a subset of drawers/cabinets to open. It shows that our approach could accurately maintain exceptionally high visual and geometric fidelity.

<sup>1</sup><https://www.klingai.com/>

#### A.2. Gaming

**Unreal setup details:** We demonstrate the utility of environments created with DRAWER for gaming applications using Unreal Engine (UE) [3]. To build an interactive game environment, we adopt the Luma Unreal Engine Plugin [9] for real-time Gaussian rendering, and use the SDF-extracted mesh for the collision model. For articulated objects, which require setting up the articulation in the game engine, we leverage the physical constraint component in Unreal, which could constrain the linear limits for objects with prismatic joints and angular limits for objects with revolute joints. For the collision model approximation, we adopt convex decomposition for rigid objects and use a higher-resolution collision model based on triangle mesh for the background scene mesh. In the interactive game, the player can shoot yellow balls with the gun and hit the surfaces and objects in the scene. When the player presses some specific keys, they can push or pull the cabinet/drawer doors in the direction they're aiming, based on where their crosshair is pointing.

**Gaming demos:** In Fig. 6, we demonstrate our interactive game in Unreal Engine with game features including shooting rigid objects segmented from the scene and opening cabinet and drawer doors. We show the comparison before and after the interaction with the scene. It could be observed that our interactive game can support multiple physical interaction types and maintaining the rendering quality at the same time.

#### A.3. Real-to-Sim-to-Real

Here we demonstrate more examples of robot learning in a real-to-sim-to-real loop. In Fig. 2, we showcase how the robot learns to open a cabinet door with the revolute joint. It uses the generated data from the simulation environment to train a policy for opening the cabinet door. It then successfully performs the same task in the real world in the zero-shot transfer setting with no extra fine tuning.

### B. Dual Representation Details

#### B.1. SDF field details

In our dual scene representation, we applied the neural signed distance function (SDF) field to capture the fine-grained geometry details in the 3D scene.

Our neural SDF  $f_{\theta}^{\text{SDF}}$  maps a 3D point  $\mathbf{x} \in \mathbb{R}^3$  and a view direction  $\mathbf{d} \in \mathbb{R}^2$  to an RGB radiance  $\mathbf{c} \in \mathbb{R}^3$  and a signed distance to the nearest surface  $s \in \mathbb{R}$ :  $s, \mathbf{c} = f_{\theta}^{\text{SDF}}(\mathbf{x}, \mathbf{d})$ . Inside  $f_{\theta}^{\text{SDF}}$ , the signed distance  $s$  only depends on 3D point



Figure 1. **More Qualitative Results on Interactable 3D Reconstruction:** We compare DRAWER’s reconstruction with [1, 2] which uses multiple images as input and selects the best output. DRAWER achieves realistic, well-aligned results, while [1, 2] show misalignment and lack realism. The left-most input images are reference images of the kitchens. To compare the performance of each method *in a fully automated pipeline*, we use the automatically generated bounding boxes in URDFormer [1] for comparison. To avoid clutter in visualization, we randomly select a subset of drawers/cabinets to open.



Figure 2. **More Results on Real-to-Sim-to-Real.** Here we showcase the robot opens the cabinet door with the revolute joint in a zero-shot transfer setting. DRAWER allows us to learn training robotic controllers using a real-to-sim-to-real loop. The inset images indicate the simulated data generation process.

$\mathbf{x}$ , and the RGB radiance  $\mathbf{c}$  depends on 3D point  $\mathbf{x} \in \mathbb{R}^3$  and view direction  $\mathbf{d} \in \mathbb{R}^2$ .

Following BakedSDF [20], we model the volume density  $\sigma$  as  $\sigma(\mathbf{x}) = \alpha \Psi_{\beta}(s)$ , where  $\Psi_{\beta}$  is a cumulative distribu-





Original Closed Cabinet

Ours - Half Open

Ours - Fully Open

KlingAI - Half Open

KlingAI - Fully Open

Figure 3. **More Qualitative Results on Articulation Simulation:** Here we show more results of the comparisons between our approach and KlingAI on the articulation simulation. Our method achieves realistic, accurate articulation, while KlingAI fails despite using manual segmentation masks and motions.



Figure 4. **Qualitative Results on Articulation Motion Simulation:** Here we show the comparisons between our predicted articulation motion trajectories and the GT trajectories. The opening speed of our predicted trajectories is aligned with the GT trajectories. The dense blue points are the GT 2D dense pixel trajectories and the dense red ones are predicted by DRAWER. We can observe that our predicted trajectories match with GT trajectories very well, which proves the accuracy of our articulation estimation.

