arXiv:2406.08474v2 [cs.CV] 13 Jun 2024

Real2Code: Reconstruct Articulated Objects via Code Generation

Zhao Mandi Stanford University
Yijia Weng Stanford University
Dominik Bauer Columbia University
Shuran Song Stanford University

Abstract

We present Real2Code, a novel approach to reconstructing articulated objects via code generation. Given visual observations of an object, we first reconstruct its part geometry using an image segmentation model and a shape completion model. We then represent the object parts with oriented bounding boxes, which are input to a fine-tuned large language model (LLM) to predict joint articulation as code. By leveraging pre-trained vision and language models, our approach scales elegantly with the number of articulated parts, and generalizes from synthetic training data to real world objects in unstructured environments. Experimental results demonstrate that Real2Code significantly outperforms previous state-of-the-art in reconstruction accuracy, and is the first approach to extrapolate beyond objects' structural complexity in the training set, and reconstructs objects with up to 10 articulated parts. When incorporated with a stereo reconstruction model, Real2Code also generalizes to real world objects from a handful of multi-view RGB images, without the need for depth or camera information. [1]

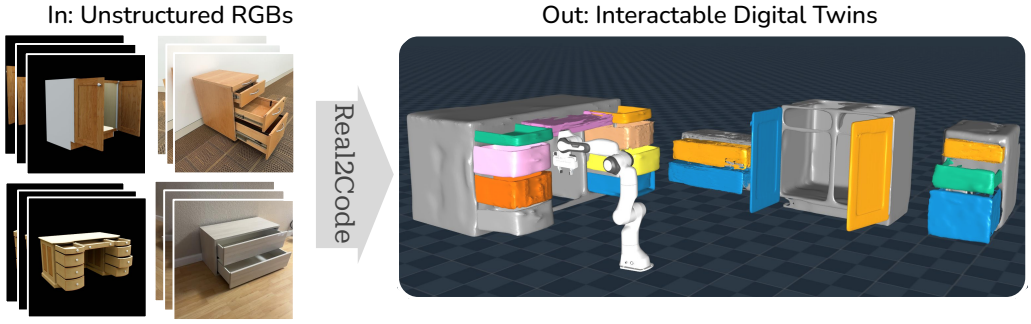

Figure 1: We propose a novel method for reconstructing articulated objects via code generation, leveraging pre-trained large language models (LLMs). Real2Code takes visual observations as input, and performs both part-level geometry reconstruction and joint prediction. When evaluated on an extensive set of real and synthetic objects with varying level of kinematic complexity, Real2Code can reconstruct complex articulated objects with up to 10 parts, and generalize to real world objects from a handful of pose-free RGB images.

1 Introduction

The ability to reconstruct real-world objects in simulation (real-to-sim) promises various downstream applications: automating asset creation for building VR/AR experiences, enabling embodied agents to verify their interaction in simulation before execution in the real world [1–3], or building large-scale

[1] Project Website: `https://real2code.github.io`

simulation environments that support data-driven policy learning [4]. We are particularly interested in articulated objects, for both their ubiquity in household and industrial settings and the unique challenges they pose in contrast to single-body rigid objects. To reconstruct articulated objects, prior learning-based methods typically train supervised [5] or test-time-optimized [6] models on synthetic objects with simple articulation structures (i.e., one or two moving parts per object). This results in limited generalization ability to objects with more complex visual appearances and kinematics. In addition, prior work only provides object part reconstructions of limited quality: the extracted meshes are often incomplete and the predicted articulation parameters require manual cleanup before being usable for simulation.

We propose **Real2Code**, a novel approach to address the above limitations. We represent object articulation with code programs, and use language modeling to predict these code programs from visual observations. This formulation scales elegantly with objects' structural complexity: to process an articulated object with multiple joints, prior methods would require either changing the output dimension of their articulation prediction model, or run multiple inferences on pairs of before- and after- interaction observations to predict one joint at a time. In contrast, the next-token prediction formulation in language modeling allows generating arbitrary-length outputs, i.e., the model architecture needs no adjustment to handle varying number of object joints. Whereas prior work on shape programs [7] needs to define task-specific code syntax, we represent objects with simulation code in Python, which takes advantage of recent progress in large language models (LLMs) that are pre-trained with code generation capabilities.

Although capable at code generation, LLMs pre-trained on text are not as equipped at predicting accurate numerical values from spatial geometry information, which is required in our task in order to obtain articulated joint parameters. To address this, we propose to use oriented bounding boxes (OBBs) as an abstraction layer that summarizes the raw sensory observation to the LLM in a concise yet precise manner. Given partial observations of an object, we first perform part-level segmentation and reconstruction via a combination of 2D segmentation and a 3D shape completion model; next, OBBs are extracted from the reconstructed object parts, and serve as input to the LLM. Instead of having to regress to continuous values, the LLMs can predict joints as a classification problem by selecting the closest OBB rotation axis and box edges.

In unstructured real world environments, another challenge is the lack of accurate depth and camera information. To address this, we incorporate a pre-trained dense stereo reconstruction model, namely DUSt3R[8], into our pipeline: we show the dense 2D-to-3D point-map prediction from DUSt3R can be combined with our fine-tuned SAM model to achieve view-consistent 3D segmentation. As a result, Real2Code can then reconstruct real world objects from only a handful of pose-free RGB images.

For more systematic evaluation, we validate Real2Code on the well-established PartNet-Mobility dataset [9], using an extensive test set of unseen objects that contain various numbers of articulated parts. Compared to the prior state of the art, Real2Code significantly improves both 3D reconstruction and joint prediction accuracy. Real2Code is the only approach to reliably reconstruct objects with more than three articulated parts, whereas prior methods fail completely on such objects. Fig. 1 highlights our results on both synthetic multi-part objects (left column of input images), where we show Real2Code can reconstruct objects with up to 10 articulated parts, and generalize to real world objects (right column of input images) using RGB images captured from in-the-wild furniture objects.

In summary, our contributions are threefold:

1. We present Real2Code, a novel approach to reconstructing articulated objects from a handful of unstructured RGB images. We formulate joint prediction as a code generation problem and adapt pre-trained large language models to specialize in this task.

2. We address part reconstruction via kinematic-aware view-consistent image segmentation and a learned 3D shape completion model, which leads to high-quality mesh extraction that generalizes to multi-part real-world objects.

3. Empirical results demonstrate that Real2Code significantly outperforms the prior state of the art at both articulation estimation and part reconstruction. To the best of our knowledge, Real2Code is the first method to accurately predict objects with more than three parts, and generalizes beyond the training dataset to objects with up to 10 articulated parts.

2 Related Work

LLMs for Visual Tasks. Pre-trained LLMs have been used for visual reasoning and grounding tasks[10, 11]. LLMs' code-generation capability has also been exploited for generating programs that solve visual tasks [12–14]. These works use zero-shot pre-trained LLMs such as GPTs [15, 16] and require prompt engineering, such as providing in-context examples, to guide the model to generate desirable outputs; in contrast, we directly fine-tune the weights of a code-generation model to specialize in our articulation prediction task, and do not use any hand-crafted prompting.

