A. Appendix

A.1. CROCS

We first provide a precise specification of CROCS.

Normalization (similar to NOCS) Let $\Omega \subset \mathbb{R}^3$ denote the surface of a 3D object or scene. For any point $\mathbf{p} = (x, y, z) \in \Omega$ in world coordinates, we first normalize its coordinates into NOCS, defined by the unit cube $[0, 1]^3$.

We determine the Axis-Aligned Bounding Box (AABB) of Ω using the minimum and maximum coordinates, $\mathbf{p}_{\text{min}}$ and $\mathbf{p}_{\text{max}}$. The geometric center of the AABB is $\mathbf{c}_\Omega = (\mathbf{p}_{\text{min}} + \mathbf{p}_{\text{max}})/2$. To ensure uniform scaling while preserving the aspect ratio, we calculate the maximum extent L of the bounding box:

$$L = \|\mathbf{p}_{\text{max}} - \mathbf{p}_{\text{min}}\|_\infty \tag{1}$$

$$= \max(x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min}). \tag{2}$$

The NOCS coordinates $\mathbf{p}' \in [0, 1]^3$ are then obtained by scaling the object and centering it at $\mathbf{h} = (0.5, 0.5, 0.5)$:

$$\mathbf{p}' = \frac{1}{L}(\mathbf{p} - \mathbf{c}_\Omega) + \mathbf{h}. \tag{3}$$

Camera-Relative Transformation Let the position of a source camera be defined in spherical coordinates by (θ, ϕ, r), corresponding to the azimuth, elevation, and radial distance, respectively. We assume the camera is always oriented to face the center of the object space.

To achieve camera-relative canonicalization, we rotate the normalized coordinates around the vertical axis (Z-axis) by the source camera's azimuth θ. This aligns the coordinate system with the viewpoint of the source camera. The transformation is applied around the center of the unit cube $\mathbf{h}$. We define the intermediate rotated coordinates $\mathbf{p}''$ as:

$$\mathbf{p}'' = R_z(\theta)(\mathbf{p}' - \mathbf{h}) + \mathbf{h}, \tag{4}$$

where $R_z(\theta) \in SO(3)$ is the standard 3D rotation matrix around the Z-axis:

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

This transformation rotates the coordinates in the X-Y plane (corresponding to the Red and Green channels in the pointmap) while preserving the Z coordinate (Blue channel). Notably, this definition is deliberately invariant to the camera elevation ϕ and distance r, a choice further elaborated in Appendix A.2.2.

Rescaling Since the object was initially scaled to fit an axis-aligned unit cube (Eq. 3), the rotation in Eq. (4) may cause the coordinates $\mathbf{p}''$ to exceed the $[0, 1]^3$ bounds (illustrated in Appendix A.2.1). A final rescaling step is necessary to ensure the coordinates remain constrained within the unit cube.

To ensure the normalization is robust and predictable across different azimuth angles, we incorporate the theoretical bounds of the transformation. We define the theoretical range $L_{th}(\theta)$ and the theoretical minimum coordinate $m_{th}(\theta)$ achievable when rotating a unit cube by θ around its center:

$$L_{th}(\theta) = |\cos\theta| + |\sin\theta|, \tag{6}$$

$$m_{th}(\theta) = 0.5 - \frac{1}{2}L_{th}(\theta). \tag{7}$$

Let Ω'' be the set of all rotated coordinates for the object. We calculate the observed minimum m_{obs} and maximum M_{obs} values across all points and all dimensions of Ω''. The effective minimum shift m_{eff} and the effective range L_{eff} are calculated by clamping the observed values with the theoretical bounds. This ensures that the normalization is robust and consistent with the geometric constraints of the rotation:

$$m_{eff} = \max(m_{obs}, m_{th}(\theta)), \tag{8}$$

$$L_{eff} = \text{clip}(M_{obs} - m_{eff}, \epsilon, L_{th}(\theta)), \tag{9}$$

where ϵ is a small positive constant (e.g., 10^{-6}) to prevent division by zero.

The final CROCS coordinates $\mathbf{p}_{\text{CROCS}} \in [0, 1]^3$ are given by the uniformly rescaled coordinates:

$$\mathbf{p}_{\text{CROCS}} = \frac{\mathbf{p}'' - m_{eff}}{L_{eff}}. \tag{10}$$

This procedure guarantees that the object remains tightly bounded by the unit cube in the camera-relative coordinate space.

A.2. Geometric Subtleties

A.2.1. CROCS Rescaling

Recall that we pre-render multiview NOCS images using a fixed reference frame. This reference frame can be labeled using the source-view camera position $(\theta, \phi, r) = (0, 0, 1)$ that captures the object in its default creator-intended pose at a default camera distance. At training time, we would like to set the source camera to arbitrary locations to ensure a rich training distribution. To change the reference frame from $\theta = 0$ to a new source-camera location θ' (we will treat the other parameters in Appendix A.2.2), we simply rotate the *ground-plane axes* of all pre-rendered NOCS maps using the SO2 rotation

matrix $[[\cos\theta', -\sin\theta'], [\sin\theta', \cos\theta']]$. The ground-plane axes are the Red and Green channels of the NOCS maps in our case.

Our on-the-fly NOCS-to-CROCS canonicalization comes with a challenge, arising from the fact that the object was scaled to fit the NOCS cube at $\theta = 0$. When we rotate the RGB reference cube to follow the primary camera to θ', then the object may no longer be bounded by the rotated cube. See Figure 6 where we visualize this issue for a fixed object scaled to fit the reference cube at $\theta = 0$.

In the upper row of Figure 6a, we get some CROCS values outside the range $[0, 1]$, because the object exceeds the boundaries of the $[0, 1]^2$ CROCS reference square. Some values of θ' such as 45, 135, 215, and 305 degrees are affected the most, because they lead to the greatest object overhang. This is not ideal—the scale of CROCS values which the model needs to predict depends on the reference frame. Since the reference frame (i.e., source camera position) is not actually supplied to the model, the model can only learn to predict the variable scale by overfitting. We aim to avoid any dependence on the reference frame. To this end, we introduce a CROCS rescaling operation to eliminate the dependence (Figure 6a, lower row).

Taking a concrete example, Figure 6b shows a cube-like object from different angles, along with the pre-rendered NOCS images (second row), and canonicalized CROCS maps without and with rescaling (third and fourth rows). Here are two observations from this figure: **1)** The front-facing edge of the cube, visible at $\theta' = 45$ as yellow in the second row, disappears in the third row because the CROCS values exceed the $[0, 1]$ range (and the color map is defined on the same range). If we rescale the values in the third row, we see the edge reappears in the fourth row. **2)** The maximum overhang (in the third row) occurs at $\theta' = 45$, where the range of observed y values expands to $[-0.2, 1.2]$. The length of this expanded interval is nearly $\sqrt{2} \approx 1.41$, which can also be deduced geometrically—it equals the length of the diagonal of the fixed square in Figure 6a.

Geometrically, we expect no object overhang at all if a given object is bounded by a sphere of diameter 1 (a tighter bound than the unit cube). We believe a lot of objects fall in this category. For this reason, our model does reasonably well even if we train it to predict CROCS without rescaling. But we observed a noticeable gain in the prior's performance on CROCS with rescaling. The latter setting removes the reference-frame-dependent scale that the model needs to predict in some cases (objects like a cube).

Note that for a given object, the rescaling needs to be performed across all multiview pointmaps (K=8 target views in our case) simultaneously, since any given view only shows a particular facet of the object. (A single pointmap may entirely miss the variation along its depth component.) Our rescaling is reminiscent of the pairwise pointmap renormalization performed by DUSt3R (see Eq. 3 in [43]). All the results in this work use CROCS with rescaling.

A.2.2. No Canonicalization for Camera Elevation or Distance

In Appendix A.2.1 we discussed how to canonicalize NOCS when switching the source camera from θ to θ'. Here, we discuss the remaining camera extrinsic parameters. Since all cameras are oriented to face the center of the object, we do not need to consider their rotations. In fact, we do not need to canonicalize for changes in the camera distance r. It only serves as a zoom parameter to augment the training data.

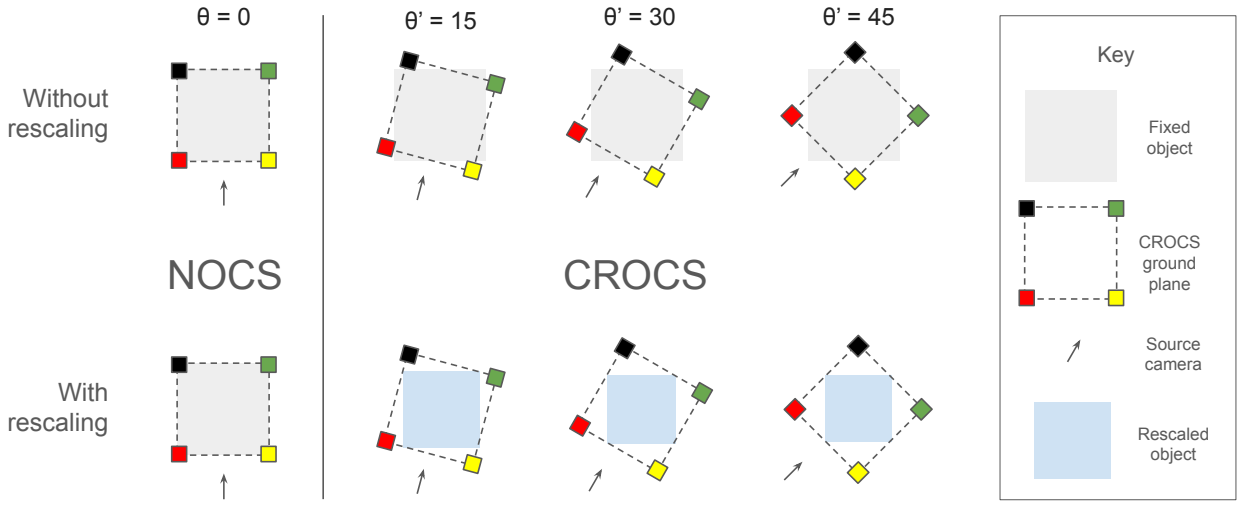
The only remaining parameter is the elevation angle ϕ, which determines the source-camera height. Recall that the target cameras are also placed at the same camera height. In fact, ϕ determines the difficulty of the multiview prediction problem. For instance, at $\phi = 90$ (a straight-down view of the object), the source and target camera locations converge to the same point—the apex of the hemisphere. In this case, the prediction task is reduced to rotating the object in the image plane.