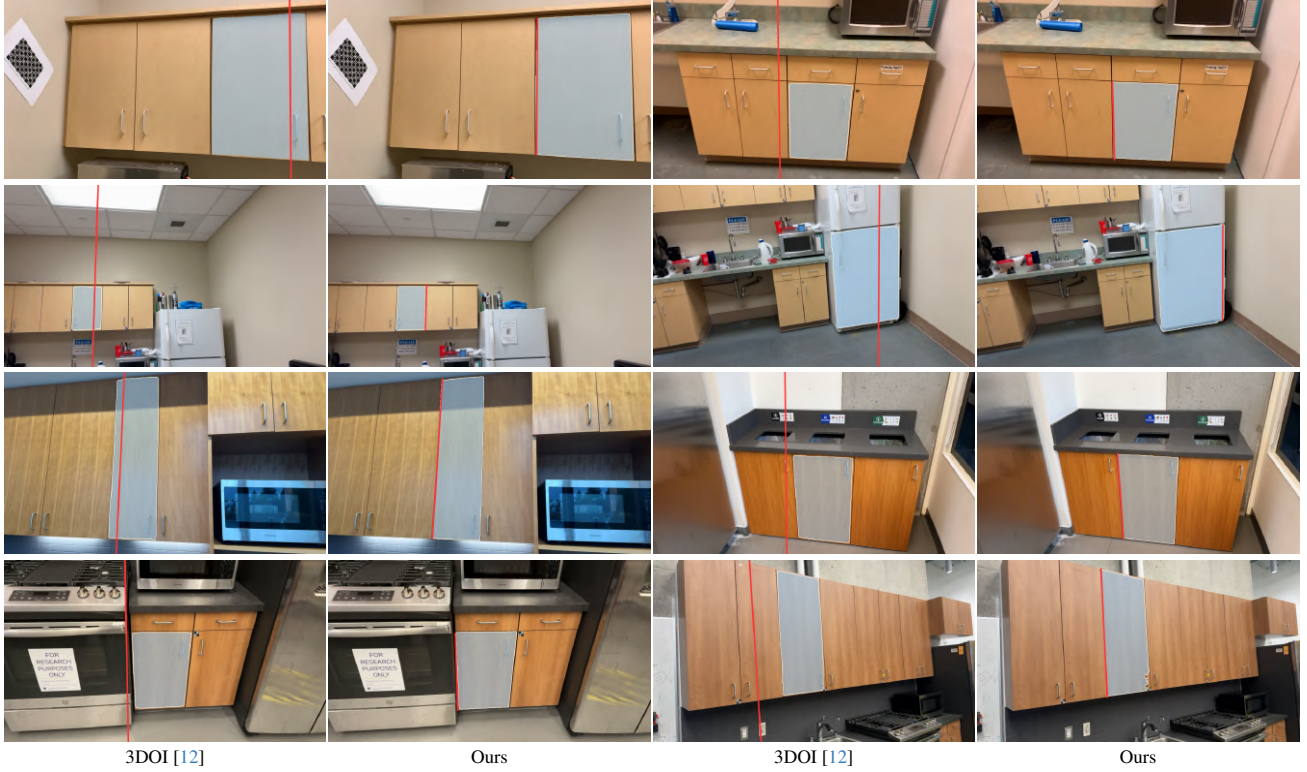


Figure 5. **More Results on Articulation Estimation:** We visualize the estimated revolute axes and articulated object masks produced by 3DOF [12] and DRAWER, demonstrating that DRAWER achieves more precise articulation estimation due to its underlying 3D geometry.

tion function of a zero-mean Laplace distribution with an annealing  $\beta$  in training. With volume rendering [10], our SDF field can render the color  $\mathbf{C}$  of a single pixel along a ray  $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$  with ray origin  $\mathbf{o}$  and direction  $\mathbf{d}$  based on a series of points  $\{\mathbf{t}_i\}$ , densities  $\{\sigma_i\}$  and colors  $\{\mathbf{c}_i\}$ :  $\mathbf{C}(\mathbf{r}) = \sum_i \mathbf{T}_i \alpha_i \mathbf{c}_i$ , where  $\mathbf{T}_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$ ,  $\alpha_i = (1 - \exp(-\sigma_i \delta_i))$  and  $\delta_i = \mathbf{t}_i - \mathbf{t}_{i-1}$ .

We minimize a loss  $\mathcal{L}_{\text{rgb}} = \sum_{\mathbf{r}} \|\mathbf{C}_{\text{gt}}(\mathbf{r}) - \mathbf{C}(\mathbf{r})\|_2^2$  between the ground truth color  $\mathbf{C}_{\text{gt}}$  and rendering color  $\mathbf{C}$ . We also apply Eikonal loss [5]  $\mathcal{L}_{\text{eik}} = (\|\nabla_{\mathbf{x}} s\| - 1)^2$  and appearance decomposition in Ref-NeRF [17] as well for better geometry regularization.

Based on the modeling volume density, we could not only render color  $\mathbf{C}$  from volume rendering, but also render the normal  $\mathbf{N}(\mathbf{r}) = \sum_i \mathbf{T}_i \alpha_i \mathbf{n}_i$  and the depth  $\mathbf{D}(\mathbf{r}) = \sum_i \mathbf{T}_i \alpha_i \mathbf{t}_i$  as well, where  $\mathbf{n}_i = \nabla_{\mathbf{x}} s / \|\nabla_{\mathbf{x}} s\|$ . Inspired by MonoSDF [22], we also use the off-the-shelf model [4] to predict 2D monocular normal and depth priors, which guide our rendering normal  $\mathbf{N}(\mathbf{r})$  and  $\mathbf{D}(\mathbf{r})$  to help the SDF Field learn better geometry with losses  $\mathcal{L}_{\text{normal}} = \|\mathbf{N}(\mathbf{r}) - \mathbf{N}_{\text{mono}}(\mathbf{r})\|_2^2$  and  $\mathcal{L}_{\text{depth}} = \sum_{\mathbf{r}} \|(a\mathbf{D}(\mathbf{r}) + b) - \mathbf{D}_{\text{mono}}(\mathbf{r})\|_2^2$ , where  $a$  and  $b$  are the scale and shift that aligns the two distribution [13].

We jointly optimize the SDF Field parameters by minimizing the following loss:  $\mathcal{L}_{\text{total}}^{\text{SDF}} = \mathcal{L}_{\text{rgb}} + \lambda_{\text{normal}} \mathcal{L}_{\text{normal}} + \lambda_{\text{depth}} \mathcal{L}_{\text{depth}} + \lambda_{\text{eik}} \mathcal{L}_{\text{eik}}$ .

We implement our SDF field based on SDFStudio [21] with the multi-resolution grid [11], which speeds up the training than the original BakedSDF MLP-based implementation. We train our SDF for 250k iterations for 4 hours in an A6000 GPU.

## B.2. Gaussian splats details

We propose anchoring the Gaussians around the zero level-set of the neural SDF field. Since repeatedly querying the learned SDF is computationally expensive, we instead extract a high-resolution mesh from the SDF and anchor Gaussians to it. Fig. 8 provides a high-level overview to the dual scene representation.