Shape Programs. Code-like programs have been studied in computer vision as a compact representation for 2D and 3D shapes. A main challenge for learning code programs is the lack of supervision, and prior work has explored using learned differentiable code executor [7], pseudo-labeling [17], differentiable rendering [18], imitation learning on code sequences [19], or reinforcement learning [20]. More recent work has explored constructing large-scale datasets of shapes[21] or scene layouts[22] and train supervised LLM-like models to generate code outputs. In contrast to ours, these prior work focuses on either individual object shapes or scene-level room layouts, but does not estimate joint articulations. In addition, instead of the task-specific code programs, such as customarily-designed language syntax [7, 17, 22] or Computer-Aided Design (CAD) code [19, 21], we represent object articulation with Python code that 1) closes matches the pre-training distribution of code-generation LLMs, which allows fine-tuning with limited data, and 2) can be directly executed by a physics simulator [23], which makes the reconstruction more usable for and requires less manual cleanup.

Articulation Model Estimation. Prior work has investigated estimating pose and joint properties of articulated objects *without* full reconstruction. A common setup is to assume physical interactions on an object to infer its articulation information: classical sampling-based algorithms[24, 25] are proposed to estimate joint parameters based on sensory inputs from an object's different configuration states; other learning-based methods train end-to-end models to predict part-level segmentation, kinematic structure, object part poses, or articulated joint parameters [26–36]. Some propose specialized neural network architectures to improve estimation performance[37–39]. Other work focuses on learning to propose the most informative physical interactions on an object to help robot manipulation[40], or to better isolate and segment articulated parts and joints[41]. These articulation estimation tasks provide useful metrics for 3D shape reasoning [28], and the predicted object pose and joint information are shown useful for robot task learning [42–45]. However, prior work typically handles objects with simple structure (i.e.,one or two moving parts) and does not address full object reconstruction. In contrast, our method handles objects with more than ten moving parts, and performs shape reconstruction via extracting part meshes.

Articulated Object Reconstruction. The most closely related to ours are methods that reconstruct both the geometry and joints of articulated objects. A popular approach is to train end-to-end models on synthetic data to jointly segment articulated parts and predict joint parameters, assuming either observations from interactions [5, 46–48] or single-stage [49–51] observations. Another approach uses per-object optimization [6, 36] without training. Based on observations of the object at two or more different joint states, it optimizes for joint parameters to match observed motion correspondence and optionally performs 3D reconstruction by extracting from learned neural rendering fields. Most existing methods assume a single joint and do not scale well with number of joints: for example, to handle an object with N joints, methods like Ditto [5] would need to move the N joints one by one, record the observations before and after each interaction, and run N inferences on each observation pair. PARIS [6] would need to optimize N neural fields and joint parameters, which may lead to a much more complex optimization landscape. The approach presented in [36] handles multiple joints but requires a complete sequence of point-cloud observations and is not able to reconstruct 3D shapes.

3 Method

We address the problem of reconstructing multi-part articulated objects from visual observations. An articulated object is composed of a set of rigid-body parts that are connected via joints. We assume joint types are either prismatic or revolute: a prismatic joint is parameterized by a joint axis $\mathbf{u}^p \in \mathbb{R}^3$ and a translation offset d; a revolute joint is parameterized by a position $\mathbf{p}^r \in \mathbb{R}$, a rotation axis $\mathbf{u}^r \in \mathbb{R}^3$, and a rotation angle θ. For an object with N moving parts, we assume each to be connected with its parent via exactly one 1-DoF joint. Therefore, the transformation between each part's frame

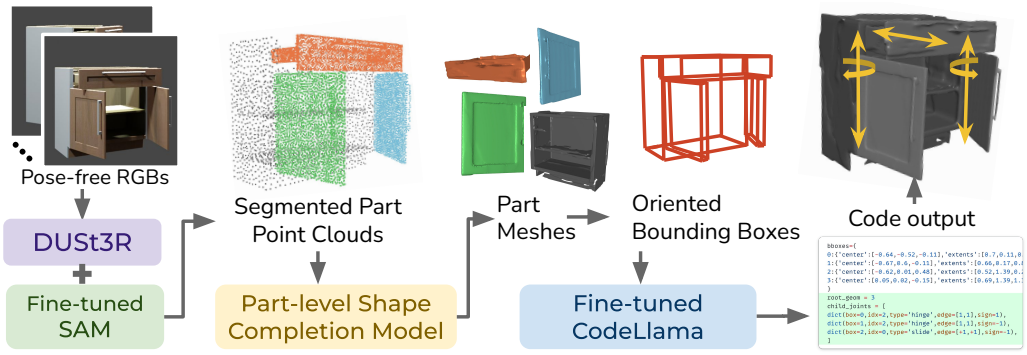

Figure 2: Overview of our proposed pipeline. Given unstructured multi-view RGB images, we leverage the pre-trained DUSt3R model [8] to obtain dense 2D-to-3D pointmaps, and use a fine-tuned 2D segmentation model[52] to perform part-level segmentation and project to segmented 3D point clouds. A learned shape-completion model takes partial point cloud inputs and predicts a dense occupancy field, which is used for part-level mesh extraction. We fine-tune a large language model (LLM) [53] that takes mesh information in the form of oriented bounding boxes, and outputs full code descriptions of the object that can directly be executed in simulation.

and its parent's frame is uniquely determined by the joint parameters — this observation motivates our OBB-based formulation described in Sec. 3.2. To obtain visual input of our system, we assume an object is manipulated such that each of its articulated joints is at a non-zero state, i.e.,$d > 0$ or $\theta > 0$, and record a set of RGB (and optionally depth) images of the object. Our system outputs a set of 3D meshes – each a reconstruction of the object's parts – and a list of joint types and parameters represented as code. The meshes and joints can then be used to create the object's digital twin in simulation and enable downstream applications.

Fig. 2 provides an overview of our proposed method. Real2Code consists of two main steps: reconstruction of object parts' geometry (Sec. 3.1) and joint estimation via LLM code generation (Sec. 3.2). Between the two steps, the oriented bounding boxes (OBBs) of the object parts serve as an abstraction layer, enabling the LLM to reason about 3D spatial information and predict accurate joint parameters. We introduce the two main steps in the following two sections.

3.1 Part Reconstruction

To reconstruct an object's part-level shape geometry, we propose a 2D-to-3D approach that is category-agnostic and able to address objects with arbitrary number of parts. First, we fine-tune a SAM model that generates 2D segmentations from RGB images, and projects them to 3D point clouds. Next, we train a shape completion model that extracts watertight meshes from the partially-observed point clouds.

3.1.1 Kinematics-aware Part Segmentation

We leverage a pre-trained 2D segmentation model to first segment object parts based on their kinematic structure. This design is motivated by the need to 1) generalize to real world data, and 2) scalably segment arbitrary number of object parts. In contrast to prior works that train 3D segmentation models with limited synthetic data [5, 9, 54], the SAM [52] model was pre-trained on a much larger scale dataset and hence generalizes well to in-the-wild real world images. Further, in contrast to prior works that infer articulation from multiple object states, we leverage SAM's strong prior to identify object parts without the need for multi-step interactions.

However, because SAM [52] is originally designed for iterative user prompting, its zero-shot predictions display uncontrollable granularity on articulated objects, i.e.,segmenting unnecessary details that require additional user input to refine. To address this, we propose to adapt the pre-trained SAM using the PartNet-Mobility [9, 54] dataset: while keeping the model's heavy-weight image encoder frozen, we fine-tune the lightweight prompt-decoder layer of SAM to take an image and one sampled 2D point prompt as input, and predict the corresponding mask that matches the object's kinematic structure. More details on the fine-tuning process are reported in appendix A.2.

