

D-Prism: Differentiable Primitives for Structured Dynamic Modeling

Supplementary Material

A. More Details for Implementation

A.1. Primitive Initialization

We provide detailed initialization parameters for all cases across different datasets to demonstrate our framework’s reproducibility. These details are included here due to space constraints in the main text. Here we list the initial primitive count K , the number of 3DGS allocated per primitive N_{gs} , and the scene scaling value s_{scene} for different scenes. This s_{scene} is multiplied by the primitive’s scale (s_1, s_2, s_3) to ensure the primitive has a moderate initial size, helping it appear within the camera’s view during the early stages of training, the same as [1].

Table 6. Initialization Settings for Training.

| Property | Dynamic Primitive Dataset | | | | | |
|-------------|---------------------------|--------------|------|--------|---------------|------------|
| | Rubik’s Cube | Treasure Box | Door | Pliers | Folding Chair | Sunglasses |
| K | 10 | 10 | 20 | 50 | 50 | 25 |
| N_{gs} | 50k | 50k | 10k | 8k | 10k | 10k |
| s_{scene} | 0.4 | 0.4 | 0.2 | 0.1 | 0.1 | 0.25 |

| Property | D-NeRF Dataset | | | | |
|-------------|----------------|------|-------------|--------|---------|
| | Jumpingjacks | Hook | Hellwarrior | Mutant | Standup |
| K | 50 | 50 | 50 | 50 | 50 |
| N_{gs} | 5k | 5k | 5k | 5k | 5k |
| s_{scene} | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 |

A.2. Details for Deformation Networks

We begin with a 5k iteration warm-up stage, during which the deformation network is frozen. This stage focuses on initially optimizing primitive shapes and 3D Gaussian color parameters. We then unfreeze the network for the main training.

We implement both the forward and backward deformation networks as Multi-Layer Perceptrons (MLPs). The architecture for both comprises $D = 8$ layers, each with a hidden dimensionality of $W = 256$. To help capture high-frequency variations, the network inputs, the primitive’s motion translation T and the temporal label t , are first mapped into a higher-dimensional Fourier space using a positional encoding function $\xi_k(p)$. We set the encoding parameter $k = 10$ for T and $k = 6$ for t .

The network culminates in a single linear layer, without an activation function, which predicts the offsets for the primitive parameters: translation (ΔT) and rotation (ΔR). These predicted offsets allow for the dynamic adjustment of the primitives. At initialization, we set the final layer’s weights for ΔT to zero to prevent large initial displacements from the deformation.

A.3. Details for Computation Resource

Our method’s training time varies across scenes. On a single NVIDIA 3090 GPU, the main training process takes 2-3 hours, and the refinement stage takes 1-2 hours. The VRAM requirement of GPU also fluctuates, typically ranging from 8G to 16G, depending on scene complexity. For objects with more detail, our primitive adaptive control strategy generates more primitives, which increases the peak GPU memory usage during training.

B. Visualization Results for Dynamic Primitive Dataset

We present visual results for all cases from Dynamic Primitive Dataset in Fig. 6, 7, 8. Additionally, we provide detailed calculation procedures for the evaluation metrics in the following subsections. We also provide the PSNR and SSIM rendering metrics for the geometry-based methods tested on Dynamic Primitive Dataset, as detailed in Tab. 7.

B.1. Details for Structured Motion Tracking Accuracy

To compute structured motion tracking accuracy, we first select a canonical ground truth mesh at one timestamp. We sample 50,000 points on it using the *trimesh* sampling function (with a fixed seed) and record their barycentric coordinates. We then use these coordinates and the ground truth meshes from all other timestamps to compute the points’ corresponding positions over time. This process yields the motion tracking ground truth results.

Next, we use the methods under evaluation to predict the positions of these 50,000 points over time. For our method, we first obtain our primitive geometry at the canonical timestamp corresponding to the initial sampling, bind each point to its nearest primitive, and then calculate the point’s motion from that primitive’s motion parameters. For methods that provide consistent meshes, we use a similar approach by binding each point to its nearest triangular face and computing its motion from the face’s motion. For methods that do not provide consistent mesh results but provide deformation networks, we first use their inverse network to map the points to their respective canonical space, and then use their forward network to map these points to each observation space. For Shape of Motion, we compute the point motion using their officially provided procedure.

