Supplementary of "Real3D: Towards Scaling Large Reconstruction Models with Real Images"

Hanwen Jiang Qixing Huang Georgios Pavlakos
The University of Texas at Austin

Project & Code: https://hwjiang1510.github.io/Real3D/

1. More LRM Details

Problem of training LRMs on multi-view real data. Training LRMs requires two properties for the training data: 1) the camera center aligns with the object center, otherwise it leads to pose ambiguity; 2) the camera has a constant distance to the object center, otherwise it leads to scale ambiguity. The two requirements are easy to handle on synthetic data.

However, real-world multi-view images do not satisfy the assumptions. To deal with the first assumption, LRM uses the sparse point cloud of COLMAP reconstruction to re-center and re-scale the world coordinate frame. However, this solution limits the accuracy and scalability of using real-world data, as COLMAP reconstruction can be inaccurate, and it is not trivial to capture detailed multi-view videos of objects and run COLMAP on each one of them. To deal with the second assumption, LRM is modified to condition on the input view camera pose ϕ^I and intrinsics K^I, formulated as $\mathbf{T} = \text{LRM}(I, \Phi^I, K^I)$. Note that both ϕ^I and K^I vary across training samples, where ϕ^I has a *non-constant* translation vector, and K^I has *non-centered* principle points due to object-region cropping. These varying camera settings across different training samples lead to limited improvements of using real-world multi-view data (MVImgNet) for training. Please see Table 9 in main paper.

2. More Training Details

Training. As we discussed, TripoSR predicts reconstruction with random scales on different inputs. The reason is that TripoSR is not conditioned on the input view camera pose and intrinsics. Thus, the model is encouraged to guess the object scale [5]. Moreover, TripoSR is trained on a set of Objaverse data rendered using different rendering settings. Thus, TripoSR usually overfits the training scales and can not predict the scales accurately for images rendered in different settings or in-the-wild real images. The scale of the object in the output triplane can vary and may not be consistent with regard to the scale variation in the input images. Specifically, the inaccurate scales are manifested as the misalignment between the rendered and original input view when using a canonical camera pose with a constant translation vector. And the misalignment of the scale is random.

We fine-tune TripoSR to solve the problem, using the Objaverse images rendered with a constant camera translation scale. We use a learning rate $4e-5$ with AdamW optimizer and warmup iteration $3,000$. It is fine-tuned with $40,000$ iterations with an equivalent batch size of 80.

For all training, we β_1, β_2, ϵ of AdamW as 0.9, 0.96, and $1e-6$. We use a weight decay of 0.05 and perform gradient clipping with the max gradient scale of 1.0. During training, we render images with a resolution of 128×128. For each pixel, we sample 128 points along its ray. For the images in the real dataset, we crop the instance with a random expanding ratio in $[1.45, 1.7]$ of the longer side of the instance bounding box. Our inputs have a resolution of $H = W = 512$ and the triplane has a resolution of $h = w = 64$. As TripoSR requires inputs with a gray background, we render a density mask $\hat{\sigma}_\Phi$ together with the color image $\hat{I}^R_\Phi$ when calculating the cycle-consistency pixel-level loss. We apply the rendered density mask to make the background gray. We use 8 GPUs with $48GB$ memory. Training lasts for 4 days.

Evaluation. As our model inherits the model architecture of TripoSR, it can not handle real data input with non-centered principle points. To evaluate the models, we select a subset of MVImgNet and CO3D of 100 instances and use the image where the center of its instance mask is closest to the image center as input. We mask the background and perform center cropping with an expanding ratio of 1.6 times the mask bounding box size. We do not have any requirements for the target novel views. Moreover, we use the provided COLMAP point cloud to normalize the camera poses. We normalize the input pose as the canonical pose ϕ. The poses of other views are normalized accordingly with similarity transformations, following LRM [4]. We evaluate CO3D and MVImgNet with 5 views for each instance and evaluate OmniObject3D with 10 views for each instance. To evaluate the self-consistency, we use intermediate camera pose with the azimuth of

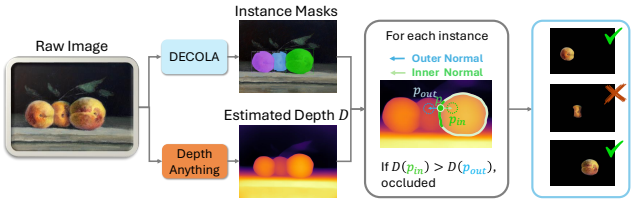

Figure 1. The occlusion detection pipeline for data curation.

