Appendix

A. Potential Solutions and How They Works

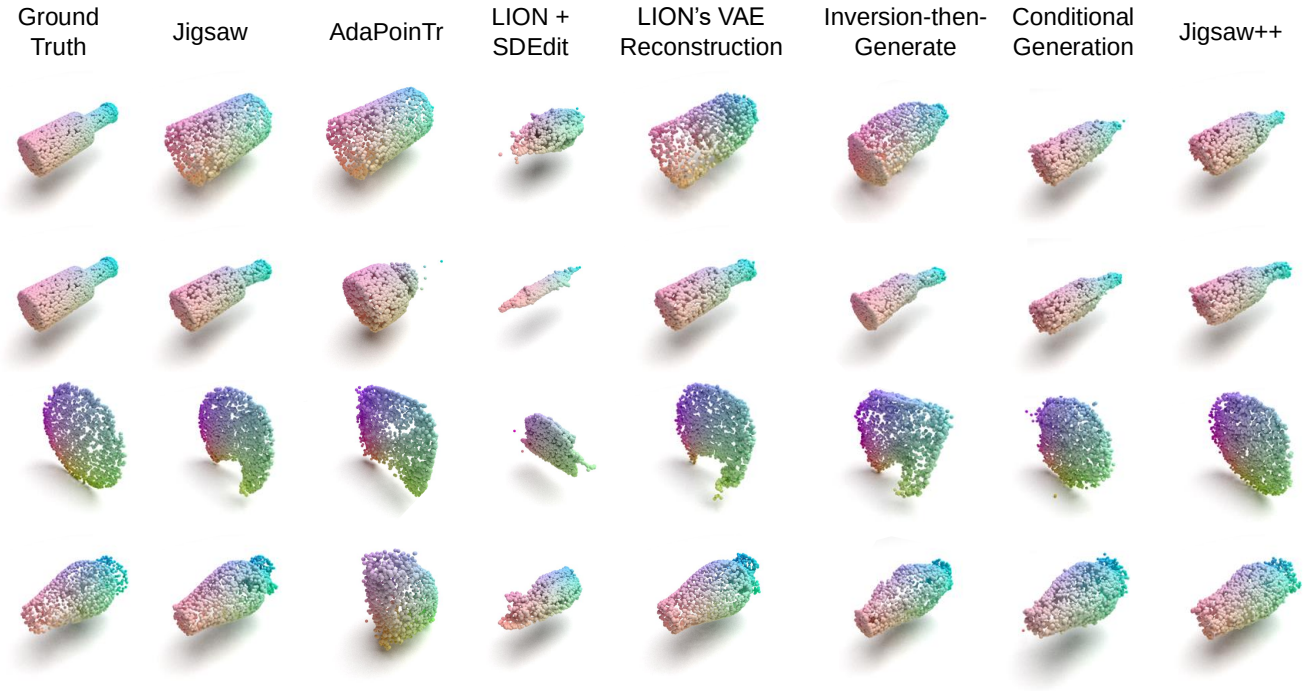

Figure 7. Qualitative demonstration of potential solutions.

Generating a complete shape prior based on a partially assembled object is a relatively new problem that is often underestimated in its complexity. We explored several intuitive solutions during the development of our method to demonstrate the challenges involved. While it is impossible to enumerate all potential solutions, we have selected representative approaches to highlight the uniqueness of our task. The difficulty in providing an accurate shape prior stems from two main challenges: (1) Lack of quantification of assembly errors: We do not know which pieces are correctly assembled and which are not. (2) Balancing shape alteration: The algorithm must adapt to varying degrees of assembly accuracy, from minor adjustments for nearly perfect assemblies to significant corrections for misplaced or incomplete pieces.

We tested four representative algorithms using state-of-the-art models and evaluated their performance on four test cases. Figure 7 illustrates the results of these experiments.

Point Cloud Completion We adopted AdaPoinTr [69] using their open-sourced code and model trained on the ShapeNet dataset. We provided the algorithm with a subset of correctly placed pieces from Jigsaw's result. The algorithm exhibited the following limitations: (a) It interpreted the subset of parts as a complete shape, resulting in no additional completion as in the first bottle. (b) With more parts in the second bottle, it completed the top slightly, but the was sparse and limited in range. (c) It produce a resonable result for the plate which most closely resembled a typical completion task, while for the vase, it over-correct the given input.