For every GS point  $\mathbf{p}_i$ , we need to parameterize its mean  $\boldsymbol{\mu}_i$ , scale  $\mathbf{S}_i$ , orientation  $\mathbf{R}_i$ , opacity  $\alpha_i$ , and colors  $\mathbf{c}_i$  encoded using spherical harmonics. For each point  $\mathbf{p}$ , we define the Gaussian as  $G(\mathbf{p}) = e^{-\frac{1}{2}(\mathbf{p}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{p}-\boldsymbol{\mu})}$ , which is multiplied by opacity  $\alpha$  in blending process. We define the covariance as  $\boldsymbol{\Sigma} = \mathbf{R} \mathbf{S} \mathbf{S}^T \mathbf{R}^T$ . Finally, the rendering  $C = \sum_{i=1}^N c_i \alpha'_i \prod_{j=1}^{i-1} (1 - \alpha'_j)$ , where  $\alpha'_j$  is the accumulated opacity.



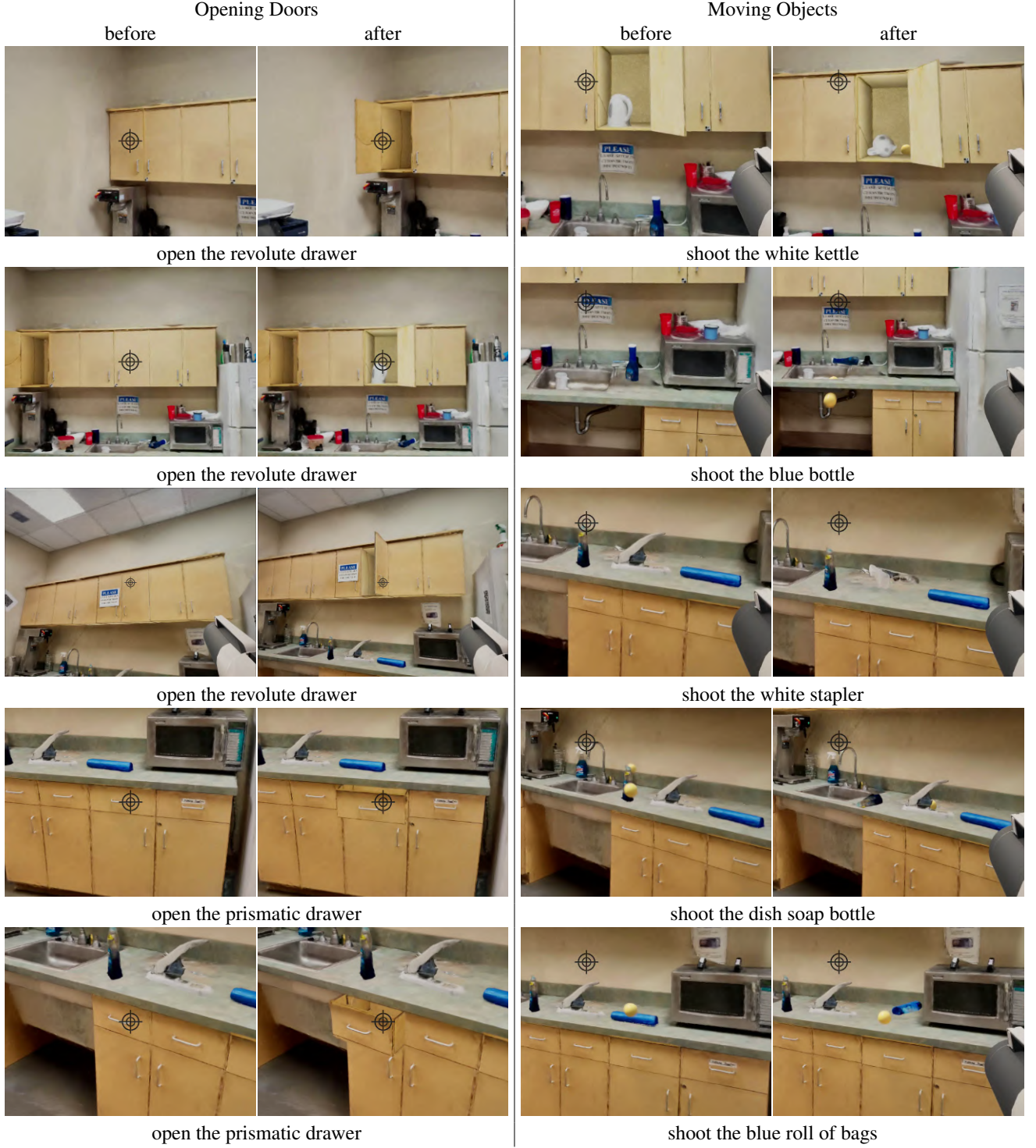


Figure 6. **DRAWER in Unreal Engine.** We demonstrate our interactive game in Unreal Engine with game features including shooting rigid objects segmented from the scene and opening cabinet and drawer doors. We zoom in on some images for better visualization. Here we show the comparison before and after the interaction with the scene. The player can shoot yellow balls with the gun. When the player presses a key, they can push or pull the cabinet/drawer doors in the direction they're aiming, based on where their crosshair is pointing.

To establish a local coordinate system for each mesh face triangle, we first map each GS point  $\mathbf{p}_i$  to its corresponding

face  $\mathbf{f}_k$  via:

$$\mathbf{h} : \mathbf{p}_i \mapsto \mathbf{f}_k$$



Figure 7. **Results of DRAWER on more environments.** DRAWER can generalize across various scenes, including the office, bedroom, and bathroom.

