ART: Articulated Reconstruction Transformer

Supplementary Material

A. More results

We provide additional results in the project webpage.

Video results. We provide both fixed-view and rotational-view renderings of the reconstructed articulated objects, with the dynamic parts moving according to the predicted articulation structure.

Export into simulator. As discussed in the main text, our part-based output can be directly converted to URDF. Combining the exported URDF with each part's textured mesh yields simulation-ready assets. We showcase several interaction scenes in the MuJoCo simulator—each featuring a humanoid robot and an articulated object—and include the corresponding videos in the project webpage.

Detailed runtime analysis. We provide a quantitative runtime comparison on a single A100 GPU in Table S1. ART completes reconstruction in 0.85s (multi-view) and 0.42s (monocular). This is orders of magnitude faster than optimization baselines with lengthy per-instance processing. This efficiency stems in part from ART's use of cross-attention layers, which reduces token processing time.

Real-world results Here we add four additional real-world results (Figure S1). While texture reconstruction is inherently limited by the synthetic training domain, ART demonstrates robust articulation-structure understanding across diverse opening configurations.

B. Rendering Articulated Object

At each training iteration, we render the articulated object both per part and in a compositional manner to obtain the final image for supervision. The rendering process queries hexa-plane feature outputs from the model to compute the per-pixel RGB and uses the predicted articulation to correctly place each dynamic part.

Our rendering pipeline follows the signed distance function (SDF) volume rendering [69]. For a camera ray defined by origin $\mathbf{o}$ and unit direction $\mathbf{v}$, we first intersect the ray with the part's axis-aligned bounding box $\boldsymbol{B}_p$ (in canonical space) and sample 3D points along the valid ray segment. Each sampled world-space point $\mathbf{x}_p(\mathbf{o}, \mathbf{v}, \mathbf{B}_p)$ is mapped to the part's normalized local coordinates $\hat{\mathbf{x}}_p = (x, y, z) \in [-1, 1]^3$, which are then used to query the hexa-plane features:

$$\mathbf{f}_{p,xy} = \begin{cases} \text{Bilinear}(\mathbf{T}_{pxy+}; x, y), & z \geq 0, \\ \text{Bilinear}(\mathbf{T}_{pxy-}; x, y), & z < 0. \end{cases} \tag{S1}$$

Here $\mathbf{T}_{pxy+}$, $\mathbf{T}_{pxy-}$ are the model's hexa-plane representation output from Eq. 2; $\mathbf{f}_{p,yz}$, $\mathbf{f}_{p,xz}$ can be obtained analogously. These features are concatenated to form the feature vector $\mathbf{f}_p$ for the spatial point $\mathbf{x} + p$. Following the architecture in [34, 61], we use two small MLPs to predict SDF value $\mathbf{s}_p$ and RGB color $\mathbf{c}_p$, respectively. In particular, the SDF value is computed as:

$$\mathbf{s}_p = \text{MLP}(\mathbf{f}_p) + \mathbf{s}_{\text{bias}}(\mathbf{x}_p). \tag{S2}$$

where the bias term $\mathbf{s}_{\text{bias}}(\mathbf{x}_p) = ||\mathbf{x}_p|| - 0.1r$ is a prior defined in the part's local space, to initialize the shape as a sphere to stabilize the training [34, 69]. The SDF value $\mathbf{s}_p$ can be converted further to volume density using the Laplace CDF [69]:

$$\sigma_p = \begin{cases} \frac{1}{2}\exp(-\frac{\mathbf{s}_p}{\beta}), & \mathbf{s}_p \geq 0, \\ 1 - \frac{1}{2}\exp(-\frac{\mathbf{s}_p}{\beta}), & \mathbf{s}_p < 0. \end{cases} \tag{S3}$$

where β is the standard deviation that controls the sharpness of the underlying surface. As noted in the main text, $\frac{1}{\beta}$ is linearly annealed over training to progressively sharpen surfaces. With σ_p and $\mathbf{c}_p$ evaluated at ray samples, we compute *per-part* color, opacity (mask), and depth via the usual transmittance accumulation [45, 69]. Normals are obtained by numerically differentiating the SDF at each sample [32].

To obtain the composited image over *all parts*, we merge the sampled points from all parts along each ray, sort their orderings by the sampled ray distance, and apply standard alpha compositing to compute the rendered images. We employ Nerfacc [30] to accelerate point sampling, ray distance sorting, and compositing.

Dynamic parts rendering. The equations above describe static per-part rendering in each part's canonical space (from the model-predicted bounding-box region). For dynamic parts at stage t, instead of physically moving the part's volume and rebuilding an oriented bounding box, we transform the camera rays into the part's instantaneous local frame (i.e., move the rays inversely), keeping the axis-aligned bounding box $\mathbf{B}_p$ unchanged.

Recall that $\mathbf{C}_p$ denotes the motion type, $\mathbf{D}_p$ the unit joint axis, $\mathbf{O}_p$ a point on that axis, and $\mathbf{S}_{p,t} \in [-1, 1]$ the normalized motion value at stage t. For a point $\mathbf{x}$ in the part's canonical space, the stage-t rigid transform $T_{p,t}$ for different motion types can be defined as:

Optimization-based	PARIS	DTA	ArtGS
Runtime (min)	50	40	9
Feed-forward	URDFormer	SINGAPO	Ours
Runtime (s)	2.37	0.54	0.85 / 0.42

Table S1. **Runtime comparison** between baselines and ART.

