CAGE: Controllable Articulation GEneration

Supplementary Material

In this supplement to the main paper, we provide implementation details (Section 1) and additional qualitative and quantitative results (Section 2). We also include a supplemental video to provide a quick introduction to our work.

1. Implementation Details

1.1. CAGE Training Details

We randomly permute the ordering of the nodes in graphs during training so that our model does not learn to rely on specific node orderings. This makes our model robust to graph isomorphic permutations at inference time. For parameterization, we normalize all the part attributes in the range of $[-1, 1]$ to feed into the diffusion model.

Our diffusion model follows the standard scheme proposed in DDPM [1]. In the forward process, we use a linear beta scheduler that maps a sequence of betas ranging from $1e-4$ to 0.02 with a total of $1,000$ diffusion steps. Each training iteration consists of 64 objects and each object is trained with 10 randomly sampled timesteps. We set an initial learning rate of $5e-4$. To schedule the learning rate, we use a warm-up strategy for 20 epochs and then decay from the initial learning rate to 0 by following the cosine function. During inference, we set 100 denoising steps for generating each sample. We train the diffusion model for $5,000$ epochs in total on a single NVIDIA A40 GPU for 13.94 hours. We show a quantitative comparison of the training time with other baselines in Table 1.

	$K = 8$, *articulation* graph			$K = 32$, *action* graph	
	NAP [2]	NAP-light	Ours-light	NAP-large	Ours
Training time (hrs)	6.05	1.76	2.74	22.87	13.94

Table 1. Training time comparison with baselines and variations under two experiment settings using 32-bit float precision. **Setting 1**: for models with $K = 8$, the *shapecode* formulation in the NAP model accounts for the majority of the computational overhead, as evidenced by the comparison with the NAP-light variant which replaces this component with a lighter part type attribute. The comparable variant of method (Ours-light) incurs a marginally higher computation time than NAP-light, while providing significantly better outputs as described in the main paper. **Setting 2**: in scenarios where the methods are extended to more parts ($K = 32$), the computational overhead for the NAP-large model escalates significantly whereas our method's training time remains comparatively lower. These results demonstrate the improved computational scalability of our approach relative to NAP, which is important for complex objects with larger numbers of distinct parts.

1.2. Part Retrieval Implementation

Given a generated articulated object abstract specification, we use a two-step approach to retrieve suitable parts and build the final articulated 3D object. We pick the base part in step 1 and the remaining parts in step 2.

Step 1: base part retrieval. We compute a Weisfeiler-Lehman graph hash [3] from the generated object kinematic tree. This hash is identical for isomorphic graphs. Since the base part (i.e. stationary part of the object) should be compatible with the generated object part motions, we anticipate that the best–matching object candidates will have the same kinematic tree topology. Hence, only candidates with the same hash (and object category, if specified in the input) from the training set are selected for further consideration. In cases where no candidates have the same hash (which happens when the object is out of distribution), we consider all candidates in the training set. We then compute the AID metric for each selected candidate and pick the base part from the candidate with best metric value. In addition, we keep the top five candidates for the next step.

Step 2: articulated part retrieval. For the remaining parts other than the base, we pick a single candidate part for each semantic part in the generated object (e.g., one drawer part for a storage with three drawers). The selected part is duplicated and resized to fill the part bounding box in the generated object abstract specification. We start with parts in the top five object candidates in step 1 and choose a part if it has the same semantic label as any of the required parts. If there are still unretrieved parts, we consider other candidates from the same category (if specified) or the whole training set.

The above procedure is designed to maximize style consistency between the retrieved parts, and create coherent objects. The full 3D mesh visualizations in the main paper and in this supplement demonstrate the results obtained using this approach.

1.3. Metrics

Here we provide implementation details for the two distances used in our evaluation metrics.