Each face  $\mathbf{f}_k$  is defined by its vertices  $\mathbf{v}_{k,0}, \mathbf{v}_{k,1}, \mathbf{v}_{k,2}$ . We compute the normalized edge direction vectors:

$$\begin{aligned} \mathbf{e}_{k,0} &= \text{normalized}(\mathbf{v}_{k,1} - \mathbf{v}_{k,0}) \\ \mathbf{e}_{k,1} &= \text{normalized}(\mathbf{v}_{k,2} - \mathbf{v}_{k,1}) \\ \mathbf{e}_{k,2} &= \text{normalized}(\mathbf{v}_{k,0} - \mathbf{v}_{k,2}) \end{aligned}$$

The face normal  $\mathbf{n}_k$  is:

$$\mathbf{n}_k = \mathbf{e}_{k,0} \times \mathbf{e}_{k,1}$$

The local coordinate axes are:

$$\mathbf{x}_k = \mathbf{e}_{k,0}, \quad \mathbf{y}_k = \mathbf{n}_k \times \mathbf{e}_{k,0}, \quad \mathbf{z}_k = \mathbf{n}_k$$

This local frame  $(\mathbf{x}_k, \mathbf{y}_k, \mathbf{z}_k)$  ensures a consistent reference for operations like mapping points and defining orientations within the mesh.

In our dual scene representation, we parameterize the Gaussian splats with the following parameters for each GS point:  $x_i, y_i, z_i$  for  $\mu_i$ ,  $s_{i,x}, s_{i,y}, s_{i,z}$  for  $\mathbf{S}_i$ , and  $\alpha_i, \phi_i, \theta_i$  for  $\mathbf{R}_i$ . We maintain the parameterization of opacity  $\alpha_i$  and colors radiance  $\mathbf{c}_i$  unchanged. Next, we will elaborate details on this.

**Parameterization of means:** To parameterize the mean positions  $\mu_i$  of the Gaussian splats, we anchor each mean  $\mu_i$  to its corresponding mesh face  $\mathbf{f}_k$  by expressing it in the face’s local coordinate system. Specifically, we decompose  $\mu_i$  into components along the face’s local axes:

$$\mu_i = x_i \mathbf{x}_k + y_i \mathbf{y}_k + z_i \mathbf{z}_k,$$

where  $\mathbf{x}_k, \mathbf{y}_k$ , and  $\mathbf{z}_k$  are the local coordinate axes of the face  $\mathbf{f}_k$ .  $\mathbf{x}_k$  and  $\mathbf{y}_k$  lie in the plane of the face.  $\mathbf{z}_k$  is the normal vector perpendicular to the face.  $x_i, y_i$ , and  $z_i$  are scalar parameters representing the projection lengths of  $\mu_i$  along the respective axes.

To ensure that the projection of the mean vector  $\mu_i$  lies within the face triangle  $\mathbf{f}_k$ , we begin by calculating its barycentric coordinates. The projection is given by  $x_i \mathbf{x}_k + y_i \mathbf{y}_k$ , where  $x_i$  and  $y_i$  are unconstrained coordinates, and  $\mathbf{x}_k$  and  $\mathbf{y}_k$  are basis vectors associated with the triangle. Using the triangle’s vertex positions  $\mathbf{v}_{k,0}, \mathbf{v}_{k,1}$ , and  $\mathbf{v}_{k,2}$ , we compute the barycentric coordinates  $\lambda_{i,0}, \lambda_{i,1}$ , and  $\lambda_{i,2}$  of the projected mean  $\mu_i$ . However, since  $x_i$  and  $y_i$  are not constrained, these barycentric coordinates may not fall within the interval  $(0, 1)$ , meaning the projection could lie outside the triangle. To map any such point back onto the triangle (specifically, onto its edges), we adjust the barycentric coordinates by clipping each  $\lambda_{i,m}$  to the range  $[0, 1]$  and then normalizing them so they sum to 1:

$$\lambda_{i,m}^{\text{reg}} = \frac{f_{\text{clip}}(\lambda_{i,m}, 0, 1)}{\sum_{m=0}^2 f_{\text{clip}}(\lambda_{i,m}, 0, 1)}$$

Where  $f_{\text{clip}}(x, a, b)$  limits the value of  $x$  to be within the range of  $[a, b]$ . With these regularized barycentric coordinates  $\lambda_{i,m}^{\text{reg}}$ , we can compute the constrained coordinates  $x_i^{\text{reg}}$  and  $y_i^{\text{reg}}$ . This ensures that the adjusted projection of  $\mu_i$  now lies within the face triangle  $\mathbf{f}_k$ .

To constrain the  $z$ -direction component of the mean vector  $\mu_i$ , we apply a clipping operation to  $z_i$  based on the scaled radius  $e_\alpha r_k$ :

$$z_i^{\text{reg}} = f_{\text{clip}}(z_i, -e_\alpha r_k, e_\alpha r_k)$$

In these expressions:  $e_\alpha$  is a predefined constant.  $r_k$  is the inradii of face triangle  $\mathbf{f}_k$ . Next, we compute the regularized mean vector  $\mu_i^{\text{reg}}$  by combining the constrained components:

$$\text{clip}(\mu_i) = \mu_i^{\text{reg}} = x_i^{\text{reg}} \mathbf{x}_k + y_i^{\text{reg}} \mathbf{y}_k + z_i^{\text{reg}} \mathbf{z}_k$$

However, the clipping function inherently stops gradients during back-propagation, preventing gradients from flowing back to the underlying parameters. To circumvent this issue and ensure proper gradient propagation, we utilize the stop-gradient operator  $\text{sg}(\cdot)$ , which halts gradients during the forward pass. We define an adjusted mean vector  $\mu_i^o$  using *straight-through estimator* as follows:

$$\mu_i^o = \text{sg}(\text{clip}(\mu_i)) + \mu_i - \text{sg}(\mu_i)$$

This formulation ensures that the forward pass uses the regularized mean  $\text{clip}(\mu_i)$ , while the gradients during back-propagation are computed with respect to the original mean



