Articulate3D: Holistic Understanding of 3D Scenes as Universal Scene Description

Supplementary Material

We provide additional details on:

- the annotation process (Sec. 8),
- the statistics of Articulate3D (Sec. 9),
- applications of Articulate3D (Sec. 10),
- experiment details and additional results (Sec. 11).

8. Annotation Process

8.1. Annotation Tool

We introduce the interface of the annotation tool used to create Articulate3D in Fig. 6. It features instance semantic segmentation functionalities, extended to enable connectivity annotation, as well as a view for articulation annotations.

An essential feature for the annotation of Articulate3D was the support both for fine-grained segmentation, as well as for an oversegmentation-based annotation. While the provided initial segments support fast and accurate annotation for big objects with flat surfaces, e.g. cabinets, fine-grained details such as buttons, knobs, switches, etc., are not recognized, as seen in Fig. 7.

8.2. Annotators Training

To ensure the quality and consistency of annotations, annotators underwent comprehensive training supported by detailed documentation and tutorials. The preparation materials included:

Segmentation Training:

- Two training videos, totaling 40 minutes, covering the segmentation tool, illustrative examples of typical cases and exceptions.
- Documentation with examples for all object classes in the fixed object label list, including: (1) images of segmented parts, (2) their corresponding connectivity graphs, and (3) the step-by-step sequence of commands used to achieve precise segmentation.
- A complete guide to the application's controls and shortcuts.

Articulation Training:

- A 15-minute training video demonstrating the articulation annotation tool and process, with examples of rotation and translation articulations.
- Documentation with examples of the different articulated parts classes.

Additionally, annotators were provided with chat support throughout the annotation process. The chat support was especially valuable during the initial review phase, allowing annotators to resolve questions, particularly around complex connectivity graphs, e.g., oven models, where controls are physically attached to the oven door but semantically part of the oven body.

9. Dataset Statistics

For a comprehensive overview of the annotated object and part labels, we refer to Fig. 9 and Fig. 10, which provide information both on the labels and on their distributions. Distributions of the 30 most frequent labels per train and test split are shown in Fig. 12.

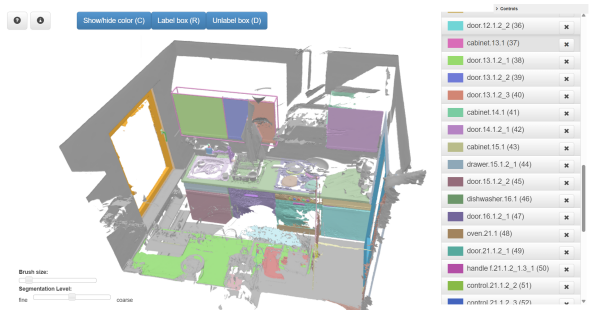
We note that the dataset also includes additional labels to address variations based on location (e.g., "cabinet" versus "wardrobe"), functional mechanisms (e.g., "faucet handle" versus "faucet ventil"), and other contextual distinctions which are not captured in the provided figures. We will include resources for mapping these additional labels and their parent categories with the dataset release.

We also provide size information for the annotated items (both objects and parts) in Fig. 11, where size is measured by the faces (triangles) count of each item. Our analysis reveals that Articulate3D includes a mix of large objects and many smaller ones with only a few faces. This highlights both the high quality of the annotations in capturing intricate details, as well as the variability within the dataset.

Our statistical analysis focuses solely on annotations added by our annotators. Since our work builds upon the ScanNet++ dataset [86], we ensure full compatibility between our annotations and those from ScanNet++. Unaltered ScanNet++ annotations (e.g., walls, floors, and sofas) are directly integrated into the final USD, while certain labels, such as "cabinet," are entirely replaced by our own annotations, as illustrated in Fig. 8. We will provide a script merging the annotations from Articulate3D and ScanNet++.