Say we raise/lower the source camera from ϕ to ϕ' (Figure 7). This would reveal more/less of the object's upper/lower surface, and hence more/less of the NOCS color representing the up-axis (Blue in our case). We could potentially correct for this by tilting the reference cube so the source camera is pointed at the same cube face as before at ϕ. This would help maintain the same color bias for the source camera. However, the target cameras would no longer point at the same cube faces respectively—rather, their distribution of colors would shift (toward the up-axis/away from it in the case of the opposite target camera) or rotate (in the image plane for other target cameras). Consequently, the distribution shifts in the CROCS colors would be inconsistent across target cameras.

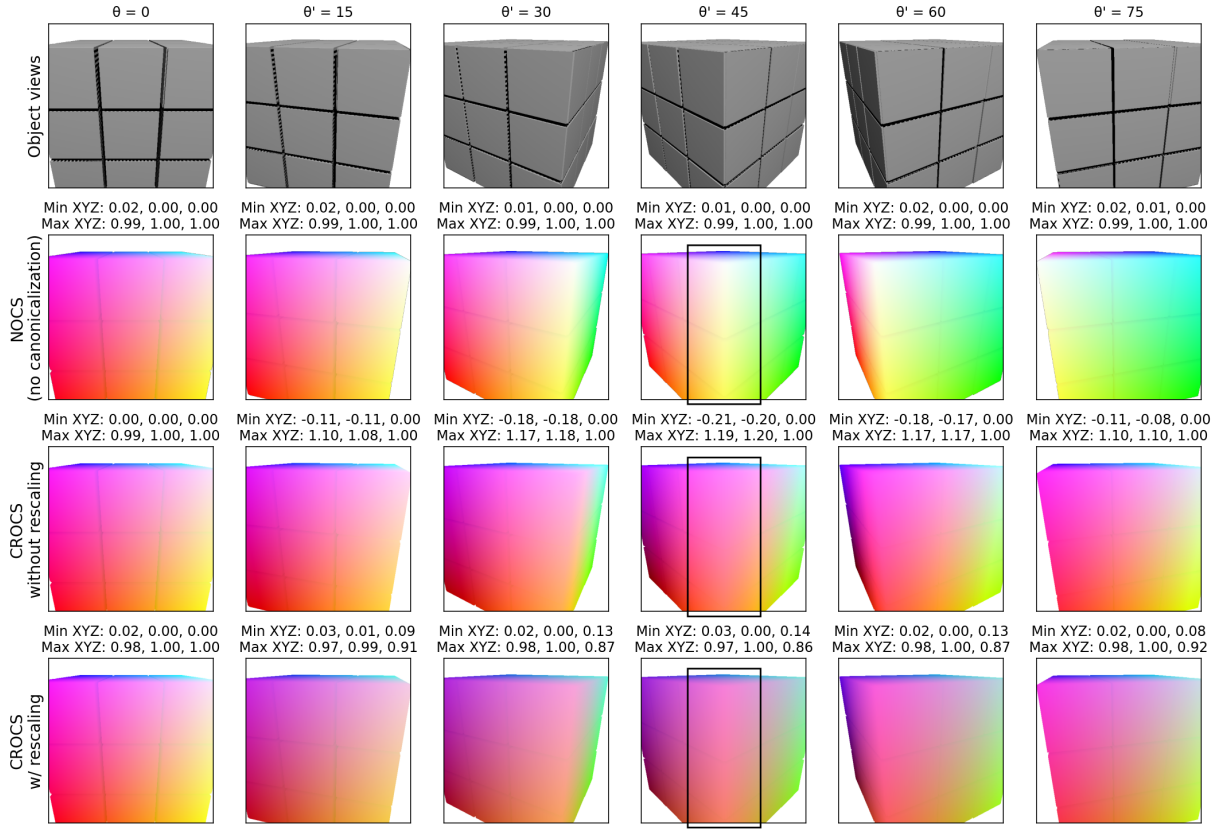
Based on this geometrical observation, we choose not to canonicalize for changes in the camera elevation ϕ'. This has the effect of leaving it to the model to infer the camera elevation from the source image (implicitly in contrast to One-2-3-45 [26]). It differs from our treatment of θ', which leaves the model oblivious to the azimuthal rotation of the cameras.

Our decision not to canonicalize for ϕ is forced by the design choice of our target camera poses, which follow the camera height of the source camera. An alternative choice would be to tilt the camera plane and the CROCS cube together, leaving the cameras at different heights in Figure 7-Middle[1]. Our choice of target poses is based on how hu-

[1]This alternative is equivalent to using a stationary multiview camera rig, but orienting the object randomly before taking pictures, as in CAT3D [10], likely LRM [15], and datasets like YCB [2]. Changing the object's orientation decouples the object's up-axis from the world's up-axis. It can leave the object in unnatural, physically unstable poses.



(a) **The general case** (simplified to 2D). **Top row:** An object that fits the NOCS reference square exactly at $\theta = 0$ is no longer bounded by the CROCS reference square at $\theta' \in \{15, 30, 45\}$. Due to object overhang, some CROCS values lie outside $[0, 1]$. The extent of the overhang depends on θ'. **Bottom row:** We rescale CROCS based on the observed range of multiview values for any θ'. This ensures the object is once again bounded tightly by $[0, 1]^2$.



(b) **A concrete example in 3D.** We show the minimum and maximum NOCS/CROCS values observed above each pointmap image. At $\theta' = 45$, we see that the front-facing edge of the cube appears squashed without rescaling, because its X and Y values lie outside the range [0, 1].

Figure 6. **CROCS rescaling.** We examine the effect of rotating the RGB reference cube used to paint the object's surface, following the source camera as it moves around a fixed object. θ (the azimuthal angle) denotes the default camera position, while θ' denotes a new position. We consider the largest possible objects—a square in 2D or cube in 3D.

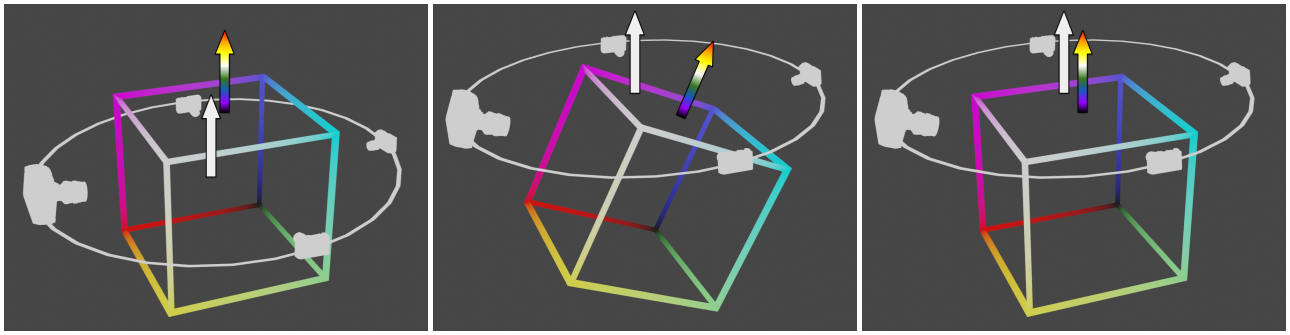


Figure 7. **CROCS when varying the camera elevation angle ϕ.** As in Fig. 3, the large camera denotes the source view; the smaller cameras denote 3 of 7 novel views. The white arrow denotes the normal direction w.r.t. the camera plane (drawn as a white circle connecting the cameras). The colored arrow denotes the normal of the CROCS color cube. **Left:** at the default angle $\phi = 0$, each camera is pointed at a particular face of the RGB color cube, and has a consistent color distribution across examples. **Middle:** when the source camera is raised to ϕ', we could hypothetically rotate the color cube by the same degree to ensure the source camera still points at the same face of the cube. The issue is—tilting the color cube toward the source camera tilts it away from the opposite camera, and also rotates the colors seen by the remaining target cameras (by ϕ' and $-\phi'$ in their respective image planes). The color shifts experienced by the four cameras are clearly inconsistent. **Right:** If we choose not to tilt the cube through ϕ', all cameras undergo a small and consistent shift in the distribution of colors toward the up-axis color (blue). If the model can infer ϕ', it can also predict the shift. We follow the *Right* approach.

mans perceive and reason about objects. We tend to see side or top-down views of *level* objects. On mental rotation tasks (e.g., when asked what an object looks like from "behind"), we tend to imagine a rotation of the object either in the image plane, or in depth about the world's vertical axis [36, 39]. We find it harder to imagine a potentially bottom-facing view. In addition to aligning with human perception, modeling object rotations at typical views/poses (e.g., top-down or side angles) could also be more useful for downstream applications.

B. Training and Evaluation Details

B.1. Dataset

We use the following sets of assets from Objaverse: 1) the LVIS subset expanded to 83k examples from the same categories. We use object type labels predicted with high confidence from [17] to expand the LVIS subset. 2) The KIUI subset [2] comprising 101k additional assets. 3) From the Objaverse-XL alignment subset [4], we used a combination of GLB, OBJ, and FBX assets after filtering out certain (a) terrains and HDRI environment maps, (b) layouts and rooms, and (c) textureless FBX assets that rendered as pink. In total, we used 620k assets (Objaverse and Objaverse-XL) for training. In addition, we held out 50k assets randomly sampled from Objaverse-XL after the filtering step.

To render NOCS images, we adapted a script from BlenderProc[3] [5] which creates a special NOCS material for the surface of a given object. We also use their Blender settings (the CYCLES engine with 1 diffuse bounce, 0 glossy bounces, and 0 ambient occlusion bounces) for rendering NOCS. We export them in the EXR format to ensure linear-light values, preserving the continuity of NOCS values on the object's surface.