Table 7. **Structured Motion Rendering Results on Dynamic Primitive Dataset.** We select dedicated test images and compute the average results across all of them. For a fair comparison, we select other geometry-based methods and using DG-Mesh*, which are its gaussian rendering results. Our structured representation simplifies the scene, which leads to some loss of detail. Despite this, our method maintains good rendering metrics. Our approach also shows excellent results in cases with large motion, such as Rubik’s Cube.

| Method | Rubik’s Cube | | Treasure Box | | Door | | Pliers | | Folding Chair | | Sunglasses | |
|----------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| | PSNR \uparrow | SSIM \uparrow | PSNR \uparrow | SSIM \uparrow | PSNR \uparrow | SSIM \uparrow | PSNR \uparrow | SSIM \uparrow | PSNR \uparrow | SSIM \uparrow | PSNR \uparrow | SSIM \uparrow |
| Ub4D | 17.199 | 0.848 | 20.858 | 0.919 | 21.877 | 0.964 | 23.520 | 0.982 | 17.707 | 0.946 | 17.907 | 0.983 |
| DG-Mesh* | 22.540 | 0.913 | 22.110 | 0.935 | 26.821 | 0.973 | 31.436 | 0.989 | 27.200 | 0.980 | 33.063 | 0.989 |
| Ours | 29.218 | 0.941 | 28.000 | 0.943 | 26.480 | 0.964 | 33.874 | 0.988 | 28.107 | 0.962 | 31.127 | 0.981 |