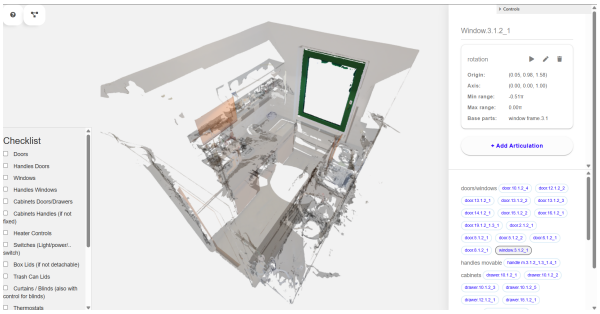
9.1. Segmentations Quality Evaluation

To assess the quality of our segmentation annotations, we conducted a control experiment involving two annotators. They were each tasked with re-annotating a "control scene"—a scene that had been previously annotated, then reviewed, and approved by the sixth annotator. The annotations were manually matched item-by-item between the two annotators, and IoU was calculated for each matched pair. The scene's average IoU served as the metric for annotation quality. Notably, the re-annotators were distinct from the original annotators. We achieved an IoU of 0.93 for the control scene.

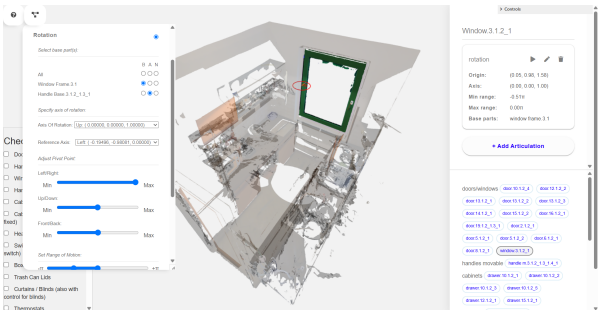


(a) Segmentation interface for annotation of object and part semantic segmentation and connectivity.

(b) The segmentation interface offers switching between color and segmentation view, enabling distinction of finer details.



(c) Articulation annotation interface. The annotators are provided with a checklist of part categories to annotate, and a sorted by label list of segmented parts, promoting a systematic annotation process.



(d) Annotation of motion parameters.

Figure 6. The annotation tool.

10. Downstream Applications

10.1. LLM-based Scene Editing

We present a pipeline for automated, semantically-aware object insertion into USD scenes. Given a USD scene from Articulate3D, a 3D object file, and the object's label, our solution produces a new USD scene with the object placed in a semantically-appropriate location. For example, in a bedroom scene, a pillow object would be placed on a bed, while in an office scene, a bottle would be placed on a desk. The pipeline uses a LLM to show the USD-understanding capabilities of LLMs, as well as to minimize user involvement. Implementation details are outlined below.

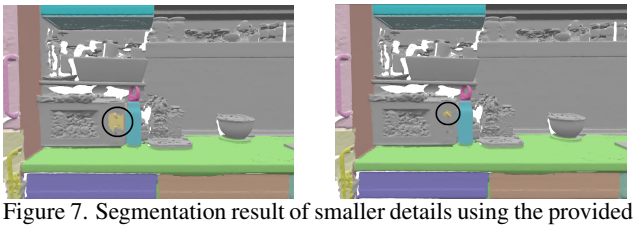


Figure 7. Segmentation result of smaller details using the provided oversegmentation (left) vs annotating on fine-grained level (right). Segments produced by the oversegmentation can prove unable to capture fine variations in the geometry, showing the need for a fine-grained segmentation support.

The method requires three inputs: (1) a USD Articulate3D scene, (2) a 3D object file, and (3) the object's label. We support various 3D file formats (e.g., OBJ, USD, GLB). Given the inputs, the pipeline extracts item labels and connectivity data from the USD scene and prompts the LLM to determine the appropriate placement target (e.g., a bed for the pillow) and the type of surface required (e.g., horizontal for a pillow, vertical for a poster). With this information, the pipeline employs RANSAC to identify a suitable placement plane on the target object. Using the plane, the input object, and an example USD insertion script defined by us, the LLM generates a script customized for the specific insertion case. The generated script is then executed to produce the updated Articulate3D USD scene.

For our pipeline, we have tested two LLMs - GPT-4o mini [48] and GPT-4o [49], both producing the desired results. We will release the pipeline as a Python CLI library.

10.2. Simulation-to-go for Robotics

Our data can be easily uploaded in a physics simulator and used for robotics policy learning, without manual adaptions. We use IsaacSim with IsaacLab [42] as simulator setup due to their USD-centricity. The utilized GPU is RTX 3090. All simulations are conducted with the Franka robot.