Point Cloud Generative Model with Editing We employed LION [70] with the SDEdit [39] model using their open-sourced code. The results showed that (a) the generated shapes with similar overall forms (e.g., thin long shape for bottle input, flat shape for plate), but (b) unable to to consistently maintain the correct object category.

Point Cloud Auto-encoders We utilized LION's [70] VAE to assess the effectiveness of reconstruction. Results showed that the output was mostly identical to the input, with only minor changes towards the desired shape. This behavior is consistent with the VAE's objective of accurate shape reconstruction.

While these methods excel in their designed tasks, they fall short in addressing the specific challenges of inferring complete shape prior for the reassembly problems.

Inversion-then-Generate We evaluate the effectiveness of direct inversion-then-generate pipeline to show how "retargeting step" influence the result. Using the same generator parameters and inversion settings as Jigsaw++ experiments, we observe that this baseline approach yields improvements on simpler cases (e.g., bottles and vases with minor variations). However, it demonstrates significant limitations when substantial modifications are required, exhibiting failure patterns similar to direct VAE reconstruction. These results suggest that the inversion-then-generate approach alone lacks the flexibility to accommodate major structural changes, underscoring the importance of our retargeting mechanism in handling complex shape difference.

Conditional Generation One potential solution is to train a conditional generative model by finetuning our first-stage model with partially assembled inputs as conditions, similar to techniques used in recent 2D generation tasks [19, 57, 76]. We implement this by incorporating partial assembly point clouds as additional input tokens during the finetuning process.

This conditional approach achieves stronger baseline performance with a Chamfer Distance of 4.8×10^{-3}, 46.3% precision, and 50.6% recall. While its Chamfer Distance matches our retargeting method, the precision falls below input level despite achieving higher recall. Qualitative analysis reveals the underlying behavior: the model excels at smoothing input geometry and completing missing regions (hence higher recall) but struggles to correct misplaced parts (resulting in lower precision). In contrast, our retargeting approach achieves a better balance among the three key challenges of this task: correcting misplaced parts, completing missing regions, and maintaining valid shape structure. This comparison validates the effectiveness of our retargeting strategy in handling the unique requirements of assembly-guided shape prior generation.

B. Implementation Details

B.1. Used Codebases and Datasets

For baseline comparison, the following codes are used:

- DGL [71]: https://github.com/hyperplane-lab/Generative-3D-Part-Assembly.
- SE(3) [63]: https://github.com/crtie/Leveraging-SE-3-Equivariance-for-Learning-3D-Geometric-Shape-Assembly/tree/main.
- Jigsaw [35]: https://github.com/Jiaxin-Lu/Jigsaw, (MIT License).
- PoinTr and AdaPoinTr [68, 69]: https://github.com/yuxumin/PoinTr, (MIT License).
- LION [70]: https://github.com/nv-tlabs/LION, (NVIDIA Source Code License).
- SDEdit [39]: https://github.com/ermongroup/SDEdit, (MIT License).

For building our methods, the following codes are referenced:

- LEAP [22]: https://github.com/hwjiang1510/LEAP.
- UViT [3]: https://github.com/baofff/U-ViT, (MIT License).
- Rectified Flow [33]: https://github.com/gnobitab/RectifiedFlow.

The following datasets are used:

- Breaking Bad Dataset [52]: doi:10.5683/SP3/LZNPKB (License as listed in the link).
- PartNet [40]: The [Pre-release v0] version at https://partnet.cs.stanford.edu/ for mesh, and the version presented with DGL [71].
- Kubric-ShapeNet [16]: The version with LEAP for camera parameters.

B.2. Parameters

We provide a detailed model parameters in Table. 3.

C. Training Details

C.1. Training resources and Inference Time

Our experiments utilized a setup featuring eight NVIDIA Tesla A100 GPUs, with all running times based on this specific GPU configuration.

The finetuning of LEAP reconstruction model takes 232 GPU hours. In the training phase for the base generative model, different datasets required varying amounts of GPU time: the Breaking Bad dataset needed 480 GPU hours, while the PartNet

Table 3. The detailed experiment parameters.