[2] https://github.com/ashawkey/objaverse_filter

[3] https://github.com/DLR-RM/BlenderProc/

B.2. Model Hyperparameters

We model images at 256×256 resolution. When encoded to latents, the dimensionality reduces to 32×32×4. Hence, each superimage comprising K=8 views has dimensionality 64×128×4.

We train the diffusion models using a standard cosine noise schedule, and $t \in [0, 1000]$. The denoiser predicts ϵ, the noise added to the superimage Z_t at time t. We use the standard diffusion loss, $L_\epsilon = \mathbb{E}_{t,\epsilon,Z_0}||\epsilon - f_\theta(Z_t, t)^2||$.

We train the prior and decoder modules with a conditioning dropout probability of 0.08 on the source image. We do not use any dropout on the CROCS images fed to the decoder to ensure it adheres to the geometry. At test time, we use a fixed classifier-free guidance weight of 2.0 on the source image for both modules. We use an exponential moving average of the model parameters for evaluation, although this is not essential.

The unPIC base model has a total of 1.1B parameters across the prior (470M), decoder (470M), and two VAEs (84M each). It takes about 1min 13s to run all model components, with classifier-free guidance over 1000 denoising steps, on an A100 GPU. Note that we did not attempt to improve the runtime, as that was not the core aim of this work. Techniques such as step distillation, or recent advances in Optimal Flow Matching [22] which facilitate straight trajectories for fast denoising, would be readily applicable to unPIC.

B.3. Choice of K=8 Views

To motivate why $K = 8$ views are sufficient to produce a good high-level 3D reconstruction, we take a larger number of ground-truth views ($K = 24$), and look at the pairwise similarity between them. This can be computed using CLIP embeddings (like our CLIP-MV metric). See Figure 8 for a heatmap of pairwise similarity scores.

The black crosses that occur in the heatmap show the frequency at which rotated views become dissimilar to each other. We see 6 black crosses on either side of the diagonal of the matrix. This number estimates how many views are sufficient to summarize the full 360-degree rotation. Hence, by generating 7 novel views ($K = 8$) for any input image, we can capture 3D structure at sufficient granularity for reconstruction.

Note that this analysis does not take into account top and bottom views of the object.

B.4. U-Net Architecture

The denoising network follows a standard design with a downsampling encoder path, a bottleneck middle path, and an upsampling decoder path with skip connections. The exact architecture is as follows, while the hyperparameters are summarized in Table 6:

- Input: A noisy latent superimage X and a time-step embedding t_{emb} derived from the diffusion time t. Any conditioning superimages (e.g., CROCS latents, Z^g) are concatenated with X along the channel axis.
- Downsampling Stack: This path consists of five levels. The superimage is downsampled $2\times$ at each step. In total, this reduces the superimage from 64×128 down to a 2×4 feature map, where each of the 8 views is represented by a 1×1 spatial embedding. Each downsampling level contains three Resnet Blocks, and the number of channels is given by `embed_dim * channel_mult[i]`, where `i` is the downsampling level. A Multi-Head Attention Block is applied at all spatial resolutions below and including 32×64.
- Middle Stack: This path consists of three Resnet Blocks with Multi-Head Attention Blocks in between them to process the lowest-resolution feature map.
- Upsampling Stack: This path mirrors the downsampling path. At each level, the feature map is upsampled, concatenated with the corresponding skip connection from the downsampling path, and passed through three Resnet Blocks. A Multi-Head Attention Block is also applied at specified resolutions.
- Output: The final feature map is processed through a normalization layer, a Swish activation, and a final 3×3 convolution to produce the denoised latent superimage of the same shape as the input X.

Core Components:

- Resnet Block: A standard residual block containing two 3×3 convolutional layers, Group Normalization, and Swish activations. The time-step embedding t_{emb} is projected and used to perform a feature-wise affine transformation (scale and shift) on the feature map after the first normalization layer.
- Multi-Head Attention Block: A standard multi-head self-attention mechanism applied over the spatial dimensions of the feature map. This allows every location in the feature map to integrate information from all other locations, enabling global communication across the different views tiled in the superimage.

Table 6. U-Net Hyperparameters

Parameter	Value
`embed_dim`	256
`channel_mult`	(1, 2, 2, 3, 3)
`num_res_blocks`	3
`per_head_channels`	64
`unet_dropout`	0.0
`norm_type`	'Group Norm'

B.5. VAE Fine-Tuning Metrics

We fine-tuned the Stable Diffusion 1.4 VAE, on CROCS and RGB images separately, using a combination of losses: Reconstruction, LPIPS, KL-Divergence, and the GAN loss. See Table 7 showing the improvement in each VAE's metrics after fine-tuning. The CROCS VAE benefits significantly, and outperforms the RGB VAE in terms of accuracy.

B.6. Assessing Multiview Consistency using MET3R

To assess multiview consistency, we attempted to run MET3R, which is a pose-free way to evaluate the consistency of generated novel views or frames of a generated video. Using ground-truth images, we found that even a 45-degree rotation of an object is too significant for MET3R. The metric produced values (see Fig 9) exceeding the ex-

Table 7. **VAE metrics** on ObjaverseXL before and after fine-tuning on (a) CROCS pointmaps and (b) RGB images.

		RGB VAE	CROCS VAE
Before Fine-Tuning	PSNR	37	38
	FID	62	16
	KL	28005	20823
After Fine-Tuning	PSNR	38	51
	FID	16	2
	KL	87496	35936

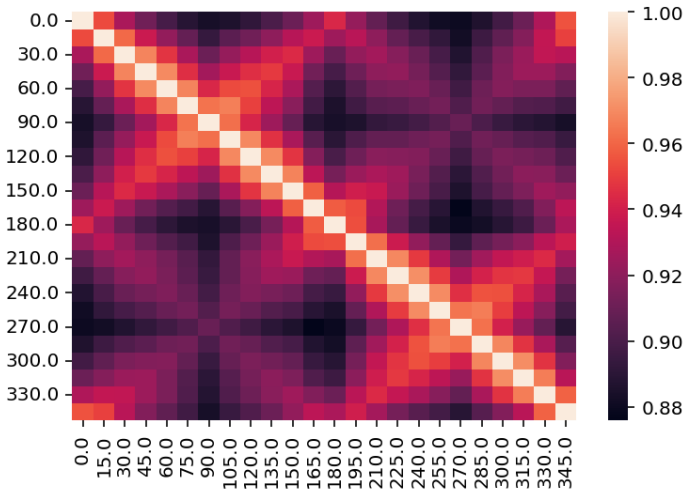


Figure 8. **Pairwise cosine similarities of CLIP embeddings for rotations of a given object**, averaged across a subset of ObjaverseXL. The x- and y-axes both show 24 views across a 360-degree rotation.

pected value of the metric on random noise (around 0.3). We ran with the standard settings, using the MAST3R backbone, DINO16 features, FeatUp upsampling, and the cosine distance.

B.7. Baselines

For single-image-to-multiview synthesis, we compare unPIC with the following baselines:

1. **CAT3D** (512x512): a SoTA-quality, geometry-free diffusion model trained using masked modeling of views on diverse datasets. While the original model is closed source, we received a Colab and checkpoint from the authors. The model uses a coarse view-sampling strategy to generate 7 anchor views given 1 source image. Subsequently, it conditions on the anchor views to generate a finer set of views using the same model. We focus on the coarse stage for our evaluation.
2. **EscherNet** (256x256): a geometry-free diffusion model equipped with a specialized camera encoding (CaPE), allowing a flexible number of input images and target views. It uses the pretrained StableDiffusion v1.5 model as its latent diffusion backbone.
3. **Free3D** (256x256): a geometry-free multiview diffusion model that uses Plücker rays to encode target camera poses and modulate the diffusion latents. It is trained on Objaverse only.
4. **InstantMesh** (512x512): a geometry-aware, feed-forward framework that generates a 3D mesh from a single image. We use the Large variant of the model. InstantMesh first employs a multi-view diffusion model, Zero123++, to produce a set of 3D-consistent views from the input. These views are then processed by a transformer-based sparse-view Large Reconstruction Model (LRM) that directly outputs a mesh. The model was trained with direct geometric supervision (depths and normals) on the mesh representation using a differentiable iso-surface extraction module (FlexiCubes). We use the frames rendered from the 3D mesh for our evaluation.
5. **One-2-3-45** (256x256): a SoTA version of Zero-1-to-3 [27]. Its diffusion-based image prior takes camera rotation and translation (R and T) parameters to transform a given image. Each novel view is sampled individually, leaving the outputs prone to multiview inconsistencies. We focus on evaluating the geometry-free stage, feeding R and T parameters to match our target poses, but using the model's own estimation of the camera height.
6. **OpenLRM** (288x288): a geometry-aware but deterministic method based on the original LRM [15]. It uses a Transformer to convert DINO features into implicit triplane representations of 3D shape. The triplane tokens are spatially indexed/queried to parameterize a NeRF model of the scene, mitigating view consistency issues and allowing image-generation at arbitrary camera poses.

C. Additional Results

Task 1: Novel-view synthesis. We provide additional multiview comparisons between unPIC and the baselines in Figure 10. We show five examples at eight target views each.

Task 2: 3D reconstruction. We visualize our point clouds and compare them with two baselines in Figure 11. We also show meshes obtained by applying a surface-reconstruction algorithm (Screened Poisson [18]) on our point clouds. We show 40 examples at one novel view each.