3.1.2 Test-time Prompting for View-consistent Segmentation.

The point-based segmentation described above allows our fine-tuned SAM to scale easily with the number of the object parts. However, this formulation also inherently lacks view consistency, as SAM is unaware of the object part correspondences across different camera views. To address this, we introduce a test-time prompting procedure to project predicted 2D masks into a view-consistent 3D segmentation. We discuss two different input settings based on the availability of depth and camera data: **1) Multi-view RGB-D and Camera Input**: we first coarsely sample 2D points on each RGB image and run the SAM model to obtain the background masks. This allows us to segment the foreground object in the different views and sample 3D points uniformly on the point cloud. Next, we project each such 3D point back onto each image, and obtain view-consistent 2D points for SAM prompting. Further, we rank the model predicted masks based on the confidence and stability scores proposed in [52], and filter them using non-maximum suppression (NMS) to produce the final 3D segmentation. **2) Multi-view Unstructured RGB Input.** To handle real world settings which often lack high-quality depth and camera information, we adopt a multi-view stereo reconstruction model to achieve part segmentation from unstructured, pose-free RGB images. We use the recently proposed DUSt3R[8] model, which is pre-trained to predict dense 3D point-maps from RGB input images. We then sample 2D points from one RGB image and find each point's corresponding point in every other RGB images via nearest-neighbor. More details are described in appendix A.3. This overall procedure of projecting between 3D to 2D prompting is similar to SA3D [55], which samples on a NeRF[56] field and uses inverse rendering to effectively prompt SAM in 3D.

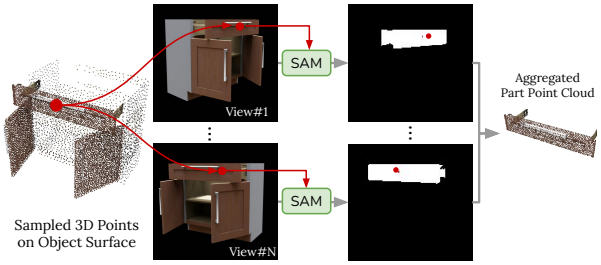

Figure 3: **View-consistent segmentation.** Illustration of our method for test-time prompting the fine-tuned SAM model. We first sample 3D points from the foreground object point clouds, project each point onto 2D RGB images captured from different camera views, which are used to prompt the model to generate view-consistent segmentations.

3.1.3 Part-level Shape Completion.

Due to frequent self-occlusion in articulated objects, e.g. the inside of a drawer is often not visible, RGB-D input does not provide full observation of each object part, and subsequently a segmented point cloud does not recover complete shape geometries. This motivates learning a shape completion model to obtain watertight meshes. Because part-segmentation is already handled in the previous step, we here tackle shape completion on the object part level. We build on top of Convolutional Occupancy Nets [57]: the model architecture consists of a PointNet++[58] point-cloud encoder, followed by a 3D Unet [59] encoder and a linear MLP decoder that predicts logits for occupancy. We use the ground-truth part meshes from PartNet-Mobility[9] to generate a dataset of partial point cloud inputs and occupancy labels. We normalize the occupancy grid using *partial* OBBs extracted from the input point cloud to avoid under-fitting the smaller-sized meshes. Marching Cubes [60] is used to extract the completed part meshes from predicted occupancy. See appendix A.2 for more details on our shape completion dataset and model training.

3.2 Articulation Prediction via Code Generation

Given a set of segmented object parts, the next step is to predict the articulation structure that connects the parts. Our approach of using LLM code generation offers several advantages: first, code is a compact representation for articulation, and when combined with LLM's ability to predict arbitrary-length outputs, it scales elegantly with the complexity of an object's kinematic structure; second, pre-trained LLMs are equipped with strong priors for both common-sense objects and for generating syntactically correct code, making them easily adaptable to our task; lastly, the LLM-generated code can be directly executed in simulation, removing the need for manual cleanup of predicted joint parameters, which is required by prior work [5]. The following sub-sections first introduce our formulation of joint parameters w.r.t. OBBs, then discuss our proposed fine-tuning procedure that adapts a pre-trained LLM to our articulation prediction task.

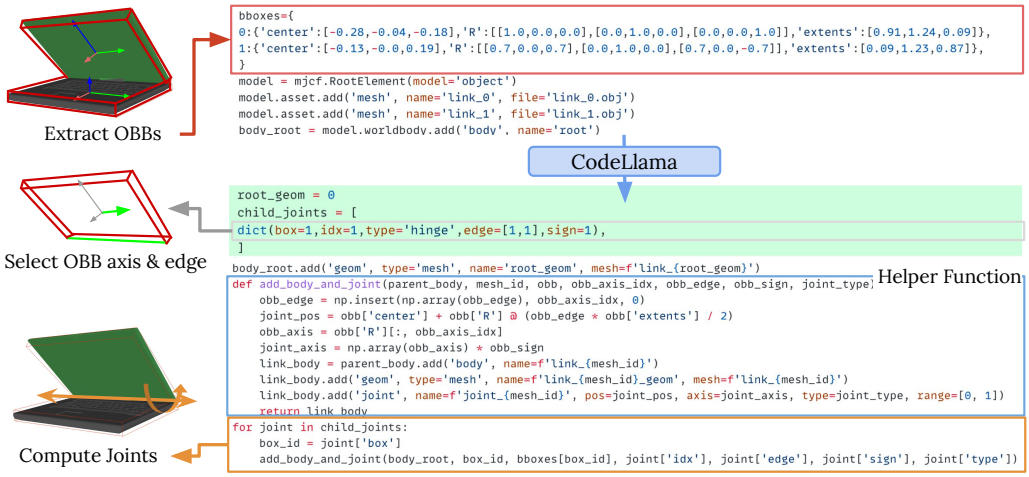

Figure 4: **Articulation Prediction as Code.** We fine-tune a Codellama model that takes in oriented bounding boxes (OBBs) for segmented parts as input, and generates joint predictions via selecting OBB rotation axes and edges (model generation is highlighted in green). A helper function is used to compute the absolute joint axis and position that assembles the object parts in simulation

3.2.1 Oriented Bounding Box as Input Abstraction.

Articulation prediction requires numerical precision at joint parameters (i.e., position and rotation) and reasoning from raw sensory input, but an LLM pre-trained on text is not naturally adept at these challenges. We address this by representing the sensory input (object point clouds) as a set of oriented bounding boxes (OBBs), each representing a segmented and completed object part. Compared to alternative object representations such as 3D point clouds or 2D images, OBBs strike a balance between **compactness** (i.e., do not require an extra feature encoder) and **preciseness** (i.e., provide numerical 3D pose information). Further, OBBs provide a reference for joint information. Recall that the pose of an object part is determined by its 1-DoF joint at a non-zero state — we can hence recover joint parameters from the observed displacement of object parts. Given an OBB of a part connected to its parent, the joint axis will be parallel to one of the three axes of the OBB's rotation matrix regardless of its joint type. We also observe many common articulated objects consist of cuboid-like parts (e.g. doors or laptops), hence the position of their corresponding revolute joints will lie close to one of the OBB edges. Combining these observations, we can re-formulate the joint axis prediction problem by selecting an OBB rotation axis as the joint axis and, for revolute joints, choosing an OBB edge parallel to the axis as the joint position. See Fig. 4 for an illustration.

