Supplementary of
MegaSynth: Scaling Up 3D Scene Reconstruction with Synthesized Data

1. MegaSynth Details

In this section, we include more details of our MegaSynth generation method. We introduce the details according to the sections in the main paper, i.e. scene floor plan, geometry and texture, and lighting.

1.1. Scene Floor Plan

We define the parameters of the scene size and object box in Table 1 and Table 2, including the categories, types, size ranges, height ranges, and probabilities. These object boxes are placed randomly in the scene, except for some categories, i.e. on-ground small box, on-roof box, and on-wall box, which have pre-defined location priors.

Scene parameters	Size range	[17.0, 30.0]
	Height range	[10.0, 15.0]
Object box parameters	# Categories	7
Large object box	Size range	[4.0, 8.0]
	Number range	[2, 5]
Small object box	Size range	[2.0, 4.0]
	Number range	[4, 8]
	Type 1	On-ground
	Prob. 1	0.5
	Height range 1	[2.0, 6.0]
	Type 2	Atop large box
	Prob. 2	0.5
	Height range 2	[2.0, 4.0]
On-roof object box	Size range	[2.0, 5.0]
	Number range	[2, 4]
	Type 1	Thin
	Prob. 1	0.5
	Height range 1	[0.5, 1.5]
	Type 2	Thick
	Prob. 2	0.5
	Height range 2	[2, 4]

Table 1. Scene floor plan details part 1.

On-wall object box	Size range	[2.0, 5.0]
	Number range	[3, 6]
	Type 1	Thin
	Prob. 1	0.5
	Height range 1	[0.5, 1.5]
	Type 2	Thick
	Prob. 2	0.5
	Height range 2	[2, 4]
Wire-frame box	Size range	[3.0, 6.0]
	Number range	[1, 3]
	Height range	[3.0, 6.0]
	Prob.	0.8
Thin stick box	Length range	[3.4, 18]
	Type 1	On-wall
	Prob. 1	1.0
	Size 1	[0.1, 0.6]
	Number 1	[5, 16]
	Type 2	In-space
	Prob. 2	0.5
	Size 2	[0.8, 1.8]
	Number 2	[2, 6]
Axis-aligned box	Size range	[2.0, 5.0]
	Number range	[1, 2]
	Prob.	0.7
	Height range	[0.2, 1.0]

Table 2. Scene floor plan details part 2.

1.2. Geometry and Texture.

We include the details of object geometry in Table 3. In detail, the probability of using cube, sphere, cylinder and cone primitives are all 0.25 for large, small, on-wall, on-roof and wireframe object. For thin stick and axis-aligned objects, we only use cubes and cylinders. Beside, fro wireframe objects, we use cube, cylinder and torus, where torus has genus, increasing the geometry and topological complexity an diversity. We apply the height field augmentations to all shape primitives except for thin sticks and axis-aligned objects.

We include the details of object textures in Table 4. After we randomly select textures and materials for all instantiated geometry primitives, we randomize the materials to improve complexity and diversity. We also have special deigns for materials of axis-aligned objects. We include details in Table 5.

Large object	Number of shape primitives	[4, 5, 6, 7, 8, 9]
	Prob. of Number of shape primitives	[0.147, 0.206, 0.294, 0.206, 0.147]
	Primitive types	Default
Small object	Number of shape primitives	[2, 3, 4, 5]
	Prob. of Number of shape primitives	[0.25,0.375,0.25,0.125]
	Primitive types	Default
On-wall object	Number of shape primitives	[2, 3, 4, 5]
	Prob. of Number of shape primitives	[0.25,0.375,0.25,0.125]
	Primitive types	Default
On-roof object	Number of shape primitives	[2, 3, 4, 5]
	Prob. of Number of shape primitives	[0.25,0.375,0.25,0.125]
	Primitive types	Default
Wireframe object	Number of shape primitives	[1, 2, 3]
	Prob. of Number of shape primitives	[0.5, 0.25, 0.25]
	Wireframe Primitive types	Torus, Cube, Sphere
	Wireframe thickness	[mean scale/30, mean scale/20]
	Sphere wireframes segments	8
	Sphere wireframes ring count	8
	Cube wireframes subdivision	[1, 2, 3]
	Cube wireframes subdivision prob.	[0.33, 0.33, 0.33]
	Torus wireframes minor radius	0.3 · mean scale
	Torus wireframes major segments	8
	Torus wireframes minor segments	8
	ProB. Adding intersecting obj.	0.5
	Types of intersecting obj.	Default
Thin stick object	Number of shape primitives	1
	Primitive types	Cube or Cylinder
Axis-aligned object	Number of shape primitives	1
	Primitive type	Cube

Table 3. Object geometry details. 'mean scale' is the average of the geometry size over the three axis.