Instantiation Distance (ID). We simplify the ID metric first proposed by NAP [2] to consider pairwise Chamfer-L1 distance between two objects in temporally synchronized articulation states. Specifically, we randomly sample 2,048 point samples per part per object and compute their Chamfer distance in five evenly spaced-out articulation states within the joint ranges of the objects. We take the average of the five distances to be the final ID value.

Abstract Instantiation Distance (AID). We also introduce AID to measure the distance between two objects with volumetric IoU (vIoU) on the part bounding boxes. Given two objects, we first scale both objects such that their overall bounding boxes are the same size. We then assign part correspondences between the two objects based on the part bounding box center distances. For each part pair, we compute a sampling-based vIoU (using $10,000$ point samples per bounding box) in five evenly spaced-out articulation states within the joint ranges of the objects. Finally, we take the average of the vIoUs over all states and parts and compute the complement ($1 - \text{vIoU}_{\text{avg}}$) as the final AID value.

2. Additional Results

Graph conditional generation. Figure 1 shows 18 randomly generated samples (with no manual selection) from our method CAGE, and the baseline NAP-large adapted from Lei et al. [2]. Both are conditioned on medium-complexity object graphs. The results are the first 18 results generated from each method. The output objects generated using our method are consistently compatible with the input graph and achieve overall high fidelity. In comparison, the objects generated using NAP often fail to conform to the graph constraint with flipped edge connections, denoted in red arrows. Moreover, many of the objects generated using NAP are unrealistic, with incorrect articulation axes and significant part-part misorientations and collisions.

Part→Motion. Figure 2 shows two sets of randomly generated samples (with no manual selection) from our method and NAP-large that are conditioned on part shape attributes. The generated motion using our method is stable and consistently compatible with the input part shape (e.g. in the microwave example, the handle on the right of the door indicates that the door is more likely to be opened from the right). The second example is a more challenging case with more parts to be coordinated. Here, we observe a few failure cases with implausible articulations in our results (columns 3 and 5 from the left). In comparison, a large proportion of the results from NAP exhibit implausible motions.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 33:6840–6851, 2020. 1

[2] Jiahui Lei, Congyue Deng, Bokui Shen, Leonidas Guibas, and Kostas Daniilidis. NAP: Neural 3D Articulation Prior. *NeurIPS*, 2023. 1, 2