$[0, 30, 60, -30, -60, 30, 60, -30, -60, 30, 60, -30, -60]$ and elevation of $[0, 0, 0, 0, 0, 30, 30, 30, 30, 60, 60, 60, 60]$.

3. Data Curation Details

We filter the instances with three criteria. First, we filter out truncated and small instances. This is achieved with simple heuristics by thresholding the instance scale and its distance to the image boundary. We use an instance scale threshold of 100 pixels and a boundary distance threshold of 10 pixels.

Second, we filter instances by their category. We empirically observe the LRM can not effectively reconstruct instances belonging to specific categories, e.g., bus. The reason is the large scale-variation between the front view and the side view. For example, when seeing the bus from a front view, the model can not reconstruct its side view with the correct scale, as the latent triplane representation has a cubic physical size. We observe that performing self-training on these instances harms the performance instead. We note this is a limitation of the Triplane-based LRM base model rather than our self-training framework.

Third, we filter the occluded instances. As shown in Fig. 1, we leverage the synergy between instance segmentation and single-view depth estimation for occlusion detection. We first detect the mask boundaries and then calculate the boundary parts that are contacting other instances. We use an erosion operation with kernel size 9 for boundary detection. The boundary is calculated as the difference between the eroded and the original instance mask. To detect boundaries that contact other instances, we use another erosion operation with kernel size 15. We erode the boundary of the current instance, then the contacting boundary is defined as its overlap region with the boundary of any other instances. We then determine whether an object is occluded based on whether it "owns" the boundary. For each instance, we sample $N = 20$ points (with replacement) on the boundary that contacts other instances. We then calculate the normal direction of the boundary at the sampled points. In detail, we use the Sobel operator to calculate the boundary tangent and normal direction. We note that we ignore the points whose 8 neighbors are all positive, during the point sampling process. We can easily know the outer and inner-mask normal directions by querying the instance mask. If the query results are both negative or positive, potentially due to the non-convex local boundary, we reject the object. Then we sample one point along each normal direction, where the sampling distance is $0.05 * s$, where $s = (b_x + b_y)/2$ and b_x, b_y are the size of the mask bounding box in x and y-axis. We then query the estimated depth at the two sampled points, denoted as D_{inner} and D_{outer}. If D_{outer}/D_{inner} is smaller than 0.95, we consider the point occluded. If half of the sampled points on the boundary are considered occluded, they vote the object as occluded.

We use DECOLA [2] and Depth Anything [6] for instance segmentation and depth estimation. We use a confidence threshold of 0.3 to filter the detection results of DECOLA. We use this low threshold for detecting all instances in the image, as any non-detected object affects the occlusion detection results. However, we empirically observe that a too-low confidence threshold, e.g., 0.1, will lead to over-segmentation and false positive detection results.