3.2.2 Fine-tuning a Code Generation LLM.

We now have an input formulation that effectively converts a regression task (i.e., predicting continuous values) to an easier classification task (i.e., selecting axes and edges) for LLMs. We use the 7B-CodeLlama [53] model for its open-source-availability and built-in priors for code generation. We construct a fine-tuning dataset using PartNet [9] objects (the same asset used to generate our segmentation and shape completion data above), where the native URDF files are converted into MJCF code [61] that 1) is in the more compact Python syntax, 2) can be executed in MuJoCo [23] physics simulation, and 3) has each object's joints assigned with respect to the corresponding part's OBB information. The LLM takes a list of part-OBB information (i.e., center, rotation, and half-lengths) as input, and outputs joint predictions as a list, where each line contains indices into the axes and edges of an OBB. More details on LLM fine-tuning can be found in appendix A.2

4 Experiments

We evaluate Real2Code and compare to baseline methods to validate the effectiveness of our approach. Sec. 4.2 describes experiments on our kinematics-aware 2D image segmentation and 3D shape completion models. Sec. 4.3 evaluates our fine-tuned code-generation model on articulation prediction.

Category	Laptop		Box		Fridge		Furniture		Furniture	
Number of Parts	2		2		2-3		2-4		5-15	
Metric	whole	part	whole	part	whole	part	whole	part	whole	part
Real2Code+gtSeg	0.57	2.33	1.54	7.65	0.51	2.04	1.46	13.3	5.84	16.8
Ditto	2.54	**2.04**	1.73	82.82	2.80	462.25	**2.25**	1105.86	**2.21**	4608.08
PARIS	84.29	206.31	15.35	158.73	20.63	1297.27	6.02	544.64	11.44	816.86
Real2Code-SegOnly	1.74	7.19	11.46	10.52	0.90	23.44	17.43	206.49	N/A	N/A
Real2Code (Ours)	**0.44**	3.02	**1.31**	**5.94**	**0.60**	**1.28**	3.47	**65.79**	19.70	**118.58**

Table 1: We evaluate surface reconstruction quality by measuring Chamfer-Distance (CD) between predicted and ground-truth meshes. Results are reported separately for each object category, where we take average CD of objects' entire surface reconstruction ('whole' column) and of all part wholes ('part' column). Objects from Storage-Furniture and Table are reported under Furniture and divided based on the number of parts.

Sec. 4.5 contains ablation studies that provide additional insights into our method. In Sec. 4.6, we test our trained models on real world object data and qualitatively demonstrate the generalization ability of Real2Code.

4.1 Experiment Setup

Datasets. We use assets from five categories in PartNet-Mobility[9] dataset: Laptop, Box, Refrigerator, Storage-Furniture and Table. The same split of 467 train and 35 test objects are used to construct our image segmentation, shape completion, and code datasets. We use Blender [62, 63] to render RGB-D and ground-truth segmentation masks for each object. The RGB-D images and masks are then used to generate part-level point clouds as partial observations. For code data, we extract OBBs from part meshes and process each object's raw URDF file into Python MJCF[61] code, where the rotation and position of each joint are relative to the OBB of the child part that this joint connects to the parent part. Refer to appendix A.1 for more details on our dataset construction.

Baselines. We compare Real2Code to the following baseline methods:

- **PARIS** [6] is the prior state-of-the-art for articulated object reconstruction. It takes multi-view RGB observations of a two-part articulated object at two different joint states, then optimizes per-part NeRF-based reconstructions and joint parameters based on motion cues from the two observed states. We render our test objects at two random joint states, run their proposed optimization procedure with 5 random initialization seeds, and report the average performance. To handle more complex objects, we modify their method to optimize for more than two parts at once. However, we observe that their design of optimizing one neural field for each part runs out of memory when the number of joints exceeds 4.

- **Ditto** [5] is an end-to-end learning method that takes in a pair of before- and after-interaction point cloud inputs and predicts implicit part shape reconstructions and joint parameters. Notably, Ditto assumes only one object part is moved at a time, which requires step-by-step interactions and observations, making evaluation less efficient. For an object with N joints, we move one part at a time, render the corresponding N pairs of point cloud observations, and run their pre-trained model N times to obtain joint parameter predictions and reconstructions of moved parts.

- **GPT-4** [16] is representative of recent state-of-the-art LLMs with strong reasoning and code-generation capability. We use it as a reference for zero-shot LLM performance on our desired task without fine-tuning. We prompt it with the same code header used in our LLM fine-tuning dataset, plus additional instructions on how to format the output, which our fine-tuned LLM does not need.

4.2 Part Segmentation and Reconstruction Experiments

3D Part-level Shape Completion.

Following the prompting procedure described in Sec. 3.1, we first run our fine-tuned SAM on images from the test set of 35 unseen objects and obtain segmented part point clouds. We observe that, because we rank and filter the mask predictions (i.e., prioritize high predicted confidence score and stability score), the low-quality masks have less impact on the final segmented point-cloud after

	2 Parts (15)			3 Parts (9)			4-5 Parts (6)			6-15 Parts (7)		
	rot↓	pos↓	type↑	rot↓	pos↓	type↑	rot↓	pos↓	type↑	rot↓	pos↓	type↑
Real2Code+gtBB	0.0	0.07	0.93	0.0	0.04	1.00	0.0	0.04	1.00	11.6	0.03	0.94
Ditto	40.04	4.04	0.57	35.57	2.47	0.70	49.77	3.20	0.43	63.06	4.16	0.30
PARIS	48.44	2.67	0.51	32.35	3.63	0.84	55.97	2.14	0.43	N/A	N/A	N/A
GPT4	57.3	0.26	0.73	10.0	0.08	0.61	45.0	0.21	0.51	**30.0**	**0.05**	0.71
Real2Code (Ours)	**7.5**	**0.08**	**0.80**	**0.0**	**0.04**	**0.89**	**0.63**	**0.07**	**0.97**	30.2	**0.05**	**0.89**

Table 2: Joint prediction results from Real2Code and baseline methods, grouped by the number of moving parts in each object. We remark that Real2Code consistently outperforms baseline methods across objects with different kinematic structures; on objects with 4 or more moving parts, Real2Code predicts joints accurately whereas baseline methods fail.

the projection step. Next, we use the segmented part point clouds as input to evaluate our learned shape completion model, and use Marching Cubes [60] on the occupancy predictions to extract meshes. Following prior work [48, 5, 6], we uniformly sample $10,000$ points on the extracted mesh surface and report the average Chamfer Distance between the extracted and ground-truth part meshes in Tab. 1. Because the predictions are semantics-agnostic, we generate permutations of the set of predicted meshes and take the permutation that results in lowest error; the same logic is used for joint prediction results.

We remark on the performance difference between Real2Code and baseline methods: the joint optimization of shapes and joints in PARIS [6] suffers from a complex loss landscape and produces unsatisfactory reconstructions, especially when the number of parts increases. Ditto [5] performs well on training categories (i.e., Laptop) but does not generalize well to unseen categories. In contrast, we factorize the problem into segmentation and shape completion, aggregate 2D segmentation from fine-tuned SAM and perform shape completion in a per-part fashion leads to better results.

Due to the partial observation and noise in the segmentation masks, simply extracting meshes from the grouped point clouds also results in subpar reconstruction results (see column 'Real2Code-SegOnly', where 'N/A' indicates the mesh extractions are too noisy to match with GT mesh). This further validates the need for our shape completion model. Overall, Real2Code achieves the best reconstruction quality and elegantly scales to a larger number of parts.

Kinematics-aware 2D Image Segmentation.

To demonstrate the effectiveness of SAM fine-tuning, we evaluate the fine-tuned model on unseen object images by uniformly sampling a grid of 32×32 query points and compare the predicted segmentation with ground-truth masks. We use NMS filtering on the predicted masks, then by sorting with the model's predicted confidence score to take the top-K masks that fill the image to more than 95% total pixels. We observe a significant improvement over zero-shot SAM: object parts are segmented much more closely following their kinematics structure, obtaining a 92% mean IoU score on the final used masks and 84% match rate to ground-truth masks.