Figure S1. **Real-world images results.**

Prismatic:

$$T_{p,t}(\mathbf{x}) \;=\; \mathbf{x} \;+\; (2r)\,\mathbf{S}_{p,t}\,\mathbf{D}_p. \tag{S4}$$

Revolute:

$$T_{p,t}(\mathbf{x}) \;=\; \mathbf{O}_p \;+\; R\big(\mathbf{D}_p,\; 2\pi\,\mathbf{S}_{p,t}\big)\,\big(\mathbf{x}-\mathbf{O}_p\big), \tag{S5}$$

where $R(\mathbf{d}, \theta)$ denotes the rotation by angle θ (radians) about axis $\mathbf{d}$.

Now, instead of actually transforming the bounding box (point), we'll inversely transform each ray $(\mathbf{o}, \mathbf{v})$ to reach the equivalent rendering results. It's straightforward that for prismatic parts, after transformation:

$$\hat{\mathbf{o}} = \mathbf{o} - 2r \cdot \boldsymbol{S}_{p,t} \cdot \boldsymbol{D}_p, \quad \hat{\mathbf{v}} = \mathbf{v}, \tag{S6}$$

and for revolute parts, similarly:

$$\hat{\mathbf{o}} = \mathbf{o} + \mathrm{Rotate}(\boldsymbol{D}_p, -2\pi \cdot \boldsymbol{S}_{p,t})(\mathbf{o} - \boldsymbol{O}_p) + \boldsymbol{O}_p, \tag{S7}$$

$$\hat{\mathbf{v}} = \mathrm{Rotate}(\boldsymbol{D}_p, -2\pi \cdot \boldsymbol{S}_{p,t})\mathbf{v}. \tag{S8}$$

We then proceed exactly as in the static case: intersect the transformed ray $(\hat{\mathbf{o}}, \hat{\mathbf{v}})$ with bounding box $\boldsymbol{B}_p$, sample points, query hexa-planes, and composite to obtain the desired renderings. The computed values from the queried features are further assigned to the original ray sampled points, which are in the current world coordinate space, for volume rendering composition.

C. Dataset Construction

A key advantage of ART is its training data: we substantially increase both the quantity and diversity of articulated assets. Prior feed-forward models [20, 37] typically rely on subsets of PartNet-Mobility [63], which offer limited geometric diversity and often unrealistic textures.

In this work, we combine three articulated-object data sources to construct our training dataset, increasing diversity and fidelity in geometry, texture, and articulation complexity. Beyond the basic information in the main text, further details on these sources are provided below.

PartNet-mobility. As previously mentioned, PartNet-Mobility provides common indoor articulated categories—bucket, dishwasher, door, laptop, microwave, oven, refrigerator, and storage furniture. In total, we collect over 300 objects from this dataset. However, many assets have unrealistic textures and low-quality surface geometry, motivating our exploration of more diverse and realistic articulated object sources.

Procedural dataset. To scale both quantity and diversity, and inspired by Infinigen-Sim [23], we adopt procedural generation to author articulated assets in Blender (with TA support). The resulting dataset includes 2,000 high-quality articulated models across six categories: laptop, dishwasher, beverage refrigerator, cabinet with drawers, bucket, and microwave oven. 3D geometry is provided in GLB/OBJ and articulation in URDF. Each category is governed by procedural rules over shape, appearance, and articulation; for each, we generate several hundred variants with randomized shapes, sizes, and materials. In principle, this pipeline can produce an unlimited number of articulated objects.

StorageFurniture dataset. We recognize that the StorageFurniture category in PartNet-Mobility spans many commonly used articulated assets, a property also noted by prior methods [4, 37] that primarily train on this category. Based on this, we construct a *StorageFurniture* dataset by recombining parts from the PartNet-Mobility storage-furniture class.

For a given object, we use its articulation tree (kinematic structure) to procedurally create new instances via compositional part assembly: original parts are replaced with geometries from other objects (rescaled as needed), followed by UV-map correction and material randomization. This yields a large, realistic, and varied set of assets—over 10,000 articulated models for training.

These three data sources collectively provide a large pool of articulated objects. Using each asset's articulation definition, we further generate random per-part trajectories, resulting in a large dataset of articulated object sequences.

D. More Implementation Details

During sequence data construction, we explicitly set the order of the parts following a certain rule: the static base part is the first, and the remaining dynamic parts are ordered from low to high, front to back and left to right. This consistent ground-truth ordering greatly improves training stability and convergence speed. We also define the *rest state* at this stage as the configuration where all dynamic parts are "closed."

The transformer in ART has 32 attention blocks with a 3 : 1 cross-/self-attention ratio, 16 heads, and a 256-dimensional embedding. It outputs 24×24 hexa-plane features that are later upsampled to the spatial resolution of 192×192. We set the maximum part count to $P_0{=}8$. Training uses AdamW with $(\beta_1, \beta_2){=}(0.9, 0.95)$. The multi-view model trains for 5 days on 64 H100 GPUs; the monoc-

ular model trains for 3 days.

ART is trained with known camera poses as part of the input to distinguish tokens across views. This design follows common LRM-style practice, where camera information serves as implicit geometric cues for multi-view 3D reconstruction. Empirically, providing calibrated poses significantly accelerates training convergence and improves accuracy compared to using fully optimizable view embeddings. While we currently assume calibrated poses, we observe that with results remain reasonable under noisy real-world pose estimates. As discussed in our limitations, developing a pose-free variant by integrating optimizable view embeddings is a meaningful direction for future work to make the system more practical for arbitrary user-captured images.

Inference details Similar to other LRM-style models, ART targets an object-centric reconstruction scenario. Consequently, it requires input images containing a single articulated object accompanied by a corresponding segmentation mask produced by off-the-shelf segmentation models. To prepare the input, we identify the longer side of the object's bounding box and crop a square region that preserves the full foreground, which is then resized to the input resolution. These standard pre-processing steps are used in both training and inference.