In-the-wild generalization. We apply unPIC to arbi-

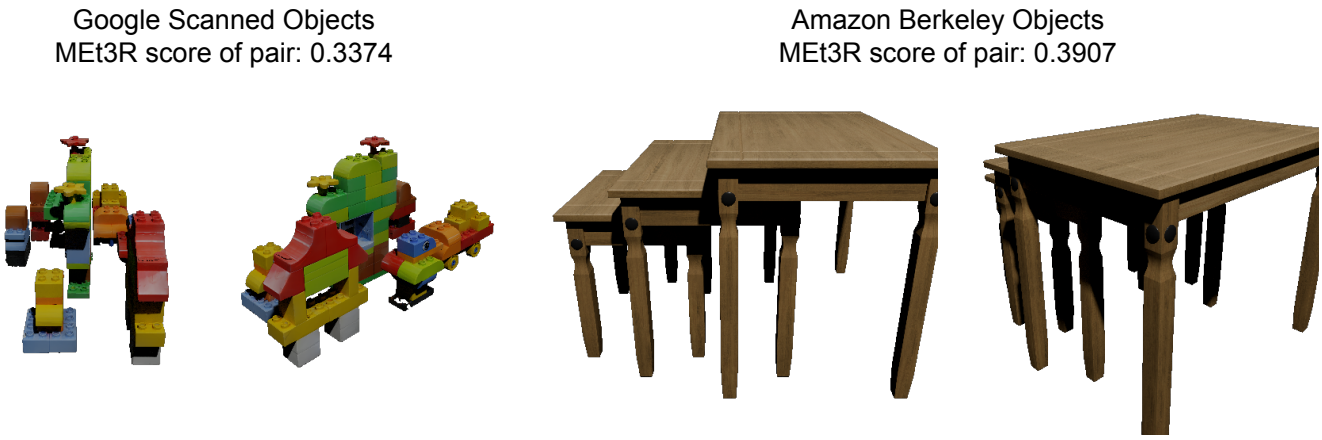


Figure 9. **MET3R as a metric for multiview consistency** produced values in excess of those expected from random noise, even on ground-truth images. The 45-degree rotation in each case is too great to facilitate a dense 3D reconstruction.

trary real-world images in Figure 12. We also cover failure cases in Figure 13.

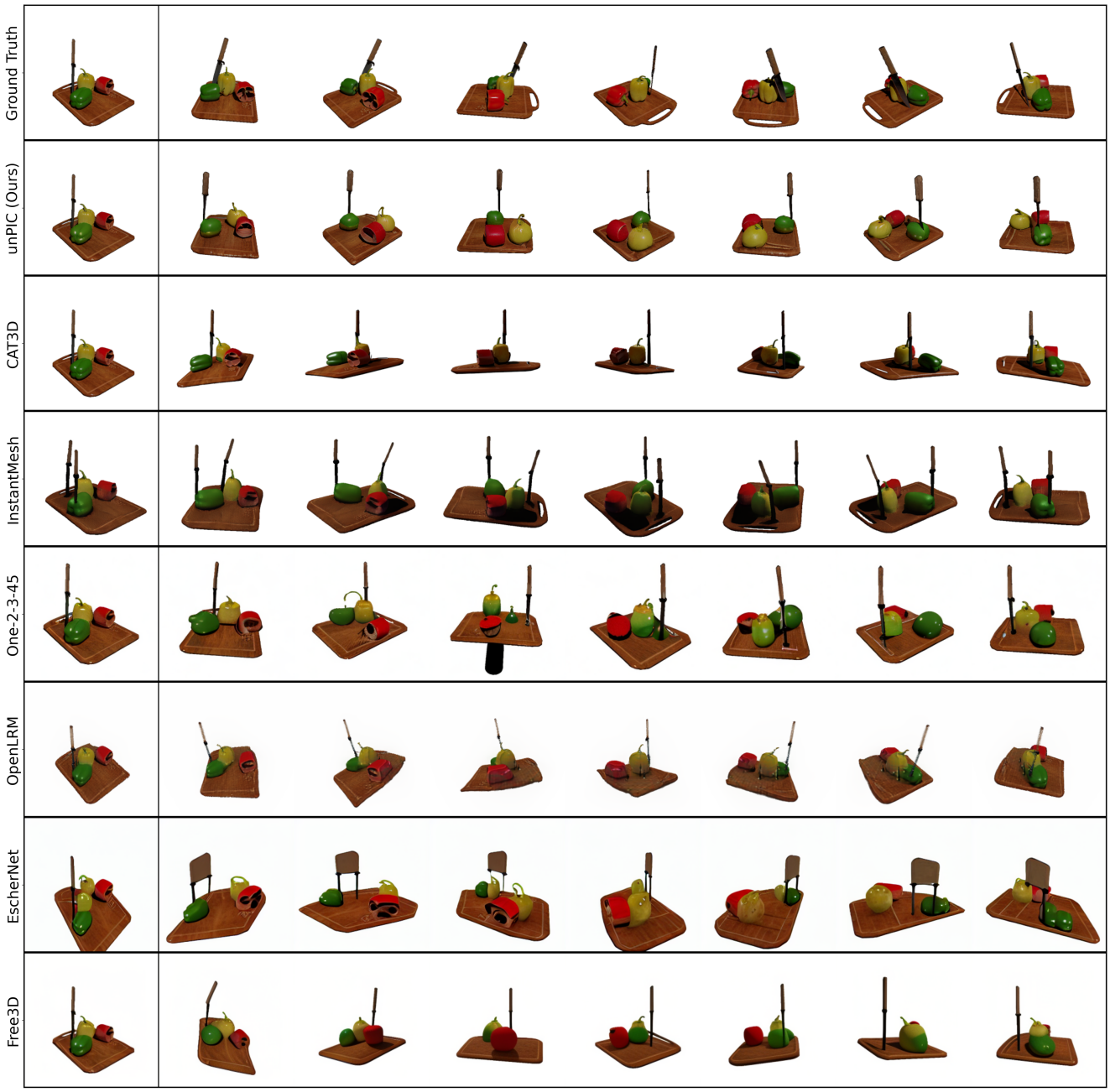


Figure 10. **Novel views from baselines versus our method** (figure continues on next page). We show an example from our ObjaverseXL test set.

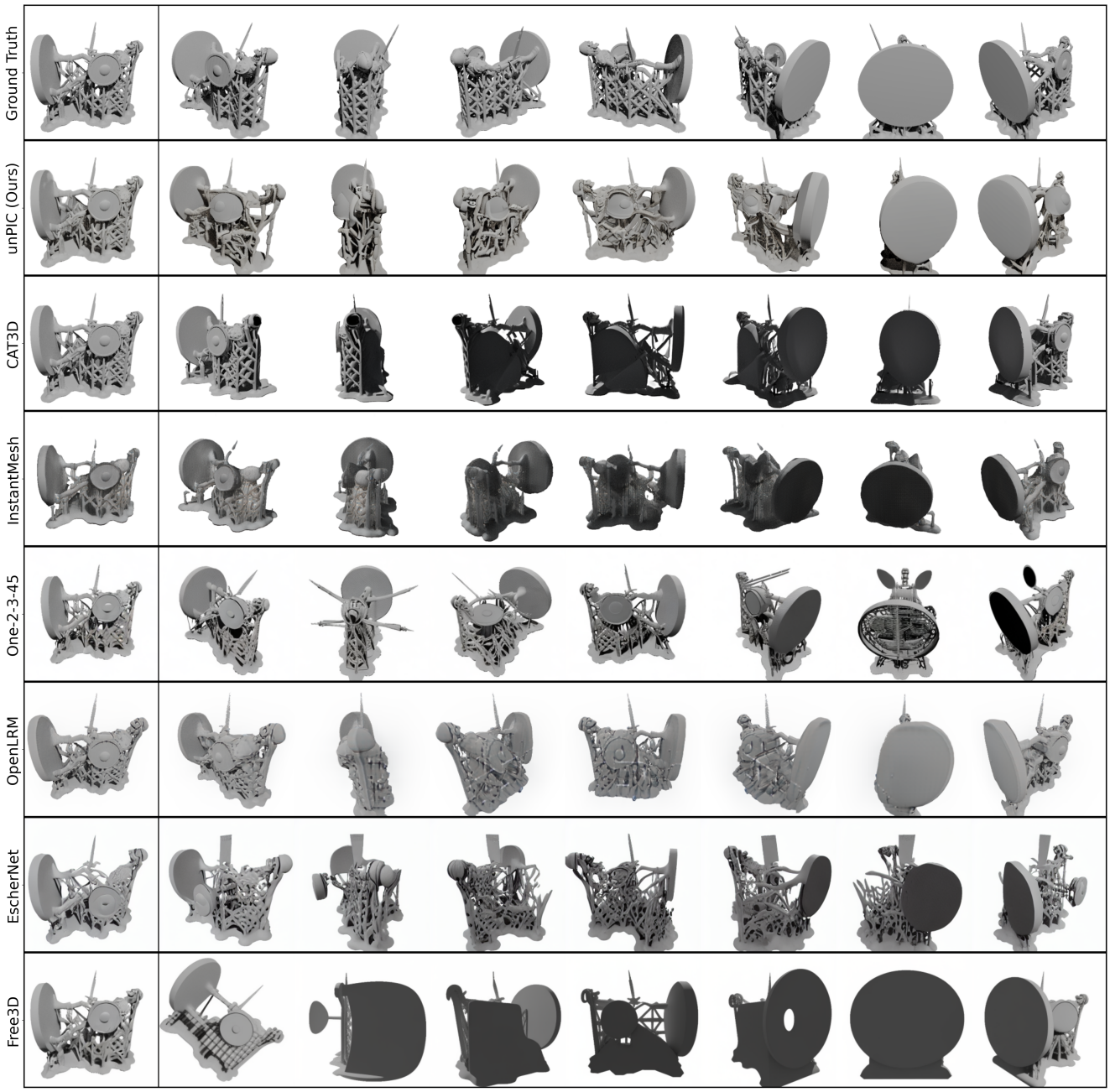


Figure 10. **Novel views from baselines versus our method** (figure continues on next page). We show an example from our ObjaverseXL test set.