4.3 Articulation Prediction Experiments

After completing part-level reconstruction on the test objects, we extract OBBs for each object part and compose a text-prompt for our fine-tuned CodeLlama model. We parse the model's code generation and append it with code header lines (e.g. import packages) such that the post-processed code can be directly executed to produce object simulation. We then evaluate the accuracy of articulation prediction by measuring the error of joint type, joint axis, and (for revolute joints only) joint position predictions.

As shown in Tab. 2, we outperform all baseline methods by a large margin. The effectiveness of our OBB abstraction is further accentuated by Real2Code+gtBB, where we feed oracle OBB to the code generation module and achieve highly accurate predictions even on unseen objects with a large number of parts.

4.4 Qualitative Results

For qualitative results, we select objects with a range of varying kinematic complexities, from a two-part laptop to a ten-part multi-drawer table. We visualize the final reconstructed objects from ours and

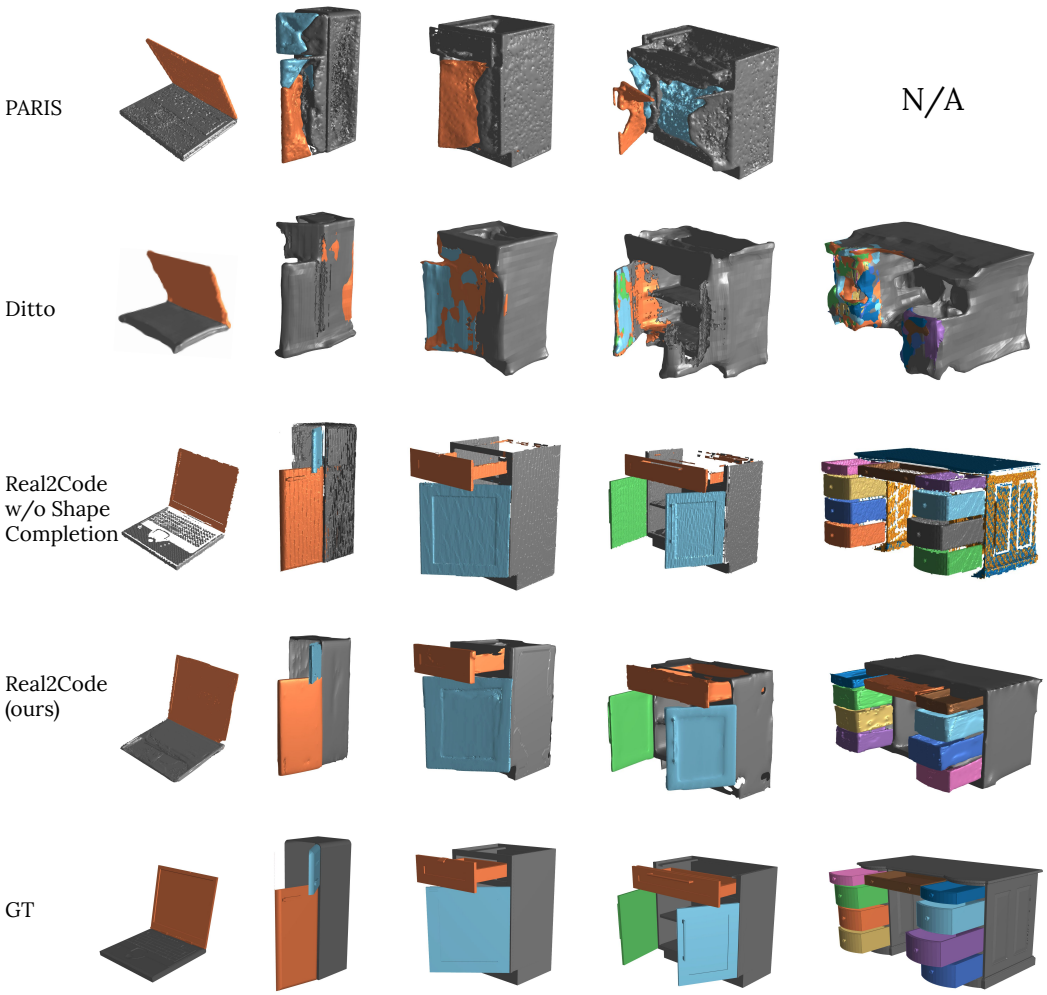

Figure 5: Qualitative results that compare Real2Code to baseline methods. We show results from objects with a range of varying kinematic complexities, from a two-part laptop to a ten-part multi-drawer table. Whereas all methods can handle the simpler laptop articulation, baseline methods struggle as the number of object parts increases, and Real2Code performs reconstruction much more accurately. PARIS runs out of memory and fails to run on the ten-part table ('N/A').

baselines methods in Fig. 5. Whereas all methods can handle the simpler laptop articulation, baseline methods struggle as the number of object part increases, and Real2Code performs reconstruction much more accurately.

4.5 Ablation Studies

To further validate our formulation of using OBB as reference for articulation prediction, we provide additional ablation experiments that use alternative input and output representation:

- **Regression on Joint Parameters.** Instead of selecting OBB rotation axis and edge, we fine-tune two more CodeLlama models to take the same input but outputs continuous numerical values for joint parameters: the first model directly predicts 3 values for each joint axis and 3 for every joint position (Sec. 4.5 row 'OBB Abs.'); the second model predicts joint axis the same way as Real2Code, but predicts joint position as a relative position to the OBB's center (Sec. 4.5 column 'OBB Rel.').

- **Provide LLM with Visual Inputs.** To verify whether OBBs provide sufficient spatial information, we fine-tuned another model with both RGB and OBB inputs. We adopt the OpenFlamingo [64, 65] approach for interleaving image embeddings with the CodeLlama model weights, and uses the same pre-trained ViT [66] weights for image encoder.

Inp.	Out	2 Parts (15) rot↓	pos↓	type↑	3 Parts (9) rot↓	pos↓	type↑	4-5 Parts (6) rot↓	pos↓	type↑	6-15 Parts (6) rot↓	pos↓	type↑
OBB	Abs.	7.5	**0.06**	0.92	**0.0**	**0.03**	**1.0**	**0.0**	0.6	0.83	**0.0**	0.7	0.73
OBB	Rot.	0.0	0.18	0.73	0.3	0.23	1.00	0.9	0.19	0.83	5.9	0.06	0.59
+RGB	Cls.	**0.0**	**0.06**	0.80	5.0	**0.03**	**1.0**	**0.0**	**0.03**	0.89	**0.0**	**0.02**	0.67
OBB	Cls.	**0.0**	0.07	**0.93**	**0.0**	0.04	**1.0**	**0.0**	0.04	**1.00**	11.6	0.03	**0.94**

Table 3: Joint prediction results from ablation experiments on Real2Code. Using the 'Regress' output formulation, the LLM is still able to output reasonable values for two or three part objects, but generates much less accurate joint positions when the number of articulated parts increase. Adding additional RGB image input yields no clear improvements from the model, which suggests the OBB input alone can provide sufficient information for articulation.

Results from the ablation experiments are reported in Tab. 3. We make the following remarks:

Regression formulation predicts less accurate joint positions. Both predicting absolute joint positions (column 'OBB Abs.') and predicting relative position from OBB center (column 'OBB Rot.') yield a higher prediction error than selecting OBB edges as joint position. In contrary, the rotation error is still on a reasonable scale: we found this is due to the model has learned to copy the correct axis column from the OBB rotation matrices contained in the input prompt. This further validates the effectiveness of using OBB as spatial representation.

