

# MeshGen: Generating PBR Textured Mesh with Render-Enhanced Auto-Encoder and Generative Data Augmentation

## Supplementary Material

### A. Implementation details

#### A.1. Auto-encoder

**Hyper-parameters.** We present our hyper-parameter setting in training auto-encoder in Tab. 4.

**Data preparation.** To calculate occupancy, the target mesh needs to be watertight, but most meshes in Objaverse do not meet this requirement. Therefore, we need to convert non-watertight meshes into watertight ones. Specifically, for each non-watertight mesh, we follow CLAY [95], first computing an unsigned distance field with a resolution of  $512^3$ . Then, we use the marching cubes algorithm with a threshold of  $2/512$  to extract the isosurface. To avoid thin surfaces inside and thin surfaces, we mark all parts not connected to the outermost region as internal, which can be quickly achieved using a connected component labeling algorithm. With off-the-shelf CUDA-based tools like torchc-umesh2sdf [20] and cc\_torch [102], the preprocessing of a mesh can be done within 0.1s on a single GPU.

**Training data.** We train our MeshGen auto-encoder on a filtered subset of Objaverse [16] consisting of approximately 150k triangle meshes. For each mesh, we first normalize into  $[-1, 1]^3$  and uniformly sample 65536 surface points on the mesh surface as the input of our auto-encoder. We then sample 100,000 spatial points near the surface by adding Gaussian noise with 0.01 std to the surface points. For random spatial points, we use stratified sampling to sample in a  $64^3$  grid uniformly. We pre-compute and store surface points, sampled spatial points, and corresponding binary occupancy values for training efficiency.

**Training details.** We first train our auto-encoder for 150 epochs in the coarse stage with a batch size of 192. During training, we select an equal number of near points and random points for supervision [91]. The model obtained after the coarse stage can reconstruct the rough shape of the original mesh but lacks details. We then train the auto-encoder for another 50 epochs with the proposed render loss and ray-based regularization incorporated and a batch size of 16. The whole training process lasts 6 days on 8 NVIDIA A100 GPUs.

#### A.2. Image-to-shape diffusion model

**Generative rendering augmentation.** In generative rendering data augmentation, to enhance the similarity between the generated images and the original image, in addition to using the normal depth ControlNet and IP-adapter, we set the initial noise to the latent of the original image with

maximum noise added. For relighting diffusion, we used IC-light [94]. Specifically, during data augmentation, we randomly select one lighting direction from the pre-defined light initial latent in IC-light (i.e., uniformly select from left, right, top, and bottom), and choose one lighting condition from a set of predefined light prompts. For filtering out low-quality images, we trained an MLP evaluator on 500 samples using CLIP embeddings of the original and relighted images to estimate quality scores. This straightforward method achieves 91% accuracy on the validation set (100 samples).

**Model setup.** Our diffusion UNet takes in the noised tri-plane latent and exploits 8 ResNet blocks with spatial self-attention as the encoder and a symmetric architecture as the decoder. We exploit DINOv2-G [48] to encode the input image and inject the extracted feature to the diffusion UNet using cross-attention [56]. For the diffusion schedule, we follow SD3 [21] to use the simple yet effective rectified flow [40] with timesteps sampled from a standard logit-normal distribution.

**Training data.** We train the image-to-shape diffusion model with our proposed augmentations on a filtered subset of GObjaverse [51], which consists of about 120k high-quality meshes. During training, we randomly select one view as condition for the diffusion UNet and apply our proposed augmentations for both geometric and image enhancement.

**Training details.** To handle input images with different elevations, since the meshes in Objaverse are aligned in the gravity axis, we force the diffusion to generate meshes with an absolute elevation equal to zero. We experimentally found that this conditioning method works better than generating meshes with a rotation in elevation, as suggested in Chen et al. [13]. We train the diffusion UNet of 16 NVIDIA A800 GPUs using bf16 precision with an effective batch size of 1536. The whole training lasts for about 18 days.

#### A.3. PBR texture generation

**Reference attention.** We demonstrate how reference attention works in figure 7. To condition the diffusion model on the reference image, we pass the noised reference image (at the same noise levels as the denoising latent) into the same diffusion UNet to obtain the key and value tensors of the reference image in self-attention layers. During sampling, the key and value tensors of the reference image are appended to the key and value tensors of the multi-views, enabling the diffusion model to perceive the reference im-

Table 4. Concrete hyper-parameter setting of our render-enhanced auto-encoder.