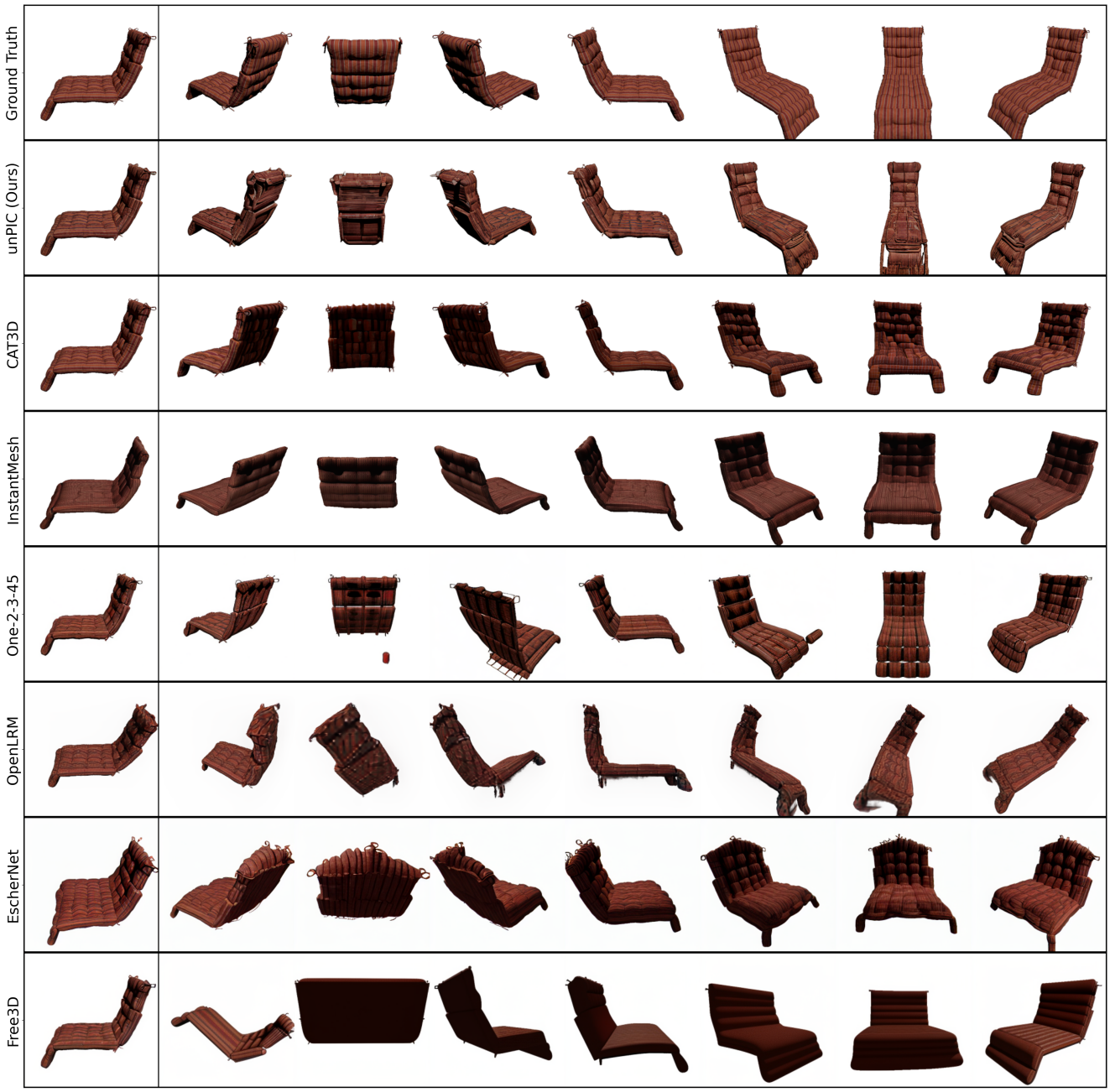


Figure 10. **Novel views from baselines versus our method** (figure continues on next page). We show an example from our Amazon Berkeley Objects test set.

Figure 10. **Novel views from baselines versus our method** (figure continues on next page). We show an example from our Digital Twin Catalog test set.

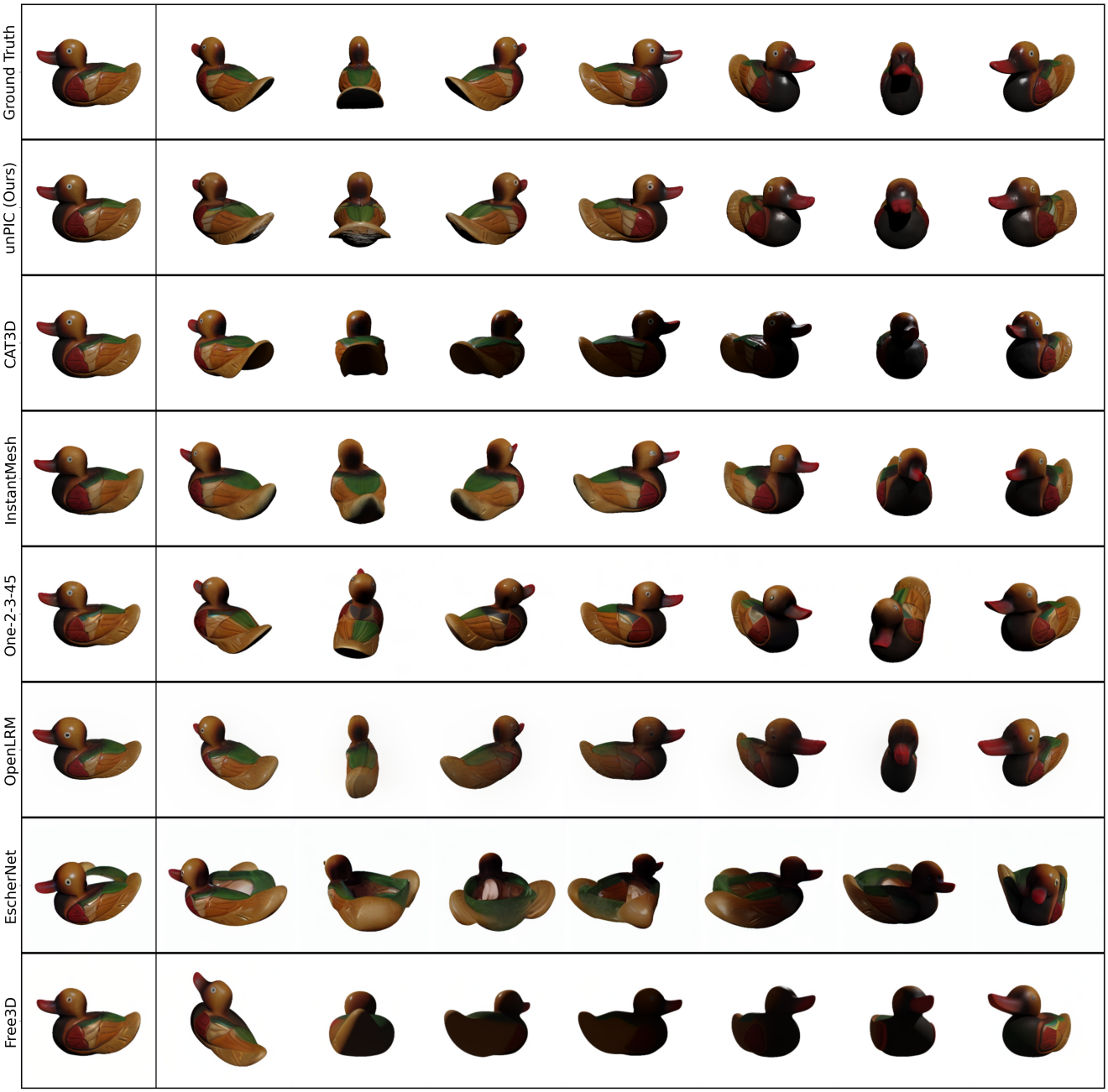
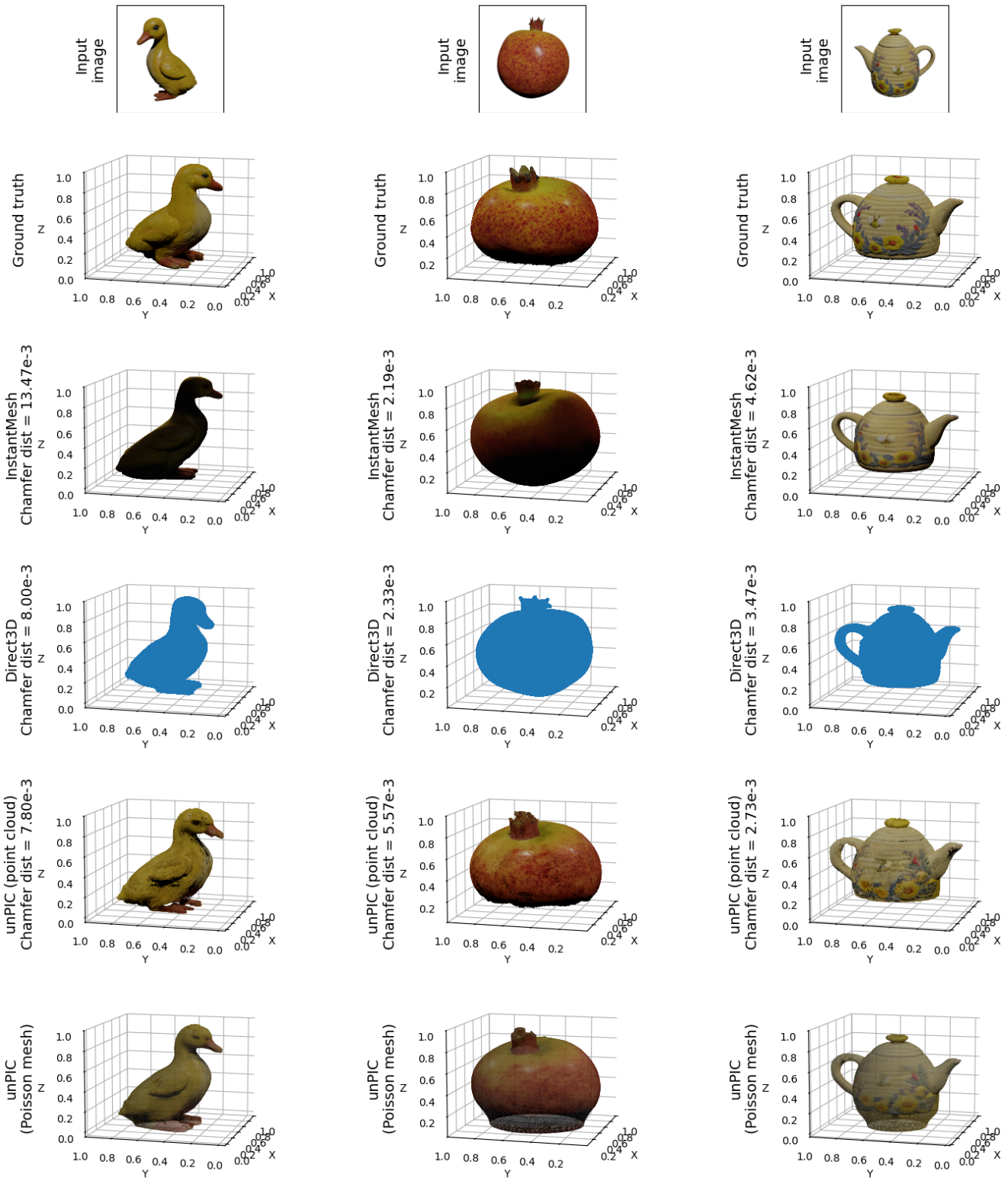
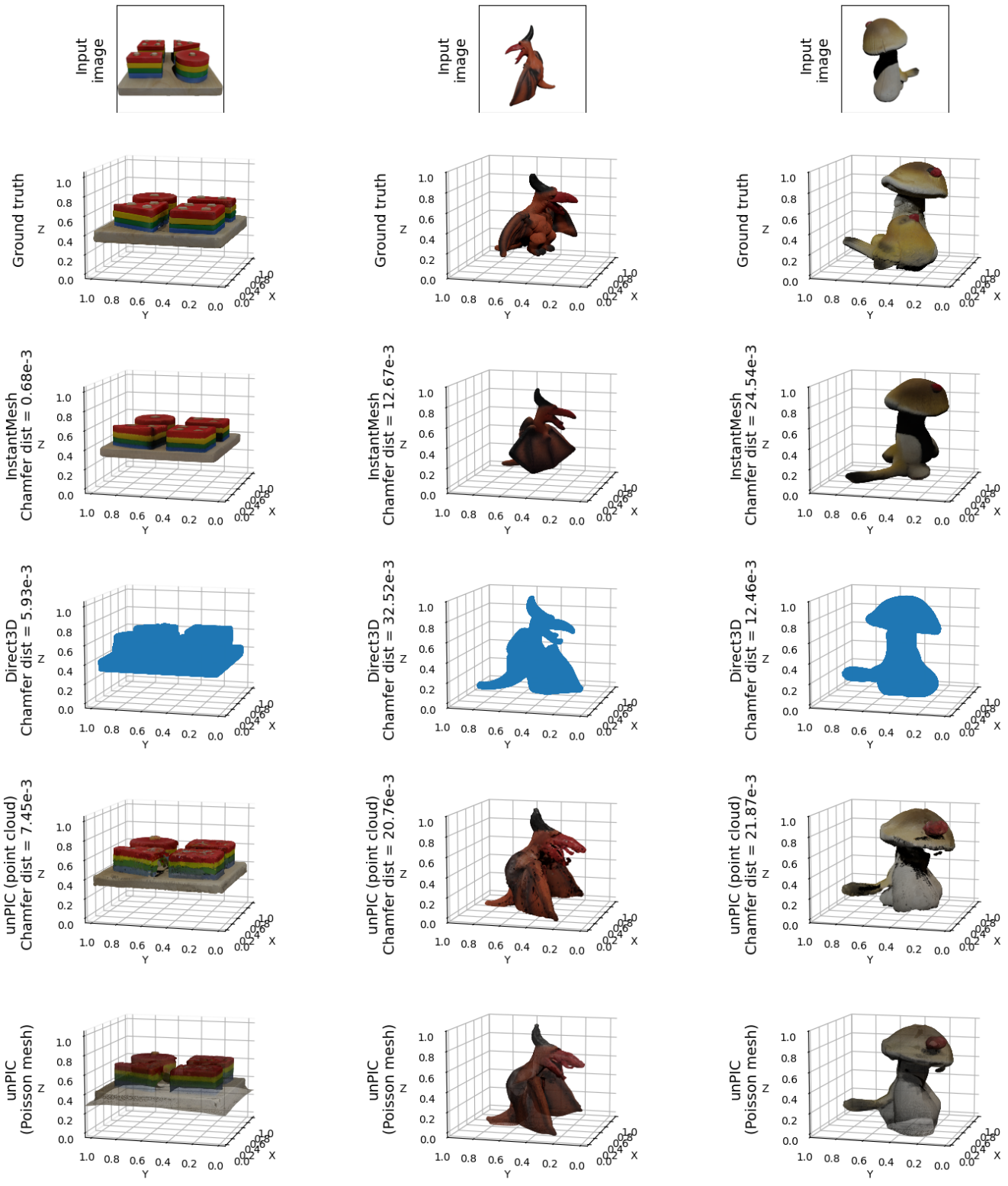