Articulate3D scenes are versatile and can be utilized with various policies, as demonstrated by their compatibility with both planner-based solutions and policy training via PPO [60]. We also note that the Articulate3D scenes can be used both for scene-level simulation, as well as for object-level, as depicted in Fig. 13. This is enabled by USD's support for easy extraction of single objects. Scene-level manipulations are also easy as objects within scenes can be removed/added or rearranged to achieve versatility. If using

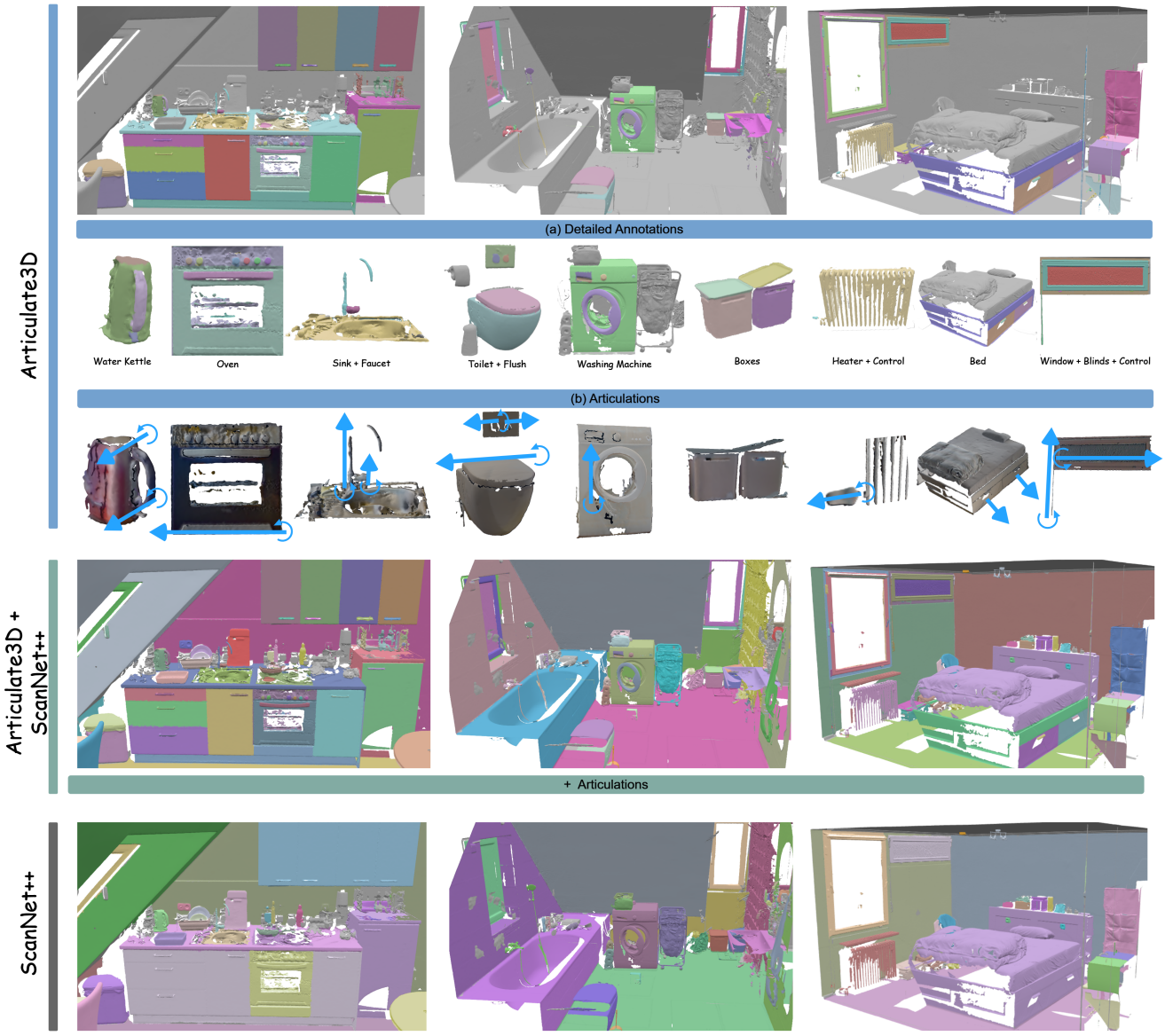


Figure 8. Overview of the annotations provided by Articulate3D, ScanNet++ [86], and their combination. Articulate3D offers (a) detailed annotations at both the object and part levels, including connectivity information, and (b) full articulation annotations. By combining ScanNet++ with Articulate3D, users gain a comprehensive scene segmentation with nearly 100% coverage, alongside detailed annotations for interactable objects and their motion specifications.

the object extraction strategy, the users obtain more than 3k articulated real-world USD objects from over 50 categories.