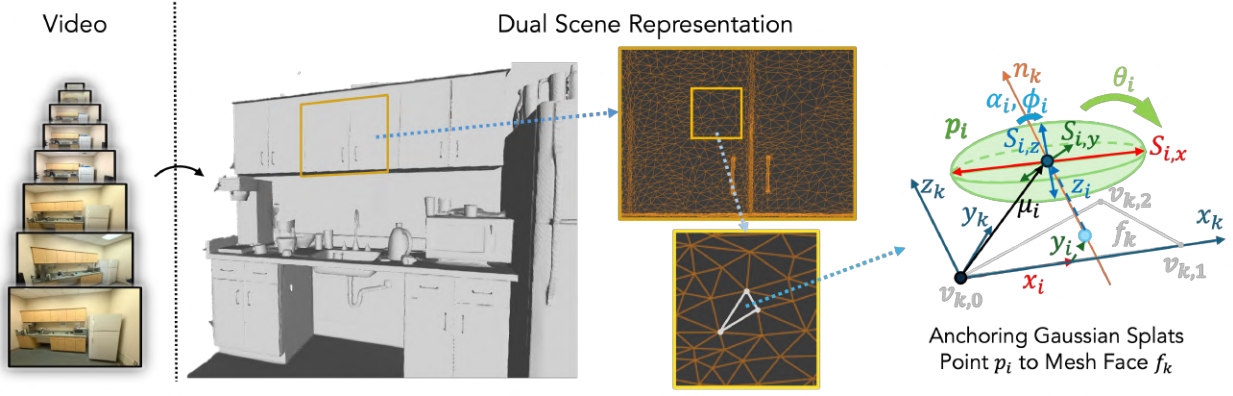


Figure 8. **Illustration of our dual scene representation.** Our dual scene representation is based on SDF reconstruction and Gaussian Splatting. We anchor gaussian splats on each triangle on the reconstructed mesh extracted from SDF.



Figure 9. **Failure cases of DRAWER.** DRAWER might occasionally fail due to catastrophic errors in foundation models. For example, when 3DO1 predicts the wrong axes and GPT-4o fails to recognize the correct face of cabinet doors.

$\mu_i$ . Finally, we use  $\mu_i^o$  in the GS rendering process to maintain the desired constraints without impeding gradient flow.

**Parameterization of scales:** To parameterize the scales, we start by defining  $\mathbf{S}_i = (S_{i,x}, S_{i,y}, S_{i,z})$ , where each component is an exponential function of learnable parameters  $s_{i,x}, s_{i,y}, s_{i,z}$ :

$$S_{i,\cdot} = \exp(s_{i,\cdot})$$

Here,  $s_{i,x}, s_{i,y}, s_{i,z}$  are the parameters that the model learns during training. To ensure that these scales remain within desired limits, we apply a clipping operation. For the  $x$  and  $y$  components, the scales are clipped as follows:

$$S_{i,\cdot}^{\text{reg}} = f_{\text{clip}}(S_{i,\cdot}, 0, s_{\alpha} r_k), \quad \text{for } S_{i,x}, S_{i,y}$$

For the  $z$  component, we introduce an additional scaling factor  $s_{\beta}$  in the clipping operation:

$$S_{i,z}^{\text{reg}} = f_{\text{clip}}(S_{i,z}, 0, s_{\alpha} s_{\beta} r_k)$$

In these expressions:  $s_{\alpha}$  and  $s_{\beta}$  are predefined scaling constants.  $r_k$  is the inradii of face triangle  $\mathbf{f}_k$ . We define  $\mathbf{S}_i^{\text{reg}}$

as:

$$\text{clip}(\mathbf{S}_i) = \mathbf{S}_i^{\text{reg}} = (S_{i,x}^{\text{reg}}, S_{i,y}^{\text{reg}}, S_{i,z}^{\text{reg}})$$

With *straight-through estimator*, we ensure proper gradient propagation by:

$$\mathbf{S}_i^o = \text{sg}(\text{clip}(\mathbf{S}_i)) + \mathbf{S}_i - \text{sg}(\mathbf{S}_i)$$

**Parameterization of orientations:** To parameterize the orientations while allowing limited rotation around the face normal direction  $\mathbf{n}_k$ , we define the rotation matrix  $\mathbf{R}_i$  as follows:

$$\mathbf{R}_i = \text{stack}(\mathbf{x}_k, \mathbf{y}_k, \mathbf{z}_k) \times \text{rot\_axis\_angle}(\mathbf{v}_i^{\text{rot}}, \phi_i) \times \text{rot}_z(\theta_i)$$

where

$$\mathbf{v}_i^{\text{rot}} = \begin{pmatrix} \sin(\alpha_i) \\ \cos(\alpha_i) \\ 0 \end{pmatrix}$$

Here,  $\alpha_i, \phi_i$ , and  $\theta_i$  are learnable parameters that control the orientation.  $\text{stack}(\mathbf{x}_k, \mathbf{y}_k, \mathbf{z}_k)$  constructs a matrix from the local coordinate axes of face triangle  $\mathbf{f}_k$ .

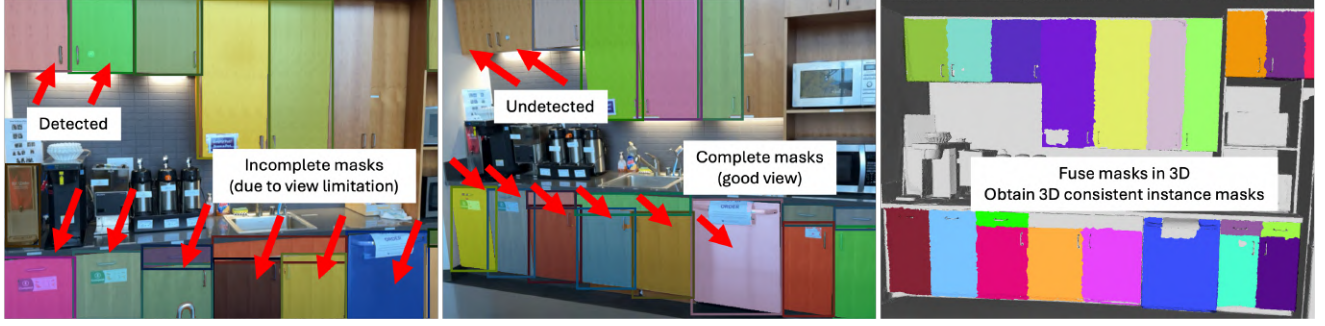


Figure 10. **Robust perception via 3D fusion.** We show the robustness of our perception here. Grounded SAM can not guarantee the correct detection of the complete mask for every door in every frame in the video. However, with the underlying 3D geometry, we could fuse masks in 3D and obtain 3d consistent instance mask.