Symbol	Meaning	Value
$N_P$	Number of points sampled from a mesh	65536
$N_z$	Number of learnable queries	3072
$n$	Number of self-attention layers	10
$d_z$	Dimension of the latent space	16
$N_s$	Number of samples for calculating ray-based regularization loss	128
$\lambda_{KL}$	Loss weight for KL loss	$10^{-6}$
$\lambda_{TV}$	Loss weight for TV loss	$5 \times 10^{-3}$
$\lambda_{MSE}$	Loss weight for normal MSE loss	1.0
$\lambda_{LPIPS}$	Loss weight for normal LPIPS loss	2.0
$\lambda_{reg}$	Loss weight for ray-based regularization loss	0.5

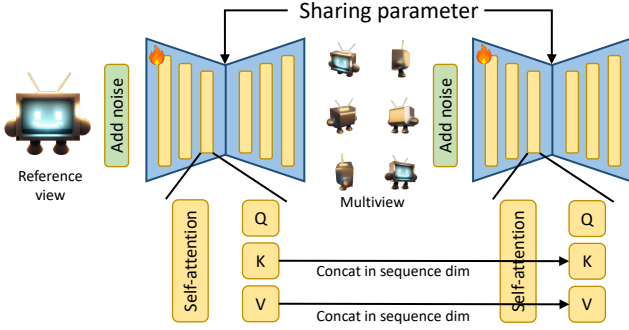


Figure 7. An illustration of reference attention.

age with more fine-grained information. This technique is also used in image editing [92] and video generation [25].

**Data preparation.** To train the geometry-conditioned ControlNet and the multi-view PBR decomposer, we render multi-view images along with corresponding normals, depth, albedo, roughness, and metallic maps from a subset of Objaverse containing PBR materials using Blender. This creates a dataset of approximately 35k multi-view images. For UV space inpainting, we calculate multi-view visible masks and back-project them into UV space to identify the invisible parts of the texture map, rendering the UV space position and normal maps. To enhance robustness, we randomly erode the visible masks in both pixel and UV spaces.

**Geometry-conditioned ControlNet.** Our geometry-conditioned sparse view generator is built upon Zero123++ [61], which generates six views according to a given front view. Specifically, Zero123++ generates a 3x2 image grid by fine-tuning Stable Diffusion [56] on Objaverse renderings. To ensure the model perceives precise depth and positional information, we did not transform depth to normalized disparity as done in the original depth ControlNet [93]; instead, we performed a

unified multi-view normalization based on camera distance and object bounding box. Specifically, the depth map is processed as  $D_{\text{normalized}} = \frac{D - \text{bias}}{\text{scale}}$ , where bias equals to camera distance minus the length of the diagonal of the bounding box (i.e. the minimal possible depth value) and the scale equals to the length of the diagonal of the bounding box.

**Multi-view target back-projection.** As detailed in the main text, the obtained multi-view PBR components are merged in UV space using back-projection with softmax. We apply a softmax operation with a temperature of 0.1 to ensure consistent textures. However, images generated by ControlNet sometimes extend beyond object boundaries, causing some pixels to be back-projected onto surfaces behind them, leading to artifacts on the final texture map. To address this, we propose a simple depth filtering technique. For each view, we identify locations in the depth map where sudden changes occur and exclude these pixels during back-projection. Our experiments demonstrate that this approach effectively reduces artifacts, and the color values of the corresponding surface points can be supplemented by other views, as shown in the middle of Fig. 8.

**PBR decomposer.** Our PBR decomposer is built on Zero123++ and incorporates multi-view shaded images using an InstructPix2Pix-based architecture. Specifically, the latent representation of the multi-view shaded image is concatenated along the channel dimension with the noisy latent. Each PBR channel is generated by the diffusion model using specific textual prompts. For instance, the model generates component  $y$  using the prompt  $y$ , where  $y \in \{\text{"metallic"}, \text{"roughness"}, \text{"albedo"}\}$ .