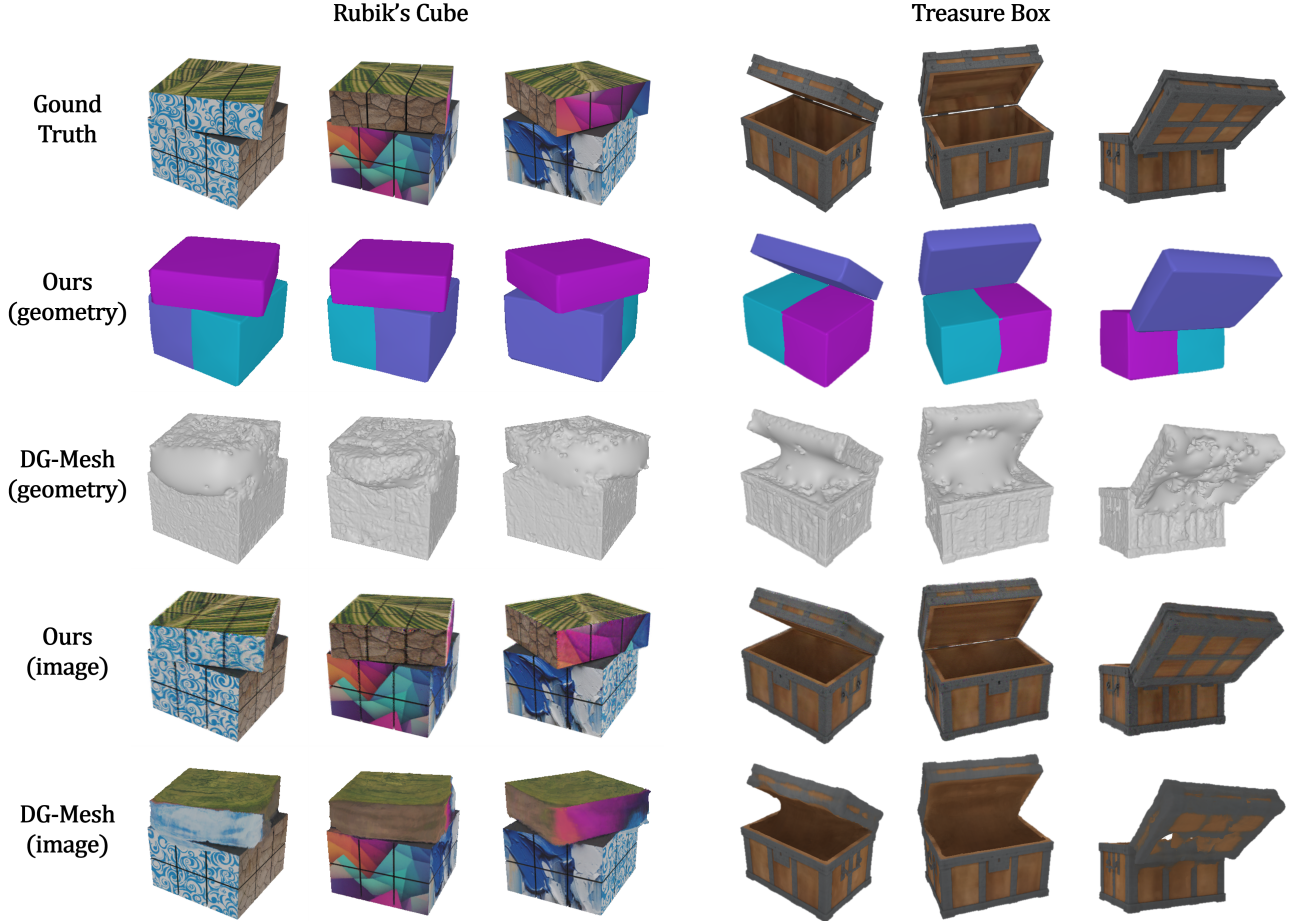


Figure 6. **Visualization Results for Dynamic Primitive Dataset.** We show both geometry results and rendering results.

B.2. Details for Structured Dynamic Reconstruction Results

For traditional mesh quality metrics, we impose a special dynamic constraint. Evaluated methods must provide consistent meshes, where the geometry at each frame shares an identical vertex count and face definition. This means the vertex indices for any given face must remain fixed over time. After obtaining these per-frame consistent results,

we compute the traditional metrics against the per-frame ground truth mesh. The final dynamic reconstruction results are the average of these metrics across all timestamps.

C. Visualization Results for D-NeRF Dataset

Here we present the complete visualization results for sequences we have used in D-NeRF Dataset, including structured dynamic geometry results and rendering results, as de-