RGB input does not yield significant improvement. We draw this conclusion from comparing row '+RGB Rel.' and 'OBB Rel.'. This suggests the OBB input provides sufficient information for articulation prediction task.

4.6 Experiments on Real World Objects

To validate the generalization ability of Real2Code, we gather a set of in-the-wild articulated objects and collect multi-view RGB data as inputs. We run Real2Code with DUSt3R[8]. Due to the achieve reconstruction from multi-view pose-free RGB images. Due to the lack of quantitative metrics, we show qualitative results in Fig. 7 that Real2Code generalizes well to in-the-wild objects and produces good quality reconstructions from RGB-only inputs. However, although the learned DUSt3R[8] model performs well on overall shape and exterior surface areas of the objects, it predicts less accurate point maps at areas inside the drawers, which is likely due to the lack of similar data in their training dataset. As a result, the segmented part point clouds display noises (second row in Fig. 7), which leads too lower quality mesh extraction from the shape completion model. See appendix A.3 for more details on our evaluation setup.

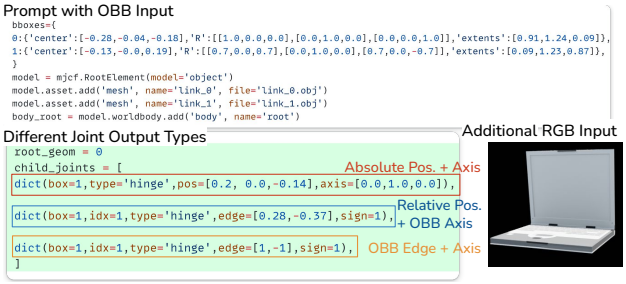

Figure 6: Qualitative comparison of the code output format in our ablation experiments. Each prediction format occupies one line. In 'Absolute Pos. + Axis', the LLM outputs continuous position and axis values; in 'Relative Pos. + OBB Axis', the LLM outputs one index into the OBB's rotation axis, and a 2D joint position relative to the selected axis; Real2Code uses 'OBB Edge + Axis', where LLM outputs index to rotation axes in an OBB, and two values to indicate the OBB edge. Bottom right of the figure shows one example of additional RGB image input to the LLM.

5 Limitations

In its current form, Real2Code still has a few limitations that point to interesting directions for future work: 1) Real2Code only considers single object with many parts, extending it to multiple object scenes would require additional object detection and preprocessing. 2) Real2Code only predicts joint parameters in terms of their type, position, and axis. To infer other joint parameters, such as joint range and friction world require additional multi-step interactions and observations. 3) We found that

Figure 7: We evaluate Real2Code on real world objects using RGB data. For each object, we use 10 pose-free RGB images captured in-the-wild and run Real2Code with DUSt3R[8]. We show one example RGB input (1st row), segmented point clouds (2nd row) and full reconstruction (3rd row) for each object.

the articulation prediction accuracy is sensitive to failures in the first 2D image segmentation module, i.e., OBBs from wrong segmentations directly obstruct the LLM reasoning of object structures; this can be improved by providing human corrective feedback as proposed in [52], i.e., a user provides additional points and prompt the model to adjust its mask predictions.

6 Conclusion

We present Real2Code, a novel method for reconstructing articulated objects that leverages code generation capability in pre-trained LLMs. We empirically show that Real2Code achieves a new state-of-the-art in both geometry reconstruction and articulation prediction and can successfully reconstruct objects with complex kinematic structures with an arbitrary number of parts, whereas prior methods fail. By reliably translating visual observation to simulatable models, we hope Real2Code unlocks new opportunities in robotics and mixed reality applications.

Acknowledgements

The authors would like to thank Zhenjia Xu for the real world data collection, Samir Gadre for helpful pointers on LLM fine-tuning, Cheng Chi for help with training our shape completion model training, and all members of REALab: Zeyi Liu, Xiaomeng Xu, Chuer Pan, Huy Ha, Yihuai Gao, Mengda Xu, Austin Patel, et. al. for valuable feedback and discussion on the paper manuscript. We also thank Stanford EE department admins Kenny Green, Steve B. Cousins and Mary K. McMahon for their support during real world data collection. This work was supported in part by the Toyota Research Institute, NSF Award #2143601, Sloan Fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.

References

[1] Lim, V., H. Huang, L. Y. Chen, et al. Planar robot casting with real2sim2real self-supervised learning, 2022. 1

[2] Wang, L., R. Guo, Q. Vuong, et al. A real2sim2real method for robust object grasping with neural surface reconstruction, 2023.

[3] Torne, M., A. Simeonov, Z. Li, et al. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation, 2024. 1

[4] Katara, P., Z. Xian, K. Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models, 2023. 2

[5] Jiang, Z., C.-C. Hsu, Y. Zhu. Ditto: Building digital twins of articulated objects from interaction, 2022. 2, 3, 4, 5, 7, 8

[6] Liu, J., A. Mahdavi-Amiri, M. Savva. Paris: Part-level reconstruction and motion analysis for articulated objects, 2023. 2, 3, 7, 8

[7] Tian, Y., A. Luo, X. Sun, et al. Learning to infer and execute 3d shape programs, 2019. 2, 3

[8] Wang, S., V. Leroy, Y. Cabon, et al. Dust3r: Geometric 3d vision made easy, 2023. 2, 4, 5, 10, 11, 16

[9] Mo, K., S. Zhu, A. X. Chang, et al. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019. 2, 4, 5, 6, 7, 15, 16

[10] Zeng, A., M. Attarian, B. Ichter, et al. Socratic models: Composing zero-shot multimodal reasoning with language, 2022. 3

[11] Hsu, J., J. Mao, J. Wu. Ns3d: Neuro-symbolic grounding of 3d objects and relations, 2023. 3

[12] Gupta, T., A. Kembhavi. Visual programming: Compositional visual reasoning without training, 2022. 3

[13] Surís, D., S. Menon, C. Vondrick. Vipergpt: Visual inference via python execution for reasoning, 2023.

[14] Subramanian, S., M. Narasimhan, K. Khangaonkar, et al. Modular visual question answering via code generation, 2023. 3

[15] Brown, T. B., B. Mann, N. Ryder, et al. Language models are few-shot learners, 2020. 3

[16] OpenAI. Gpt-4 technical report, 2023. 3, 7

[17] Jones, R. K., H. Walke, D. Ritchie. Plad: Learning to infer shape programs with pseudo-labels and approximate distributions, 2022. 3

[18] Liang, Y.-C. Learning to infer 3d shape programs with differentiable renderer. *ArXiv*, abs/2206.12675, 2022. 3

[19] Willis, K. D. D., Y. Pu, J. Luo, et al. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences, 2021. 3

[20] Tulsiani, S., H. Su, L. J. Guibas, et al. Learning shape abstractions by assembling volumetric primitives. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1466–1474, 2016. 3

[21] Ganin, Y., S. Bartunov, Y. Li, et al. Computer-aided design as language, 2021. 3

[22] Avetisyan, A., C. Xie, H. Howard-Jenkins, et al. Scenescript: Reconstructing scenes with an autoregressive structured language model, 2024. 3

[23] Todorov, E., T. Erez, Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. 2012. 3, 6, 17

[24] Huang, X., I. D. Walker, S. Birchfield. Occlusion-aware multi-view reconstruction of articulated objects for manipulation. *Robotics Auton. Syst.*, 62:497–505, 2014. 3

[25] Katz, D., M. Kazemi, J. A. Bagnell, et al. Interactive segmentation, tracking, and kinematic modeling of unknown 3d articulated objects. In *2013 IEEE International Conference on Robotics and Automation*, pages 5003–5010. IEEE, 2013. 3

[26] Hu, R., W. Li, O. Van Kaick, et al. Learning to predict part mobility from a single static snapshot. *ACM Transactions on Graphics (TOG)*, 36(6):1–13, 2017. 3

[27] Yi, L., H. Huang, D. Liu, et al. Deep part induction from articulated object pairs. *ACM Transactions on Graphics*, 37(6):1–15, 2018.

