GET3D: A Generative Model of High Quality 3D Textured Shapes Learned from Images

Jun Gao[1,2,3] **Tianchang Shen**[1,2,3] **Zian Wang**[1,2,3] **Wenzheng Chen**[1,2,3]
Kangxue Yin[1] **Daiqing Li**[1] **Or Litany**[1] **Zan Gojcic**[1] **Sanja Fidler**[1,2,3]

NVIDIA[1] University of Toronto[2] Vector Institute[3]

{jung, frshen, zianw, wenzchen, kangxuey, daiqingl, olitany, zgojcic, sfidler}@nvidia.com

Abstract

As several industries are moving towards modeling massive 3D virtual worlds, the need for content creation tools that can scale in terms of the quantity, quality, and diversity of 3D content is becoming evident. In our work, we aim to train performant 3D generative models that synthesize textured meshes that can be directly consumed by 3D rendering engines, thus immediately usable in downstream applications. Prior works on 3D generative modeling either lack geometric details, are limited in the mesh topology they can produce, typically do not support textures, or utilize neural renderers in the synthesis process, which makes their use in common 3D software non-trivial. In this work, we introduce GET3D, a **G**enerative model that directly generates **E**xplicit **T**extured **3D** meshes with complex topology, rich geometric details, and high fidelity textures. We bridge recent success in the differentiable surface modeling, differentiable rendering as well as 2D Generative Adversarial Networks to train our model from 2D image collections. GET3D is able to generate high-quality 3D textured meshes, ranging from cars, chairs, animals, motorbikes and human characters to buildings, achieving significant improvements over previous methods. Our project page: https://nv-tlabs.github.io/GET3D

1 Introduction

Diverse, high-quality 3D content is becoming increasingly important for several industries, including gaming, robotics, architecture, and social platforms. However, manual creation of 3D assets is very time-consuming and requires specific technical knowledge as well as artistic modeling skills. One of the main challenges is thus scale – while one can find 3D models on 3D marketplaces such as Turbosquid [4] or Sketchfab [3], creating many 3D models to, say, populate a game or a movie with a crowd of characters that all look different still takes a significant amount of artist time.

To facilitate the content creation process and make it accessible to a variety of (novice) users, generative 3D networks that can produce high-quality and diverse 3D assets have recently become an active area of research [5, 14, 43, 46, 53, 68, 75, 60, 59, 69, 23]. However, to be practically useful for current real-world applications, 3D generative models should ideally fulfill the following requirements: **(a)** They should have the capacity to generate shapes with detailed geometry and arbitrary topology, **(b)** The output should be a textured mesh, which is a primary representation used by standard graphics software packages such as Blender [15] and Maya [1], and **(c)** We should be able to leverage 2D images for supervision, as they are more widely available than explicit 3D shapes.

Prior work on 3D generative modeling has focused on subsets of the above requirements, but no method to date fulfills all of them (Tab. 1). For example, methods that generate 3D point clouds [5,

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Method	Application	Representation	Supervision	Textured mesh	Arbitrary topology
OccNet [43]	3D generation	Implicit	3D	✗	✓
PointFlow [68]	3D generation	Point cloud	3D	✗	✓
Texture3D [53]	3D generation	Mesh	2D	✓	✗
StyleNerf [25]	3D-aware NV	Neural field	2D	✗	✓
EG3D [8]	3D-aware NV	Neural field	2D	✗	✓
PiGAN [7]	3D-aware NV	Neural field	2D	✗	✓
GRAF [57]	3D-aware NV	Neural field	2D	✗	✓
Ours	3D generation	Mesh	2D	✓	✓

Table 1: Comparison with prior works. (NV: Novel view synthesis.)

68, 75] typically do not produce textures and have to be converted to a mesh in post-processing. Methods generating voxels often lack geometric details and do not produce texture [66, 20, 27, 40]. Generative models based on neural fields [43, 14] focus on extracting geometry but disregard texture. Most of these also require explicit 3D supervision. Finally, methods that directly output textured 3D meshes [54, 53] typically require pre-defined shape templates and cannot generate shapes with complex topology and variable genus.

Recently, rapid progress in neural volume rendering [45] and 2D Generative Adversarial Networks (GANs) [34, 35, 33, 29, 52] has led to the rise of 3D-aware image synthesis [7, 57, 8, 49, 51, 25]. However, this line of work aims to synthesize multi-view consistent images using neural rendering in the synthesis process and does not guarantee that meaningful 3D shapes can be generated. While a mesh can potentially be obtained from the underlying neural field representation using the marching cube algorithm [39], extracting the corresponding texture is non-trivial.

In this work, we introduce a novel approach that aims to tackle all the requirements of a practically useful 3D generative model. Specifically, we propose GET3D, a **G**enerative model for 3D shapes that directly outputs **E**xplicit **T**extured **3D** meshes with high geometric and texture detail and arbitrary mesh topology. In the heart of our approach is a generative process that utilizes a differentiable *explicit* surface extraction method [60] and a differentiable rendering technique [47, 37]. The former enables us to directly optimize and output textured 3D meshes with arbitrary topology, while the latter allows us to train our model with 2D images, thus leveraging powerful and mature discriminators developed for 2D image synthesis. Since our model directly generates meshes and uses a highly efficient (differentiable) graphics renderer, we can easily scale up our model to train with image resolution as high as 1024×1024, allowing us to learn high-quality geometric and texture details.

We demonstrate state-of-the-art performance for unconditional 3D shape generation on multiple categories with complex geometry from ShapeNet [9], Turbosquid [4] and Renderpeople [2], such as chairs, motorbikes, cars, human characters, and buildings. With explicit mesh as output representation, GET3D is also very flexible and can easily be adapted to other tasks, including: **(a)** learning to generate decomposed material and view-dependent lighting effects using advanced differentiable rendering [12], without supervision, **(b)** text-guided 3D shape generation using CLIP [56] embedding.

2 Related Work

We review recent advances in 3D generative models for geometry and appearance, as well as 3D-aware generative image synthesis.

3D Generative Models In recent years, 2D generative models have achieved photorealistic quality in high-resolution image synthesis [34, 35, 33, 52, 29, 19, 16]. This progress has also inspired research in 3D content generation. Early approaches aimed to directly extend the 2D CNN generators to 3D voxel grids [66, 20, 27, 40, 62], but the high memory footprint and computational complexity of 3D convolutions hinder the generation process at high resolution. As an alternative, other works have explored point cloud [5, 68, 75, 46], implicit [43, 14], or octree [30] representations. However, these works focus mainly on generating geometry and disregard appearance. Their output representations also need to be post-processed to make them compatible with standard graphics engines.

More similar to our work, Textured3DGAN [54, 53] and DIBR [11] generate textured 3D meshes, but they formulate the generation as a deformation of a template mesh, which prevents them from generating complex topology or shapes with varying genus, which our method can do. PolyGen [48] and SurfGen [41] can produce meshes with arbitrary topology, but do not synthesize textures.

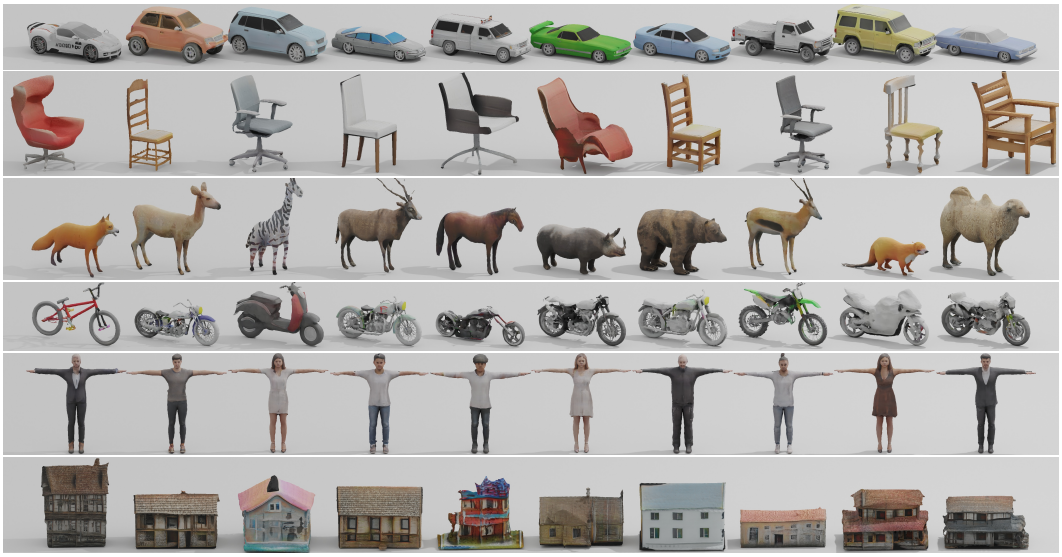


Figure 1: We export our **generated shapes** and visualize them in Blender. GET3D is able to generate diverse shapes with arbitrary topology, high quality geometry, and texture.

3D-Aware Generative Image Synthesis Inspired by the success of neural volume rendering [45] and implicit representations [43, 14], recent work started tackling the problem of 3D-aware image synthesis [7, 57, 49, 26, 25, 76, 8, 51, 58, 67]. However, neural volume rendering networks are typically slow to query, leading to long training times [7, 57], and generate images of limited resolution. GIRAFFE [49] and StyleNerf [25] improve the training and rendering efficiency by performing neural rendering at a lower resolution and then upsampling the results with a 2D CNN. However, the performance gain comes at the cost of a reduced multi-view consistency. By utilizing a dual discriminator, EG3D [8] can partially mitigate this problem. Nevertheless, extracting a textured surface from methods that are based on neural rendering is a non-trivial endeavor. In contrast, GET3D directly outputs textured 3D meshes that can be readily used in standard graphics engines.

3 Method

We now present our GET3D framework for synthesizing textured 3D shapes. Our generation process is split into two parts: a geometry branch, which differentiably outputs a surface mesh of arbitrary topology, and a texture branch that produces a texture field that can be queried at the surface points to produce colors. The latter can be extended to other surface properties such as for example materials (Sec. 4.3.1). During training, an efficient differentiable rasterizer is utilized to render the resulting textured mesh into 2D high-resolution images. The entire process is differentiable, allowing for adversarial training from images (with masks indicating an object of interest) by propagating the gradients from the 2D discriminator to both generator branches. Our model is illustrated in Fig. 2. In the following, we first introduce our 3D generator in Sec 3.1, before proceeding to the differentiable rendering and loss functions in Sec 3.2.

3.1 Generative Model of 3D Textured Meshes

We aim to learn a 3D generator $M, E = G(\mathbf{z})$ to map a sample from a Gaussian distribution $\mathbf{z} \in \mathcal{N}(0, \mathbf{I})$ to a mesh M with texture E.

Since the same geometry can have different textures, and the same texture can be applied to different geometries, we sample two random input vectors $\mathbf{z}_1 \in \mathbb{R}^{512}$ and $\mathbf{z}_2 \in \mathbb{R}^{512}$. Following StyleGAN [34, 35, 33], we then use non-linear mapping networks f_{geo} and f_{tex} to map $\mathbf{z}_1$ and $\mathbf{z}_2$ to intermediate latent vectors $\mathbf{w}_1 = f_{\text{geo}}(\mathbf{z}_1)$ and $\mathbf{w}_2 = f_{\text{tex}}(\mathbf{z}_2)$ which are further used to produce *styles* that control the generation of 3D shapes and texture, respectively. We formally introduce the generator for geometry in Sec. 3.1.1 and the texture generator in Sec. 3.1.2.

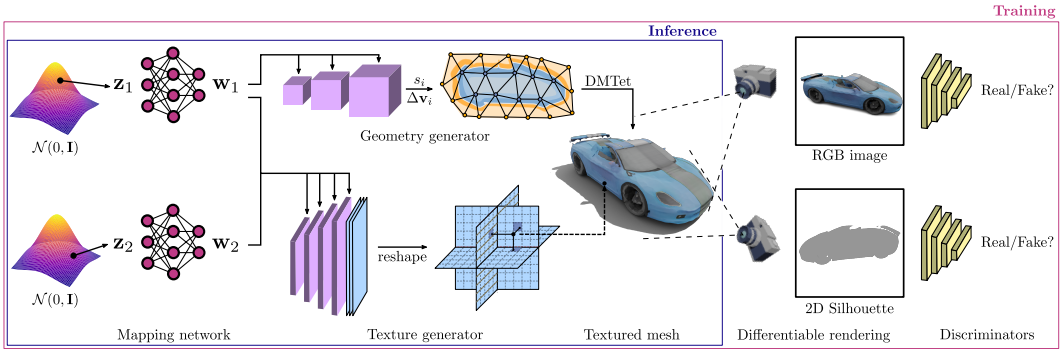


Figure 2: Overview of **GET3D:** We generate a 3D SDF and a texture field via two latent codes. We utilize DMTet [60] to extract a 3D surface mesh from the SDF, and query the texture field at surface points to get colors. We train with adversarial losses defined on 2D images. In particular, we use a rasterization-based differentiable renderer [37] to obtain RGB images and silhouettes. We utilize two 2D discriminators, each on RGB image, and silhouette, respectively, to classify whether the inputs are real or fake. The whole model is end-to-end trainable. Note that we additionally provide an improved version of our Generator in Appendix A.5 and Fig. C.