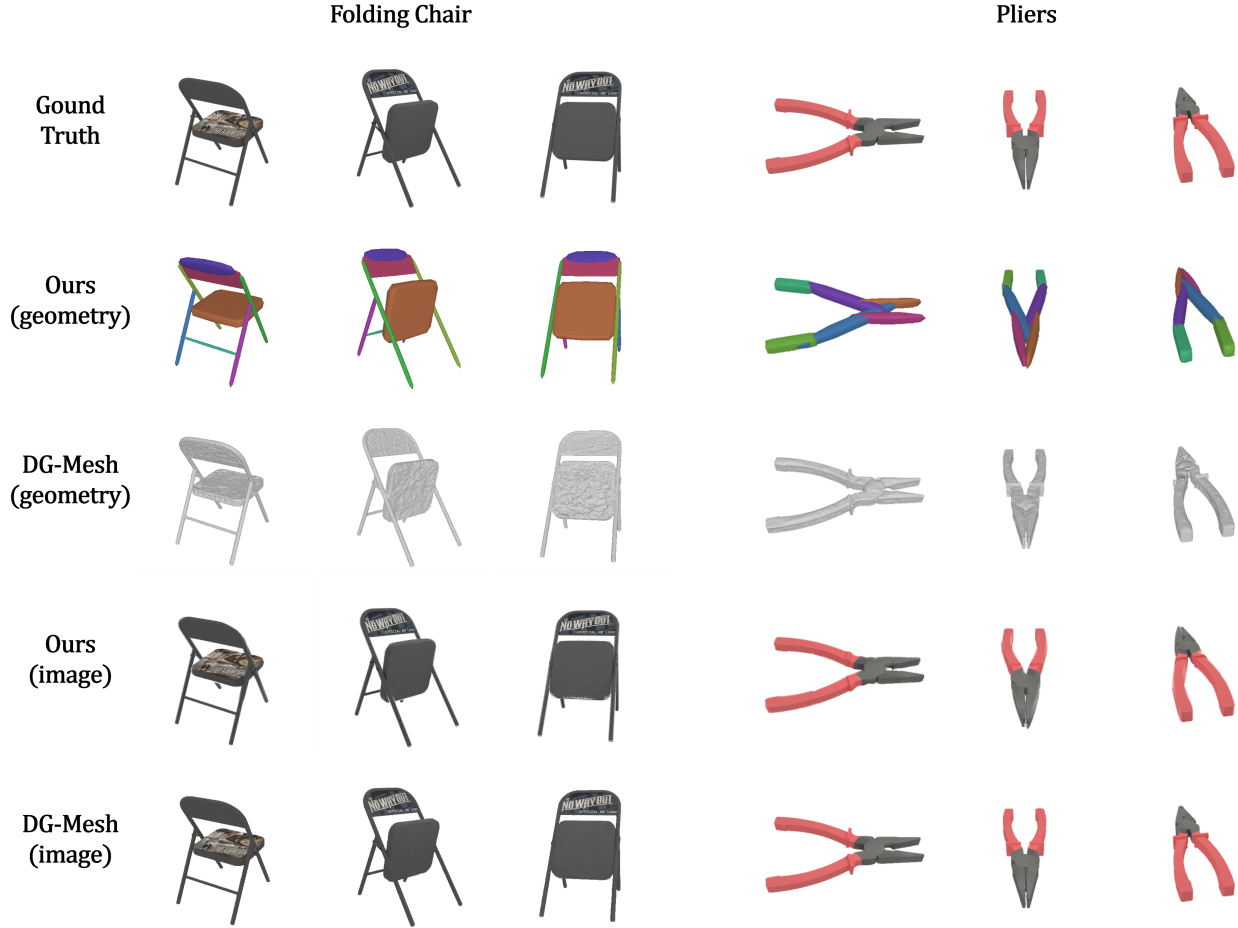


Figure 7. **Visualization Results for Dynamic Primitive Dataset.** We show both geometry results and rendering results.

tailed in Fig. 9, 10, 11. The structured results make the object more interpretable and facilitate further operations on the object’s parts, such as articulation and editing.

D. Visualization Results for Real World Data

We also test our method on real-world scenarios. Here we select and test two cases from Self-Captured iPhone Dataset [2] and DyCheck Dataset [3], visualizing their results in Fig. 12. These causal video cases, which feature sparse viewpoints and imprecise annotations, are indeed challenging for our method. Additionally, we have recorded two dynamic video sequences of real world robotic scenarios, including a moving robotic arm and a dexterous hand. These videos were processed following the data processing pipeline from [2]. With adequate viewpoint coverage, our method performs well on these sequences, as shown in Fig. 13.

E. Visualization for Applications

Our method provides a structured geometric representation that enables easy editing of the reconstructed results. First, as demonstrated in the teaser, we can easily perform motion pattern transfer. For instance, we can transfer the opening motion of the Treasure Box lid to the top layer of the Rubik’s Cube, or apply the rotation of the Rubik’s Cube to the lid, as shown in Fig. 14.

We can also perform various physics-based motion simulations on individual object parts. For instance, we simulate the Rubik’s Cube rotating upon impact and stopping due to friction, or the Treasure Box lid accelerating open under force and slowly settling back due to joint damping. Visualizations of this application are available in our supplementary video.

Additionally, we can articulate our humanoid reconstructions. Leveraging our structured representation, we can control limb movements in Blender by simply setting up a skeleton and Inverse Kinematics (IK) parameters based on the structured representation, eliminating the need for skin-