$\text{rot\_axis\_angle}(\mathbf{v}_i^{\text{rot}}, \phi)$  represents the rotation matrix for rotating around axis  $\mathbf{v}_i^{\text{rot}}$  by angle  $\phi$ .  $\text{rot}_z(\theta_i)$  is the rotation matrix for rotating around the  $z$ -axis by angle  $\theta_i$ .

We will explain the rotation components then. Firstly, we start with an initial alignment with face triangle with the matrix stack  $(\mathbf{x}_k, \mathbf{y}_k, \mathbf{z}_k)$ . It aligns the coordinate system with the local axes of the face triangle  $\mathbf{f}_k$ . This sets the foundation for further rotations.

Secondly, we rotate around an axis in the  $xy$ -plane. We define an axis  $\mathbf{v}_i^{\text{rot}}$  in the  $xy$ -plane using  $\alpha_i$ . We rotate around this axis by angle  $\phi_i$  using  $\text{rot\_axis\_angle}(\mathbf{v}_i, \phi_i)$ . This allows for limited tilting away from the face normal  $\mathbf{n}_k$ .

Finally, we rotate around the  $z$ -axis. We apply  $\text{rot}_z(\theta_i)$  to capture any rotation within the plane perpendicular to the  $z$ -axis.

To limit the tilting (leaning) of the orientation away from the  $z$ -axis, we constrain the angle  $\phi_i$ :

$$\text{clip}(\phi_i) = \phi_i^{\text{reg}} = f_{\text{clip}}(\phi_i, 0, \phi^{\text{max}})$$

This ensures that  $\phi_i$  stays within the range  $[0, \phi^{\text{max}}]$ , preventing excessive tilting.

With *straight-through estimator*, we ensure proper gradient propagation by:

$$\phi_i^o = \text{sg}(\text{clip}(\phi_i) + \phi_i - \text{sg}(\phi_i))$$

**Adaptive density control:** In the process of adaptive density control, which includes splitting, duplicating, and culling of Gaussian points, we adjust the mapping of GS points to mesh faces. Each GS point  $\mathbf{p}_i$  is associated with a mesh face  $\mathbf{f}_k$  through the mapping function

$$\mathbf{h} : \mathbf{p}_i \mapsto \mathbf{f}_k.$$

By maintaining this mapping relationship, we ensure that even when multiple GS points are associated with a single mesh face, they remain well-integrated within the mesh structure.

When duplicating GS points during adaptive density control, we copy all the parameters from the original GS point to the duplicates. This straightforward replication ensures consistency and maintains the spatial and geometric properties of the points.

Splitting GS points requires more careful handling to ensure that the new points are properly positioned and scaled. The steps involved are:

1) We copy rotation parameters and  $z$ -component directly: we directly copy the rotation parameters  $\alpha_i, \phi_i, \theta_i$ , and the  $z$ -component  $z_i$  from the original GS point to the new split points.

2) We shrink the scales of the GS points by a factor of  $s^{\text{srk}} = 1.6$ . This is done by adjusting the exponentiated scales:

$$\exp(s_{i,\cdot}^{\text{new}}) = \frac{1}{s^{\text{srk}}} \times \exp(s_{i,\cdot}).$$

3) We determine new means for split points. We sample the new mean positions  $\mu_i^{\text{new}}$  for the split GS points within a radius  $r_i^{\text{max}}$ . The radius  $r_i^{\text{max}}$  is calculated as the minimum of:

1. the smallest of the original  $x$ - and  $y$ -scale components  $\min(S_{i,x}, S_{i,y})$
2. the minimum distance from the projected point  $x_i \mathbf{x}_k + y_i \mathbf{y}_k$  to the edges of the face triangle  $\mathbf{f}_k$ .

This ensures that the new means are not only within the original GS point's scale but also confined within the face triangle they belong to. By sampling within this radius, we



maintain spatial coherence and prevent the new GS points from overlapping or extending beyond the boundaries of  $\mathbf{f}_k$ .

**Training details:** For training loss: in each iteration, we render colors  $\mathbf{C}$ , accumulations  $\mathbf{A}$ , and depths  $\mathbf{D}$  for each image frame. Given GT colors  $\mathbf{C}_{\text{gt}}$  and mesh projected depth  $\mathbf{D}_{\text{mesh}}$ , we have:  $\mathcal{L}_{\text{rgb}} = \|\mathbf{C}_{\text{gt}} - \mathbf{C}\|_2^2$ ,  $\mathcal{L}_{\text{accu}} = \|\mathbf{A} - 1\|_2^2$ , and  $\mathcal{L}_{\text{depth}} = \|\mathbf{D}_{\text{mesh}} - \mathbf{D}\|_2^2$ . Totally, the loss is  $\mathcal{L}_{\text{total}}^{GS} = \mathcal{L}_{\text{rgb}} + \lambda_{\text{accu}}\mathcal{L}_{\text{accu}} + \lambda_{\text{depth}}\mathcal{L}_{\text{depth}}$ . We train our GS for 30k iterations following [6]. Our implementation is based on NeRFStudio [15].

## C. Articulation Details

### C.1. Scene decomposition details

**Articulated object decomposition:** Given a set of posed images, we first adopt Grounded SAM [7, 8, 14] to segment all objects of interest across all frames. Here, Grounded SAM takes natural language as input and outputs corresponding image masks on each input image frame. We select some related words including *drawer door*, *cabinet door*, *fridge door* etc. as input.