3.1.1 Geometry Generator

We design our geometry generator to incorporate DMTet [60], a recently proposed differentiable surface representation. DMTet represents geometry as a signed distance field (SDF) defined on a deformable tetrahedral grid [22, 24], from which the surface can be differentiably recovered through marching tetrahedra [17]. Deforming the grid by moving its vertices results in a better utilization of its resolution. By adopting DMTet for surface extraction, we can produce explicit meshes with arbitrary topology and genus. We next provide a brief summary of DMTet and refer the reader to the original paper for further details.

Let (V_T, T) denote the full 3D space that the object lies in, where V_T are the vertices in the tetrahedral grid T. Each tetrahedron $T_k \in T$ is defined using four vertices $\{\mathbf{v}_{a_k}, \mathbf{v}_{b_k}, \mathbf{v}_{c_k}, \mathbf{v}_{d_k}\}$, with $k \in \{1, \ldots, K\}$, where K is the total number of tetrahedra, and $\mathbf{v}_{i_k} \in V_T, \mathbf{v}_{i_k} \in \mathbb{R}^3$. In addition to its 3D coordinates, each vertex $\mathbf{v}_i$ contains the SDF value $s_i \in \mathbb{R}$ and the deformation $\Delta \mathbf{v}_i \in \mathbb{R}^3$ of the vertex from its initial canonical coordinate. This representation allows recovering the explicit mesh through differentiable marching tetrahedra [60], where SDF values in continuous space are computed by a barycentric interpolation of their value s_i on the deformed vertices $\mathbf{v}'_i = \mathbf{v}_i + \Delta \mathbf{v}_i$.

Network Architecture We map $\mathbf{w}_1 \in \mathbb{R}^{512}$ to SDF values and deformations at each vertex $\mathbf{v}_i$ through a series of conditional 3D convolutional and fully connected layers. Specifically, we first use 3D convolutional layers to generate a feature volume conditioned on $\mathbf{w}_1$. We then query the feature at each vertex $\mathbf{v}_i \in V_T$ using trilinear interpolation and feed it into MLPs that outputs the SDF value s_i and the deformation $\Delta \mathbf{v}_i$. In cases where modeling at a high-resolution is required (e.g. motorbike with thin structures in the wheels), we further use volume subdivision following [60].

Differentiable Mesh Extraction After obtaining s_i and $\Delta \mathbf{v}_i$ for all the vertices, we use the differentiable marching tetrahedra algorithm to extract the explicit mesh. Marching tetrahedra determines the surface topology within each tetrahedron based on the signs of s_i. In particular, a mesh face is extracted when $\text{sign}(s_i) \neq \text{sign}(s_j)$, where i, j denotes the indices of vertices in the edge of tetrahedron, and the vertices $\mathbf{m}_{i,j}$ of that face are determined by a linear interpolation as $\mathbf{m}_{i,j} = \frac{\mathbf{v}'_i s_j - \mathbf{v}'_j s_i}{s_j - s_i}$. Note that the above equation is only evaluated when $s_i \neq s_j$, thus it is differentiable, and the gradient from $\mathbf{m}_{i,j}$ can be back-propagated into the SDF values s_i and deformations $\Delta \mathbf{v}_i$. With this representation, the shapes with arbitrary topology can easily be generated by predicting different signs of s_i.

3.1.2 Texture Generator

Directly generating a texture map consistent with the output mesh is not trivial, as the generated shape can have an arbitrary genus and topology. We thus parameterize the texture as a texture field [50].

Specifically, we model the texture field with a function f_t that maps the 3D location of a surface point $\mathbf{p} \in \mathbb{R}^3$, conditioned on the $\mathbf{w}_2$, to the RGB color $\mathbf{c} \in \mathbb{R}^3$ at that location. Since the texture field depends on geometry, we additionally condition this mapping on the geometry latent code $\mathbf{w}_1$, such that $\mathbf{c} = f_t(\mathbf{p}, \mathbf{w}_1 \oplus \mathbf{w}_2)$, where $\oplus$ denotes concatenation.

Network Architecture We represent our texture field using a tri-plane representation, which is efficient and expressive in reconstructing 3D objects [55] and generating 3D-aware images [8] . Specifically, we follow [8, 35] and use a conditional 2D convolutional neural network to map the latent code $\mathbf{w}_1 \oplus \mathbf{w}_2$ to three axis-aligned orthogonal feature planes of size $N \times N \times (C \times 3)$, where $N = 256$ denotes the spatial resolution and $C = 32$ the number of channels.

Given the feature planes, the feature vector $\mathbf{f}^t \in \mathbb{R}^{32}$ of a surface point $\mathbf{p}$ can be recovered as $\mathbf{f}^t = \sum_e \rho(\pi_e(\mathbf{p}))$, where $\pi_e(\mathbf{p})$ is the projection of the point $\mathbf{p}$ to the feature plane e and $\rho(\cdot)$ denotes bilinear interpolation of the features. An additional fully connected layer is then used to map the aggregated feature vector $\mathbf{f}^t$ to the RGB color $\mathbf{c}$. Note that, different from other works on 3D-aware image synthesis [8, 25, 7, 57] that also use a neural field representation, we only need to sample the texture field at the locations of the surface points (as opposed to dense samples along a ray). This greatly reduces the computational complexity for rendering high-resolution images and guarantees to generate multi-view consistent images by construction.

3.2 Differentiable Rendering and Training

In order to supervise our model during training, we draw inspiration from Nvdiffrec [47] that performs multi-view 3D object reconstruction by utilizing a differentiable renderer. Specifically, we render the extracted 3D mesh and the texture field into 2D images using a differentiable renderer [37], and supervise our network with a 2D discriminator, which tries to distinguish the image from a real object or rendered from the generated object.

Differentiable Rendering We assume that the camera distribution $\mathcal{C}$ that was used to acquire the images in the dataset is known. To render the generated shapes, we randomly sample a camera c from $\mathcal{C}$, and utilize a highly-optimized differentiable rasterizer Nvdiffrast [37] to render the 3D mesh into a 2D silhouette as well as an image where each pixel contains the coordinates of the corresponding 3D point on the mesh surface. These coordinates are further used to query the texture field to obtain the RGB values. Since we operate directly on the extracted mesh, we can render high-resolution images with high efficiency, allowing our model to be trained with image resolution as high as 1024×1024.

Discriminator & Objective We train our model using an adversarial objective. We adopt the discriminator architecture from StyleGAN [34], and use the same non-saturating GAN objective with R1 regularization [42]. We empirically find that using two separate discriminators, one for RGB images and another one for silhouettes, yields better results than a single discriminator operating on both. Let D_x denote the discriminator, where x can either be an RGB image or a silhouette. The adversarial objective is then be defined as follows:

$$L(D_x, G) = \mathbb{E}_{\mathbf{z}\in\mathcal{N},c\in\mathcal{C}}[g(D_x(R(G(\mathbf{z}), c)))] + \mathbb{E}_{I_x\in p_x}[g(-D_x(I_x)) + \lambda||\nabla D_x(I_x)||_2^2], \quad (1)$$

where $g(u)$ is defined as $g(u) = -\log(1+\exp(-u))$, p_x is the distribution of real images, R denotes rendering, and λ is a hyperparameter. Since R is differentiable, the gradients can be backpropagated from 2D images to our 3D generators.

Regularization To remove internal floating faces that are not visible in any of the views, we further regularize the geometry generator with a cross-entropy loss defined between the SDF values of the neighboring vertices [47]:

$$L_{\text{reg}} = \sum_{i,j\in\mathbb{S}_e} H\left(\sigma(s_i), \text{sign}\left(s_j\right)\right) + H\left(\sigma(s_j), \text{sign}\left(s_i\right)\right), \quad (2)$$

where H denotes binary cross-entropy loss and σ denotes the sigmoid function. The sum in Eq. 2 is defined over the set of unique edges $\mathbb{S}_e$ in the tetrahedral grid, for which $\text{sign}(s_i) \neq \text{sign}(s_j)$.

The overall loss function is then defined as:

$$L = L(D_{\text{rgb}}, G) + L(D_{\text{mask}}, G) + \mu L_{\text{reg}}, \quad (3)$$

where μ is a hyperparameter that controls the level of regularization.

Category	Method	COV (%, ↑)		MMD (↓)		FID (↓)	
		LFD	CD	LFD	CD	Ori	3D
Car	PointFlow [68]	51.91	57.16	1971	0.82	-	-
	OccNet [43]	27.29	42.63	1717	**0.61**	-	-
	Pi-GAN [7]	0.82	0.55	6626	25.54	52.82	104.29
	GRAF [57]	1.57	1.57	6012	10.63	49.95	52.85
	EG3D [8]	60.16	49.52	1527	0.72	15.52	21.89
	Ours	**66.78**	**58.39**	**1491**	0.71	**10.25**	**10.25**
	Ours+Subdiv.	62.48	55.93	1553	0.72	12.14	12.14
	Ours (improved G)	59.00	47.95	1473	0.81	10.60	10.60
Chair	PointFlow [68]	49.58	**71.87**	3755	**3.03**	-	-
	OccNet [43]	61.10	67.13	3494	3.98	-	-
	Pi-GAN [7]	53.76	39.65	4092	6.65	65.70	120.53
	GRAF [57]	50.23	39.28	4055	6.80	43.82	61.63
	EG3D [8]	58.31	50.14	3444	4.72	38.87	46.06
	Ours	69.08	69.91	3167	3.72	23.28	23.28
	Ours+Subdiv.	**71.59**	70.84	**3163**	3.95	**23.17**	**23.17**
	Ours (improved G)	71.96	71.96	3125	3.96	22.41	22.41

Category	Method	COV (%, ↑)		MMD (↓)		FID (↓)	
		LFD	CD	LFD	CD	Ori	3D
Mbike	PointFlow [68]	50.68	63.01	4023	**1.38**	-	-
	OccNet [43]	30.14	47.95	4551	2.04	-	-
	Pi-GAN [7]	2.74	6.85	8864	21.08	72.67	131.38
	GRAF [57]	43.84	50.68	4528	2.40	83.20	113.39
	EG3D [8]	38.36	34.25	4199	2.21	66.38	89.97
	Ours	**67.12**	**67.12**	3631	1.72	65.60	65.60
	Ours+Subdiv.	63.01	61.64	**3440**	1.79	**54.12**	**54.12**
	Ours (improved G)	69.86	65.75	3393	1.79	48.90	48.90
Animal	PointFlow [68]	42.70	74.16	4885	**1.68**	-	-
	OccNet [43]	56.18	75.28	4418	2.39	-	-
	Pi-GAN [7]	31.46	30.34	6084	8.37	36.26	150.86
	GRAF [57]	60.67	61.80	5083	4.81	42.07	52.48
	EG3D [8]	74.16	58.43	4889	3.42	40.03	83.47
	Ours	**79.77**	**78.65**	**3798**	2.02	**28.33**	**28.33**
	Ours+Subdiv.	66.29	74.16	3864	2.03	28.49	28.49
	Ours (improved G)	74.16	82.02	3767	1.97	27.18	27.18

Table 2: **Quantitative evaluation of generation results**: ↑: the higher the better, ↓: the lower the better. The best scores are highlighted in bold. MMD-CD scores are multiplied by 10^3. The results of *Ours (improved G)* were obtained after the review process by improving the design of the generator network architecture G (see Appendix A.5 for more details).

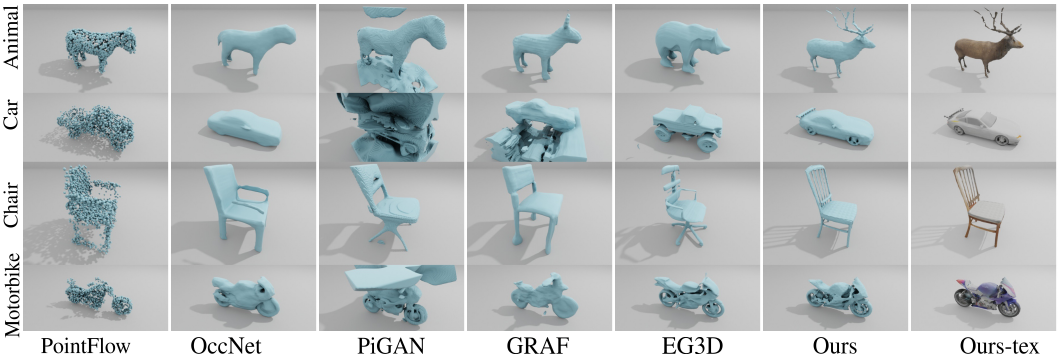


Figure 3: Qualitative comparison of GET3D to the baseline methods in terms of extracted 3D geometry. GET3D is able to generate shapes with much higher geometric detail across all categories.