**UV space inpainting.** Our UV space inpainter is a multi-channel ControlNet trained on top of the LoRA fine-tuned Stable Diffusion 1.5 [56]. The input to our inpainting model is a 9-channel image: the first three channels represent the normal map in UV space, the middle three channels represent the position map, and the last three channels contain

Table 5. **Quantitative ablation study on the proposed data augmentation and comparison with Direct3D.** FS and CD denote f-score and chamfer distance.  $FS_{sym}$ ,  $FS_{asym}$ , and  $FS_{complex}$  represent f-scores for symmetric, asymmetric objects, and complex lighting images. Parentheses indicate metric changes for each modification.

Method	FS $\uparrow$	CD $\downarrow$	$FS_{sym}$ $\uparrow$	$FS_{asym}$ $\uparrow$	$FS_{complex}$ $\uparrow$
+ render-enhanced VAE	0.907	0.061	0.932	0.881(+.04)	0.841(+.008)
+ generative rendering	0.918	0.053	0.944	0.892(+.05)	<b>0.897(+.064)</b>
+ geometric alignment	0.955	0.035	0.962	<b>0.947(+.11)</b>	0.855(+.020)
Ours (full)	<b>0.970</b>	<b>0.028</b>	<b>0.974</b>	<b>0.965</b>	<b>0.902</b>

the masked texture map, with pixel values set to -1 in regions that requires inpainting. During inference, we follow ControlNet inpainting [93], applying masking in the latent space to maintain consistency in areas that do not require inpainting.

## B. More experiments

### B.1. More ablations

**Number of points in auto-encoder.** We use 65536 points per mesh in auto-encoding. This decision stems from observing that VAE reconstruction quality saturates at a high number of points (Fig. 9).

**Quantitative ablation of generative data augmentation.** As shown in Tab. 5, geometric alignment enhances performance on asymmetric objects, and generative rendering improves results under images with complex lighting conditions. Compared to Direct3D, each model design leads to significant improvements across all metrics.

**The effectiveness of ray-based regularization.** We show in the left part of Fig. 8 an example obtained using an auto-encoder trained without ray-based regularization. Without ray-based regularization, the training of the auto-encoder quickly becomes unstable, resulting in severe floaters in the reconstructed mesh.

**Quantitative ablation study on render-enhanced auto-encoder.** To better assess the importance of incorporating render loss in our render-enhanced auto-encoder, we propose several variants and demonstrate the corresponding accuracy and volumeIoU on a validation set of Objaverse consisting of 2048 objects in Tab. 6. Here, “base” represents the case with only BCE loss, while “w/. 3D GAN loss” represents incorporating the 3D patch-based GAN loss proposed in Zheng et al. [100].

As shown in Tab. 6, removing either the MSE loss or the LPIPS loss leads to a certain performance drop. Moreover, compared to the 3D patch-based GAN loss, the proposed render-based perceptual loss is more beneficial for auto-encoder training.

**UV-space texture inpainting.** In the right part of Fig. 8, we compare the mesh obtained without UV inpainting. The figure clearly shows that without UV inpainting, colors may

be missing from regions that are not visible from the fixed viewpoints of the multi-view diffusion. UV space inpainting effectively fills these regions, enhancing both the visual quality and realism of the model.

### B.2. More results

**Quantitative evaluation on texture generation.** The results in Tab. 7 show that MeshGen significantly outperforms other baselines across all metrics under both original and relighted settings, improving both reconstruction and generation metrics by a large margin.

**More comparison with LRMs.** In Fig. 14, we present more qualitative comparisons with large reconstruction models on image-to-shape generation, including InstantMesh [83], MeshLRM [78], and MeshFormer [39]. Our method outperforms other large reconstruction models in capturing geometric details, such as the webbed feet of a frog and the handle of a backpack.

**More comparison with 3D native methods.** In Fig. 15, we present more qualitative comparisons with 3D native diffusion models, including CraftsMan [34], 3DTopia-XL [11], and a more recent model with billion-level parameters, WaLa [58]. Clearly, the meshes generated by other methods are significantly lower in quality compared to ours and fail to produce meshes similar to the images, which introduces additional challenges for subsequent texturing. Although WaLa uses more parameters and a much larger training dataset than we do, thanks to our proposed augmentations, our method greatly outperforms WaLa in terms of mesh quality and image-shape alignment.