		Breaking Bad Dataset		PartNet		
	Parameter	base	retargeting	base	regargeting	description
Training	epoch	500	100	1000	400	training epochs
	bs	32	32	16	16	batch size
	lr	0.0001	0.00002	0.0001	0.00002	learning rate
	optimizer	Adam	Adam	Adam	Adam	optimizer during training
	scheduler	Cosine	-	Cosine	-	learning rate scheduler
	min_lr	1e-6	-	1e-6	-	minimum learning rate for Cosine scheduler
	frames	5	5	5	5	input frames to the image encoder
Model	N	100	100	100	100	sample steps in Rectified Flow
	N_r	-	4	-	4	reverse sampling steps in retargting
	α	-	0.5	-	0.5	scaling factor for latents during retargeting
	depth	12		768		depth of UViT
	d	768		768		token dimension in UViT

categories required 40 GPU hours for Lamp, 216 GPU hours for Chair, and 240 GPU hours for Table. Additionally, the "retargeting" stage fine-tuning took 480 GPU hours for the Breaking Bad dataset and half of base model for each PartNet category.

For inference, reverse sampling of a single instance on one GPU took 0.2 seconds, and forward generation took 5 seconds. The complete processing time for one instance, includes rendering and reconstruction, was approximately 7.5 seconds on average.

C.2. Trained Models

On the Breaking Bad dataset of the fracture assembly problem, LEAP [22] is first finetuned using rendered mesh data. One generation model is trained for the entire subset without categorical information. Then, this model is finetuned and "retargeted" based on data computed by Jigsaw for the reconstruction task. The same model without finetuning on SE(3) [63] is used for testing on SE(3) [63] model.

On the PartNet of the part assembly problem, LEAP is first finetuned using rendered mesh data for all three categories. For each category, one generation model is trained, which results to three base generative models. These models are finetuned independently for the reconstruction task based on data computed by DGL.

C.3. Data Visualization

Figure 8 provides a visualization of the rendered data utilized in our experiments. In the top row, the partially assembled object is shown in point cloud format, which is used as input during the "retargeting" phase and for testing. The bottom row features the rendered complete objects, which are based on the mesh data from the dataset. Due to the superior continuity of this mesh data, it is selected as the ground truth for guiding the training of the generative model and the image-to-3D model, LEAP.

D. Additional Results

D.1. Metric Distribution

Fig. 9 presents a comprehensive analysis of metric distributions, illustrating the impact of our generative approach on reconstruction quality. While the incorporation of generative models introduces inherent uncertainties - manifesting as point displacement, omission and addition of geometric features, or even shape change - the quantitative improvements are substantial. Notably, Jigsaw++ demonstrates a remarkedly lower peak in Chamfer distance distribution compared to baseline methods, with a larger proportion of samples achieving high precision and recall scores. These improvements in geometric accuracy, when considered alongside the assembly performance metrics (Table 2, Right), provide compelling evidence that the learned shape priors serve as a valuable constraint for the reassembly task. The integration of complete shape priors introduces an additional layer of geometric reasoning that effectively guides the reconstruction process, particularly in challenging cases.