4 Experiments

We conduct extensive experiments to evaluate our model. We first compare the quality of the 3D textured meshes generated by GET3D to the existing methods using the ShapeNet [9] and Turbosquid [4] datasets. Next, we ablate our design choices in Sec. 4.2. Finally, we demonstrate the flexibility of GET3D by adapting it to downstream applications in Sec. 4.3. Additional experimental results and implementation details are provided in Appendix.

4.1 Experiments on Synthetic Datasets

Datasets For evaluation on ShapeNet [9], we use three categories with complex geometry – *Car*, *Chair*, and *Motorbike*, which contain 7497, 6778, and 337 shapes, respectively. We randomly split each category into training (70%), validation (10 %), and test (20 %), and further remove from the test set shapes that have duplicates in the training set. To render the training data, we randomly sample camera poses from the upper hemisphere of each shape. For the *Car* and *Chair* categories, we use 24 random views, while for *Motorbike* we use 100 views due to less number of shapes. As models in ShapeNet only have simple textures, we also evaluate GET3D on an *Animal* dataset (442 shapes) collected from TurboSquid [4], where textures are more detailed and we split it into training, validation and test as defined above. Finally, to demonstrate the versatility of GET3D, we also provide qualitative results on the *House* dataset collected from Turbosquid (563 shapes), and *Human Body* dataset from Renderpeople [2] (500 shapes). We train a separate model on each category.

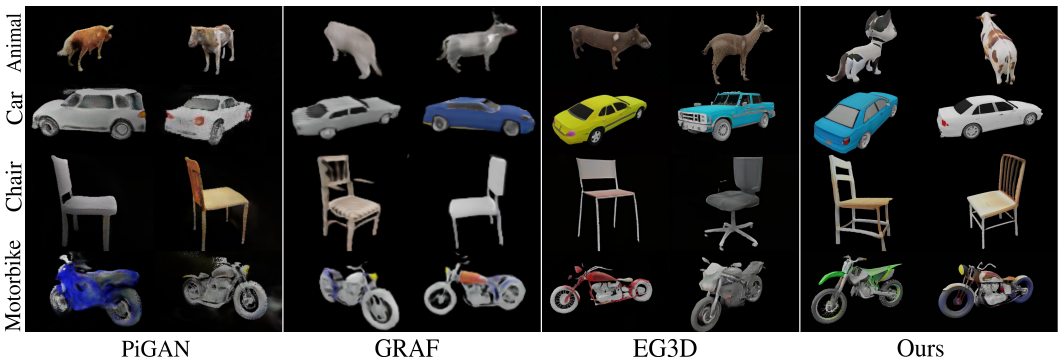


Figure 4: Qualitative comparison of GET3D to the baseline methods in terms of generated 2D images. GET3D generates sharp textures with high level of detail.

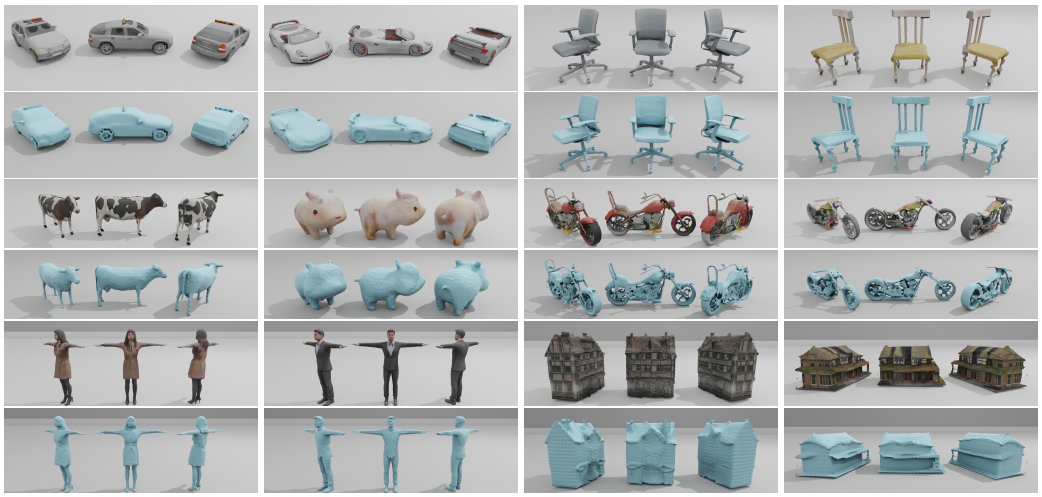


Figure 5: **Shapes generated by GET3D rendered in Blender.** GET3D generates high-quality shapes with diverse texture, high-quality geometry, and complex topology. Zoom-in for details.

Baselines We compare GET3D to two groups of works: **1)** 3D generative models that rely on 3D supervision: PointFlow [68] and OccNet [43]. Note that these methods only generate geometry without texture. **2)** 3D-aware image generation methods: GRAF [57], PiGAN [7], and EG3D [8].

Metrics To evaluate the quality of our synthesis, we consider both the geometry and texture of the generated shapes. For geometry, we adopt metrics from [5] and use both Chamfer Distance (CD) and Light Field Distance [10] (LFD) to compute the Coverage score and Minimum Matching Distance. For OccNet [43], GRAF [57], PiGAN [7] and EG3D [8], we use marching cubes to extract the underlying geometry. For PointFlow [68], we use Poisson surface reconstruction to convert a point cloud into a mesh when evaluating LFD. To evaluate texture quality, we adopt the FID [28] metric commonly used to evaluate image synthesis. In particular, for each category, we render the test shapes into 2D images, and also render the generated 3D shapes from each model into 50k images using the same camera distribution. We then compute FID on the two image sets. As the baselines from 3D-aware image synthesis [57, 7, 8] do not directly output textured meshes, we compute FID score in two ways: (**i**) we use their neural volume rendering to obtain 2D images, which we refer to as FID-Ori, and (**ii**) we extract the mesh from their neural field representation using marching cubes, render it, and then use the 3D location of each pixel to query the network to obtain the RGB values. We refer to this score, that is more aware of the actual 3D shape, as FID-3D. Further details on the evaluation metrics are available in the Appendix B.3.

Experimental Results We provide quantitative results in Table. 2 and qualitative examples in Fig. 3 and Fig. 4. Additional results are available in the supplementary video. Compared to OccNet [43] that

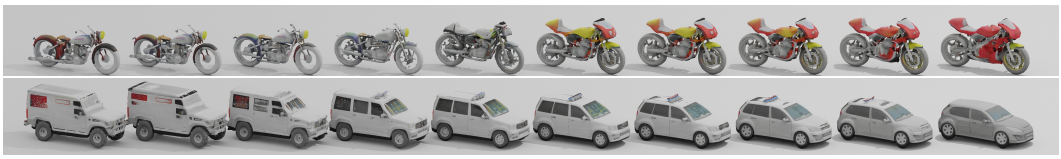


Figure 6: **Shape interpolation.** We interpolate both geometry and texture latent codes from left to right.

Figure 7: **Shape variation.** We locally perturb each latent code to generate different shapes.

uses 3D supervision during training, GET3D achieves better performance in terms of both diversity (COV) and quality (MMD), and our generated shapes have more geometric details. PointFlow [68] outperforms GET3D in terms of MMD on CD, while GET3D is better in MMD on LFG. We hypothesize that this is because PointFlow directly optimizes on point locations, which favours CD. GET3D also performs favourably when compared to 3D-aware image synthesis methods, we achieve significant improvements over PiGAN [7] and GRAF [57] in terms of all metrics on all datasets. Our generated shapes also contain more detailed geometry and texture. Compared with recent work EG3D [8]. We achieve comparable performance on generating 2D images (FID-ori), while we significantly improve on 3D shape synthesis in terms of FID-3D, which demonstrates the effectiveness of our model on learning actual 3D geometry and texture.

Since we synthesize textured meshes, we can export our shapes into Blender[1]. We show rendering results in Fig. 1 and 5. GET3D is able to generate shapes with diverse and high quality geometry and topology, very thin structures (motorbikes), as well as complex textures on cars, animals, and houses.

Shape Interpolation GET3D also enables shape interpolation, which can be useful for editing purposes. We explore the latent space of GET3D in Fig. 6, where we interpolate the latent codes to generate each shape from left to right. GET3D is able to faithfully generate a smooth and meaningful transition from one shape to another. We further explore the local latent space by slightly perturbing the latent codes to a random direction. GET3D produces novel and diverse shapes when applying local editing in the latent space (Fig. 7).

4.2 Ablations

We ablate our model in two ways: **1)** w/ and w/o volume subdivision, **2)** training using different image resolutions. Further ablations are provided in the Appendix C.3.

Ablation of Volume Subdivision As shown in Tbl. 2, volume subdivision significantly improves the performance on classes with thin structures (e.g., motorbikes), while not getting gains on other classes. We hypothesize that the initial tetrahedral resolution is already sufficient to capture the detailed geometry on Chairs and Cars, and hence the subdivision cannot provide further improvements.

Ablating Different Image Resolutions We ablate the effect of the training image resolution in Tbl. 3. As expected, increased image resolution

Class	Img Res	COV (%, ↑)		MMD (↓)		FID (↓)
		LFD	CD	LFD	CD	
Car	128^2	9.28	8.25	2224	1.30	39.21
	512^2	52.32	44.13	1593	0.80	13.19
	1024^2	**66.78**	**58.39**	**1491**	**0.71**	**10.25**
Chair	128^2	38.25	33.98	3886	5.90	43.04
	512^2	68.80	**69.92**	**3149**	3.90	30.16
	1024^2	**69.08**	67.87	3167	**3.74**	**23.28**
Mbike	512^2	**68.49**	**65.75**	**3421**	1.74	74.04
	1024^2	67.12	64.38	3631	**1.73**	**65.60**
Animal	512^2	77.53	**78.65**	3828	**2.01**	29.75
	1024^2	**79.78**	**78.65**	**3798**	2.03	**28.33**

Table 3: **Ablating the image resolution.** ↑: higher is better, ↓: lower is better.

[1]We use xatlas [71] to get texture coordinates for the extracted mesh, from where we can warp our 3D mesh into a 2D plane and obtain the corresponding 3D location on the mesh surface for any position on the 2D plane. We then discretize the 2D plane into an image, and for each pixel, we query the texture field using corresponding 3D location to obtain the RGB color to get the texture map.

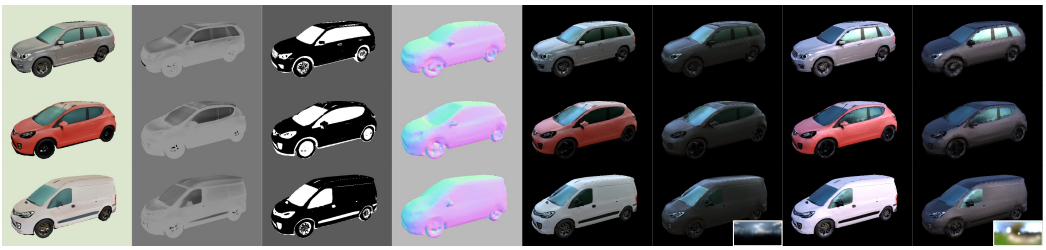


Figure 8: **Material generation and relighting.** Despite being unsupervised, our model generates reasonable material properties, and can be realistically rendered with real-world HDR panoramas (bottom right). Normals are computed from the generated mesh. Note how specular effects change under two different lighting conditions.

improves the performance in terms of FID and shape quality, as the network can see more details, which are often not available in the low-resolution images. This corroborates the importance of training with higher image resolution, which are often hard to make use of for implicit-based methods.

4.3 Applications

4.3.1 Material Generation for View-dependent Lighting Effects

GET3D can easily be extended to also generate surface materials that are directly usable in modern graphics engines. In particular, we follow the widely used Disney BRDF [6, 32] and describe the materials in terms of the base color ($\mathbb{R}^3$), metallic ($\mathbb{R}$), and roughness ($\mathbb{R}$) properties. As a result, we repurpouse our texture generator to now output a 5-channel reflectance field (instead of only RGB). To accommodate differentiable rendering of materials, we adopt an efficient spherical Gaussian (SG) based deferred rendering pipeline [12]. Specifically, we rasterize the reflectance field into a G-buffer, and randomly sample an HDR image from a set of real-world outdoor HDR panoramas $\mathcal{S}_{\text{light}} = \{L_{SG}\}_K$, where $L_{SG} \in \mathbb{R}^{32\times 7}$ is obtained by fitting 32 SG lobes to each panorama. The SG renderer [12] then uses the camera c to render an RGB image with view-dependent lighting effects, which we feed into the discriminator during training. Note that GET3D does not require material supervision during training and learns to generate decomposed materials in an unsupervised manner.