4. More Results and Ablations

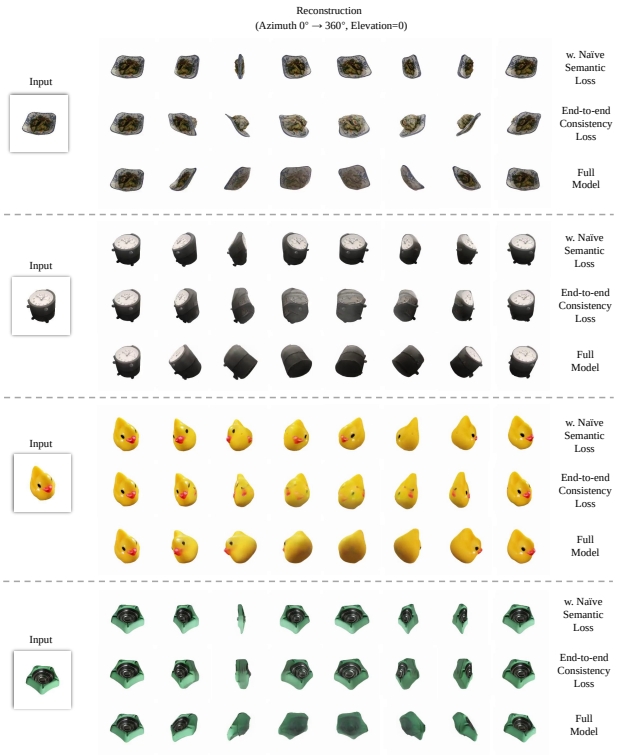

Figure 2. Visualization of semantic-guidance loss ablation experiments on MVImgNet.

Performance Gain over using Multi-View Data. In Table 9 of main paper for ablating the effectiveness of using multi-view real-world MVImgNet data, we choose OpenLRM [3] as it is the best open-source model that is trained with MVImgNet data. We note that training with multi-view real-world MVImgNet data requires pose and

Method	Eval. on GT Novel Views — *Novel View Synthesis Quality*								
	MVImgNet			CO3D			OmniObject3D		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
LRM [3]	19.75	0.864	0.112	18.31	0.849	0.126	18.20	0.831	0.144
LRM* [3]	20.16	0.867	0.105	18.82	0.852	0.119	18.53	0.837	0.138
ΔLRM* / Δmulti-view	0.410	0.003	0.007	0.510	0.003	0.007	0.330	0.006	0.006
TripoSR† [5]	17.37	0.830	0.170	15.94	0.812	0.181	17.28	0.810	0.180
TripoSR [5]	19.81	0.864	0.116	18.44	0.848	0.127	19.43	0.847	0.128
Real3D§ (ours)	20.33	0.869	0.111	19.03	0.854	0.119	19.98	0.854	0.121
Δours§	0.520	0.005	0.005	0.590	0.006	0.008	0.550	0.003	0.007

Table 1. Evaluation results on the real-world in-domain MVImageNet dataset. We note that LRM* is trained on both synthetic Objaverse and multi-view data of real-world MVImgeNet as an oracle comparison (results in gray). TripoSR† is the original TripoSR without our fine-tuning. Real3D§ is trained on single-view images of MVImgNet without access to the multi-view information. We use 1 image for each object instance; in contrast, LRM* uses 30 multi-view images (in average) for each object. We highlight the best results. We also include the gain (Δ) by using real data. ΔLRM* is same as Δmulti-view in Table 9 of main paper.

Figure 3. Qualitative comparison of the model without cycle-consistency loss (first row of each example) and with cycle-consistency loss (second row of each example).

intrinsics conditioning as we discussed in Sec. 3 of the main paper. Our base model TripoSR does not support this feature, as it does not have the image pose and intrinsics conditioning branch.

Effectiveness of Self-Training. To further evaluate the effectiveness of self-training, we compare LRM* and a Real3D version trained with MVImgNet single-view images. In this comparison, LRM* and Real3D have a similar real-world training data distribution. We note that LRM* is trained with multi-view data, where each instance of MVImgNet contains about 30 views. In contrast, Real3D only uses one image of each instance. Thus, Real3D uses the same number of

Figure 4. Visual comparison with prior works and ground-truth novel views.

shape instances for training as LRM*, but the number of training images is 30× less. As shown in Table 1, Real3D outperforms LRM* and achieves larger improvements in most of the results, demonstrating the effectiveness of our self-training strategy.

	MVImgNet	CO3D	WildImgs	OmniObject3D
SF3D	19.58	18.34	18.68	19.84
SF3D + ours	**20.43**	**19.10**	**19.31**	**20.36**

Table 2. PSNR of SF3D and ours. For rendering, we decrease points sampled on each ray from 128 to 64 to fit SF3D into our GPUs in all experiments.