Articulate3D scenes are decimated using quadratic decimation. In this way we enable learning on multiple environments in parallel, speeding up training.

Planner-based policies. We apply a planner-based policy only requiring a specification of an object of interest (e.g., a cabinet), a movable part (drawer) and the corresponding interactable part (handle). All articulation parameters and configurations are derived from the USD scene file.

Proximal policy optimization (PPO) policies. Articulate3D scenes support training in simulation, which we demonstrate by learning a drawer-opening policy via PPO, with 87% success rate. We use 1024 environments and train for 30k iterations with 40 steps per environment, with learning rate 5.0e-4 and adaptive schedule.

10.3. Cross-Domain Experiment - URDFormer

As illustrated in Sec. 6, we evaluated URDFormer [7] on both Articulate3D and Multiscan dataset to demonstrated

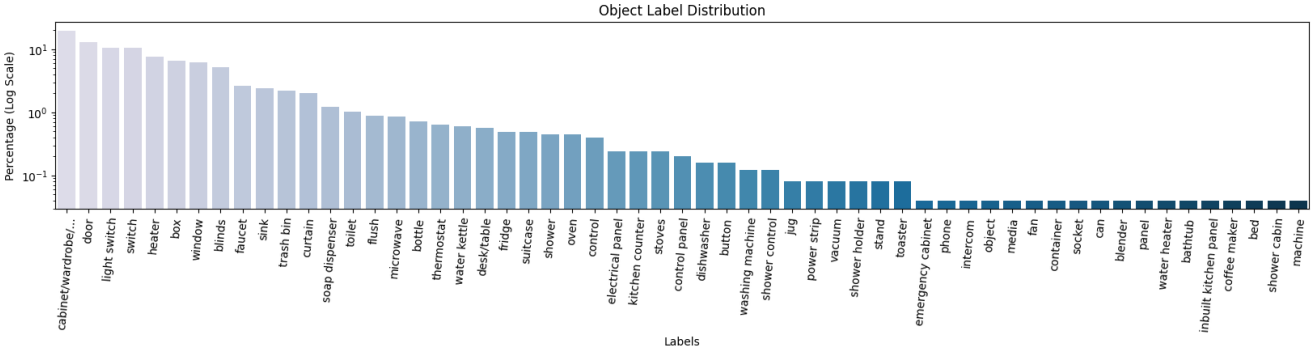


Figure 9. Distribution of the object-level labels.

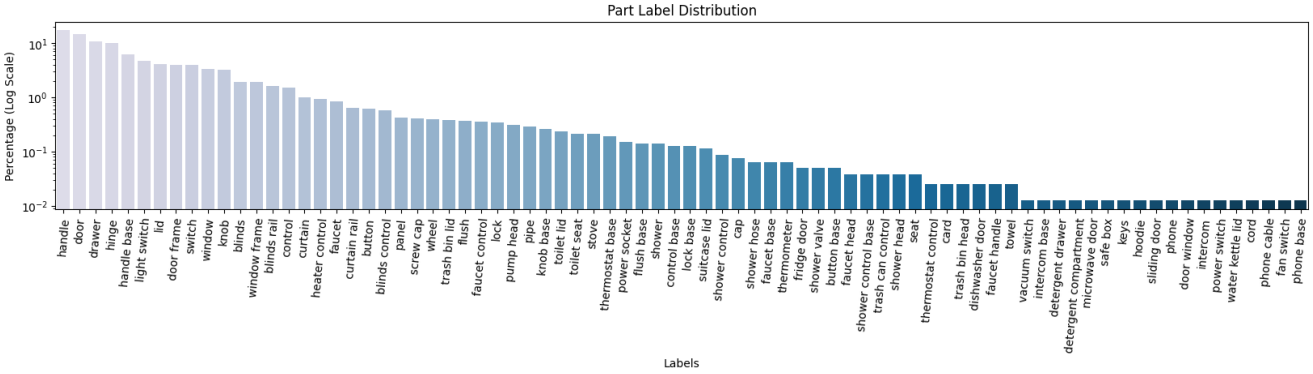


Figure 10. Distribution of the part-level labels.