We provide qualitative results of generated surface materials in Fig. 8. Despite unsupervised, GET3D discovers interesting material decomposition, e.g., the windows are correctly predicted with a smaller roughness value to be more glossy than the car's body, and the car's body is discovered as more dielectric while the window is more metallic. Generated materials enable us to produce realistic relighting results, which can account for complex specular effects under different lighting conditions.

4.3.2 Text-Guided 3D Synthesis

Similar to image GANs, GET3D also supports text-guided 3D content synthesis by fine-tuning a pre-trained model under the guidance of CLIP [56]. Note that our final synthesis result is a textured 3D mesh. To this end, we follow the dual-generator design from styleGAN-NADA [21], where a trainable copy G_t and a frozen copy G_f of the pre-trained generator are adopted. During optimization G_t and G_f both render images from 16 random camera views. Given a text query, we sample 500 pairs of noise vectors $\mathbf{z}_1$ and $\mathbf{z}_2$. For each sample, we optimize the parameters of G_t to minimize the directional CLIP loss [21] (the source text labels are "car", "animal" and "house" for the corresponding categories), and select the samples with minimal loss. To accelerate this process, we first run a small number of optimization steps for the 500 samples, then choose the top 50 samples with the lowest losses, and run the optimization for 300 steps. The results and comparison against a SOTA text-driven mesh stylization method, Text2Mesh [44], are provided in Fig. 9. Note that, [44] requires a mesh of the shape as an input to the method. We provide our generated meshes from the frozen generator as input meshes to it. Since it needs mesh vertices to be dense to synthesize surface details with vertex displacements, we further subdivide the input meshes with mid-point subdivision to make sure each mesh has 50k-150k vertices on average.

Figure 9: **Text-guided 3D synthesis.** Note that Text2Mesh [44] requires 3D mesh geometry as input. To fulfil the requirement, we provide our generated geometry as its input mesh.

5 Conclusion

We introduced GET3D, a novel 3D generative model that is able to synthesize high-quality 3D textured meshes with arbitrary topology. GET3D is trained using only 2D images as supervision. We experimentally demonstrated significant improvements on generating 3D shapes over previous state-of-the-art methods on multiple categories. We hope that this work brings us one step closer to democratizing 3D content creation using A.I..

Limitations While GET3D makes a significant step towards a practically useful 3D generative model of 3D textured shapes, it still has some limitations. In particular, we still rely on 2D silhouettes as well as the knowledge of camera distribution during training. As a consequence, GET3D was currently only evaluated on synthetic data. A promising extension could use the advances in instance segmentation and camera pose estimation to mitigate this issue and extend GET3D to real-world data. GET3D is also trained per-category; extending it to multiple categories in the future, could help us better represent the inter-category diversity.

Broader Impact We proposed a novel 3D generative model that generates 3D textured meshes, which can be readily imported into current graphics engines. Our model is able to generate shapes with arbitrary topology, high quality textures and rich geometric details, paving the path for democratizing A.I. tool for 3D content creation. As all machine learning models, GET3D is also prone to biases introduced in the training data. Therefore, an abundance of caution should be applied when dealing with sensitive applications, such as generating 3D human bodies, as GET3D is not tailored for these applications. We do not recommend using GET3D if privacy or erroneous recognition could lead to potential misuse or any other harmful applications. Instead, we do encourage practitioners to carefully inspect and de-bias the datasets before training our model to depict a fair and wide distribution of possible skin tones, races or gender identities.

6 Disclosure of Funding

This work was funded by NVIDIA. Jun Gao, Tianchang Shen, Zian Wang and Wenzheng Chen acknowledge additional revenue in the form of student scholarships from University of Toronto and the Vector Institute, which are not in direct support of this work.

References

[1] Autodesk Maya, https://www.autodesk.com/products/maya/overview. Accessed: 2022-05-19.

[2] Renderpeople, http://https://renderpeople.com/. Accessed: 2022-05-19.

[3] Sketchfab, https://sketchfab.com/. Accessed: 2022-05-19.

[4] Turbosquid by Shutterstock, https://www.turbosquid.com/. Accessed: 2022-05-19.

[5] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018.

[6] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, volume 2012, pages 1–7. vol. 2012, 2012.

[7] Eric Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proc. CVPR*, 2021.

[8] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16123–16133, 2022.

[9] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[10] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003.

[11] Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances In Neural Information Processing Systems*, 2019.

[12] Wenzheng Chen, Joey Litalien, Jun Gao, Zian Wang, Clement Fuji Tsang, Sameh Khalis, Or Litany, and Sanja Fidler. DIB-R++: Learning to predict lighting and material with a hybrid differentiable renderer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[13] Yanqin Chen, Xin Jin, and Qionghai Dai. Distance measurement based on light field geometry and ray tracing. *Optics Express*, 25(1):59–76, 2017.

[14] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[15] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[16] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34, 2021.

[17] Akio Doi and Akio Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems*, 74(1):214–224, 1991.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[19] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12873–12883, 2021.

[20] Matheus Gadelha, Subhransu Maji, and Rui Wang. 3d shape induction from 2d views of multiple objects. In *2017 International Conference on 3D Vision (3DV)*, pages 402–411. IEEE, 2017.

[21] Rinon Gal, Or Patashnik, Haggai Maron, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. Stylegan-nada: Clip-guided domain adaptation of image generators. *ACM Transactions on Graphics (TOG)*, 41(4):1–13, 2022.

[22] Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. In *Advances In Neural Information Processing Systems*, 2020.

[23] Jun Gao, Chengcheng Tang, Vignesh Ganapathi-Subramanian, Jiahui Huang, Hao Su, and Leonidas J Guibas. Deepspline: Data-driven reconstruction of parametric curves and surfaces. *arXiv preprint arXiv:1901.03781*, 2019.

[24] Jun Gao, Zian Wang, Jinchen Xuan, and Sanja Fidler. Beyond fixed grid: Learning geometric image representation with a deformable grid. In *European Conference on Computer Vision*, pages 108–125. Springer, 2020.

[25] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d aware generator for high-resolution image synthesis. In *International Conference on Learning Representations*, 2022.

[26] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds. In *ICCV*, 2021.

[27] Philipp Henzler, Niloy J. Mitra, and Tobias Ritschel. Escaping plato's cave: 3d shape from adversarial rendering. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[28] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[29] Xun Huang, Arun Mallya, Ting-Chun Wang, and Ming-Yu Liu. Multimodal conditional image synthesis with product-of-experts GANs. In *ECCV*, 2022.

[30] Moritz Ibing, Gregor Kobsik, and Leif Kobbelt. Octree transformer: Autoregressive 3d shape generation on hierarchically structured sequences. *arXiv preprint arXiv:2111.12480*, 2021.

[31] James T. Kajiya. The rendering equation. SIGGRAPH '86, page 143–150, 1986.

[32] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4(3), 2013.

[33] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. In *Proc. NeurIPS*, 2021.

[34] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[35] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proc. CVPR*, 2020.

[36] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[37] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020.

[38] Daiqing Li, Junlin Yang, Karsten Kreis, Antonio Torralba, and Sanja Fidler. Semantic segmentation with generative models: Semi-supervised learning and strong out-of-domain generalization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[39] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[40] Sebastian Lunz, Yingzhen Li, Andrew Fitzgibbon, and Nate Kushman. Inverse graphics gan: Learning to generate 3d shapes from unstructured 2d data. *arXiv preprint arXiv:2002.12674*, 2020.

[41] Andrew Luo, Tianqin Li, Wen-Hao Zhang, and Tai Sing Lee. Surfgen: Adversarial 3d shape synthesis with explicit surface discriminators. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16238–16248, 2021.

[42] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Which training methods for gans do actually converge? In *International Conference on Machine Learning (ICML)*, 2018.

[43] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[44] Oscar Michel, Roi Bar-On, Richard Liu, Sagie Benaim, and Rana Hanocka. Text2mesh: Text-driven neural stylization for meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13492–13502, 2022.

[45] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[46] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *ACM Transactions on Graphics (TOG), Siggraph Asia 2019*, 38(6):Article 242, 2019.

[47] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting triangular 3d models, materials, and lighting from images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8280–8290, 2022.

[48] Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. Polygen: An autoregressive generative model of 3d meshes. *ICML*, 2020.

[49] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[50] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4531–4540, 2019.

[51] Roy Or-El, Xuan Luo, Mengyi Shan, Eli Shechtman, Jeong Joon Park, and Ira Kemelmacher-Shlizerman. Stylesdf: High-resolution 3d-consistent image and geometry generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13503–13513, 2022.

[52] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[53] Dario Pavllo, Jonas Kohler, Thomas Hofmann, and Aurelien Lucchi. Learning generative models of textured 3d meshes from real-world images. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[54] Dario Pavllo, Graham Spinks, Thomas Hofmann, Marie-Francine Moens, and Aurelien Lucchi. Convolutional generation of textured 3d meshes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[55] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, 2020.

[56] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[57] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[58] Katja Schwarz, Axel Sauer, Michael Niemeyer, Yiyi Liao, and Andreas Geiger. Voxgraf: Fast 3d-aware image synthesis with sparse voxel grids. *ARXIV*, 2022.

[59] Tianchang Shen, Jun Gao, Amlan Kar, and Sanja Fidler. Interactive annotation of 3d object geometry using 2d scribbles. In *European Conference on Computer Vision*, pages 751–767. Springer, 2020.

[60] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[61] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[62] Edward J Smith and David Meger. Improved adversarial systems for 3d object generation and reconstruction. In *Conference on Robot Learning*, pages 87–96. PMLR, 2017.

[63] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[64] Jiaping Wang, Peiran Ren, Minmin Gong, John Snyder, and Baining Guo. All-frequency rendering of dynamic, spatially-varying reflectance. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–10. 2009.

[65] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010.

[66] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29, 2016.

[67] Yinghao Xu, Sida Peng, Ceyuan Yang, Yujun Shen, and Bolei Zhou. 3d-aware image synthesis via learning structural and textural representations. In *CVPR*, 2022.

[68] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4541–4550, 2019.

[69] Kangxue Yin, Jun Gao, Maria Shugrina, Sameh Khamis, and Sanja Fidler. 3dstylenet: Creating 3d shapes with geometric and texture style variations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12456–12465, 2021.

[70] Kangxue Yin, Jun Gao, Maria Shugrina, Sameh Khamis, and Sanja Fidler. 3dstylenet: Creating 3d shapes with geometric and texture style variations. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2021.

[71] Jonathan Young. xatlas, 2021. https://github.com/jpcy/xatlas.

[72] Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. Physg: Inverse rendering with spherical gaussians for physics-based material editing and relighting. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[73] Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. Image gans meet differentiable rendering for inverse graphics and interpretable 3d neural rendering. In *International Conference on Learning Representations*, 2021.

[74] Yuxuan Zhang, Huan Ling, Jun Gao, Kangxue Yin, Jean-Francois Lafleche, Adela Barriuso, Antonio Torralba, and Sanja Fidler. Datasetgan: Efficient labeled data factory with minimal human effort. In *CVPR*, 2021.

[75] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5826–5835, 2021.

[76] Peng Zhou, Lingxi Xie, Bingbing Ni, and Qi Tian. Cips-3d: A 3d-aware generator of gans based on conditionally-independent pixel synthesis. *arXiv preprint arXiv:2110.09788*, 2021.

[77] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We provide extensive experiments in Sec. 4.
   (b) Did you describe the limitations of your work? [Yes] We provide the discussion on limitations in Sec. 5, and failure cases in Appendix.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We provide the discussion in the Broader Impact paragraph with further discussions in the main paper.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We have released the code here `https://github.com/nv-tlabs/GET3D`
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide training details in both Sec. 4 in the main paper and Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Training existing 3D models, including ours, on large-scale 3D datasets is too computation costly to repeat multiple times.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We provide in the Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We used ShapeNet [9] core dataset, Turbosquid [4], Renderpeople [2] in Sec. 4.1. We also used official code to reproduce baselines with citations. In particular, OccNet [43], PiGAN [7], Pointflow [68], Eg3D [8].
   (b) Did you mention the license of the assets? [Yes] We provided the license of ShapeNet, Turbosquid and RenderPeople in the Appendix.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] We did not include new assets in Supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] We provide discussion on this in the Appendix.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] We provide discussion on this in Appendix.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

In this Appendix, we first provide detailed description of the GET3D network architecture (Sec. A.1-A.4) along with the training procedure and hyperparameters (Sec. A.6). We then describe the datasets (Sec. B.1), baselines (Sec. B.2), and evaluation metrics (Sec. B.3). Additional qualitative results, ablation studies, robustness analysis, and results on the real dataset are available in Sec. C. Details and additional results of the material generation for view-dependent lighting effects are provided in Sec. D. Sec E contains more information about the text-guided shape generation experiments as well as more additional qualitative results. The readers are also kindly referred to the accompanying video (*demo.mp4*) that includes 360-degree renderings of our results (more than 400 generated shapes for each category), detailed zoom-ins, interpolations, material generation, and shapes generated with text-guidance.