[28] Wang, X., B. Zhou, Y. Shi, et al. Shape2motion: Joint analysis of motion parts and attributes from 3d shapes, 2019. 3

[29] Michel, F., A. Krull, E. Brachmann, et al. Pose estimation of kinematic chain instances via object coordinate regression. In *British Machine Vision Conference*. 2015.

[30] Li, X., H. Wang, L. Yi, et al. Category-level articulated object pose estimation, 2020.

[31] Zeng, V., T. E. Lee, J. Liang, et al. Visual identification of articulated object parts. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2443–2450. IEEE, 2021.

[32] Huang, J., H. Wang, T. Birdal, et al. Multibodysync: Multi-body segmentation and motion estimation via 3d scan synchronization. In *Proceedings of the Computer Vision and Pattern Recognition (CVPR)*. 2021.

[33] Tseng, W.-C., H. Liao, Y.-C. Lin, et al. Cla-nerf: Category-level articulated neural radiance field. *2022 International Conference on Robotics and Automation (ICRA)*, pages 8454–8460, 2022.

[34] Abdul-Rashid, H., M. Freeman, B. Abbatematteo, et al. Learning to infer kinematic hierarchies for novel object instances. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8461–8467. IEEE, 2022.

[35] Jiang, H., Y. Mao, M. Savva, et al. Opd: Single-view 3d openable part detection. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIX*, pages 410–426. Springer, 2022.

[36] Liu, S., S. Gupta, S. Wang. Building rearticulable models for arbitrary 3d objects from 4d point clouds. In *Proceedings of the Computer Vision and Pattern Recognition (CVPR)*. 2023. 3

[37] Buchanan, R., A. Röfer, J. Moura, et al. Online estimation of articulated objects with factor graphs using vision and proprioceptive sensing. *ArXiv*, abs/2309.16343, 2023. 3

[38] Heppert, N., T. Migimatsu, B. Yi, et al. Category-independent articulated object tracking with factor graphs. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022.

[39] Sun, X., H. Jiang, M. Savva, et al. Opdmulti: Openable part detection for multiple objects. *ArXiv*, abs/2303.14087, 2023. 3

[40] Mo, K., L. Guibas, M. Mukadam, et al. Where2act: From pixels to actions for articulated 3d objects, 2021. 3

[41] Gadre, S. Y., K. Ehsani, S. Song. Act the part: Learning interaction strategies for articulated object part discovery, 2021. 3

[42] Liu, L., W. Xu, H. Fu, et al. Akb-48: A real-world articulated object knowledge base, 2022. 3

[43] An, B., Y. Geng, K. Chen, et al. Rgbmanip: Monocular image-based robotic manipulation through active object pose estimation. *ArXiv*, abs/2310.03478, 2023.

[44] Geng, H., H. Xu, C. Zhao, et al. Gapartnet: Cross-category domain-generalizable object perception and manipulation via generalizable and actionable parts, 2023.

[45] Geng, H., Z. Li, Y. Geng, et al. Partmanip: Learning cross-category generalizable part manipulation policy from point cloud observations, 2023. 3

[46] Hsu, C.-C., Z. Jiang, Y. Zhu. Ditto in the house: Building articulation models of indoor scenes through interactive perception. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3933–3939, 2023. 3

[47] Nie, N., S. Y. Gadre, K. Ehsani, et al. Structure from action: Learning interactions for articulated object 3d structure discovery, 2023.

[48] Mu, J., W. Qiu, A. Kortylewski, et al. A-sdf: Learning disentangled signed distance functions for articulated shape representation, 2021. 3, 8

[49] Heppert, N., M. Z. Irshad, S. Zakharov, et al. Carto: Category and joint agnostic reconstruction of articulated objects. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21201–21210, 2023. 3

[50] Kawana, Y., Y. Mukuta, T. Harada. Unsupervised pose-aware part decomposition for man-made articulated objects. In *European Conference on Computer Vision*, pages 558–575. Springer, 2022.

[51] Wei, F., R. Chabra, L. Ma, et al. Self-supervised neural articulated shape and appearance models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15816–15826. 2022. 3

[52] Kirillov, A., E. Mintun, N. Ravi, et al. Segment anything, 2023. 4, 5, 11, 15, 16

[53] Rozière, B., J. Gehring, F. Gloeckle, et al. Code llama: Open foundation models for code, 2023. 4, 6, 16

[54] Xiang, F., Y. Qin, K. Mo, et al. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020. 4

[55] Cen, J., Z. Zhou, J. Fang, et al. Segment anything in 3d with nerfs, 2023. 5

[56] Mildenhall, B., P. P. Srinivasan, M. Tancik, et al. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. 5

[57] Peng, S., M. Niemeyer, L. Mescheder, et al. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*. 2020. 5

[58] Qi, C., L. Yi, H. Su, et al. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Neural Information Processing Systems*. 2017. 5, 16

[59] Özgün Çiçek, A. Abdulkadir, S. S. Lienkamp, et al. 3d u-net: Learning dense volumetric segmentation from sparse annotation, 2016. 5

[60] Lorensen, W. E., H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 163–169. Association for Computing Machinery, New York, NY, USA, 1987. 5, 8, 16

[61] Tunyasuvunakool, S., A. Muldal, Y. Doron, et al. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. 6, 7, 15

[62] Community, B. O. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 7, 15

[63] Denninger, M., D. Winkelbauer, M. Sundermeyer, et al. Blenderproc2: A procedural pipeline for photorealistic rendering. *Journal of Open Source Software*, 8(82):4901, 2023. 7, 15

[64] Alayrac, J.-B., J. Donahue, P. Luc, et al. Flamingo: a visual language model for few-shot learning, 2022. 9

[65] Awadalla, A., I. Gao, J. Gardner, et al. Openflamingo: An open-source framework for training large autoregressive vision-language models, 2023. 9

[66] Dosovitskiy, A., L. Beyer, A. Kolesnikov, et al. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. 9

[67] Fuji Tsang, C., M. Shugrina, J. F. Lafleche, et al. Kaolin: A pytorch library for accelerating 3d deep learning research. `https://github.com/NVIDIAGameWorks/kaolin`, 2022. 15

[68] Lin, T.-Y., P. Goyal, R. Girshick, et al. Focal loss for dense object detection, 2018. 15

[69] Sudre, C. H., W. Li, T. Vercauteren, et al. *Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations*, page 240–248. Springer International Publishing, 2017. 15

[70] Hu, E. J., Y. Shen, P. Wallis, et al. Lora: Low-rank adaptation of large language models, 2021. 16

[71] 3D Scanner App. 3D Scanner App. `https://3dscannerapp.com/`, 2024. Accessed: 2024-03-13. 16

[72] Polycam. Polycam - LiDAR & 3D Scanner for iPhone & Android. `https://poly.cam/`, 2024. Accessed: 2024-03-07. 17

A Appendix

A.1 Dataset Preparation Details

Base: PartNet-Mobility Object Assets. We use the same set of 468 training and 41 testing objects from 4 categories in PartNet-Mobility [9]. The raw dataset contains a rich collection of object meshes, textures, and URDF files that contain articulation information. We further process the data as follows:

RGB-D Image Rendering We render each object individually using Blender [62, 63] for 5 loops. For each rendering loop, the object is centered at the scene origin and the rendering camera poses are randomly sampled; we render 12 RGB-D images and all the segmentation masks corresponding to the all the object parts. During rendering, we also randomly sample joint states in the object such that all its doors or drawers are partially open — we make the assumption that all the parts our train and test objects are partially open to remove ambiguity and provide more observation view into object insides.