the value of Articulate3D in cross-domain generalization. URDFormer consist of global scene-object arrangement generation and local object-part generation. We focus on the second part and assume the poses of articulated articulated object are given. To obtain data required for fine-tuning/evaluation of URDFormer, we generate pairs of images and 3D articulated object by selecting 3 top images per object in which the object are most visible and then crop the images to fit the object. URDFormer introduces biases in the framework design for object URDF generation: it generates bounding boxes of parts of articulated object and then fit the boxes with the part mesh from PartNet-Mobility [80], and it also assumes the articulation axis is aligned with one of edges of those bounding boxes. Thus, for fair evaluation, we focused the accuracy of the generated bounding boxes instead of the biased geometry or motion parameter. Generally, given the cropped image of articulated objects, we apply URDFormer to predict the bounding boxes and motion type (rotation, translation and background) of the movalble parts and evaluate the part detection accuracy by comparing the predicted part bboxes and the ground truth ones.

10.4. Connectivity Graph Prediction

To demonstrate the value of Articulate3D in 3D holistic scene understanding, particularly in the context of connectivity graphs within articulated objects, we conduct experiments on the task of connectivity graph prediction. As introduced in Sec. 3, a connectivity graph refers to the hierarchies between different partegments. In the graph, the "root" part is the base to which other parts are attached. A "child" refers to a part that belongs to the parent part. We train a network on Articulate3D that takes the 3D point cloud of objects' parts as input and predicts the connectivity graph of the parts by inferring relationship between each part pair.

Method. We draw inspiration from the existing scene-graph-based framework [58] and apply a similar network for the task. As shown in Fig. 15, PointNet [3] is applied to extract the information of geometry and appearance of part segments, outputting per-part feature vectors. Then we treat each part segment as a node and feed the feature vectors into a graph attention network [71] to extract the relationship between the nodes. The features of the two node are concatenated and fed into a MLP to infer the pairwise relationship between them ($\{no\ relationship,\ parent\ of,\ child\ of\}$).

Result. In Table. 9, we show the accuracy of the connectivity graph prediction. From the table we can see that (1) the random guess could achieve $36\,\%$ accuracy of edge prediction by randomly picking one relationship from $\{no\ relationship,\ parent\ of,\ child\ of\}$ for each pair of part segments, and (2) it has poor performance in the connectivity graph prediction for objects as only 6.1% of

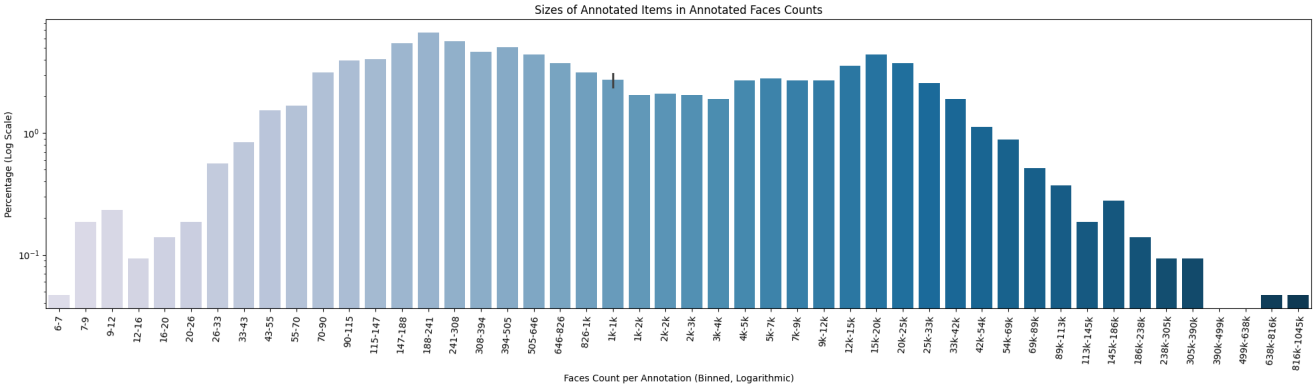


Figure 11. Distribution of face counts per annotated item using logarithmic binning. Articulate3D features numerous high-detail small annotations alongside larger objects