A Details of Our Model

In Sec. 3 we have provided a high level description of GET3D. Here, we provide the implementation details that were omitted due to the lack of space. Please consult the Figure B and Figure 2 in the main paper for more context. Source code is available at our project webpage

A.1 Mapping Network

Following StyleGAN [34, 35], our mapping networks f_{geo} and f_{tex} are 8-layer MLPs in which each fully-connected layer has 512 hidden dimensions and a leaky-ReLU activation (Figure B). The mapping networks are used to map the randomly sampled noise vectors $\mathbf{z}_1 \in \mathbb{R}^{512}$ and $\mathbf{z}_2 \in \mathbb{R}^{512}$ to the latent vectors $\mathbf{w}_1 \in \mathbb{R}^{512}$ and $\mathbf{w}_2 \in \mathbb{R}^{512}$ as $\mathbf{w}_1 = f_{\text{geo}}(\mathbf{z}_1)$ and $\mathbf{w}_2 = f_{\text{tex}}(\mathbf{z}_2)$.

A.2 Geometry Generator

The geometry generator of GET3D starts from a randomly initialized feature volume $\mathbf{F}_{\text{geo}} \in \mathbb{R}^{4\times4\times4\times256}$ that is shared across the generated shapes, and is learned during training. Through a series of four modulated 3D convolution blocks (*ModBlock3D* in Figure B), the initial volume is up-sampled to a feature volume $\mathbf{F}'_{\text{geo}} \in \mathbb{R}^{32\times32\times32\times64}$ that is conditioned on $\mathbf{w}_1$. Specifically, in each *ModBlock3D*, the input feature volume is first upsampled by a factor of two using trilinear interpolation. It is then passed through a small 3D ResNet, where the residual path uses a 3D convolutional layer with kernel size 1x1x1, and the main path applies two *conditional* 3D convolutional layers with kernel size 3x3x3. To perform the conditioning, we follow StyleGAN2 [35] and first map the latent vector $\mathbf{w}_1$ to *style* $\mathbf{h}$ through a learned affine transformation ($\mathbf{A}$ in Figure B). The *style* $\mathbf{h}$ is then used to modulate ($\mathbf{M}$) and demodulate ($\mathbf{D}$) the weights of the convolutional layers as:

$$\mathbf{M} : \theta'_{i,j,k,l,m} = h_i \cdot \omega_{i,j,k,l,m}, \tag{4}$$

$$\mathbf{D} : \theta''_{i,j,k,l,m} = \theta'_{i,j,k,l,m} / \sqrt{\sum_{i,k,l,m} {\theta'_{i,j,k,l,m}}^2}, \tag{5}$$

where θ and θ'' are the original and modulated weight, respectively. h_i is the *style* corresponding to the ith input channel, j is the output channel dimension, and k, l, m denote the spatial dimension of the 3D convolutional filter.

Once we obtain the final feature volume $\mathbf{F}'_{\text{geo}}$, the feature vector $\mathbf{f}'_{\text{geo}} \in \mathbb{R}^{64}$ of each vertex $\mathbf{v}$ in the tetrahedral grid can be obtained through trilinear interpolation. We additionally feed the coordinates of the point $\mathbf{p}$ to a $[\sin(\mathbf{p}), \cos(\mathbf{p})]$ positional encoding (PE) and concatenate the output with the feature vector $\mathbf{f}'_{\text{geo}}$. To decode the concatenated feature vector into the vertex offset $\Delta\mathbf{v} \in \mathbb{R}^3$ or the SDF value $s \in \mathbb{R}$, we pass it through three conditional FC layers (*ModFC* in Figure B). The modulation and demodulation in these layers is done analogously to Eq. 5. All the layers, except for the last, are followed by the leaky-ReLU activation function. In the last layer, we apply `tanh` to either normalize the SDF prediction s to be within [-1, 1], or normalize the $\Delta\mathbf{v}$ to be within [-$\frac{1}{\text{tet-res}}$, $\frac{1}{\text{tet-res}}$], where tet-res denotes the resolution of our tetrahedral grid, which we set to 90 in all the experiments.

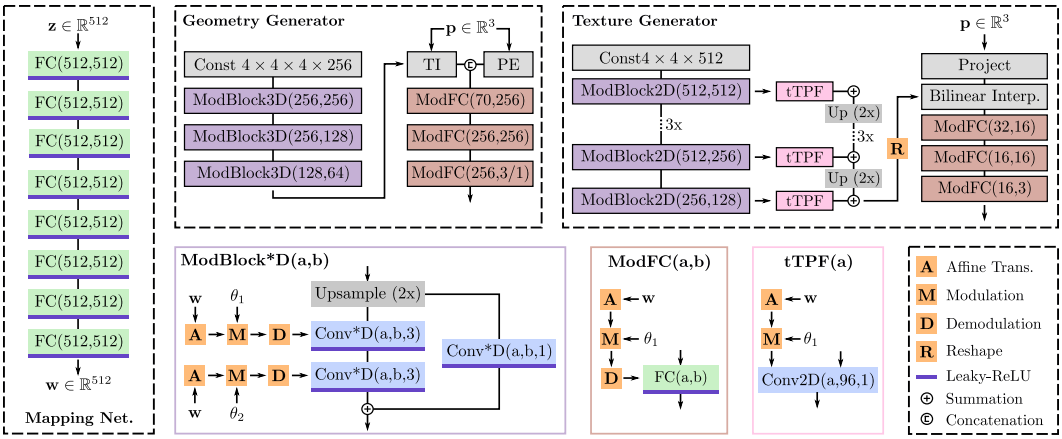


Figure B: **Network architecture of GET3D**. TI and PE denote trilinear interpolation and positional encoding, respectively. FC(a, b) represents a fully connected layer with a and b denoting the input and output dimension, respectively. Similarly, Conv3D(a, b, c) denotes a 3D convolutional layer with a input channels, b output channels, and kernel dimension $c \times c \times c$. In the Texture Generator, the block ModBlock2D(512,512) is repeated four times. All convolutional layers have stride 1.

Note that for simplicity, we remove all the noise vector from StyleGAN [34, 35] and only have stochasticity in the input $\mathbf{z}$. Furthermore, following practices from DEFTET [22] and DMTET [60], we us two copies of the geometry generator. One generates the vertex offsets $\Delta\mathbf{v}$, while the other outputs the SDF values s. The architecture of both is the same, except for the output dimension and activation function of the last layer.

Volume Subdivision: In cases where modeling at a high-resolution is required (e.g. motorbike with thin structures in the wheels), we further use volume subdivision following DMTET [60]. As illustrated in Fig. A, we first subdivide the tetrahedral grid and compute SDF values of the new vertices (midpoints) by averaging the SDF values on the edge. Then we identify tetrahedra that have vertices with different SDF signs. These are the tetrahedra that intersect with the underlying surface encode by SDF. To refine the surface at increased grid resolution after subdivision, we further predict the residual on SDF values and deformations to update s and $\Delta\mathbf{v}$ of the vertices in **identified** tetrahedra. Specifically, we use an additional 3D convolutional layer to upsample feature volume $\mathbf{F}'_{\text{geo}}$ to $\mathbf{F}''_{\text{geo}}$ of shape $64 \times 64 \times 64 \times 8$ conditioned on w_1. Then, following the steps described above, we use trilinear interpolation to obtain per-vertex feature, concatenate it with PE and decode the residuals δs and $\delta\mathbf{v}$ using conditional FC layers. The final SDF and vertex offset are computed as:

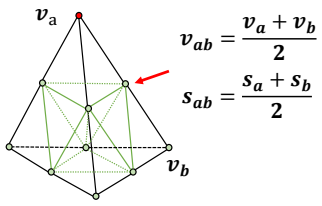


Figure A: With volume subdivision, each tetrahedron is divided into 8 smaller tetrahedra by connecting midpoints.

$$s' = s + \delta s, \ \Delta\mathbf{v}' = \Delta\mathbf{v} + \delta\mathbf{v}. \tag{6}$$

A.3 Texture Generator

We adapt the generator architecture from StyleGAN2 [35] to generate a tri-plane representation of the texture field. Similar as in the geometry generator, we start from a randomly initialized feature grid $\mathbf{F}_{\text{tex}} \in \mathbb{R}^{4\times4\times512}$ that is shared across the shapes, and is learned during training. This initial feature grid is up-sampled to a feature grid $\mathbf{F}'_{\text{tex}} \in \mathbb{R}^{256\times256\times96}$ that is conditioned on $\mathbf{w}_1$ and $\mathbf{w}_2$. Specifically, we use a series of six modulated 2D convolution blocks (*ModBlock2D* in Figure B). The *ModBlock2D* blocks are the same as the *ModBlock3D* blocks, except that the convolution is 2D and that the conditioning is on $\mathbf{w}_1 \oplus \mathbf{w}_2$, where $\oplus$ denotes concatenation. Additionally, the output of each *ModBlock2D* block is passed through a conditional *tTPF* layer that applies a conditional 2D convolution with kernel size 1x1. Note that, following the practices from StyleGAN2 [35], the conditioning in the *tTPF* layers is performed only through *modulation* of the weights (no *demodulation*).

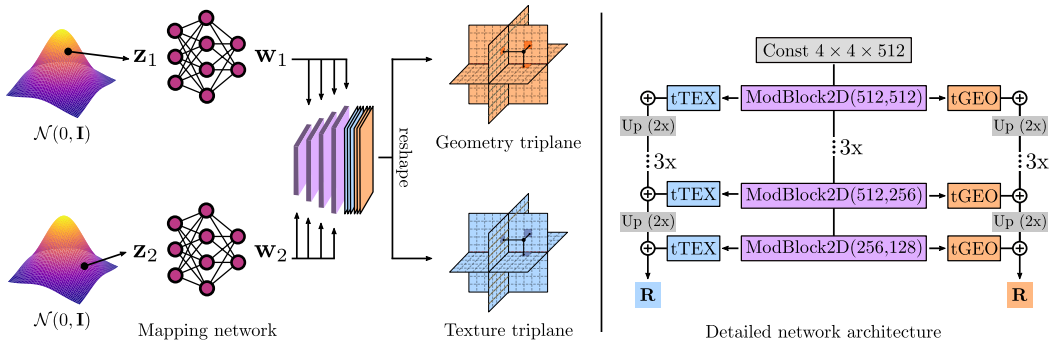


Figure C: **Improved generator architecture of GET3D**. High-level overview of our 3D **Generator** *left* and detailed architecture *right*. Different to the model architecture proposed in the main paper, the new generator shares the same backbone network for both geometry and texture generation. This improves the information flow and enables better disentanglement of the geometry and texture.

The output of the last *tTPF* layer is then reshaped into three axis-aligned feature planes of size $256 \times 256 \times 32$.

To obtain the feature $\mathbf{f}_{\text{tex}} \in \mathbb{R}^{32}$ of a surface point $\mathbf{p} \in \mathbb{R}^3$, we first project $\mathbf{p}$ onto each plane, perform bilinear interpolation of the features, and finally sum the interpolated features:

$$\mathbf{f}_{\text{tex}} = \sum_e \rho(\pi_e(\mathbf{p})), \tag{7}$$

where $\pi_e(\mathbf{p})$ is the projection of the point $\mathbf{p}$ to the feature plane e and $\rho(\cdot)$ denotes bilinear interpolation of the features. Color $\mathbf{c} \in \mathbb{R}^3$ of the point $\mathbf{p}$ is then decoded from $\mathbf{f}^t$ using three conditional FC layers (*ModFC*) conditioned on $\mathbf{w}_1 \oplus \mathbf{w}_2$. The hidden dimension of each layer is 16. Following StyleGAN2 [35], we do not apply normalization to the final output.

A.4 2D Discriminator

We use two discriminators to train GET3D: one for the RGB output and one for the 2D silhouettes. For both, we use exactly the same architecture as the discriminator in StyleGAN [34]. Empirically, we have observed that conditioning the discriminator on the camera pose leads to canonicalization of the shape orientations. However, discarding this conditioning only slightly affects the performance, as shown in Section C.3. In fact, we primarily use this conditioning to enable the evaluation of geometry using evaluation metrics, which assume that the shapes are generated in the canonical frame.

A.5 Improved Generator

The motivation for sampling two noise vectors ($\mathbf{z}_1$, $\mathbf{z}_2$) in the generator is to enable disentanglement of the geometry and texture, where geometry is to be treated as a first-class citizen. Indeed, the geometry should only be controlled by the geometry latent code, while the texture should be able to not only adapt to the changes in the texture latent code, but also to the changes in geometry, i.e. a change in the geometry latent should propagate to the texture. However, in the original design of the GET3D generator (c.f. Sec. 3 and Fig. 2) the information flow from the geometry to the texture generator is very limited—concatenation of the two latent codes (Fig. B). Such a weak connection makes it hard to learn the disentanglement of geometry and texture and the texture generator can even learn to ignore the texture latent code (Fig. D.).