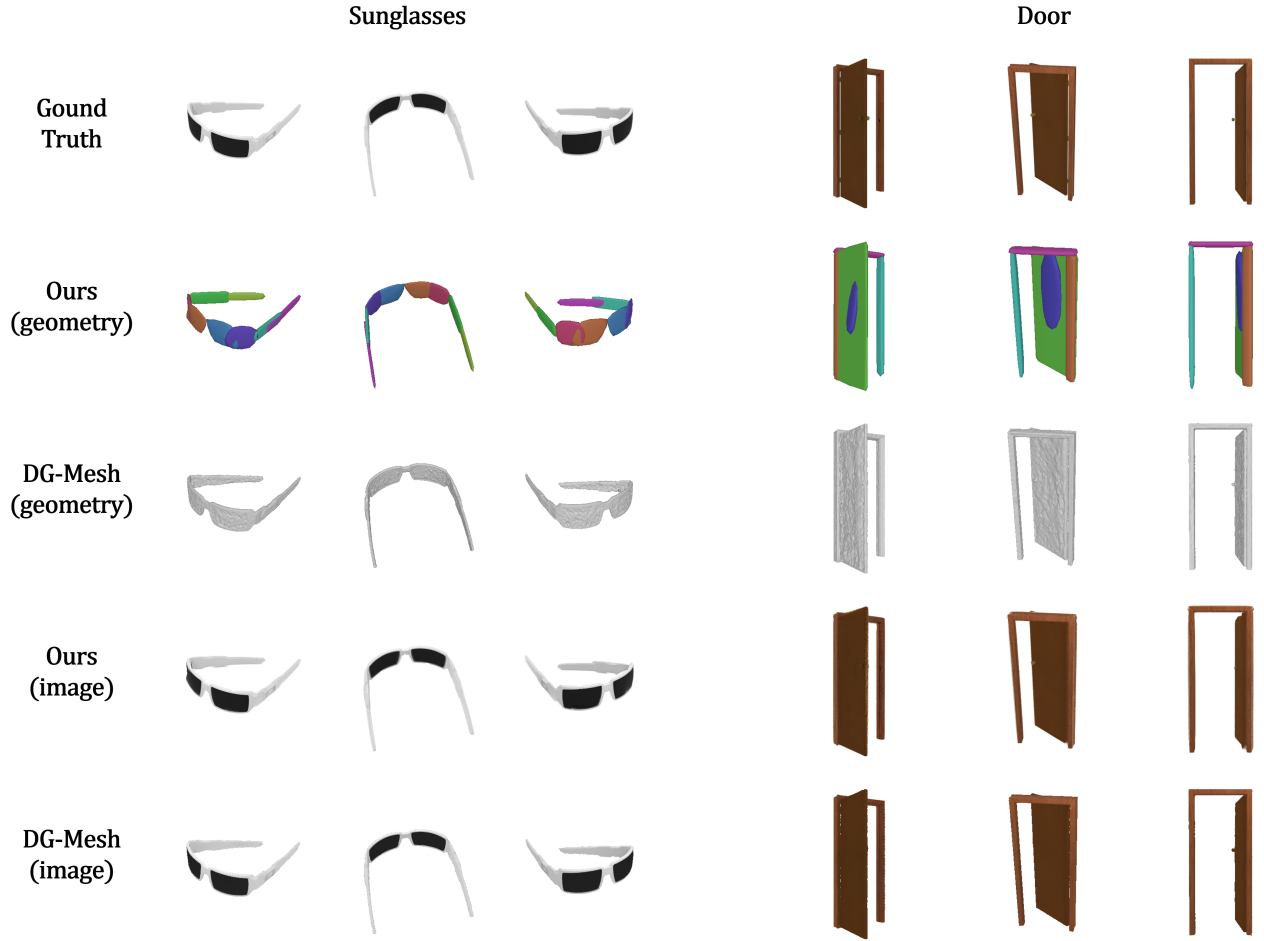


Figure 8. **Visualization Results for Dynamic Primitive Dataset.** We show both geometry results and rendering results.

ning, as shown in Fig. 15.

Comprehensive visualizations of all these applications are provided in our supplementary video.

References

- [1] T. Monnier, J. Austin, A. Kanazawa, A. Efros, and M. Aubry. Differentiable blocks world: Qualitative 3d decomposition by rendering primitives. In *Advances in Neural Information Processing Systems*, volume 36, pages 5791–5807, 2023. 1
- [2] I. Liu, H. Su, and X. Wang. Dynamic gaussians mesh: Consistent mesh reconstruction from dynamic scenes. *arXiv preprint arXiv:2404.12379*, 2024. 3
- [3] H. Gao, R. Li, S. Tulsiani, B. Russell, and A. Kanazawa. Monocular dynamic view synthesis: A reality check. In *Advances in Neural Information Processing Systems*, volume 35, pages 33768–33780, 2022. 3