Prob. modify mat.	0.5
Prob. modify mat. of slot	0.4
Prob. specular scene	0.2
Basic roughness range	[0.001, 0.2]
Basic metallic range	[0.001, 1.0]
Specular roughness range	[0.0, 0.05]
Specular metallic range	[0.6, 1.0]

Table 4. Material details 1.

Glass IOR range	[1.4, 1.6]
Glass roughness range	[0.001, 0.1]
Prob. Axis-aligned object glass	0.8

Table 5. Material details 2.

1.3. Lighting Details

We include the lighting details of sunlight in Table 6. We include details of luminous objects and light bulbs in Table 7.

Prob. sunlight	0.6
Sunlight strength	[0.2, 2.0]
Prob. window glass	0.5
Prob. window bar	0.5

Table 6. Sunlight details.

2. Model and Training Details

We include more model and training details as follows.

Luminous objects	Applied objects	Thin sticks
	Prob.	0.7
	Prob. slot	0.2
	Strength range 1	[0.2, 2.0]
	Prob. strength range 1	0.9
	Strength range 2	[5.0, 8.0]
	Prob. strength range 2	0.1
Light bulb	Num. range	[2, 5]
	Strength range 1	[0.2, 2.0]
	Prob. strength range 1	0.9
	Strength range 2	[5.0, 8.0]
	Prob. strength range 2	0.1

Table 7. Luminous objects and light bulbs details.

Training input and target view sampling. For each training sample in a batch, we randomly sample input views and target views from a pool of 48 views following LRM training strategy [1]. The number of input views is always 32. The number of target views is 12 for 128-resolution experiments, and 8 for 256-resolution experiments to balance the compute cost. We allow the overlap between input and target views during training. On the MegaSynth dataset, the set of 48 views are randomly sampled. On the real training data, we evenly sample frames within a distance range, which is sampled from the range of 64 to 128.

Camera pose normalization. The cameras of the input views are normalized with a random global scale between 1.1 and 1.6. For Gaussian rendering, we clip the predicted Gaussian scale of 0.135. We set a near plane of the Gaussian renderer as 0.1.

Learning rate and scheduler. In the pre-training stage, we use a peak learning rate of $4e-4$. In the tuning stage using real-world data, we use a smaller peak learning rate of $1e-4$. For joint training or training exclusively on real data, we use a learning rate of $4e-4$. All experiments adopt a warm-up of 3000 iterations and cosine learning rate decay.

Batch size. For both 128×128 and 256×256 resolution training, we use a batch size of 4 per GPU. The experiments are launched on 64 A100 GPUs thus the global batch size is 256.

Training iterations. The training iterations for Res-128 and Res-256 are 120K and 80K for each training stage (i.e., pre-training and fine-tuning stages, as well as joint-training), respectively. The final learning rate is decreased to 0 at the end of training. Specifically, we end the pre-training stage at 75K and 55K iterations for experiments on resolution 128 and 256, respectively. Thus, The effective learning rate at the end of pre-training stage is around $1e-4$. The reason is we observe that training with more iterations, especially with a learning rate smaller than $1e-4$, leads to overfit on MegaSynth and makes the fine-tuning stage fail.

Training time cost. It takes GS-LRM 7 days for pre-training and fine-tuning, and it takes 4 days for joint-training, under

resolution 128×128. It takes 11 days for pre-training and fine-tuning, and it takes 6 days for joint training on resolution 256×256.

Gaussian Settings. We use spherical harmonics of 3 for 3D Gaussians. We follow all other training hyper-parameters as the original GS-LRM [2] and Long-LRM [3].

Loss weights. We set the weights of point location loss (on synthetic data) and perceptual loss as 0.4 and 0.2, respectively. For joint training, we set the probability of sampling data from real and synthetic data as the same. For ablations, we run experiments with resolution 128×128 using GS-LRM.

Training view rendering settings. For MegaSynth rendering, we sample 36 and 12 cameras in the outer and inner parts of the scenes, respectively. We sample the FoV of cameras within the range of 45 to 70 degrees.

Other details. In our ablation, quality control means we only use four basic object types without wireframes, think structure and axis-aligned object, without material randomization, and using only ambient lighting. We render MegaSynth with Blender Cycles ray-tracing. Geometry are augmented shape primitives. Textures are sampled from datasets, e.g. MatSynth. It takes 10K CPU cores in total to create data in 3 days (Res-256), where scene creation and rendering take 0.5 and 2.5 days respectively. Rendering cost is same as other synthetic data when using same rendering tool, while our scene creation is much faster and scalable.

3. More Results

We include more visualization results with 32 input views and 32 rendered target views as well as the ground-truth target views in Fig. 1 and Fig. 2.

References

[1] 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.