Due to viewpoint variations and specular reflections, objects are not always fully visible, resulting in significant variations in mask quality. To address this issue and ensure reliable mask estimations across frames, we employ a multi-step approach involving mask projection, fusion, and filtering.

For each input image, we use Grounded SAM to generate several related masks. These masks are then projected onto the 3D scene mesh obtained in SDF field reconstruction, resulting in segmented 3D partial meshes corresponding to each mask. Within each partial 3D mesh, we extract the indices of the faces covered by the projected mask.

We construct a graph representing the entire mesh, where nodes represent the faces of the mesh, edges connect adjacent faces, and edge weights are determined by the frequency with which two connected faces appear together in the same segmented 3D partial mesh across all masks. This frequency reflects how often the two faces are co-projected from different masks.

Using the Louvain algorithm [16, 23], we detect communities (clusters) at various clustering resolutions. Each cluster corresponds to a group of faces that frequently appear together in the projected masks, indicating they belong to the same object or surface region. Finally, we obtain the corresponding 3D partial mesh by aggregating the faces within each cluster.

For each obtained 3D partial mesh cluster, we project it back into every image frame to generate a 2D mask. These projected masks are compared with the original masks generated

by Grounded SAM using the Intersection over Union (IoU) metric. If no Grounded SAM mask in any image frame has an IoU exceeding a predefined threshold with the projected mask, the corresponding 3D partial mesh cluster is discarded. This ensures that only clusters corresponding to reliable and consistent object masks across frames are retained.

Leveraging the advanced visual grounding capabilities of Vision-Language Models (VLMs) [19], we employ GPT4o to assess the quality of the remaining masks. The evaluation criteria include: 1) Object Type: Ensuring the mask corresponds to the desired object type. 2) Masked Region Completeness: Ensuring the object within the mask is complete and not partially occluded. 3) Singularity of Object: Confirming that the mask covers a single object rather than multiple instances (e.g., avoiding masks that cover multiple doors when only one is desired). Masks that do not meet these criteria are discarded, enhancing the overall quality.

Filtered masks are projected onto the 3D scene mesh, and IoU values between different 3D segmented meshes are computed. Duplicated segmented articulated objects (i.e., multiple meshes representing the same object) are identified through high IoU values and removed, ensuring that each object is uniquely represented.

For each remaining segmented mesh, we project it into each image frame to obtain a projected mask. From this mask, we derive point prompts indicating key positions on the object. Using these point prompts, the Segment Anything Model (SAM) [7] is employed to re-segment the objects in the images. This step refines the segmentation, leveraging SAM’s capabilities to produce more precise and consistent masks across frames.

This process yields, for each object  $i$ , its high-quality 2D masks  $\mathbf{M}_{i,j}$  in each image  $j$ , and its partial 3D geometry in either mesh form  $\mathcal{M}_i^{\text{obj}} = (\mathbf{V}_i, \mathbf{F}_i)$  or SDF.

**Rigid object decomposition:** For rigid object decomposition, we follow a method similar to that described in [18]. Firstly, we begin by locating the rigid object in the scene either by Grounded SAM from text prompts including *Bottle*, *Bowl*, *Cup* etc. or through user clicks. Once the object is located in one view, we propagate the masks across views to ensure consistent segmentation of the object throughout different perspectives. We then extract the corresponding 3D bounding box of the rigid object. Finally, we could adjust the underlying SDF field according to the bounding box location. This involves modifying the SDF values within the bounding box to isolate the object from the rest of the scene. We can then extract the rigid object with marching cubes in the SDF field. By adjusting the SDF field instead of direct mesh segmentation, the hole below the

object is automatically filled.

## C.2. Physical reasoning details

Once we identify all interactable objects, the next step is to estimate their physics-related attributes to effectively model and simulate their physical dynamics. Specifically, we aim to determine the articulation types and axes of articulated objects. To achieve this, we adopt a two-pronged approach. The first prong leverages a specialized vision foundation model 3DOI [12] to predict object hinges and affordances. This model excels in identifying articulation types and affordance maps but may produce less accurate object masks and revolute axes. To enhance the overall accuracy, we integrate the predictions of 3DOI with our reconstructed 3D geometry and high-quality, filtered articulated object masks.

Here we elaborate on the details of how we integrate the predictions of 3DOI [12] with our reconstructed 3D geometry. We first use the filtered articulated object masks obtained in Section C.1 and sample a point at the center of each mask. This point serves as input to the 3DOI model. The 3DOI model processes the input point and predicts the articulated object mask, the articulation type (e.g., revolute or prismatic joint), the revolute axis, if applicable, and the affordance map, indicating where forces are likely to be applied. For more details on the 3DOI model, please refer to [12].

3DOI provides robust predictions for articulation types and affordance maps but may inaccurately predict object masks and revolute axes. To leverage the model’s strengths, we combine its predictions with our accurate 3D geometry and filtered object masks. We focus on the affordance map from the 3DOI predictions, which is a 2D matrix matching the input image’s dimensions. This map assigns higher values to pixels where force application is more probable. We assume the force application point as the pixel within our articulated object mask that holds the maximum value in the affordance map. This pixel represents the position where we are most likely to apply force to interact with the object.

We start with computing the minimum 3D oriented bounding box for the articulated object’s geometry. For objects like doors, which are thin in one dimension, the bounding box reflects this by being thin along one axis. We then designate the thin direction as the z-axis. The other two dimensions become the x-axis and y-axis. We then calculate the position of the force application point within the object’s local coordinate system. The coordinates  $(x, y)$  are normalized to the range  $[0, 1]$ , where smaller values of  $x$  and  $y$  indicate positions closer to the left and top edges of the object, respectively.