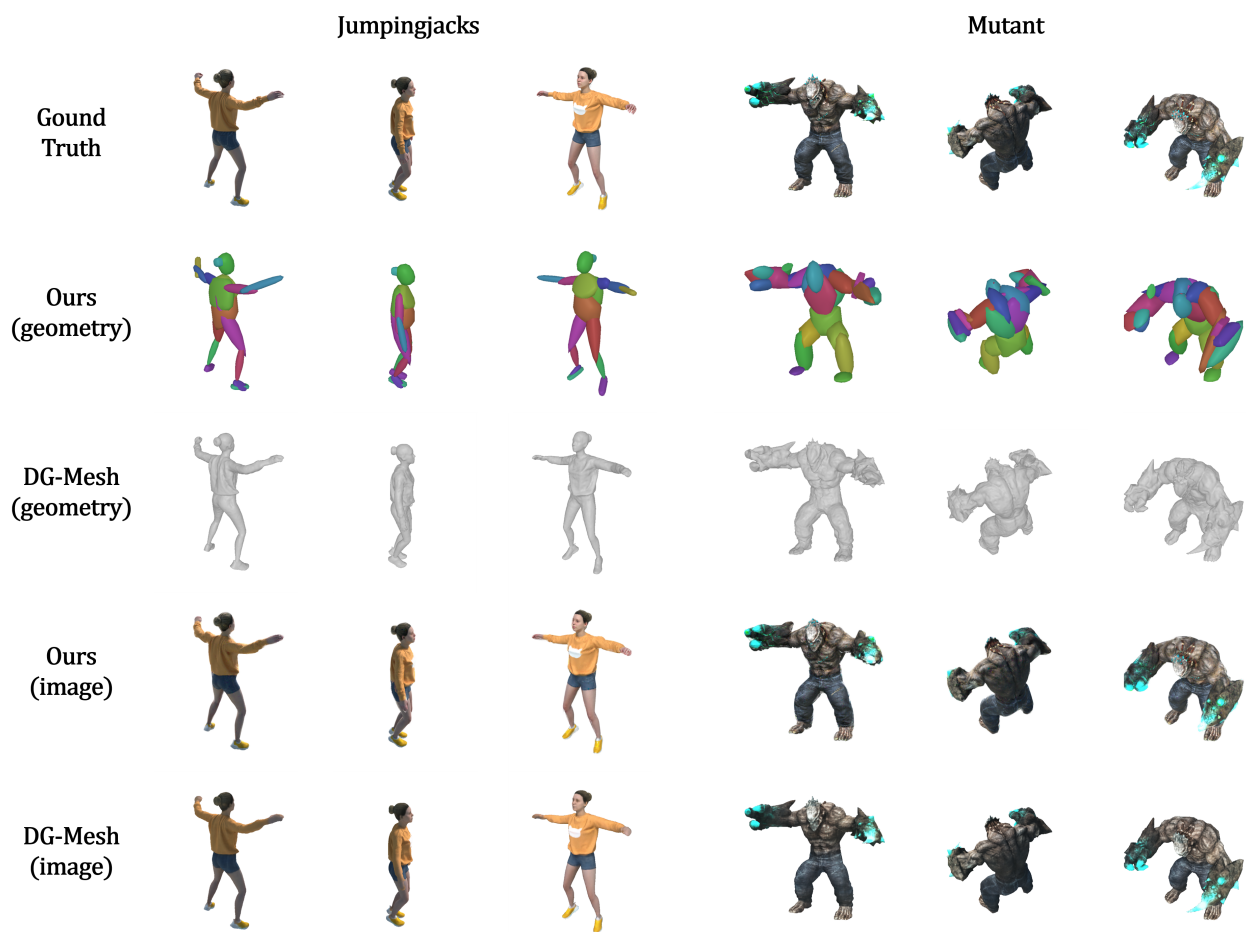


Figure 9. **Visualization Results for D-NeRF Dataset.** We show both geometry results and rendering results.