[2] Kai Zhang, Sai Bi, Hao Tan, Yuanbo Xiangli, Nanxuan Zhao, Kalyan Sunkavalli, and Zexiang Xu. Gs-lrm: Large reconstruction model for 3d gaussian splatting. *arXiv preprint arXiv:2404.19702*, 2024.

[3] Chen Ziwen, Hao Tan, Kai Zhang, Sai Bi, Fujun Luan, Yicong Hong, Li Fuxin, and Zexiang Xu. Long-lrm: Long-sequence large reconstruction model for wide-coverage gaussian splats. *arXiv preprint 2410.12781*, 2024.

Figure 1. Visualizaton of input views (first row of each example), render target view and ground-truth target views (last two rows of each example). We include results on the DL3DV benchmark data.

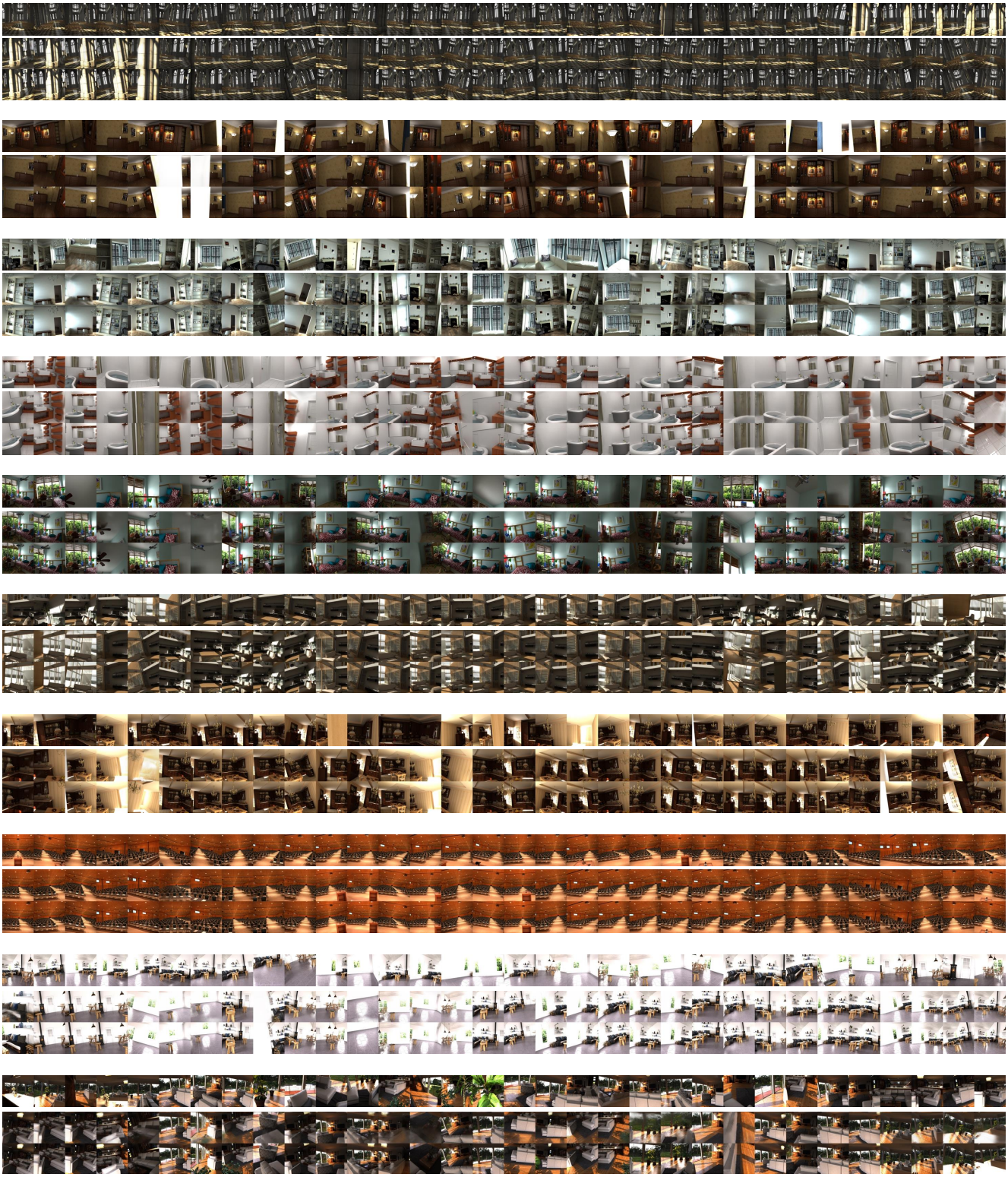

Figure 2. Visualizaton of input views (first row of each example), render target view and ground-truth target views (last two rows of each example). We include results on the Hypersim data.