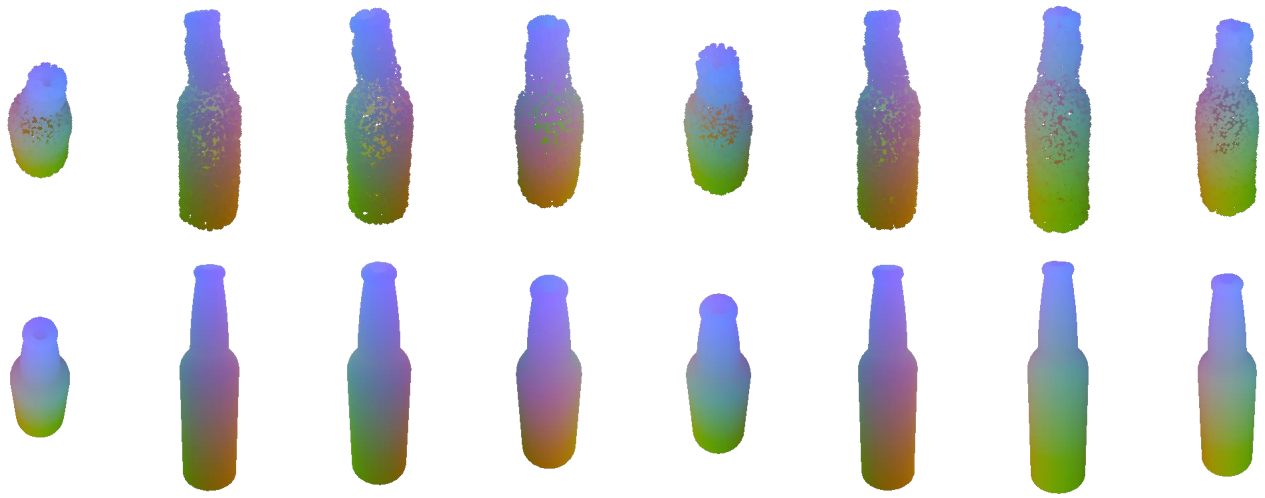

Figure 8. Visualization of one instance from the Breaking Bad Dataset. Top: The input partially assembled object is presented in point cloud format, which is employed both in the ”retargeting” phase and for testing purposes. Bottom: Input complete objects rendered from meshes. Those data are used to create ground truth data for the training phases of the generative model and the image-to-3D model LEAP.

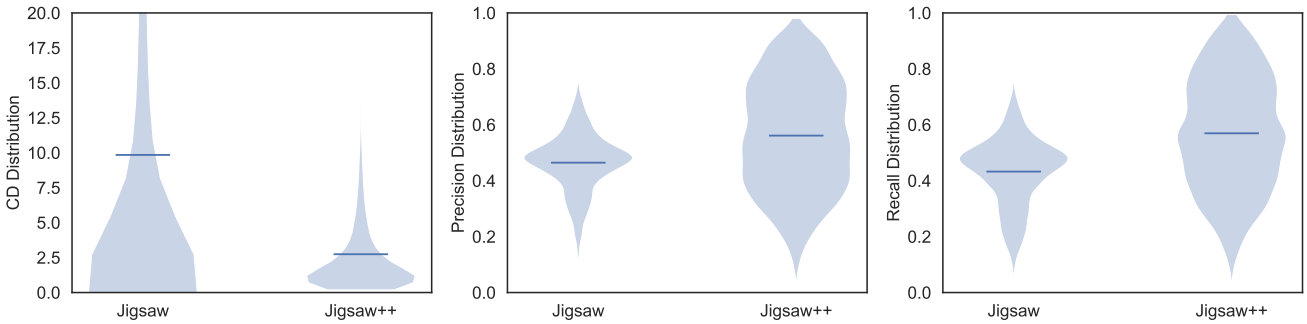

Figure 9. Metric distribution of Jigsaw and Jigsaw++. The metric Chamfer Distance is truncated by $[0.0, 20.0]$ for better visual quality.

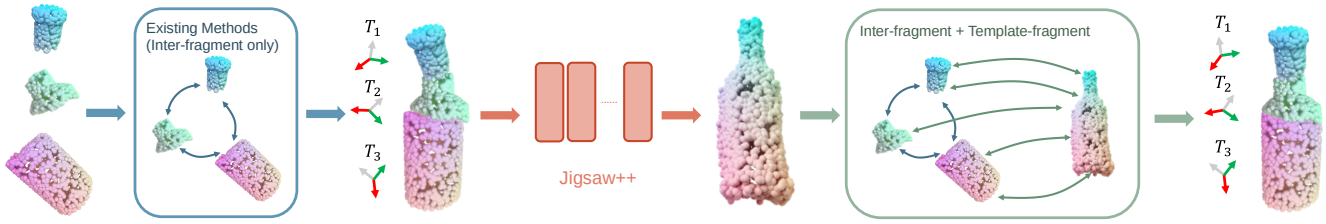

Figure 10. Apply Jigsaw++ to assembly workflow. Left: Use existing methods to compute an initial assembly result and compose a partially assembled object. Middle: Jigsaw++ generates a complete shape prior from this partial assembly, providing global context unavailable to local matching methods. Right: The shape prior guides refinement of fragment transformations through template matching.

E. Apply Jigsaw++ To Assembly Problem

To clarify how Jigsaw++ enhances practical assembly tasks, we present a complete pipeline overview in Fig. 10. Our method serves as an intermediate step that provides complete shape prior as an additional level of information to improve assembly accuracy.