This empirical observation motivated us to improve the design of the generator network, after the initial submission, by improving the information flow, which in turn better supports the disentanglement of the geometry and texture. To this end, our improved generator shares the same backbone network for both geometry and texture generation, as shown in Fig. C. In particular, we follow SemanticGAN [38] and use StyleGAN2 [35] backbone. Each ModBlock2D (modulated with the geometry latent code $\mathbf{w}_1$), now has two tTPF branches, one for generating the geometry feature (*tGEO*), and the other for generating texture features (*tTEX*). The output of this backbone network are two feature triplanes, one for geometry and one for texture. To predict the SDF value and deformation

Figure D: **Disentanglement of geometry and texture achieved by the original model depicted in Fig. 2**. In each row, we show shapes generated from the same texture latent code, while changing the geometry latent code. In each column, we show shapes generated from the same geometry latent code, while changing the texture code. The original model fails to achieve good disentanglement.

for each vertex in the tetrahedral grid, we project the vertex onto each of the geometry triplanes, obtain its feature vector using Eq. 7, and finally use a ModFC to decode s_i and $\Delta \mathbf{v}_i$. The prediction of the color in the texture branch remains unchanged.

Qualitative result of the geometry and texture disentanglement achieved with this improved generator is depicted in Fig. E and F. Shared backbone network allows us to achieve much better disentanglement of geometry and texture (Fig. D vs Fig. E), while also achieving better quantitative metrics on the task of unconditional generation (Tab. 2).

A.6 Training Procedure and Hyperparameters

We implement GET3D on top of the official PyTorch implementation of StyleGAN2 [35][2]. Our training configuration largely follows StyleGAN2 [35] including: using a minibatch standard deviation in the discriminator, exponential moving average for the generator, non-saturating logistic loss, and R1 Regularization. We train GET3D along with the 2D discriminators from scratch, without progressive training or initialization from pretrained checkpoints. Most of our hyper-parameters are adopted form styleGAN2 [35]. Specifically, we use Adam optimizer with learning rate 0.002 and $\beta = 0.9$. For R1 regularization, we set the regularization weight γ to 3200 for chair, 80 for car, 40 for animal, 80 for motorbike, 80 for renderpeople, and 200 for house. We follow StyleGAN2 [35] and use lazy regularization, which applies R1 regularization to discriminators only every 16 training steps. Finally, we set the hyperparameter μ that controls the SDF regularization to 0.01 in all the experiments. We train our model using a batch size of 32 on 8 A100 GPUs for all the experiments. Training a single model takes about 2 days to converge.

B Experimental Details

B.1 Datasets

We evaluate GET3D on ShapeNet [9], TurboSquid [4], and RenderPeople [2] datasets. In the following, we provide their detailed description and the preprocessing steps that were used in our evaluation. Detailed statistic of the datasets is available in Table A.

[2]StyleGan3: https://github.com/NVlabs/stylegan3 (NVIDIA Source Code License)

Figure E: **Disentanglement of geometry and texture achieved by the improved model depicted in Fig. C**. In each row, we show shapes generated from the same texture latent code, while changing the geometry latent code. In each column, we show shapes generated from the same geometry latent code, while changing the texture code. The disentanglement in this model is poor. Comparing with Fig. D, this improved model achieves significant better disentanglement of geometry and texture.

Figure F: **Shape Interpolation.** We interpolate the latent code from top-left corner to the bottom-right corner. In each row, we keep the texture latent code fixed and interpolate the geometry latent code. In each column, we keep the geometry latent code fixed and interpolate the texture latent code. GET3D adequately disentangles geometry and texture, while also providing a meaningful interpolation for both geometry or texture.

ShapeNet[3] [9] contains more than 51k shapes from 55 different categories and is the most commonly used dataset for benchmarking 3D generative models[4]. Prior work [68, 75] typically uses the categories *Airplane*, *Car*, and *Chair* for evaluation. Herein, we replace the category *Airplane* with *Motorcycle*, which has more complex geometry and contains shapes with varying genus. *Car*, *Chair*, and *Motorcycle* contain 7497, 6778, and 337 shapes, respectively. We random split the shapes of each category into training (70%), validation (10%), and test (20%) and remove from the test set shapes that have duplicates in the training set.

TurboSquid[5] [4] is a large collection of various 3D shapes with high-quality geometry and texture, and is thus well suited to evaluate the capacity of GET3D to generate shapes with high-quality

[3]The ShapeNet license is explained at https://shapenet.org/terms

[4]Herein, we used ShapeNet v1 Core subset obtained from https://shapenet.org/

[5]https://www.turbosquid.com, we obtain consent via an agreement with TurboSquid, and following license at https://blog.turbosquid.com/turbosquid-3d-model-license/

Dataset	# Shapes	# Views per shape	Rotation Angle	Elevation Angle
ShapeNet Car	7497	24	$[0, 2\pi]$	$[\frac{1}{3}\pi, \frac{1}{2}\pi]$
ShapeNet Chair	6778	24	$[0, 2\pi]$	$[\frac{1}{3}\pi, \frac{1}{2}\pi]$
ShapeNet Motorbike	337	100	$[0, 2\pi]$	$[\frac{1}{3}\pi, \frac{1}{2}\pi]$
Turbosquid Animal	442	100	$[0, 2\pi]$	$[\frac{1}{4}\pi, \frac{1}{2}\pi]$
Turbosquid House	563	100	$[0, 2\pi]$	$[\frac{1}{3}\pi, \frac{1}{2}\pi]$
Renderpeople	500	100	$[0, 2\pi]$	$[\frac{1}{3}\pi, \frac{1}{2}\pi]$

Table A: **Dataset statistics**.

details. To this end, we use the category *Animal* that contains 442 textured shapes with high diversity ranging from cats, dogs, and lions, to bears and deer [60, 70]. We again randomly split the shapes into training (70%), validation (10%), and test (20%) set. Additionally, we provide qualitative results on the category *House* that contains 563 shapes. Since we perform only qualitative evaluation on *House*, we use all the shapes for training.

RenderPeople[6] [2] is a large dataset containing photorealistic 3D models of real-world *humans*. We use it to showcase the capacity of GET3D to generate high-quality and diverse characters that can be used to populate virtual environments, such as games or even movies. In particular, we use 500 models from the whole dataset for training and only perform qualitative analysis.

Preprocessing To generate the data, we first scale each shape such that the longest edge of its bounding-box equals $\mathbf{e}_m$, where $\mathbf{e}_m = 0.9$ for *Car*, *Motorcycle*, and *Human*, $\mathbf{e}_m = 0.8$ for *House*, and $\mathbf{e}_m = 0.7$ for *Chair* and *Animal*. For methods that use 2D supervision (Pi-GAN, GRAF, EG3D, and our model GET3D), we then render the RGB images and silhouettes from camera poses sampled from the upper hemisphere of each object. Specifically, we sample 24 camera poses for *Car* and *Chair*, and 100 poses for *Motorcycle*, *Animal*, *House*, and *Human*. The rotation and elevation angles of the camera poses are sampled uniformly from a specified range (see Table A). For all camera poses, we use a fixed radius of 1.2 and the fov angle of $49.13°$. We render the images in Blender [15] using a fixed lighting, unless specified differently.

For the methods that rely on 3D supervision, we follow their preprocessing pipelines [68, 43]. Specifically, for Pointflow [68] we randomly sample 15k points from the surface of each shape, while for OccNet [43] we convert the shapes into watertight meshes by rendering depth frames from random camera poses and performing TSDF fusion.

B.2 Baselines

PointFlow [68] is a 3D point cloud generative model based on continuous normalizing flows. It models the generative process by learning a distribution of distributions. Where the former, denotes the distribution of shapes, and the latter the distribution of points given a shape [68]. PointFlow generates only the geometry, which is represented in the form of a point cloud. To generate the results of [68], we use the original source code provided by the authors[7] and train the models on our data. To compute the metrics based on LFD, we convert the output point clouds (10k points) to a mesh representation using Open3D [77] implementation of Poisson surface reconstruction [36].

OccNet [43] is an implicit method for 3D surface reconstruction, which can also be applied to unconditional generation of 3D shapes. OccNet is an autoencoder that learns a continuous mapping from 3D coordinates to occupancy values, from which an explicit mesh can be extracted using marching cubes [39]. When applied to unconditional 3D shape generation, OccNet is trained as a variational autoencoder. To generate the results of [43], we use the original source code provided by the authors[8] and train the models on our data.

[6] We follow the license of Renderpeople https://renderpeople.com/general-terms-and-conditions/

[7] PointFlow: https://github.com/stevenygd/PointFlow (MIT License)

[8] OccNet: https://github.com/autonomousvision/occupancy_networks (MIT License)

GRAF [57] is a generative model that tackles the problem of 3D-aware image synthesis. GRAF's underlying representation is a neural radiance field—conditioned on the shape and appearance latent codes—parameterized using a multi-layer perceptron with positional encoding. To synthesize novel views, GRAF utilizes a neural volume rendering approach similar to Nerf [45]. In our evaluation, we use the source code provided by the authors[9] and train GRAF models on our data.

Pi-GAN [7] similar to GRAF, Pi-GAN also tackles the problem of 3D-aware image synthesis, but uses a Siren [61] network—conditioned on a randomly sampled noise vector—to parameterize the neural radiance field. To generate the results of Pi-GAN [7], we use the original source code provided by the authors[10] and train the models on our data.

EG3D [8] is a recent model for 3D-aware image synthesis. Similar to our method, EG3D builds upon the StyleGAN formulation and uses a tri-plane representation to parameterize the underlying neural radiance field. To improve the efficiency and to enable synthesis at higher resolution, EG3D utilizes neural rendering at a lower resolution and then upsamples the output using a 2D CNN. The source code of EG3D was provided to us by the authors. To generate the results, we train and evaluate EG3D on our data.

B.3 Evaluation Metrics

To evaluate the performance, we compare both the texture and geometry of the *generated* shapes S_g to the *reference* ones S_r.

B.3.1 Evaluating the Geometry

To evaluate the geometry, we use all shapes of the test set as S_r, and synthesize five times as many generated shapes, such that $|S_g| = 5|S_r|$, where $|\cdot|$ denotes the cardinality of a set. Following prior work [68, 14], we use Chamfer Distance d_{CD} and Light Field Distance d_{LFD} [13] to measure the similarity of the shapes, which is in turn used to compute Coverage (COV) and Minimum Matching Distance (MMD) evaluation metrics.

Let $X \in S_g$ denote a generated shape and $Y \in S_r$ a reference one. To compute d_{CD}, we first randomly sample $N = 2048$ points $X_p \in \mathbb{R}^{N\times 3}$ and $Y_p \in \mathbb{R}^{N\times 3}$ from the surface of the shapes X and Y, respectively[11] . The d_{CD} can then be computed as:

$$d_{\text{CD}}(X_p, Y_p) = \sum_{\mathbf{x}\in X_p} \min_{\mathbf{y}\in Y_p} ||\mathbf{x}-\mathbf{y}||_2^2 + \sum_{\mathbf{y}\in Y_p} \min_{\mathbf{x}\in X_p} ||\mathbf{x}-\mathbf{y}||_2^2. \tag{8}$$

While Chamfer distance has been widely used in the field of 3D generative models and reconstruction [11, 22, 60], LFD has received a lot attention in computer graphics [13]. Inspired by human perception, LFD measures the similarity between the 3D shapes based on their appearance from different viewpoints. In particular, LFD renders the shapes X and Y (represented as explicit meshes) from a set of selected viewpoints, encodes the rendered images using Zernike moments and Fourier descriptors, and computes the similarity over these encodings. Formal definition of LFD is available in [13]. In our evaluation, we use the official implementation to compute d_{LFD}[12].

We combine these similarity measures with the evaluation metrics proposed in [5], which are commonly used to evaluate 3D generative models:

- **Coverage (COV)** measures the fraction of shapes in the *reference* set that are matched to at least one of the shapes in the *generated* set. Formally, COV is defined as

$$\text{COV}(S_g, S_r) = \frac{|\{\text{argmin}_{X\in S_r} D(X,Y)\,|Y \in S_g\}|}{|S_r|}, \tag{9}$$

  where the distance metric D can be either d_{CD} or d_{LFD}. Intuitively, COV measures the diversity of the generated shapes and is able to detect mode collapse. However, COV does

[9] GRAF: https://github.com/autonomousvision/graf (MIT License)

[10] Pi-GAN: https://github.com/marcoamonteiro/pi-GAN (License not provided)

[11] For PointFlow [68], we directly use N points generated by the model.

[12] LFD: https://github.com/Sunwinds/ShapeDescriptor/tree/master/LightField/3DRetrieval_v1.8/3DRetrieval_v1.8 (License not provided)

not measure the quality of individual generated shapes. In fact, it is possible to achieve high COV even when the generated shapes are of very low quality.