Figure 10. **Novel views from baselines versus our method** (continued). We show an example from our Digital Twin Catalog test set.



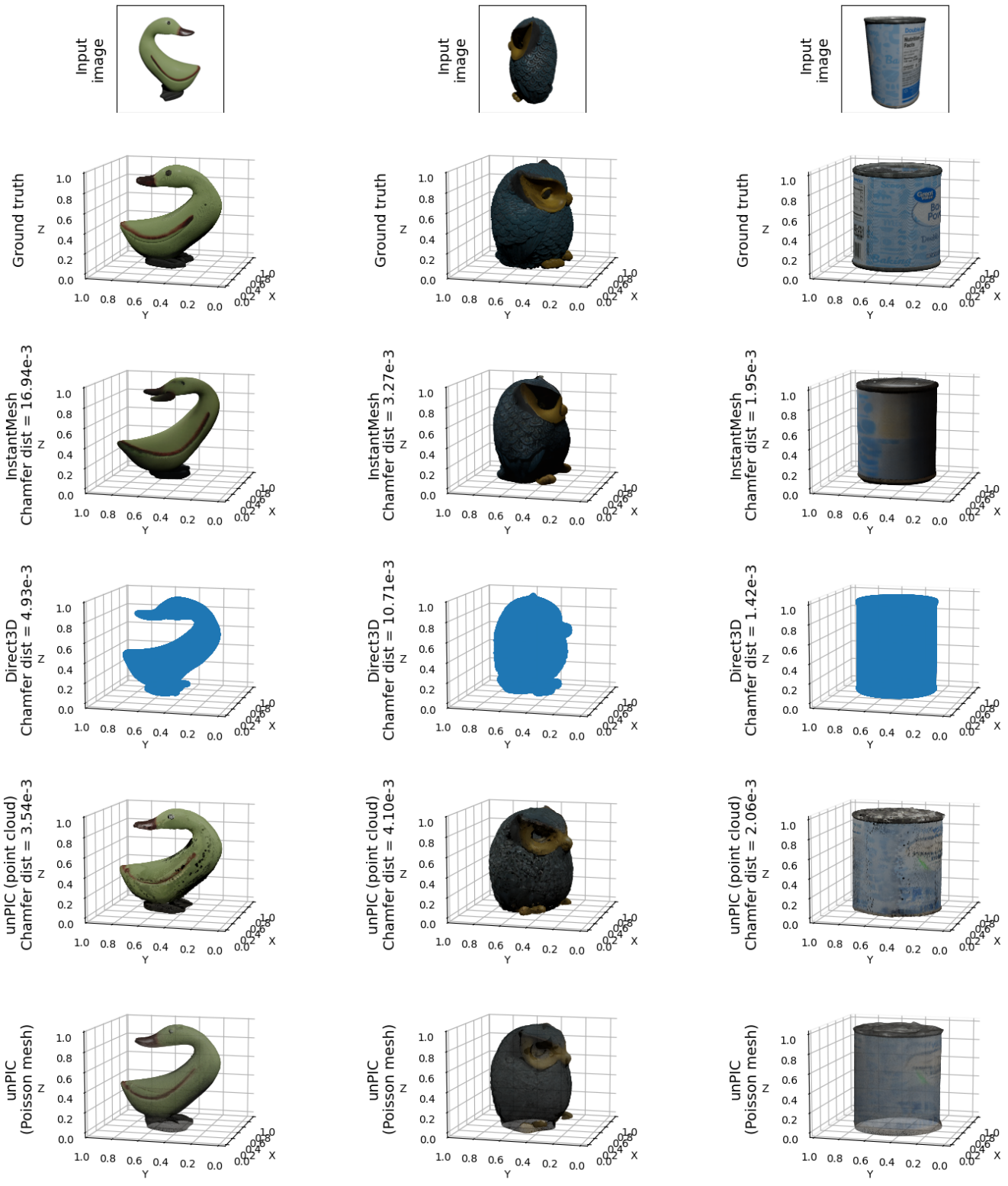
Example 1 Example 2 Example 3

Figure 11. **3D reconstruction results from unPIC vs baseline methods (Part 1 of 8).** We show a 195° rotation of the object relative to the input image. The bottom row (unPIC's reconstructed mesh) is visualized with transparency to show the full 3D shape. (Figure continues on next page.)