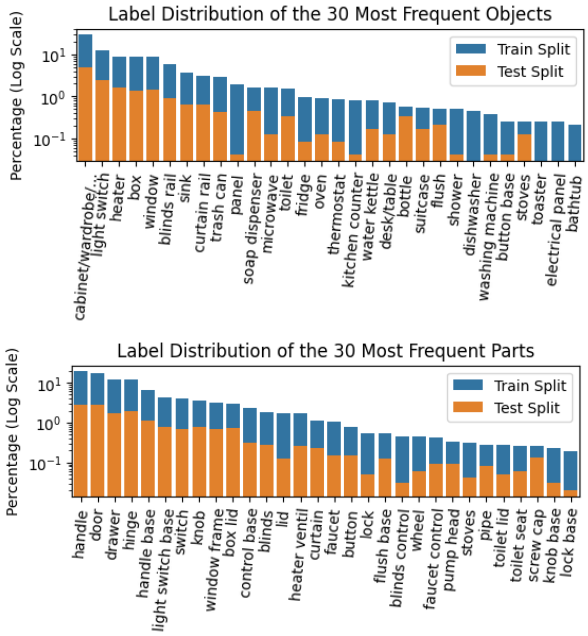


Figure 12. Distributions of the 30 most frequent object and part labels, per split.

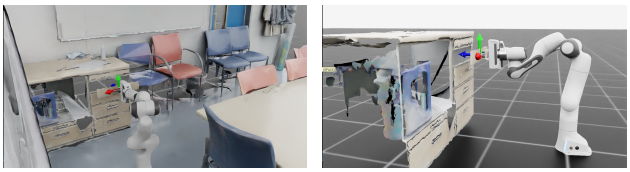


Figure 13. Simulation of an Articulate3D scene (left) vs simulation of a single object from an Articulate3D scene (right).

objects' connectivity graph are correctly predicted. Compared to the random guess, by training on Articulate3D , our network can achieve much better performance in both edge prediction (72.7%) and connectivity graph prediction (31.1%). In order to further understand the performance of our model in connectivity or edge prediction,

Method	Acc_{edge} (%)	Acc_{obj} (%)
Random	36.0	6.1
Ours	72.7	31.1

Table 9. Accuracy of connectivity graph prediction. Acc_{edge} represents the percentage of edges that are correctly recognized in $\{no\ relationship,\ parent\ of,\ child\ of\}$; Acc_{obj} represents the percentage of objects that has correct connectivity graph with all the edges within it correctly recognized.

we plot the ROC curves as shown in Fig. 14. As the edge prediction is a 3-category classification problem, we transform the results into two binary classification problems for straightforward ROC plot: connectivity classification ($\{no\ relationship, \{\ parent\ of,\ child\ of\}\}$) and hierarchical classification ($\{\ parent\ of,\ child\ of\}\}$). From Fig. 14 we can see that our model has good performance (with 0.88) in predicting the hierarchical relationship between part segments (whether a is the parent of b or the opposite), while it is more challenging to tell whether two parts have hierarchies within the connectivity graph (AUC 0.74).

11. Implementation and Experiment Details

This section introduces the implementation and experiment details of the experiments in Sec. 4 and Sec. 5 as well as additional results.

11.1. Implementation Details

We take 15% of the scenes (42 scenes) in Articulate3D as the test set for evaluations. For the training and evaluation of all the methods, we downsample the point cloud with a voxel size of 2cm from the original laser scans in [86], in order to achieve balance between preserving geometric details and computation resources. For the task of movable part segmentation, all points in the