[3] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011. 1

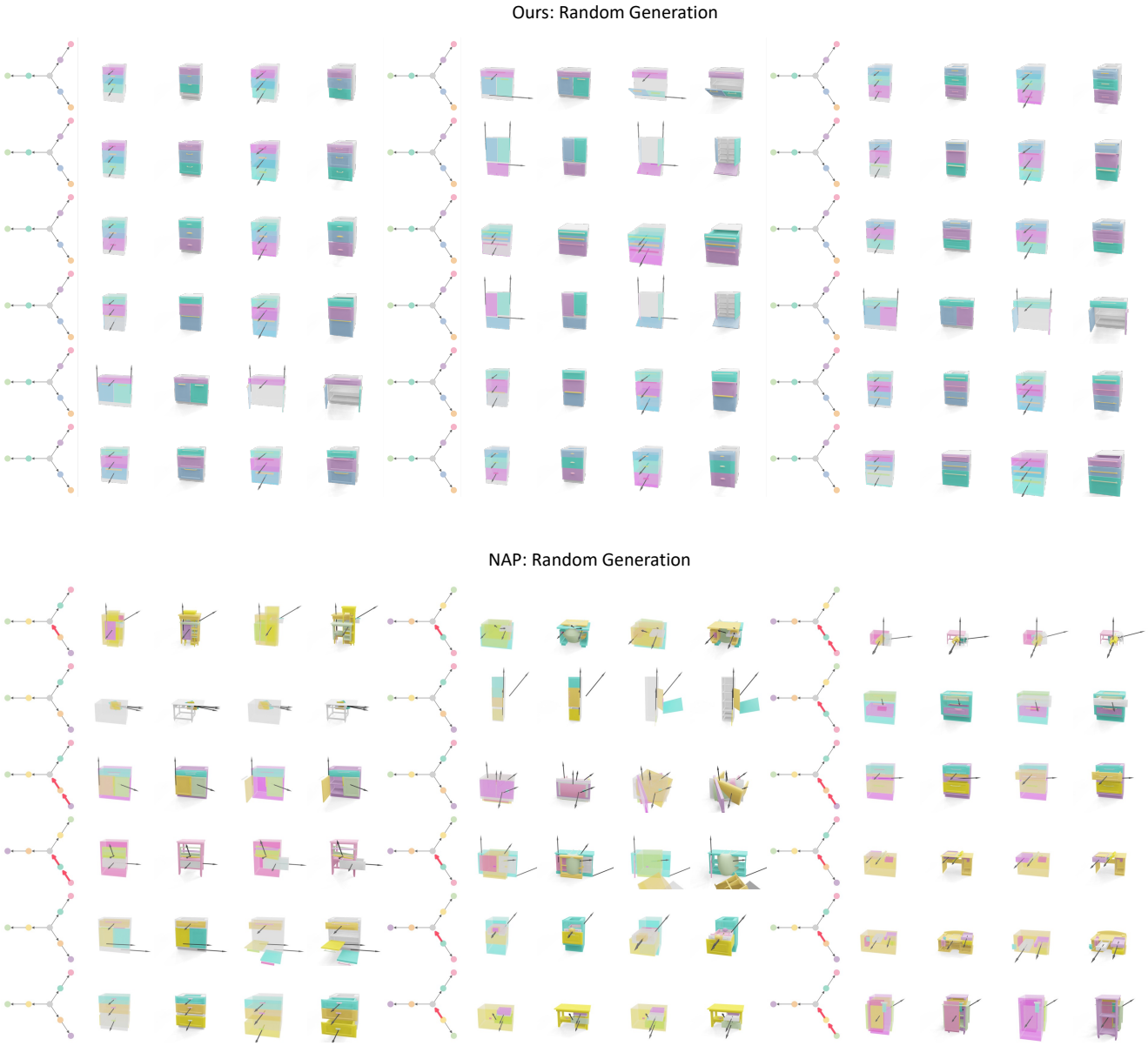

Figure 1. **Graph conditional generation**: we show 18 randomly generated samples (without manual selection) produced with our method (top) and NAP-large (bottom). **Summary**: Our results are consistently compatible with the input graph and achieve overall high fidelity. In comparison, the results from NAP often fail to conform to the graph constraint with many flipped edge connections denoted by red arrows in the output part hierarchy graph. Moreover, many objects generated with NAP-large exhibit unrealistic part-part overlaps and incorrectly oriented motion axes. **Setup**: The results from ours and NAP-large are both conditioned on a medium-complexity object articulation graph identical to the graph shown in the top-left entry. The results are selected from the first 18 results generated from each model. For every set of five columns, the first column shows the node hierarchy of the generated parts. The second and third columns depict the object in its resting state in abstract and complete mesh form. Columns four and five represent the object in a fully open state, again in abstract and mesh form.

Figure 2. **Part → Motion**: randomly generated samples (with no manual picking) from our method and NAP-large for two example inputs shown at the top. **Summary**: our generated motions are stable and consistently compatible with the input part shape (e.g. in the microwave example, the handle on the right of the door indicates that the door is more likely to be opened from the right). The second example is more challenging with more parts to be coordinated. As expected, we observe some failure cases in our results, with implausible articulation motions (see outputs in the sets at column 3 and column 5, from the left). In this more challenging case, NAP by comparison produces implausible motions in a much larger proportion of the generated results with almost all generated objects exhibiting part collisions or unrealistic motion axes. **Setup**: Every set of two columns shows a generated object, with the first column showing the abstract bounding box form with joint axes and the second column showing the final retrieved part meshes, both in the fully open state.