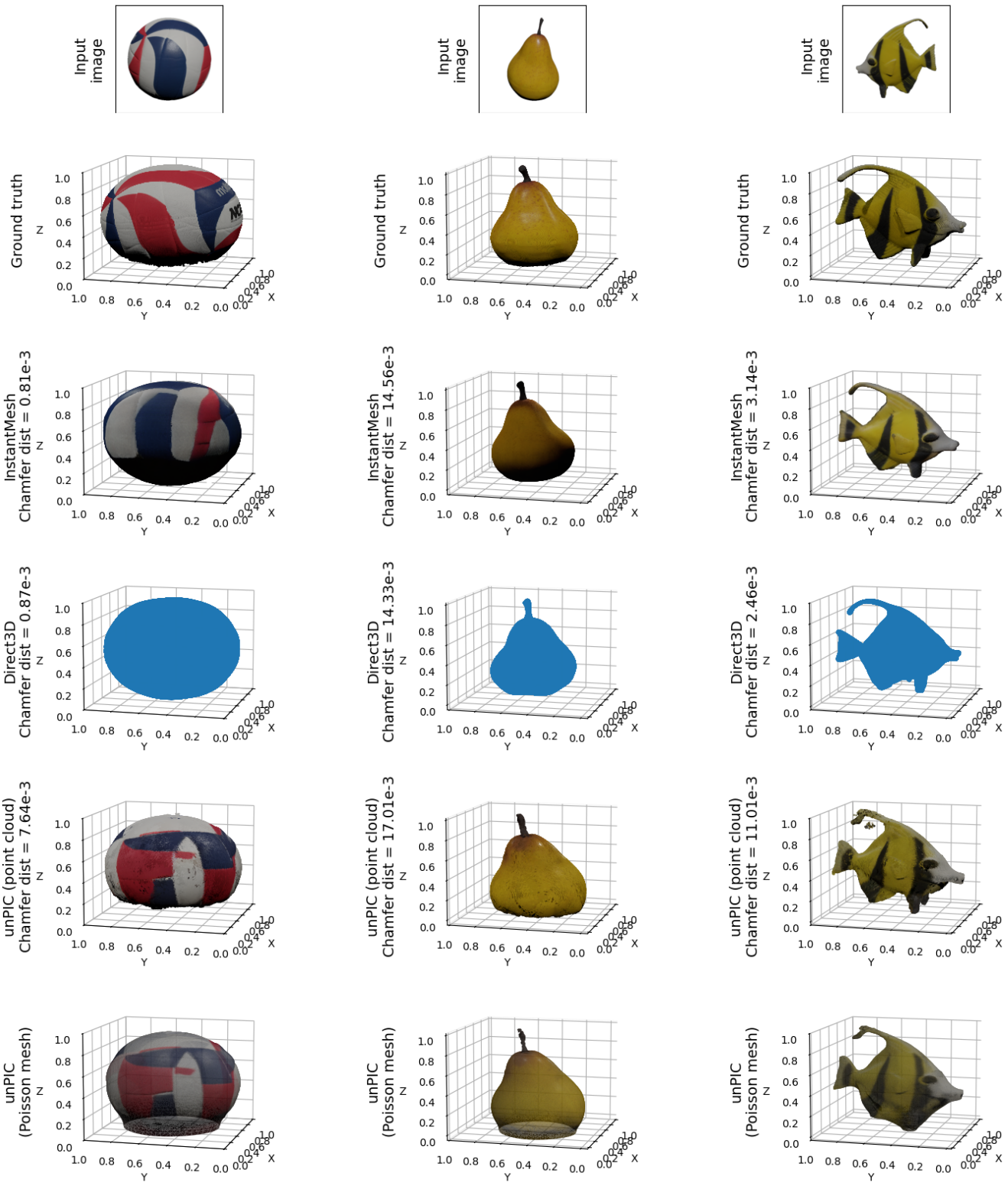
Example 4 Example 5 Example 6

Figure 11. **3D reconstruction results from unPIC vs baseline methods (Part 2 of 8).** We show a 195° rotation of the object relative to the input image. The bottom row (unPIC's reconstructed mesh) is visualized with transparency to show the full 3D shape. (Figure continues on next page.)



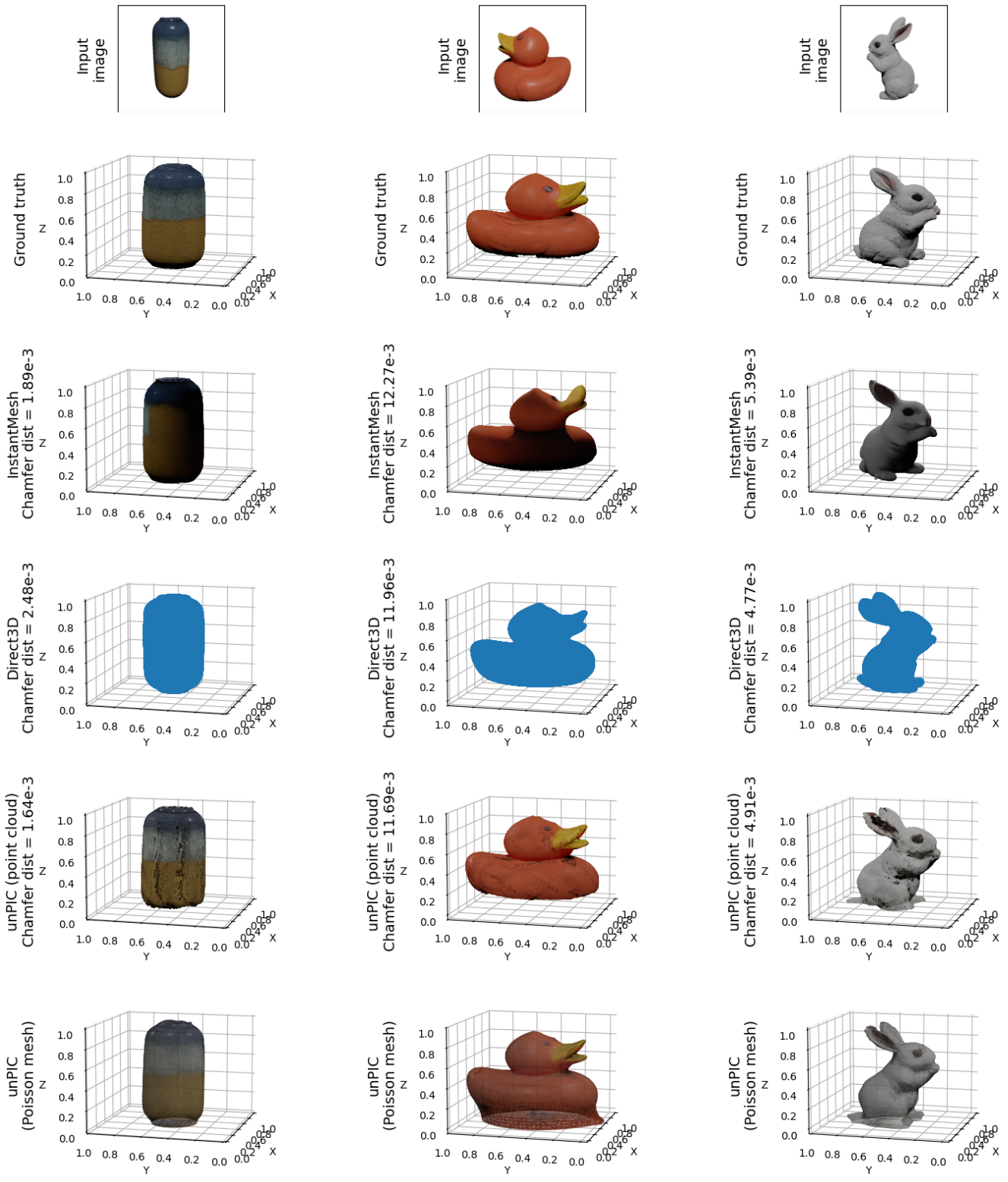
Example 7 Example 8 Example 9

Figure 11. **3D reconstruction results from unPIC vs baseline methods (Part 3 of 8).** We show a 195° rotation of the object relative to the input image. The bottom row (unPIC's reconstructed mesh) is visualized with transparency to show the full 3D shape. (Figure continues on next page.)



Example 10 Example 11 Example 12

Figure 11. **3D reconstruction results from unPIC vs baseline methods (Part 4 of 8).** We show a 195° rotation of the object relative to the input image. The bottom row (unPIC's reconstructed mesh) is visualized with transparency to show the full 3D shape. (Figure continues on next page.)

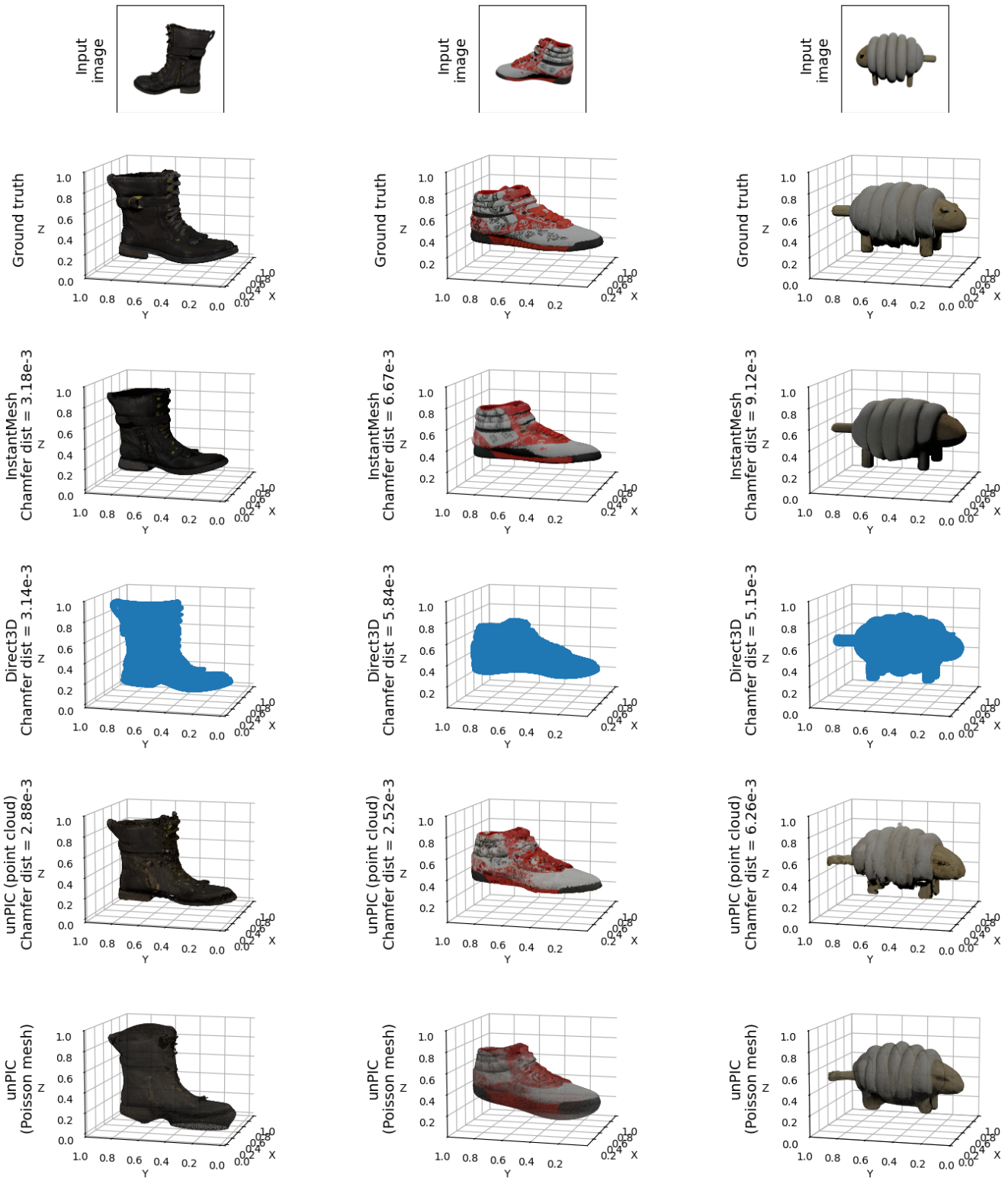


Example 13 Example 14 Example 15

Figure 11. **3D reconstruction results from unPIC vs baseline methods (Part 5 of 8).** We show a 195° rotation of the object relative to the input image. The bottom row (unPIC's reconstructed mesh) is visualized with transparency to show the full 3D shape. (Figure continues on next page.)