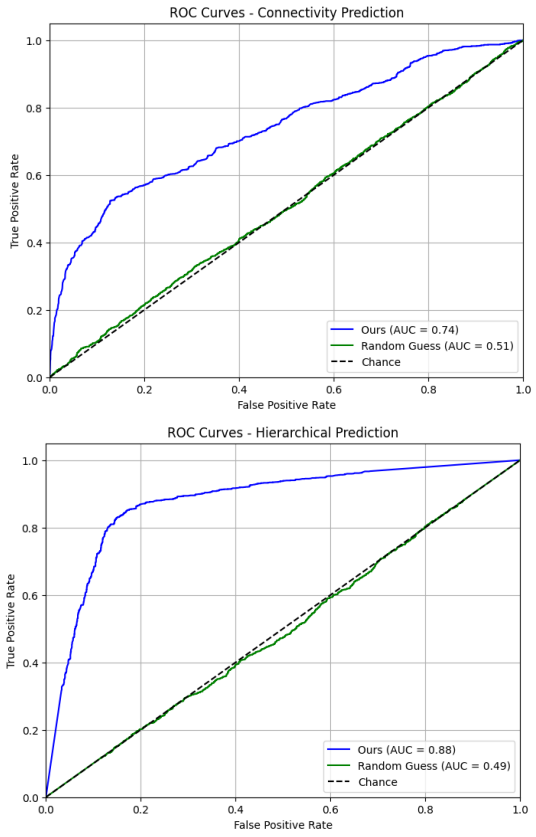


Figure 14. ROC curve of connectivity and hierarchical relationship of part segments prediction of our method.

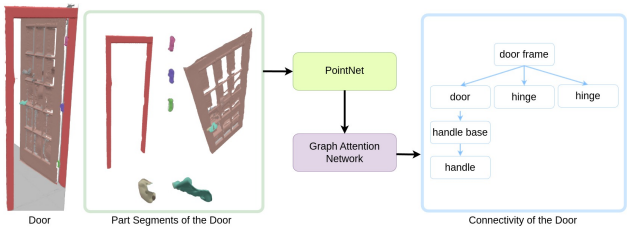


Figure 15. Connectivity Graph Prediction of Parts of Articulated Objects.

scene will be semantically segmented into one of the classes $\{background, rotation, translation\}$. For points of $\{rotation, translation\}$, instance labels are furthered assigned for instance segmentation of movable part.

Softgroup†. As described in the main paper, we adopt the framework of Softgroup[73] for this task. We firstly pretrain Softgroup[73] for 2000 epochs in the task of semantic segmentation, and then further train the semantic segmentation network together with instance grouping branch for movable part segmentation and the articulation branch for articulation parameters prediction for 220 epochs. For interactable part segmentation, we also pretrain the network with semantic segmentation for 2000 epochs and then another 360 epochs for instance segmentation of interactable

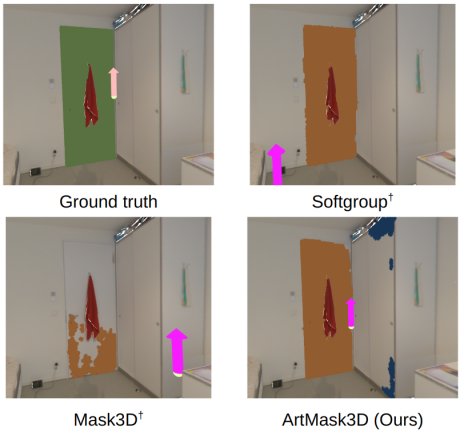


Figure 16. Movable part segmentation and articulation predictions (origin and axis).

parts. The training batch size is 6, on 2 NVIDIA A100-40g GPUs. Learning rate is 0.002.

Mask3D†. For Mask3D†, we firstly pretrain it for semantic-instance segmentation of movable part for 1000 epochs and then further train it for joint tasks of movable part segmentation and articulation prediction for 920 epochs. For interactable part segmentation, we train the network for 1400 epochs. The training batch size is 1, on 1 NVIDIA A100-40g Gpu and the learning rate is 0.0001. In order that the input of large scenes in Articulate3D fits in the memory of the training GPU, we randomly crop the input scenes into a $6 \times 6\ m^2$ cuboid during training.

11.2. Additional Results

Qualitative results of movable part segmentation and motion prediction can be seen in Fig. 16. In order to further understand the performance of the proposed USDNet, we analyze its performance across objects of various sizes and categories. In Fig. 17, we show the performance of USDNet in the task of movable part segmentation and articulation parameter prediction over the movable parts with different number of points. From the figure we can see that for the task of movable part segmentation and articulation origin prediction, USDNet performs better with middle-sized parts (with $1739 \sim 5207$ points) than with the small or large parts (less than 1739 or more than 5207 points). On the other hand, articulation axis prediction is more challenging than origin prediction for the small- and middle-sized objects.