- **Minimum Matching Distance (MMD)** complements COV metric, by measuring the quality of the individual generated shapes. Formally, MMD is defined as

$$\text{MMD}(S_g, S_r) = \frac{1}{|S_r|} \sum_{X \in S_r} \min_{Y \in S_g} D(X, Y), \tag{10}$$

where D can again be either d_{CD} or d_{LFD}. Intuitively, MMD measures the quality of the generated shapes by comparing their geometry to the closest reference shape.

B.3.2 Evaluating the Texture and Geometry

To evaluate the quality of the generated textures, we adopt the Fréchet Inception Distance (FID) metric, commonly used to evaluate the synthesis quality of 2D images. In particular, for each category, we render 50k views of the generated shapes (one view per shape) from the camera poses randomly sampled from the predefined camera distribution, and use all the images in the test set. We then encode these images using a pretrained Inception v3 [63] model[13], where we consider the output of the last pooling layer as our final encoding. The FID metric can then be computed as:

$$\text{FID}(S_g, S_r) = ||\boldsymbol{\mu}_g - \boldsymbol{\mu}_r||_2^2 + \text{Tr}[\boldsymbol{\Sigma}_g + \boldsymbol{\Sigma}_r - 2(\boldsymbol{\Sigma}_g \boldsymbol{\Sigma}_r)^{1/2}]||, \tag{11}$$

where Tr denotes the trace operation. $\boldsymbol{\mu}_g$ and $\boldsymbol{\Sigma}_g$ are the mean value and covariance matrix of the generated image encoding, while $\boldsymbol{\mu}_r$ and $\boldsymbol{\Sigma}_r$ are obtained from the encoding of the test images.

As briefly discussed in the main paper, we use two variants of FID, which differ in the way in which the 2D images are rendered. In particular, for FID-Ori, we directly use the neural volume rendering of the 3D-aware image synthesis methods to obtain the 2D images. This metric favours the baselines that were designed to directly generate valid 2D images through neural rendering. Additionally, we propose a new metric, FID-3D, which puts more emphasis on the overall quality of the generated 3D shape. Specifically, for the baselines which do not output a textured mesh, we extract the geometry from their underlying neural field using marching cubes [39]. Then, we find the intersection point of each pixel ray with the generated mesh and use the 3D location of the intersected point to query the RGB value from the network. In this way, the rendered image is a more faithful representation of the underlying 3D shape and takes the quality of both geometry and texture into account. Note that FID-3D and FID-Ori are identical for methods that directly generate textured 3D meshes, as it is the case with GET3D.

C Additional Results on the Unconditioned Shape Generation

In this section we provide additional results on the task of unconditional 3D shape generation. First, we perform additional qualitative comparison of GET3D with the baselines in Section C.1. Second, we present further qualitative results of GET3D in Section C.2. Third, we provide additional ablation studies in Section C.3. We also analyse the robustness and effectiveness of GET3D. Specifically, in Sec. C.4 and C.5, we evaluate GET3D trained with noisy cameras and 2D silhouettes predicted by 2D segmentation networks. We further provide addition experiments on StyleGAN generated realistic dataset from GANverse3D [73] in Sec. C.6. Finally, we provide additional comparison with EG3D [8] on human character generation in Sec. C.7.

C.1 Additional Qualitative Comparison with the Baselines

Comparing the Geometry of Generated Shapes We provide additional visualization of the 3D shapes generated by GET3D and compare them to the baseline methods in Figure Q. GET3D is able to generate shapes with complex geometry, different topology, and varying genus. When compared to the baselines, the shapes generated by GET3D contain more details and are more diverse.

[13] Inception network checkpoint path: http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz

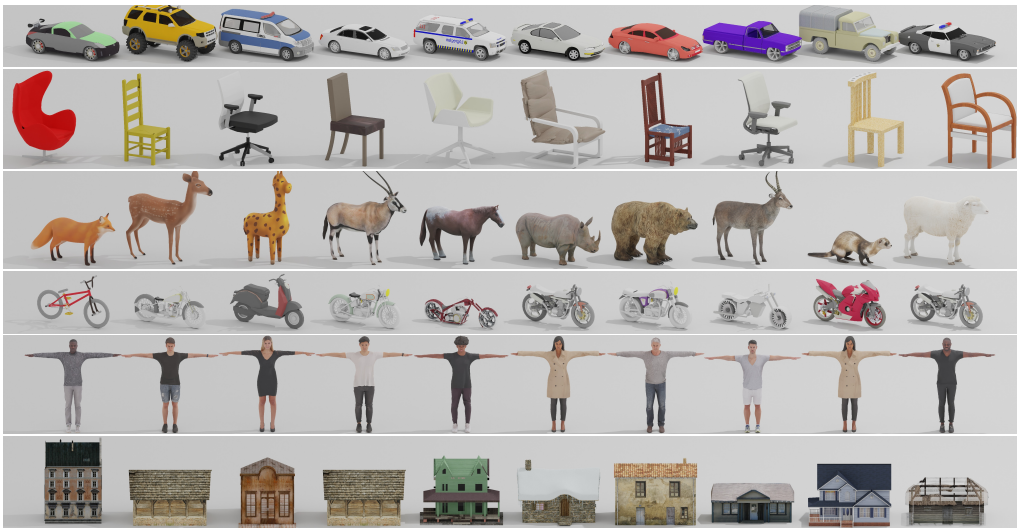


Figure G: **Shape retrieval of our generated shapes**. We retrieve the closest shape in the training set for each of shapes we showed in the Figure 1. Our generator is able to generate novel shapes that are different from the training set

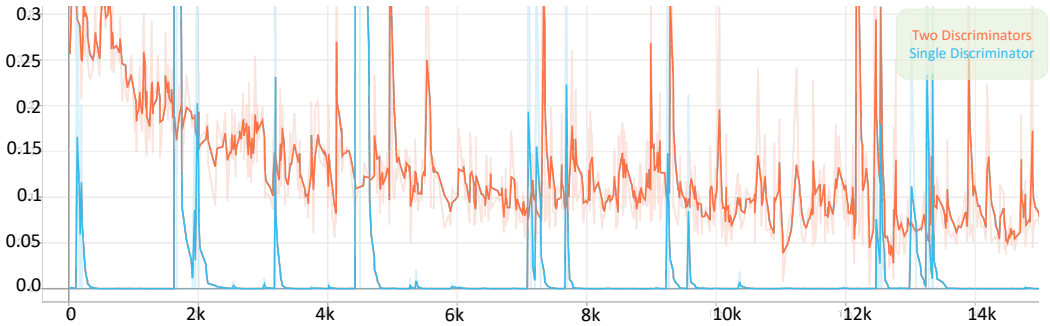


Figure H: **Training loss curve for discriminator.** We compare the training dynamics of using a single discriminator on both RGB image and 2D silhouette, with the ones using two discriminators for each image, respectively. The horizontal axis represents the number of images that the discriminators have seen during training (mod by 1000). Two discriminators greatly reduce training instability and help us obtain good results.

Comparing the Synthesized Images We provide additional results on the task of 2D image generation in Figure R. Even though GET3D is not designed for this task, it produces comparable results to the strong baseline EG3D [8], while significantly outperforming other baselines, such as PiGAN [7] and GRAF [57]. Note that GET3D directly outputs 3D textured meshes, which are compatible with standard graphics engines, while extracting such representation from the baselines is non-trivial.

C.2 Additional Qualitative Results of GET3D

We provide additional visualizations of the generated geometry and texture in Figures S-X. GET3D can generate high quality shapes with diverse textures across all the categories, from chairs, cars, and animals, to motorbikes, humans, and houses. Accompanying video (*demo.mp4*) contains further visualizations, including detailed 360° turntable animations for 400+ shapes and interpolation results.

Closest Shape Retrieval To demonstrate that GET3D is capable of generating novel shapes, we perform shape retrieval for our generated shapes. In particular, we retrieve the closest shape in the training set for each of shapes we showed in the Figure 1 by measuring the CD between the generated

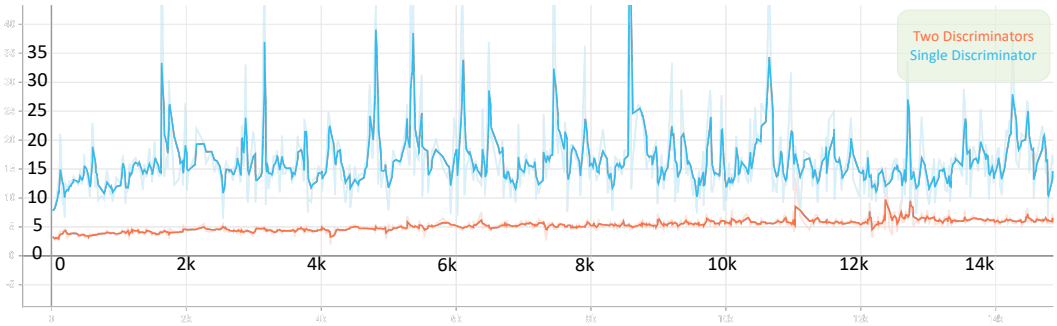


Figure I: **Training loss curve for generator.** We compare the training dynamics for using single discriminator on both RGB image and 2D silhouette with two discriminators for each image, respectively. The horizontal axis represents the number of images discriminator have seen during training (mod by 1000).

Model	FID
GET3D w.o. Camera Condition	11.63
GET3D w/ Camera Condition	10.25

Table B: **Ablations on using camera condition**: We ablate using camera condition for discriminator. We train the model on Shapenet Car dataset.

shape and all training shapes. Results are provided in Figure G. All generated shapes in Figure 1 significantly differ from their closest shape in the training set, exhibiting different geometry and texture, while still maintaining the quality and diversity.

Volume Subdivision We provide further qualitative results highlighting the benefits of volume subdivision in Figure Y. Specifically, we compare the shapes generated with and without volume subdivision on ShapeNet motorbike category. Volume subdivision enables GET3D to generate finer geometric details like handle and steel wire, which are otherwise hard to represent.

C.3 Additional Ablations Studies

We now provide additional ablation studies in an attempt to further justify our design choices. In particular, we first discuss the design choice of using two dedicated discriminators for RGB images and 2D silhouettes, before ablating the impact of adding the camera pose conditioning to the discriminator.

C.3.1 Using Two Dedicated Discriminators

We empirically find that using a single discriminator on both RGB image and silhouettes introduces significant training instability, which leads to divergence when training GET3D. We provide a comparison of the training dynamics in Figure H and I, where we depict the loss curves for the generator and discriminator. We hypothesize that the instability might be caused by the fact that a single discriminator has access to both geometry (from 2D silhouettes) and texture (from RGB image) of the shape, when classifying whether the image is real or not. Since we randomly initialize our geometry generator, the discriminator can quickly overfit to one aspect—either geometry or texture—and thus produces bad gradients for the other branch. A two-stage approach in which two discriminators would be used in the first stage of the training, and a single discriminator in the later stage, when the model has already learned to produce meaningful shapes, is an interesting research direction, which we plan to explore in the future.

C.3.2 Ablation on Using Camera Condition for Discriminator

Since we are mainly operating on synthetic datasets in which the shapes are aligned to a canonical direction, we condition the discriminators on the camera pose of each image. In this way, GET3D

Figure J: **Additional qualitative results of GET3D trained with noisy cameras.** We render generated shapes in Blender. The visual quality is similar to original GET3D in the main paper.

Figure K: **Additional qualitative results of GET3D trained with predicted 2D silhouettes (Mask-Black).** We render generated shapes in Blender. The visual quality is similar to original GET3D in the main paper.

Figure L: **Additional qualitative results of GET3D trained with predicted 2D silhouettes (Mask-Random).** We render generated shapes in Blender. The visual quality is similar to original GET3D in the main paper.

learns to generate shapes in the canonical orientation, which simplifies the evaluation when using metrics that assume that the input shapes are canonicalized. We now ablate this design choice. Specifically, we train another model without the conditioning and evaluate its performance in terms of FID score. Quantitative results are given in Table. B. We observe that removing the camera pose conditioning, only slightly degrades the performance of GET3D (-1.38 FID). This confirms that our model can be successfully trained without such conditioning, and that the primary benefit of using it is the easier evaluation.

Method	FID
GET3D - original	10.25
GET3D - noisy cameras	19.53
GET3D - predicted 2D silhouettes (Mask-Black)	29.68
GET3D - predicted 2D silhouettes (Mask-Random)	33.16

Table C: Additional quantitative results for noisy cameras and using predicted 2D silhouettes on Shapenet Car dataset.

C.4 Robustness to Noisy Cameras

To demonstrate the robustness of GET3D to imperfect cameras poses, we add Gaussian noises to the camera poses during training. Specifically, for the rotation angle, we add a noise sampled from a Gaussian distribution with zero mean, and 10 degrees variance. For the elevation angle, we also add a noise sampled from a Gaussian distribution with zero mean, and 2 degrees variance. We use ShapeNet Car dataset [9] in this experiment.