Mesh Pre-processing. The original PartNet-Mobility assets contain highly fine-grained meshes, i.e.,one drawer part is comprised of more than ten panel or bar-shaped meshes. To prepare data for part-level shape completion, we group these fine-grained meshes such that meshes from the same object part are merged into one single mesh. Mesh textures are ignored during grouping, resulting in grouped texture-less part-level meshes. The RGB-D images and masks are then used to generate part-level point clouds as partial observations. We use Kaolin [67] to sample label occupancy values from object part meshes.

Code-Generation Data. To prepare data for fine-tuning code-generation LLMs, we first use the rendered RGB-D images and segmentation masks to obtain *ground-truth* part-level point-clouds, which are used to extract oriented-bounding boxes (OBBs) for each part. Next, we take the raw object URDF files and generate a shorter copy with our grouped part meshes. Because the raw URDF/XML syntax contain long unnecessary details, we manually translate them into Python-like MJCF [61] code, which are a lot more compact and familiar to the pre-trained LLMs. Finally, for each of the 5 rendering loops per object, we re-write the object code again to replace the absolute joint information with the relative position and rotation of each joint with respect to the extracted OBBs. We further augment the data by randomly rotating the OBBs along the z-axis, 5 times per object. This results in $468 \times 5 \times 5 = 11700$ training samples for LLM fine-tuning.

A.2 Model Training Details

SAM Fine-tuning. The fine-tuning data consists of $28,020$ RGB images, and each image corresponds to a set of binary segmentation masks, one per each object part plus a background mask. We fine-tune only the decoder layers of pre-trained SAM[52] on this custom dataset while keeping the rest of the model weights frozen. Each fine-tuning batch contains 24 RGB images; for every RGB image in the batch, we sample 16 prompt points uniformly across each image's ground-truth masks, i.e.,only sample points from the positive mask area. Hence each training batch of size $B = 24$ contains 24 images and 24×16 pairs of prompt point and ground-truth masks. Following the original paper [52], we update the model with a weighted average of Focal Loss [68], Dice Loss [69] and MSE IoU prediction loss.

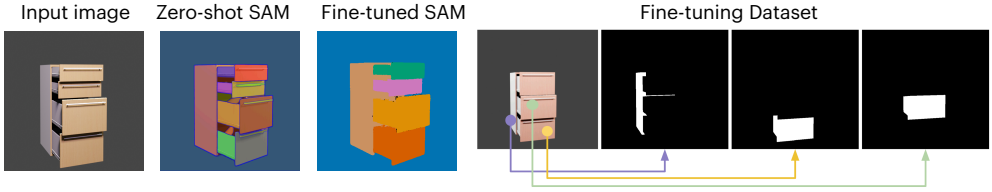

Figure 8: **Kinematics-aware SAM Fine-tuning.** Given an RGB input image, the pre-trained zero-shot SAM[52] produces unnecessarily detailed segmentation masks (column Zero-shot SAM'. We construct a dataset of objects' RGB images and kinematics-aligned ground-truth masks (column 'Fine-tuning Dataset'). The model is fine-tuned to take one image and one sampled 2D query point and predict the corresponding part mask. We compare the output of the model after fine-tuning on the same image (column 'Fine-tuned SAM').

Figure 9: **3D part segmentation from Pose-free RGB images.** Illustration of how DUSt3R[8] is used to achieve 3D part segmentation from unstructured RGB images. For each object, we take around 10 pose-free RGB images as input to the pre-trained DUSt3R[8] model, which outputs a set of globally-aligned 2D-to-3D dense point-maps, i.e.,every 2D pixel on each image is matched to a point in 3D. This correspondence enables cross-view pixel matching via finding nearest-neighbor in 3D space. We can therefore sample view-consistent 2D points for prompting our fine-tuned SAM model, and the resulting segmented masks are grouped into 3D part segmentation.

Training Shape Completion Model. We use $6,260$ pairs of partial point clouds and size 96^3 occupancy grids and train our PointNet++ [58] based occupancy prediction model from scratch. For a training batch of size B, we sample B point clouds of size 2048, and sample $B \times 12,000$ query points on the label occupancy grids. Notably, because object parts are of different scales, we normalize the occupancy grid using *partial* OBBs extracted from the input point cloud to avoid under-fitting the smaller-sized meshes. When sampling training query points, we found sampling 25% occupied works the best for balancing between occupied areas and empty space, and we add a random shifting step on the occupied grids to improve model accuracy on the near-surface areas. At test time, we query on a 96^3 grid and use Marching Cubes [60] to extract the completed part meshes.

Fine-tuning Code Generation LLM. We use the pre-trained Codellama[53]-7B model on our code dataset, which contains code samples generated from PartNet[9] objects as described above. We use LoRA [70], a low-rank weight fine-tuning technique, to fine-tune the model with the next-token prediction loss. For training efficiency, we compress the training sequences by removing unnecessary empty character spaces and overhead code lines (such as package import statements). The resulting training set contains under 800 tokens per sequence for objects with up to 7 parts (i.e., 6 articulated joints). Despite the short training data, we found the model to be able to extrapolate to unseen test set objects with up to 15 parts.

A.3 Details on Real World Evaluations

Data Collection. We collect data from a set of common furniture objects, including cabinets, laptops, night stands, dressers, ranging from 1 to 3 moving parts. Each object is scanned using a LiDAR-equipped iPhone camera and 3dScanner App [71] to capture a set of RGB images from the front 180° view. We then select 10 RGB images per object, and crop and resize them into 512×512 images used by SAM [52] and DUSt3R[8].

Part Segmentation from Unstructured RGB images. Fig. 9 visualizes the DUSt3R model output on an example object in: notably, the model predicts dense point-maps on the object's surface area that can be globally-aligned into a object point cloud; but the 3D points are less accurate on the partially occluded areas, such as the inside of the drawer. This is likely due to these areas are less common in the model's pre-training dataset. Also notice that, because we sample each 2D point from one RGB image first and uses nearest neighbor in the predicted 3D point-map to find its matching 2D point in another image, it might find a wrong match if the point is occluded and not visible in the other image. We address this by manually setting a distance threshold, and decide a match cannot be found if its 3D point's distance to the nearest neighbor is above set threshold.

Figure 10: We demonstrate that Real2Code can be used for labeling and animating real world objects. We evaluate Real2Code on scanned real objects from Polycam[72] and export the resulting mesh and joints in MuJoCo [23]. Blue arrows indicate the simulated joint axis and position; mesh corresponding to the moving part is colored in green.

B Additional Results on Animating Scanned Real World Objects

In addition to object reconstruction from raw RGB images, we show Real2Code can also be used to animate scanned objects. We use real world scanned object meshes uploaded by users of the Polycam [72] App, and use our Blender rendering pipeline to render RGB-D images. We evaluate our image segmentation, shape completion, and code generation models on these images, and demonstrate only the qualitative results due to the lack of ground-truth data. We execute the final model output code to show the objects can be simulated in MuJoCo[23]. See Fig. 10 for visualizations. These real world objects feature complex visual appearance – outside our SAM fine-tuning distribution –, but Real2Code is still able to successfully segment parts and predict reasonable joint positions and rotations.