Next, we perform the articulation type inference based on the coordinate. If the position of the point is at the left ( $x < 0.25$ )

or right ( $x > 0.75$ ) of the door, the object is likely to have a revolute joint that opens from left to right or right to left. If the force point is near the center ( $0.25 \leq x \leq 0.75$ ), the object could have either a revolute or prismatic joint. In this situation, if the point is not near the top or bottom of the drawer door ( $0.25 \leq y \leq 0.75$ ), it’s probably a prismatic joint (e.g., a drawer sliding out). If the point is located at center top ( $0.25 \leq x \leq 0.75$  and  $y < 0.25$ ) or center bottom ( $0.25 \leq x \leq 0.75$  and  $y > 0.75$ ), then further analysis is needed. In this case, where the articulation type is ambiguous based on position alone, we rely on the articulation type prediction from the 3DOI model. For example, if the point is at the center top or bottom and 3DOI predicts a revolute joint, it suggests the object opens by rotating from top to bottom or vice versa.

Once the articulation type is inferred, for revolute joints, we still need to predict the revolute axis. We leverage our high-quality 3D geometry to accurately predict the revolute axis. This is done by identifying the corresponding edge of the object mask that aligns with the axis of rotation. We then project this edge into 3D space to obtain the 3D revolute axis.

To further enhance our predictions, we adopt GPT4o Inference. We provide GPT4o with images of the masked articulated objects. And GPT4o is queried for articulation information, including type and possible rotation directions. If GPT4o’s predictions differ from our inferences based on 3DOI predictions, we consult another VLM. This additional VLM acts as an arbitrator to resolve discrepancies and determine the final articulation type. Based on the confirmed articulation type and rotation direction, we use the corresponding edge of our object mask to define the rotation axis. This axis is then utilized in modeling and simulating the articulated object’s physical dynamics.

## D. Evaluation details

**Articulation Understanding Comparison** In the comparison with URDFormer [1] and Digital Cousin [2], the results of URDFormer and Digital Cousin are all in different coordinates. When we calculate the correct perception result, we first try to match the perception results with the GT layout. Specifically, we first divide the GT layout and the prediction layout into upper and lower parts. The upper parts are all cabinet doors which are fixed, attached to the wall, and hanging in the air. The lower parts are all cabinet doors which are placed on the ground. For each part of the layout, we assign a x-y coordinate for each door. To handle differences in scale between layouts, we normalize x-coordinates to a (0-1) range while preserving relative positions. We then conduct a pattern-preserving matching process that treats components at the same x-coordinate as a structural pattern



unit. We only consider a match valid when both GT and prediction have identical pattern structures (same number of components) at corresponding x-positions. This strict matching approach ensures that vertically-aligned components are matched as complete units, preserving the structural integrity of cabinet arrangements. Our evaluation metrics include total predictions, accurate predictions (number of correctly matched components), prediction recall (percentage of GT components matched), and prediction accuracy (percentage of predictions matching GT).

**Articulation Inference Comparison** In the comparison with 3DOI [12], for a fair comparison, we only compare on those cabinet doors which are recognized by our perception method. The 3DOI model takes a 2d point prompt as input, and we provide the 3DOI model with that in comparison.

## E. Failure cases mode

While DRAWER is generally robust to perception errors (See Fig. 10 for illustration), it occasionally fails due to catastrophic errors in foundation models. Fig. 9 showcases a few failure cases: (left) both 3DOI [64] and GPT-4o incorrectly suggest the revolute joint is on the right, while the ground truth hinge is on the left; (right) both Grounded SAM [68] and GPT-4o misperceive the side of the cabinet as a door.

## References

- [1] Zoey Chen, Aaron Walsman, Marius Memmel, Kaichun Mo, Alex Fang, Karthikeya Vemuri, Alan Wu, Dieter Fox, and Abhishek Gupta. Urdformer: A pipeline for constructing articulated simulation environments from real-world images. *arXiv*, 2024. 1, 2, 10
- [2] Tianyuan Dai, Josiah Wong, Yunfan Jiang, Chen Wang, Cem Gokmen, Ruohan Zhang, Jiajun Wu, and Li Fei-Fei. Acdc: Automated creation of digital cousins for robust policy learning. *arXiv*, 2024. 1, 2, 10
- [3] Epic Games. Unreal engine. 1
- [4] Gonzalo Martin Garcia, Karim Abou Zeid, Christian Schmidt, Daan de Geus, Alexander Hermans, and Bastian Leibe. Fine-tuning image-conditional diffusion models is easier than you think. *arXiv*, 2024. 4
- [5] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv*, 2020. 4
- [6] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *TOG*, 2023. 9
- [7] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *ICCV*, 2023. 9
- [8] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv*, 2023. 9
- [9] Luma AI. Luma unreal engine plugin. 1
- [10] Nelson Max. Optical models for direct volume rendering. *TOG*, 1995. 4
- [11] Thomas Müller. tiny-cuda-nn, 2021. 4
- [12] Shengyi Qian and David F Fouhey. Understanding 3d object interaction from a single image. In *ICCV*, 2023. 4, 10, 11
- [13] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. 2020. 4
- [14] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024. 9
- [15] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, et al. Nerfstudio: A modular framework for neural radiance field development. in *arXiv*, 2023. 9
- [16] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*. 9
- [17] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-nerf: Structured view-dependent appearance for neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5481–5490. IEEE, 2022. 4
- [18] Hongchi Xia, Zhi-Hao Lin, Wei-Chiu Ma, and Shenlong Wang. Video2game: Real-time interactive realistic and browser-compatible environment from a single video. In *CVPR*, 2024. 9
- [19] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv*, 2023. 9
- [20] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. Baked sdf: Meshing neural sdfs for real-time view synthesis. In *SIGGRAPH Conference*, 2023. 2
- [21] Zehao Yu, Anpei Chen, Bozidar Antic, Songyou Peng, Apratim Bhattacharyya, Michael Niemeyer, Siyu Tang, Torsten Sattler, and Andreas Geiger. Sdfstudio: A unified framework for surface reconstruction, 2022. 4
- [22] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *arXiv*, 2022. 4
- [23] Jicun Zhang, Jiyu Fei, Xueping Song, and Jiawei Feng. An improved louvain algorithm for community detection. *Mathematical Problems in Engineering*, 2021. 9