	MVImgNet	CO3D	WildImgs	OmniObject3D
No real data	19.81	18.44	18.18	19.43
Controlled real data	20.27	18.96	18.74	19.98
Controlled + in-the-wild real data	**20.53**	**19.18**	**19.00**	**20.17**

Table 3. PSNR results of self-training with only controlled real data and both controlled and in-the-wild real data.

Original TripoSR Performance. We also report the performance of original TripoSR (denoted as TripoSR†) in Table 1. Due to its random scale prediction, we observe low evaluation metrics for TripoSR†. We use a grid search to find the best evaluation metrics by using different camera-to-world distances.

Visualization of Ablations. We visualize the reconstruction of ablated models in Fig. 2 and Fig. 3. Using naive consistency loss makes the model copy the front of the object to the back of the reconstruction. Using an end-to-end cycle-consistency loss makes the reconstructions deformed in a wrong manner. Our full model can reconstruct the geometry correctly, especially the concave local geometry. Using our cycle consistency loss improves the texture and the geometry of the reconstruction, by avoiding texture leaks, unnatural and deformed shapes in general.

Generalization to other base models. Other than TripoSR, we test with another base model, SF3D [1]. We show results in Table 2, where the improvements demonstrates the generalization of our self-training method to other base models. Note that to fit SF3D to our 40GB GPUs, we use a smaller number of points sampled on each ray, reducing it from 128 to 64. This generally degrades the evaluation results. However, this does not affect our observation that our self-training method improves the base model.

Detailed analysis of real data. We curate 1) 200K images from well-controlled data, e.g. ImageNet; and 2) 100K images from in-the-wild style Internet data, e.g. OpenImages. Tab. 3 shows that both subsets contribute to better performance. We uphold ethics by avoiding sensitive resources like LION-5B and excluding human-related categories.

5. More Visualizations

We include additional visualizations in Fig. 4. We observe that methods with generative priors usually suffer from unrealistic reconstruction, where the synthesized novel views of real objects are incorrect. This leads to the compounding error of the two-stage generation-then-reconstruction framework. Moreover, we also observe these methods usually suffer from not photo-realistic reconstruction at the back views and unaligned reconstruction content with the input images. In some other cases, they can produce high-quality reconstruction, while the reconstruction content, object scale, and object pose are different from the input image.

Figure 5. Visualization of meshes predicted by InstantMesh (first row of each example), TripoSR (second row of each example) and our method (last row of each example).

Figure 6. Representative failure cases of Real3D reconstructions.

6. Mesh Visualization

We include the mesh visualization in Fig. 5. We observe that the other two baselines perform worse than Real3D, particularly in cases with non-common object shapes, while InstantMesh specifically struggles to faithfully reconstruct thin structures.

7. Failure Cases

We include some failure cases of Real3D in Fig. 6. We observed that the reconstruction quality of Real3D can be compromised in cases with very unusual viewpoints, e.g., upside-down views, as well as in cases with low image quality, e.g., blurry images with potentially truncated content.

References

[1] Mark Boss, Zixuan Huang, Aaryaman Vasishta, and Varun Jampani. Sf3d: Stable fast 3d mesh reconstruction with uv-unwrapping and illumination disentanglement. *arXiv preprint arXiv:2408.00653*, 2024.

[2] Jang Hyun Cho and Philipp Krähenbühl. Language-conditioned detection transformer. *arXiv preprint arXiv:2311.17902*, 2023.

[3] Zexin He and Tengfei Wang. Openlrm: Open-source large reconstruction models. https://github.com/3DTopia/OpenLRM, 2023.

[4] Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. Lrm: Large reconstruction model for single image to 3d. *arXiv preprint arXiv:2311.04400*, 2023.

[5] Dmitry Tochilkin, David Pankratz, Zexiang Liu, Zixuan Huang, Adam Letts, Yangguang Li, Ding Liang, Christian Laforte, Varun Jampani, and Yan-Pei Cao. Triposr: Fast 3d object reconstruction from a single image. *arXiv preprint arXiv:2403.02151*, 2024.

[6] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. *arXiv preprint arXiv:2401.10891*, 2024.