**Compare with large reconstruction models with texture.** To comprehensively compare our approach with large reconstruction models, we compare the final generated textured mesh in Fig. 10. It is evident from the figure that our method not only exceeds the previous best large reconstruction models in geometry but also produces clearer and more consistent textures.

**PBR joint generation v.s. PBR decomposition.** In addition to our proposed multi-view PBR decomposer, previous methods like HyperHuman [41] and CLAY [95] have suggested using expert branches to generate PBR components

Table 6. Quantitative ablation study on the proposed render-enhanced auto-encoder.

Setting	Accuracy $\uparrow$	VolumeIoU $\uparrow$
w/o. $\mathcal{L}_{normal}^{MSE}$	95.972	89.977
w/o. $\mathcal{L}_{normal}^{LPIPS}$	96.021	90.044
w/. 3D patch GAN loss	96.224	90.149
base	94.745	87.164
Ours	<b>96.987</b>	<b>91.045</b>

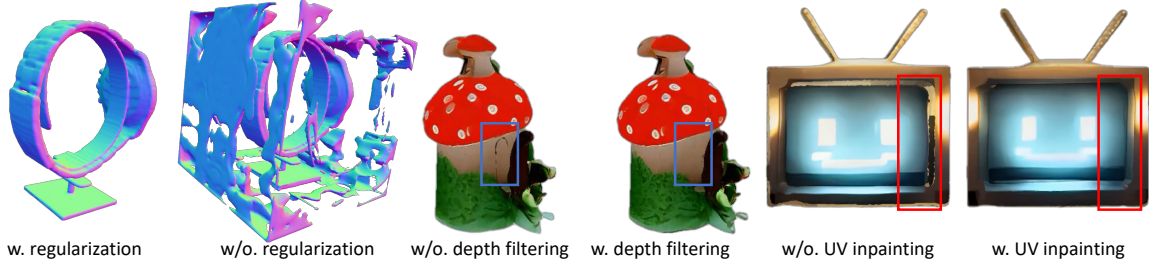


Figure 8. Ablations on ray-based regularization, depth filtering, and UV space inpainting.

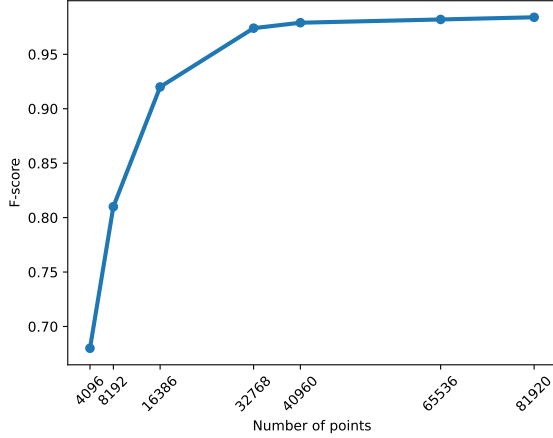


Figure 9. Ablations on the number of points used in mesh auto-encoding.

Table 7. **Quantitative comparisons on PBR texture generation.** <sup>†</sup> denotes methods unable to generate PBR textures; baked-in textures are used as albedo for relighting.

Method	Generative		Original lighting		Relighted	
	FID↓	KID↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓
TEXTure	28.03	7.6	18.74	0.157	16.21 <sup>†</sup>	0.188 <sup>†</sup>
Fantasia3D	24.16	5.22	19.04	0.134	17.42	0.155
Paint3D	25.28	5.19	19.2	0.144	16.04 <sup>†</sup>	0.192 <sup>†</sup>
Ours	<b>20.14</b>	<b>3.21</b>	<b>22.87</b>	<b>0.089</b>	<b>22.44</b>	<b>0.098</b>

from image prompts. Specifically, they use a shared backbone diffusion network and different input/output adapters to enable information sharing across different channels while exploiting information in pre-training. To compare this approach with our proposed PBR decomposer, we trained a multi-view PBR component generator based on the expert branch approach using the same data. As shown in Fig. 16, we compare the PBR components generated by both methods for the same reference image and mesh. It can be seen that the model using the expert branch tends to produce PBR channels with color blending and inaccuracies, leading to unreasonable metallic and roughness outputs. We believe this is because different PBR channels should interact with the reference image in distinct ways during generation, and the shared parts of the expert branch hinder learning these different interactions. Additionally,

compared to the expert branch-based model, our PBR decomposer inherently includes all multi-view information in the input, resulting in better consistency in the generated results. Furthermore, due to our limited data (or possibly because our method requires less data), we consider comparing both models with larger datasets as future work.