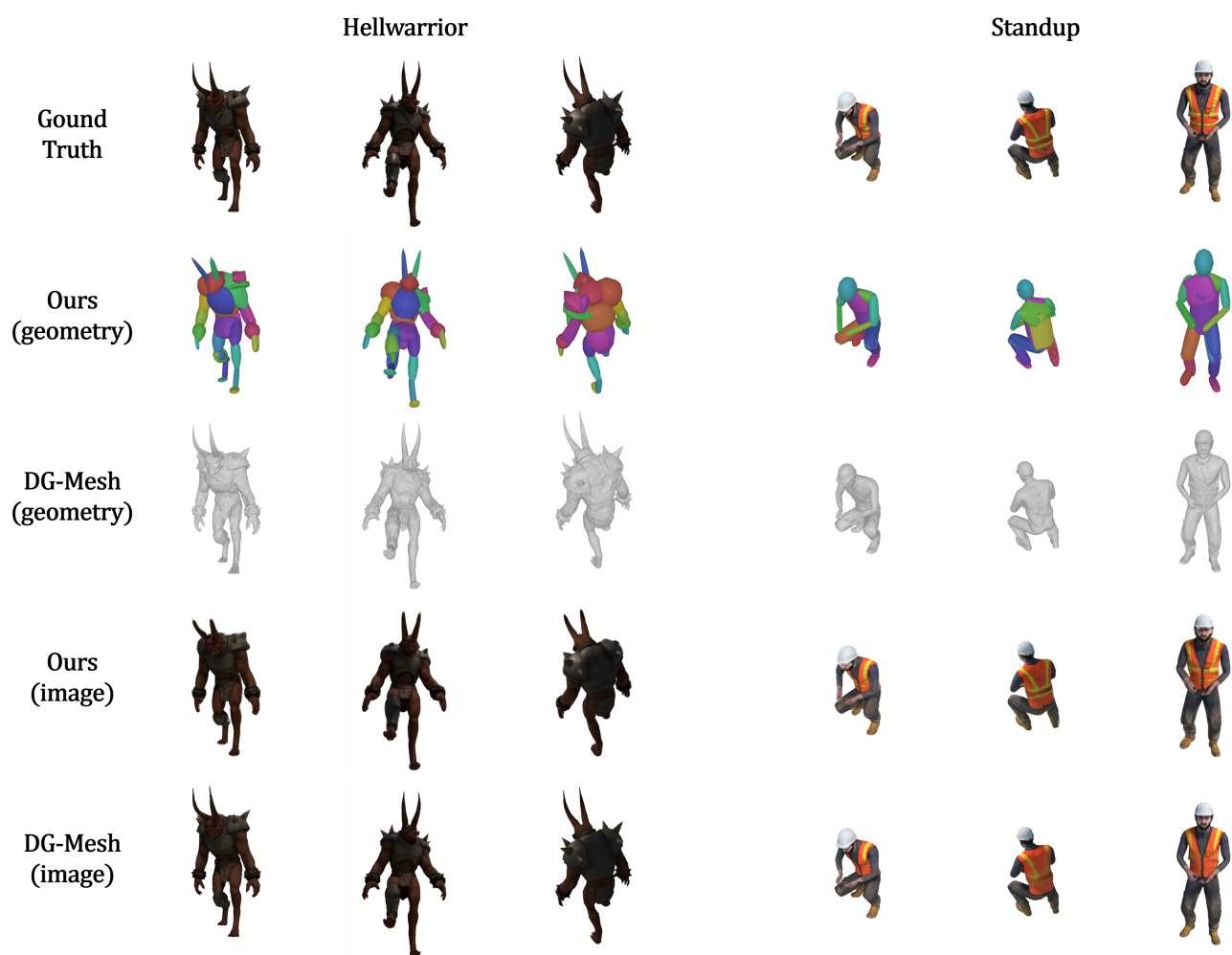


Figure 10. **Visualization Results for D-NeRF Dataset.** We show both geometry results and rendering results.

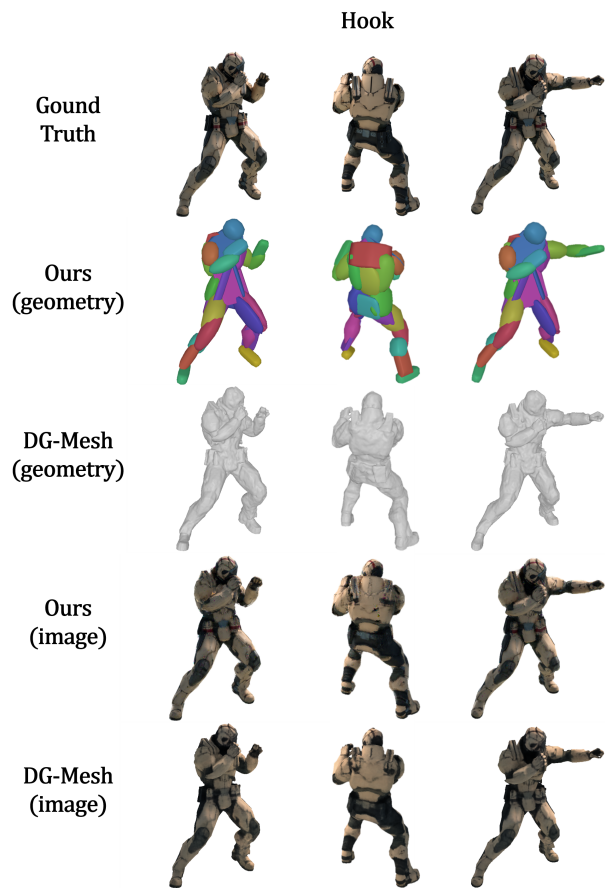


Figure 11. **Visualization Results for D-NeRF Dataset.** We show both geometry results and rendering results.