We also show the performance of USDNet across the movable parts of different semantic categories. We select 6 dominant categories with significant numbers in both training and test set. In Fig. 18, we can see that USDNet performs better in the category of drawer (middle sized objects) than the other categories (small and large objects), which is in accordance with the results shown in Fig. 17. It is notice-

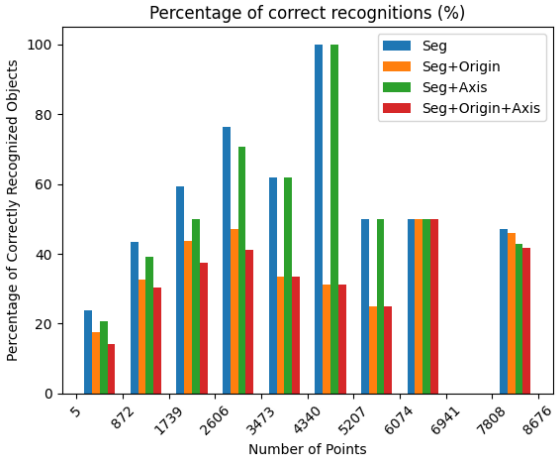


Figure 17. Percentage of correct recognitions v.s. size of movable parts

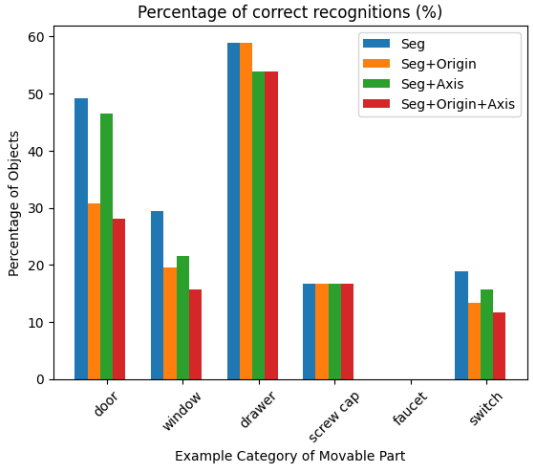


Figure 18. Percentage of correct recognitions v.s. categories of movable parts

able that USDNet fails in the prediction of all the faucets, which is mainly because the 3D meshes of faucets are not well reconstructed and incomplete due to the complicated geometry and reflective surfaces of them.

12. Details of Universal Scene Description

As mentioned in the main paper, USD organizes a scene into hierarchical entities, primitives (prims)–the building blocks representing all objects and relationships in the scene. Prims support a nested structure, where complex objects (e.g., cabinets) are represented as parent prims containing child sub-prims, modeling both individual items and grouped components of the scene.

Each prim can be assigned various attributes, e.g., position, scale, orientation, geometry, and appearance. Custom attributes can be introduced, enabling dataset-specific data to be embedded directly within the scene, such as physical properties (e.g., mass), material details, or semantic labels.

USD also supports joints that define movable connections, e.g., door hinges, as well as fixed joints. This enables the representation of both fixed and dynamic object relationships, making USD particularly suited for applications requiring detailed articulation and interaction modeling.

USD offers a highly robust, standardized format for representing complex 3D scenes, making it an ideal choice for Articulate3D. It supports rich 3D data representations, including mesh geometry, semantic segmentations, connectivity and articulation definitions, and physical attributes - all within a single, unified file. It offers an efficient, lightweight alternative to datasets like MultiScan [40], which require multiple scans of articulated objects in their open and closed states. USD's structure also supports non-destructive edits, facilitating scene manipulations.

USD's relevance and utility are increasingly recognized within the research community. NVIDIA's Isaac Sim [42, 46], built on USD, is gaining popularity as the state-of-the-art simulator for research with its high-fidelity environment for realistic physics-based simulations [23, 27, 56, 72, 91]. USD is also widely used in research going beyond robotics simulations for projects like procedural scene generation [52] and knowledge graph conversion for semantic querying [45] .