**Compare with commercial products.** In Fig. 11, we compare our method with existing non-open-source commercial products. The results for Direct3D are sourced from their paper, while those for HyperHuman Rodin are generated on their official website without the “symmetric” tags. Although our method is currently limited by lacking high-quality data and computational resources, resulting in slightly lower mesh quality compared to commercial products, our proposed augmentation allows for better alignment with the images while other commercial products tend to generate symmetrical objects. We believe that with increased computational power and more high-quality data, our method can match the mesh quality of commercial products while preserving image-shape alignment.

**Real-world images.** To validate the performance of our method on real-world objects, we present a set of textured meshes generated from casual captures in Fig. 12. As shown in Fig. 12, our method is capable of generating reasonable shapes and consistent textures when processing real objects, demonstrating the generalization ability of our pipeline.

**PBR decomposition results.** In Fig. 17, we present the intrinsic channels estimated using our proposed multi-view PBR decomposer. The results show that our PBR decomposer can accurately infer the PBR components of objects by leveraging multi-view information and can still generate multi-view consistent results under complex lighting conditions.

## C. Limitations

Although our method has made some progress in native image-to-3D generation, there are still limitations in the following three areas.

1. Due to the limited resolution of multi-view diffusion generation and the constraints of the auto-encoder used,

our texture model struggles to accurately reproduce high-frequency details, such as the text on the box in the left part of Fig. 13. We believe that using more advanced network architectures with more high-quality data could achieve higher-resolution multi-view generation.

2. Our texture model finds it challenging to accurately capture textures and lighting effects from input images when dealing with objects with complex high-frequency information and lighting conditions, as shown by the face in the center of Fig. 13.
3. Our geometry and texture generation model currently cannot effectively handle transparent objects, as illustrated by the object on the right in Fig. 13.

Addressing these limitations will be the focus of our future research.



Figure 10. Qualitative comparison on textured meshes with state-of-the-art large reconstruction models, including InstantMesh [83], MeshLRM [78] and MeshFormer [39].

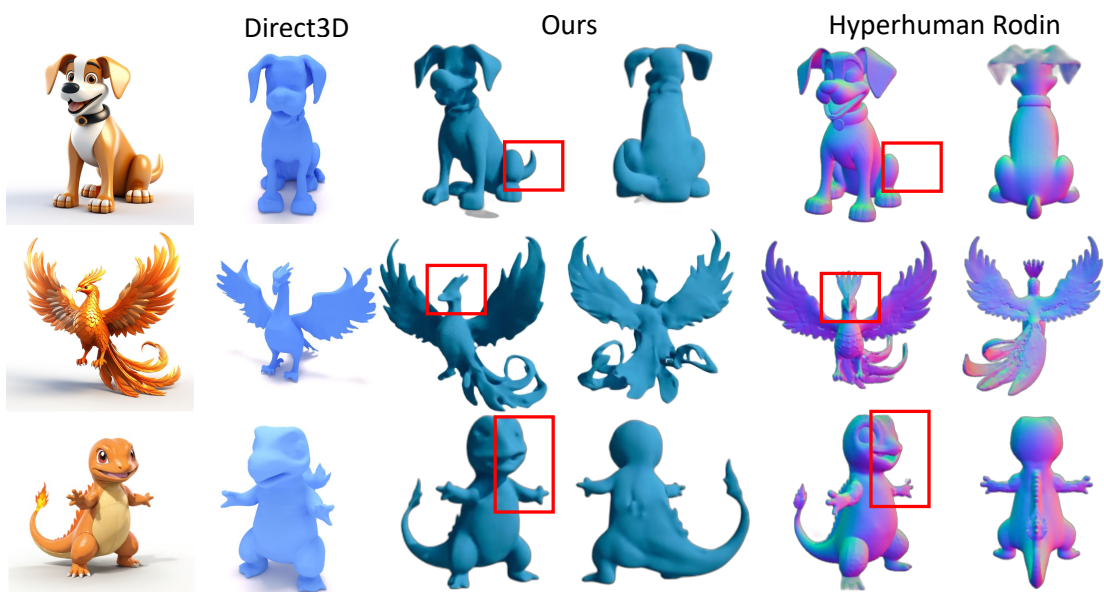


Figure 11. Comparison with non-open-source commercial products, including Direct3D and Hyperhuman Rodin.