Figure 12. **Visualization Results for Real World Data.** We show both geometry results and rendering results.

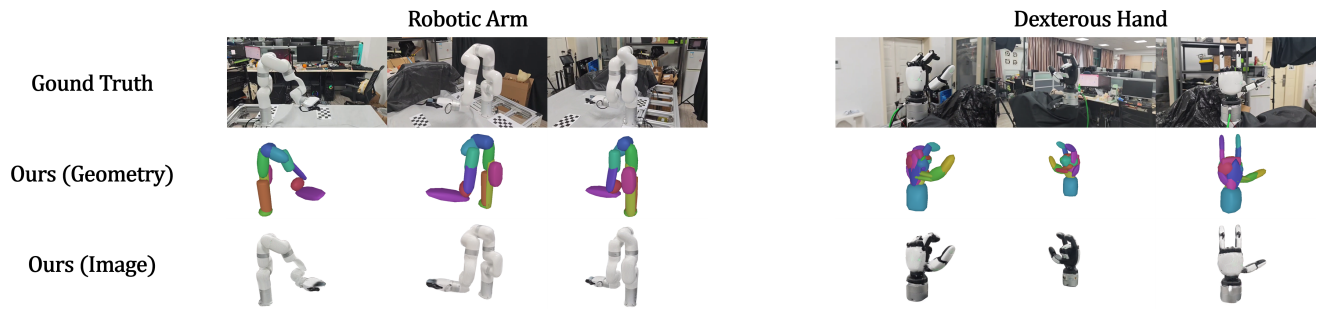


Figure 13. **Visualization Results for Real World Robotic Scenarios.** We show both geometry results and rendering results.



(a) From Treasure Box to Rubik's Cube.



(b) From Rubik's Cube to Treasure Box.

Figure 14. **Visualization for Motion Pattern Transfer.** Here we swap the original motions of the Treasure Box and the Rubik's Cube, assigning new motion patterns to both dynamic objects.

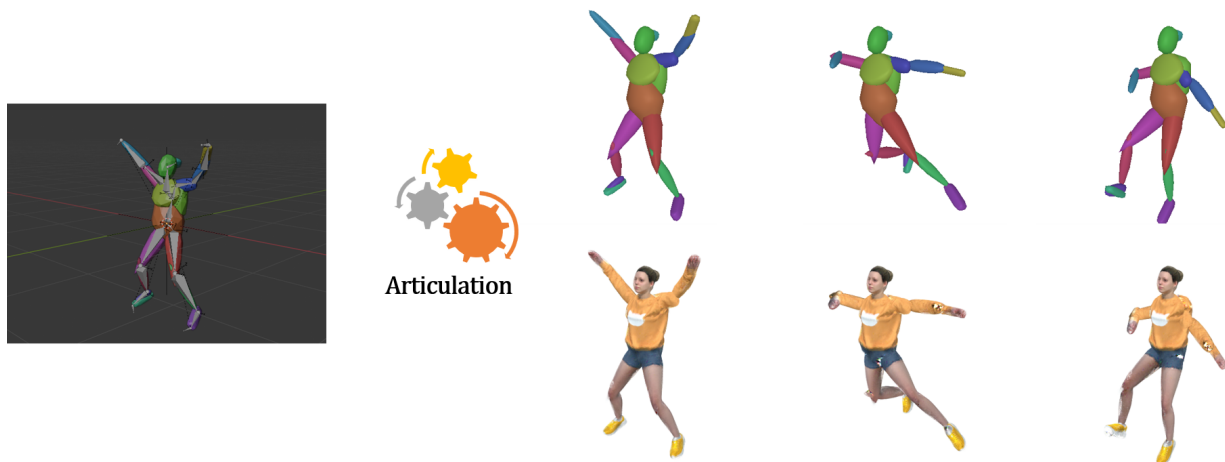


Figure 15. **Visualization of Easy Articulation.** After simple operations, such as simple skeleton annotation in software like Blender, we can obtain a structured articulable representation, enabling free articulation editing.