Given fragment point clouds, the pipeline operates in three phases:

1. **Initial Assembly:** Existing methods (e.g., Jigsaw, SE(3)) compute initial fragment placements using local geometric features, producing a partially assembled object.
2. **Shape Prior Generation:** Jigsaw++ processes this partial assembly to generate a complete shape prior in the same point

cloud representation as the input fragments.

3. **Assembly Refinement:** The shape prior guides fragment placement optimization (through geometric matching in our example) between fragments and the complete shape. This produces refined transformation matrices for each fragment, improving the final assembly accuracy.

Importantly, this pipeline can be extended in future work by developing more sophisticated matching algorithms between fragments and shape priors, or by incorporating the shape prior directly into existing assembly optimization objectives.

F. Visualization

We present detailed visualization of results on the Breaking Bad Dataset (Fig. 12) and PartNet (Fig. 13). We also present a visualization on several examples we tested on Fantastic Breaks [27] (Fig. 11). We use the same model trained on Breaking Bad Dataset. Please note that Fantastic Breaks only involves 2-pieces samples and all objects are real-world objects that doesn't exist in the Breaking Bad Dataset. For the fracture assembly problem, we additionally visualize the experiment described in Table 2 Right where we use this shape prior generated by Jigsaw++ to guide assembly. Each instances are organized in the order of "partial input - Jigsaw++ - Ground Truth" vertically.

G. Broader Impacts

This paper tackles object reassembly problem, which has no known negative impact on society as whole. On the contrary, its application in archaeology and medication would benefits research in other areas. Our method utilizes 3D generative model, which we hope could address several hard problems overlook by the current researches. The data we use are all objects datasets. Although we see no immediate negative use cases or content from this model, we acknowledge the necessity of handling the generative model with care to prevent any potential harm.

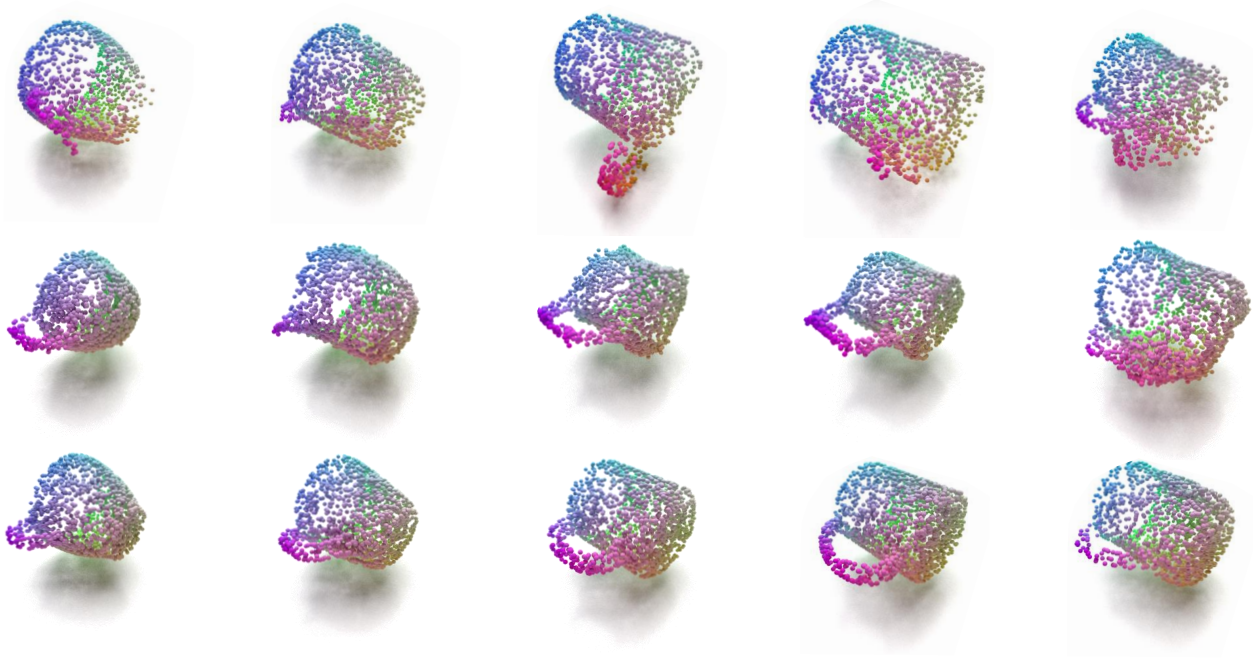

Figure 11. Detailed visualization of results on the FantasticBreaks.