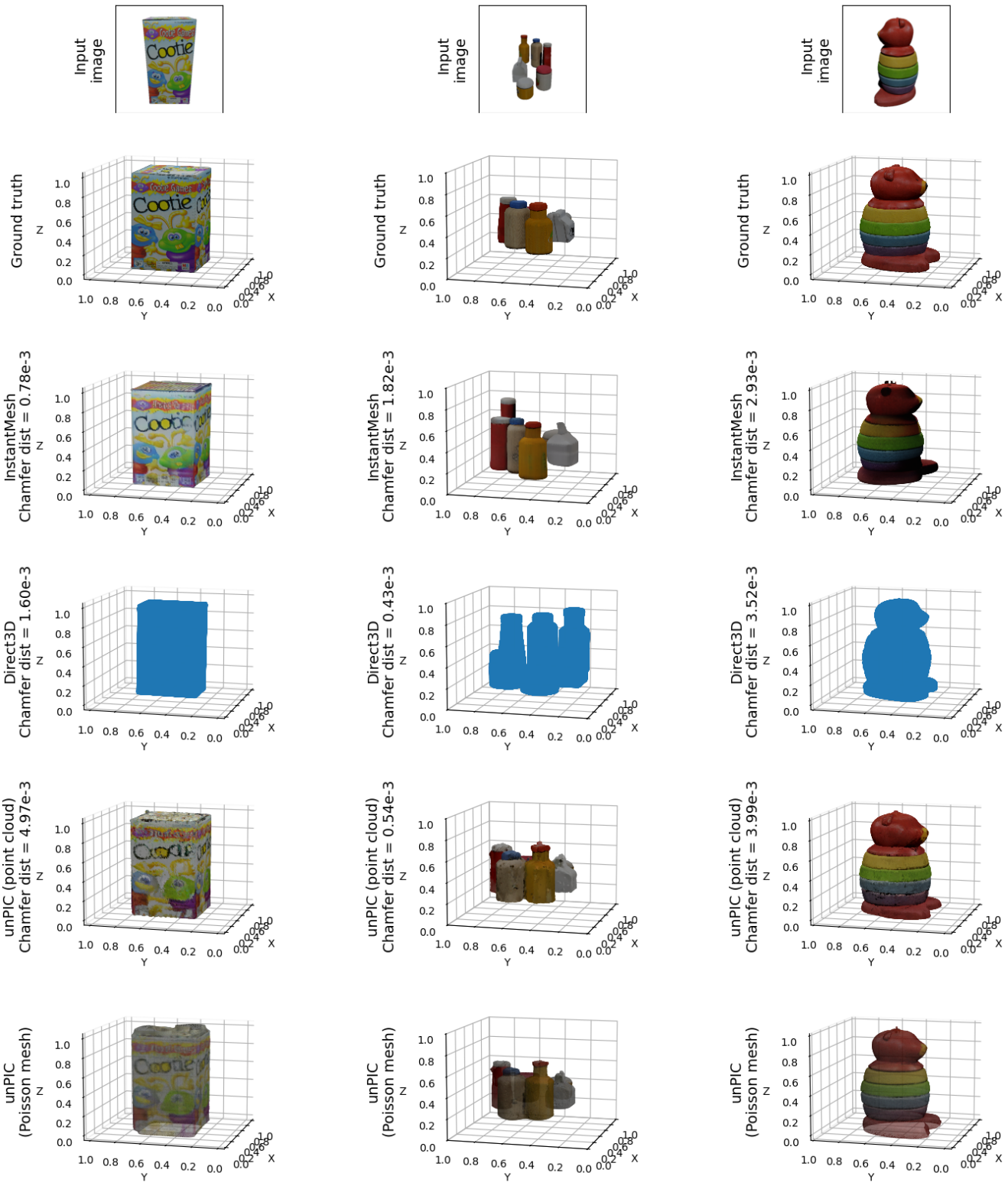
Example 16 Example 17 Example 18

Figure 11. **3D reconstruction results from unPIC vs baseline methods (Part 6 of 8).** We show a 195° rotation of the object relative to the input image. The bottom row (unPIC's reconstructed mesh) is visualized with transparency to show the full 3D shape. (Figure continues on next page.)



Example 19 Example 20 Example 21

Figure 11. **3D reconstruction results from unPIC vs baseline methods (Part 7 of 8).** We show a 195° rotation of the object relative to the input image. The bottom row (unPIC's reconstructed mesh) is visualized with transparency to show the full 3D shape. (Figure continues on next page.)



Example 22 Example 23 Example 24

Figure 11. **3D reconstruction results from unPIC vs baseline methods (Part 8 of 8).** We show a 195° rotation of the object relative to the input image. The bottom row (unPIC's reconstructed mesh) is visualized with transparency to show the full 3D shape. (Figure continues on next page.)

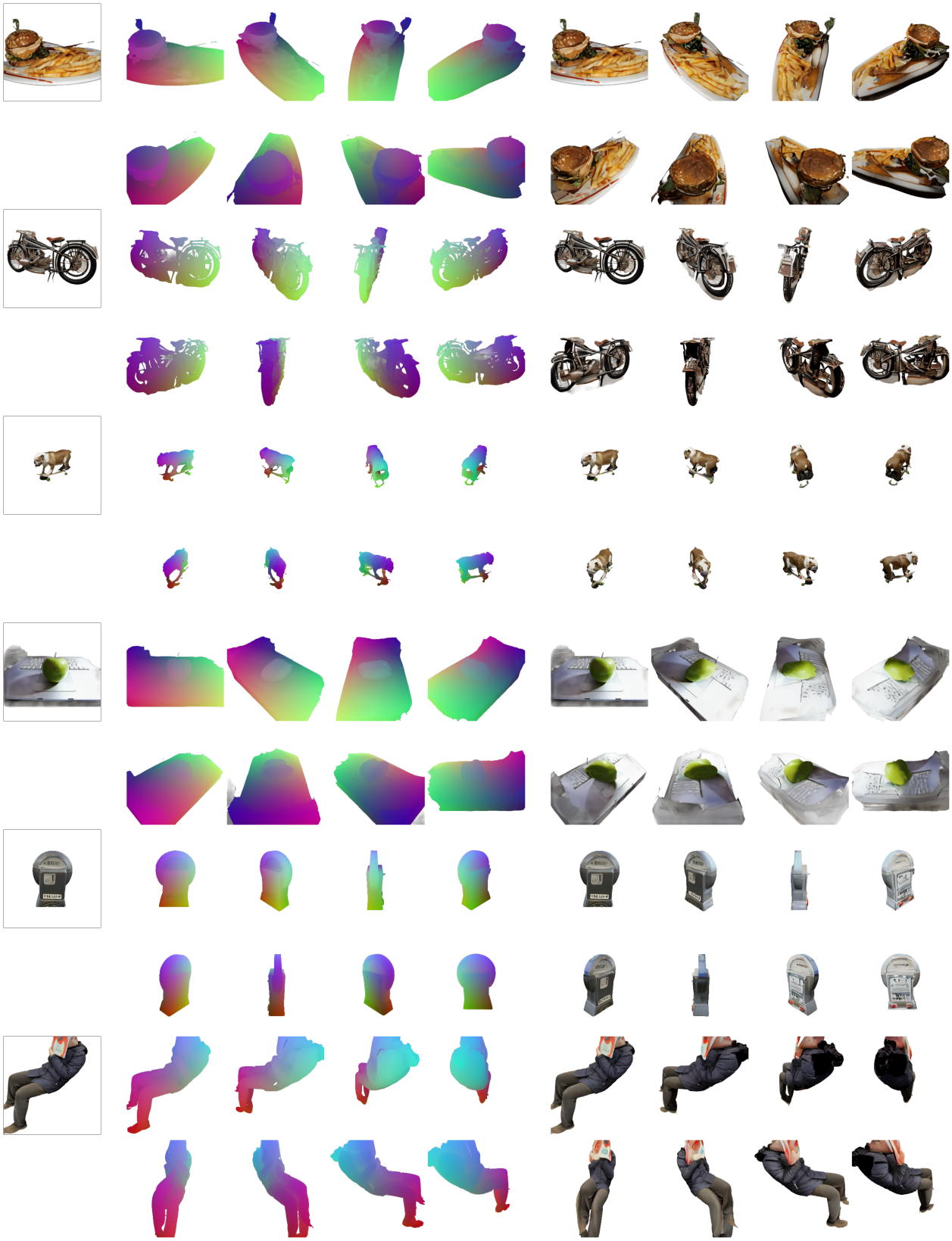


Figure 12. **unPIC on in-the-wild images.** Each pair of rows shows: the input image in the top-left box, predicted CROCS in the middle (2x4 grid), and predicted novel-view images to the right (2x4).

Figure 13. **Failure cases on in-the-wild images.** Each pair of rows shows: the input image in the top-left box, predicted CROCS in the middle (2x4 grid), and predicted novel-view images to the right (2x4). In the first example, unPIC rotates the person in the image plane, likely due to interpreting the source image as an overhead view. In the second example, unPIC predicts plausible poses, but fails to texture the images correctly, likely due to the out-of-distribution case of multiple humans in the scene.