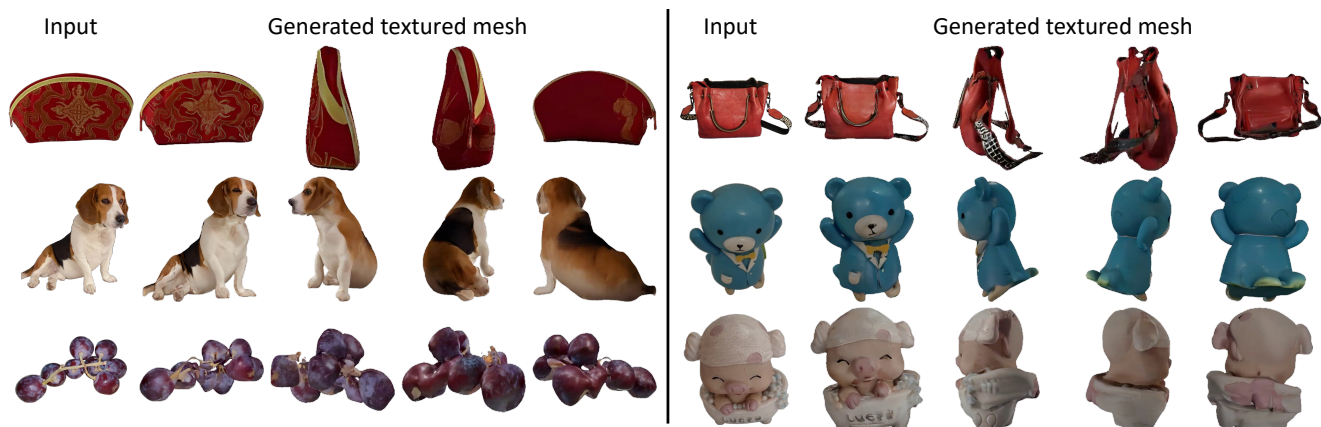


Figure 12. Performance of MeshGen on real-world captures.

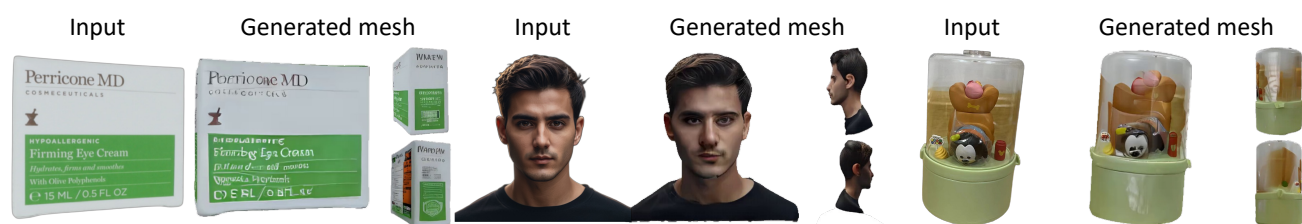


Figure 13. Some typical failure cases of MeshGen.



Figure 14. More comparisons with large reconstruction models, including InstantMesh [83], MeshLRM [78], MeshFormer [39].



Figure 15. More comparisons with 3D native generation models, including, CraftsMan [34], 3DTopia-XL [11], and WaLa [58].

Albedo		Metallic		Roughness	
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					

Albedo		Metallic		Roughness	
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					

Figure 16. Comparison of PBR decomposer and expert branch on PBR component generation.



Figure 17. Intrinsic channels estimated using our multi-view PBR decomposer. The proposed PBR decomposer can handle images with complicated material under different lighting conditions.