Figure 12. Detailed visualization of results on the Breaking Bad Dataset.

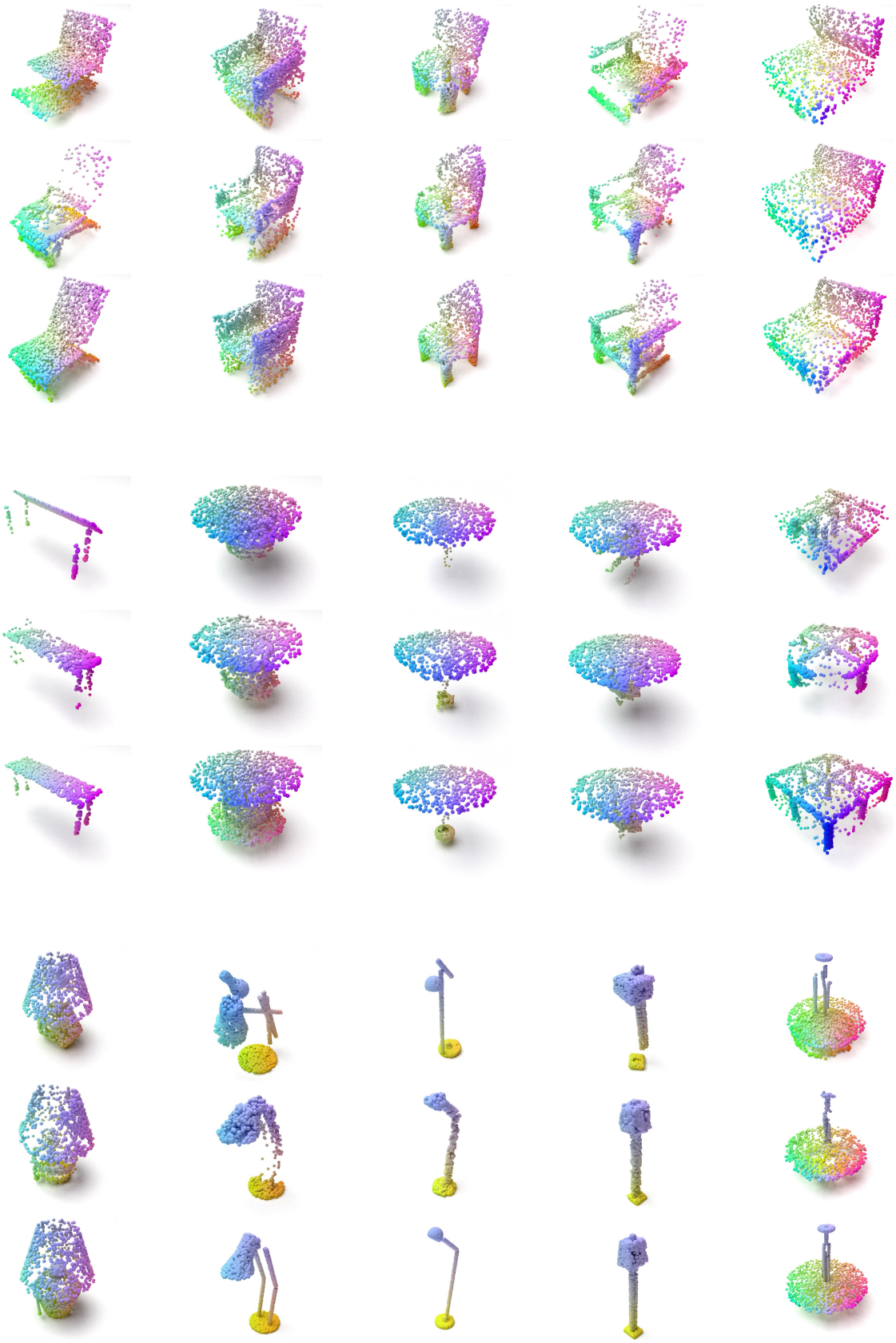

Figure 13. Detailed visualization of results on the PartNet.

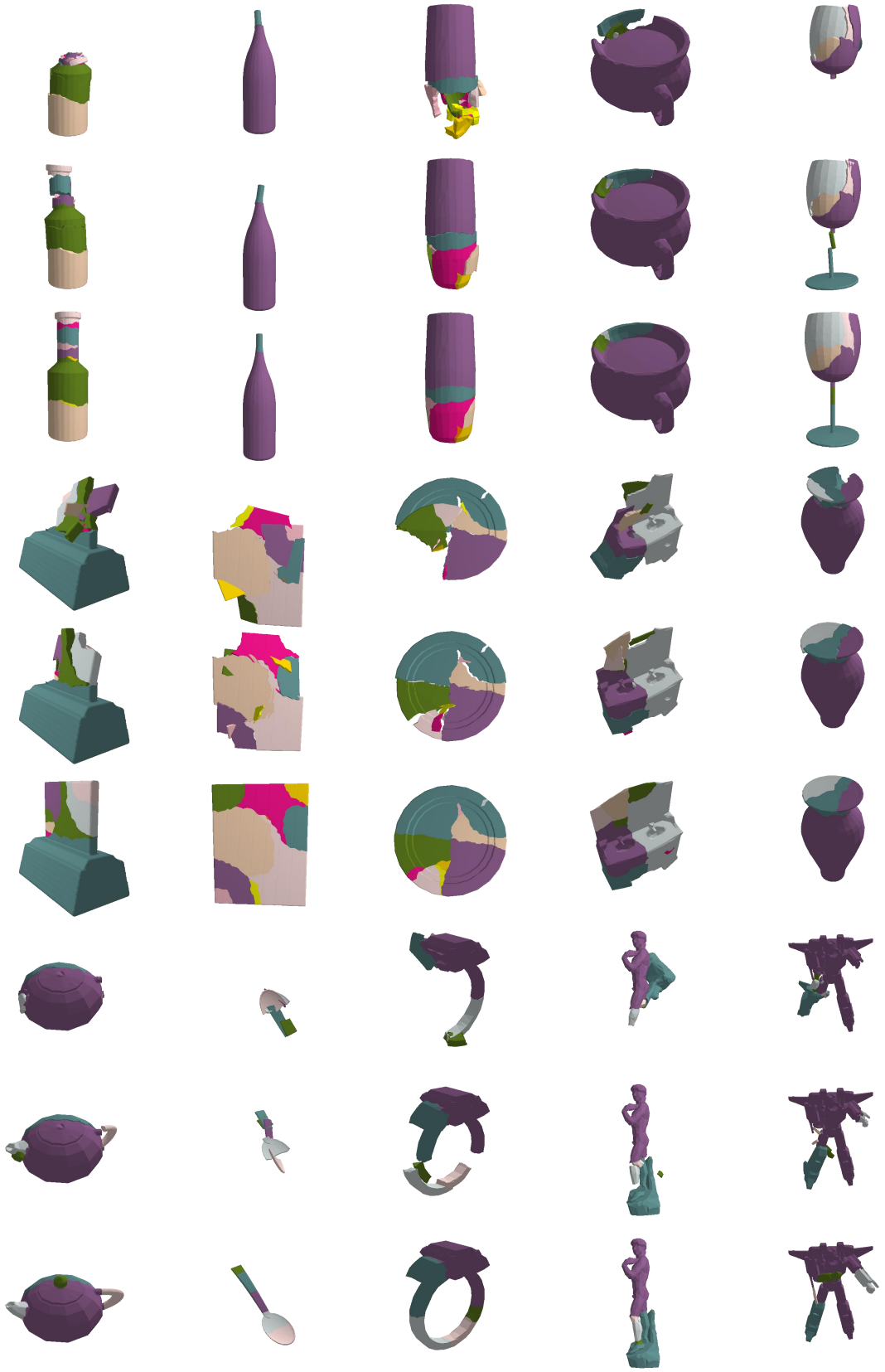

Figure 14. Visualization of fracture assembly performance with original-shape matching with the shape prior generated by Jigsaw++.