The quantitative results are provided in Table C and qualitative examples are depicted in Figure J. Adding camera noise harms the FID metric, whereas we observe only little degradation in visual

Figure M: **Additional qualitative results of GET3D trained with "real" GANverse3D [73] data.** We render generated shapes in Blender.

Figure N: We show randomly sampled 2D images and silhouettes from GANverse3D [73] data. Note the realism of the images and the imperfections of the 2D silhouettes.

quality. We hypothesize that the drop in the FID is a consequence of the camera pose distribution mismatch, which occurs as result of rendering the testing dataset, used to calculate the FID score, with a camera pose distribution without added noise. Nevertheless, based on the visual quality of the generated shapes, we conclude that GET3D is robust to a moderate level of noise in the camera poses.

C.5 Robustness to Imperfect 2D Silhouettes

To evaluate the robustness of GET3D when trained with imperfect 2D silhouettes, we replace ground truth 2D masks with the ones obtained from Detectron2[14] using pretrained PointRend checkpoint, mimicking how one could obtain the 2D segmentation masks in the real world. Since our training images are rendered with the black background, we use two approaches to obtain the 2D silhouettes: i) we directly feed the original training image into Detectron2 to obtain the predicted segmentation mask (we refer to this as Mask-Black), and ii) we add a background image, randomly sampled from PASCAL-VOC 2012 dataset (we refer to this as Mask-Random). In this setting, the pretrained Detectron2 model achieved 97.4 and 95.8 IoU for the Mask-Black and Mask-Random versions, respectively. We again use the Shapenet Car dataset [9] in this experiment.

Experimental Results Quantitative results are summarized in Table C, with qualitative examples provided in Figures K and L. Although we observe drop in the FID scores, qualitatively the results are still similar to the original results in the main paper. Our model can generate high quality shapes even when trained with the imperfect masks. Note that, in this scenario, the training data for GET3D is different from the testing data that is used to compute the FID score, which could be one of the reasons for worse performance.

C.6 Experiments on "Real" Image

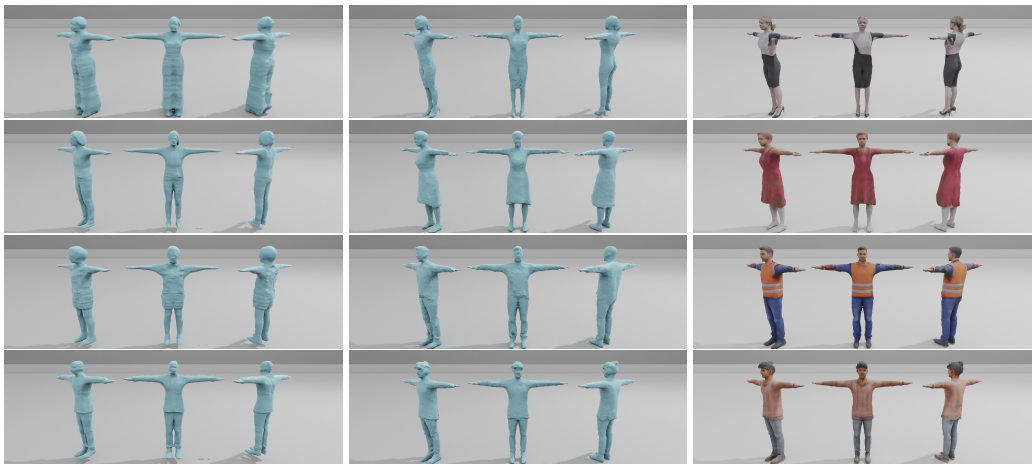
Since many real-world datasets lack camera poses, we follow GANverse3D [73] and utilize pretrained 2D StyleGAN to generate a realistic car dataset. We train GET3D on this dataset to demonstrate the potential applications to real-world data.

Experimental Setting Following GANverse3D [73], we manipulate the latent codes of 2D StyleGAN and generate multi-view car images. To obtain the 2D segmentation of each image, we use

[14] https://github.com/facebookresearch/detectron2

Method	FID (↓)	
	Ori	3D
EG3D [8]	**13.77**	60.42
GET3D	14.27	**14.27**

Table D: Additional quantitative comparison with EG3D [8] on *Human Body* dataset [2].



EG3D [8] Ours Ours-Tex

Figure O: **Additional qualitative comparison on *Human Body* dataset.** We compare our method with EG3D [8] on the extracted geometry.

DatasetGAN [74] to predict the 2D silhouette. We then use SfM [65] to obtain the camera initialization for each generated image. We visualize some examples of this dataset in Fig N and refer the reader to the original GANverse3D paper for more details. Note that, in this dataset both cameras and 2D silhouettes are imperfect.

Experimental Results We provide qualitative examples in Fig. M. Even when faced with the imperfect inputs during training, GET3D is still capable of generating reasonable 3D textured meshes, with variation in geometry and texture.

C.7 Comparison with EG3D on Human Body

Following the suggestion of the reviewer, we also train EG3D model on the *Human Body* dataset rendered from Renderpeople [2] and compare it to the results of GET3D.

Quantitative results are available in Table D and qualitative comparisons in Figure O. GET3D achieves comparable performance to EG3D [8] in terms of generated 2D images (FID-ori), while significantly outperforming it on 3D shape synthesis (FID-3D). This once more demonstrates the effectiveness of our model in learning actual 3D geometry and texture.

D Material Generation for View-dependent Lighting Effects

In modern computer graphics engines such as Blender [15] and Unreal Engine [32], surface properties are represented by material parameters crafted by graphics artists. To make the generated assets graphics-compatible, one direct extension of our method is to also generate surface material properties. In this section, we describe how GET3D is able to incorporate physics-based rendering models, predicting SVBRDF to represent view-dependent lighting effects such as specular surface reflections.

As described in main paper Sec. 4.3.1, two modules need to be adapted to facilitate material generation. Namely, the texture generation and the rendering process. Specifically, we repurpose the texture

generator branch to predict the Disney BRDF properties [6, 32] on the surface as a reflectance field. Specifically, the texture generator now outputs a 5-channel reflectance property, including surface base color $\mathbf{c}_{\text{base}} \in \mathbb{R}^3$, roughness $\beta \in \mathbb{R}$ and metallic $m \in \mathbb{R}$ parameters.

Note that different from a texture field, rendering the reflectance field requires one additional shading step after rasterization into the G-buffer. Thus, the second adaptation is to replace the texture rasterization with an expressive rendering model capable of rendering the reflectance field. According to the non-emissive rendering equation [31], the outgoing radiance L_{o} at the camera direction $\boldsymbol{\omega}_{\text{o}}$ is given by:

$$L_{\text{o}}(\boldsymbol{\omega}_{\text{o}}) = \int_{\mathcal{S}^2} L_{\text{i}}(\boldsymbol{\omega}_{\text{i}}) f_{\text{r}}(\boldsymbol{\omega}_{\text{i}}, \boldsymbol{\omega}_{\text{o}}; \mathbf{c}_{\text{base}}, \beta, m)(\mathbf{n} \cdot \boldsymbol{\omega}_{\text{i}})^{+} \, \mathrm{d}\boldsymbol{\omega}_{\text{i}}, \tag{12}$$

where L_{i} is the incoming radiance, f_{r} is the BRDF, $\mathbf{n}$ is the normal direction on the surface points, $\mathbf{n} \cdot \boldsymbol{\omega}_{\text{i}}$ is the cosine foreshortening term, $\boldsymbol{\omega}_{\text{i}}$ is incoming light direction sampled on sphere $\mathcal{S}^2$, while $(\mathbf{n} \cdot \boldsymbol{\omega}_{\text{i}})^{+} = \max(\mathbf{n} \cdot \boldsymbol{\omega}_{\text{i}}, 0)$ constrains the integration over the positive hemisphere. Standard ray tracing technique adopts Monte Carlo sampling methods to estimate this integral, but this incurs large computation and memory cost. Inspired by [64, 72, 12], we instead employ a spherical Gaussian (SG) rendering framework [12], which approximates every term in Eq. (12) with SGs and allows us to analytically compute the outgoing radiance without sampling any rays, from where we can obtain the RGB color for each pixel in the image. We refer the reader to [12] for more details.

Similar to the original training pipeline, we randomly sample light from a set of real-world outdoor HDR panoramas (detailed in the following "Datasets" paragraph) and render the generated 3D assets into 2D images using cameras sampled from the camera distribution of training set. We train the model using the same method as in the main paper by adopting the discriminators to encourage the perceptual realism of the rendered images under arbitrary real-world lighting, along with a second discriminator on the 2D silhouettes to learn the geometry. Note that no supervision from material ground truth is used during training, and the material decomposition emerges in a fully unsupervised manner. When equipped with a physics-based rendering models, GET3D successfully predicts reasonable surface material parameters, generating delicate models which can be directly utilized in stand rendering engines like Blender [15] and Unreal [32].

Datasets We collect a set of 724 outdoor HDR panoramas from HDRIHaven[15], DoschDesign[16] and HDRMaps[17], which cover a diverse range of real-world lighting distribution for outdoor scenes. We also apply random flipping and random rotation along azimuth as data augmentation. During training, we convert all the environment maps to SG lighting representations, where we adopt 32 SG lobes, optimizing their directions, sharpness and amplitudes such that the approximated lighting is close to the environment map. We optimize 7000 iterations with MSE loss and Adam optimizer. The converged SG lighting can preserve the most contents in the environment map.

As ShapeNet dataset [9] does not contain consistent material definition, we additionally collect 1494 cars from Turbosquid [4] with materials consistently defined with Disney BRDF. To render the dataset using Blender [15], we follow the camera configuration of ShapeNet Car dataset, and randomly select from the collected set of HDR panoramas as lighting. In the dataset, the groundtruth roughness for car windows is in the range of $[0.2, 0.4]$ and the metallic is set to 1; for car paint, the groundtruth roughness is in the range of $[0.3, 0.6]$ and the metallic is set to 0. We disable complex materials such as the transparency and clear coat effects, such that the rendered results can be interpreted by the basic Disney BRDF properties including base color, metallic and roughness.

Evaluation metrics Since we aim to generate 3D assets that can be used in graphics workflow to produce realistic 2D renderings, we quantitatively evaluate the realism of the 2D rendered images under real-world lighting using FID score.

Comparisons To the best of our knowledge, up to date no generative model can directly generate complex geometry (meshes) with material information. We therefore only compare different version of our model. In particular, we compare the results to the texture prediction version of GET3D,

[15] polyhaven.com/hdris (License: CC0)

[16] doschdesign.com (License: doschdesign.com/information.php?p=2)

[17] hdrmaps.com (License: Royalty-Free)

where we do not use material and directly predict RGB color for the surface points. We then ablate the effects of using real-world HDR panoramas for lighting, which are typically hard to obtain. To this end, we manually use two spherical Gaussians for ambient lighting and a random directions to simulate the lighting when rendering the generated shapes during training, and try to learn the materials under this simulated lighting.

Results The quantitative FID scores are provided in Table E. With material generation, the FID score improves by more than 2 points when compared to the texture prediction baseline (18.53 vs 20.78). This indicates that the material generation version of GET3D has better capacity and improved realism compared to the texture only baseline. When using the simulated lighting, instead of real-world HDR panorama, the FID score gets slightly worse but still produces reasonable performance. We further provide additional qualitative results in Fig. P visualizing rendering results of generated assets under different real-world lighting conditions. We import our generated assets in Blender and show animated visualization in the accompanied video (*demo.mp4*).

Method	FID
Ours (Texture)	20.78
Ours + Material (Ambient and directional light)	22.83
Ours + Material (Real-world light)	**18.53**

Table E: Quantitative FID results of material generation.

E Text-Guided 3D Synthesis

Technical details. As briefly described in Sec. 4.3.2, our text-guided 3D synthesis method follows the dual-Generator design from StyleGAN-NADA [21], and uses the directional CLIP loss [21]. In particular, at each optimization iteration, we randomly sample $N = 16$ camera views and render N paired images using two generators: the frozen one (G_f) and the trainable one (G_t). The directional CLIP loss can then be computed as:

$$L_{clip} = 1 - \frac{1}{N}\sum_{i=1}^{N}\frac{\Delta I_i \cdot \Delta T}{|\Delta I_i| \cdot |\Delta T|} \tag{13}$$

where $\Delta I_i = E(R(G_t(w), c_i)) - E(R(G_f(w), c_i))$ is the translation of the CLIP embeddings (E) from the rendering with G_f to the rendering with G_t, under camera c_i and ΔT is the CLIP embedding translation from the class text label to the provided query text. In our implementation, we used two pre-trained CLIP models with different Vision Transformers ('ViT-32/B' and 'ViT-B/16') [18] for different level of details, and follow the text augmentation as in the StyleGAN-NADA codebase[18].

[18] https://github.com/rinongal/StyleGAN-nada (MIT License)

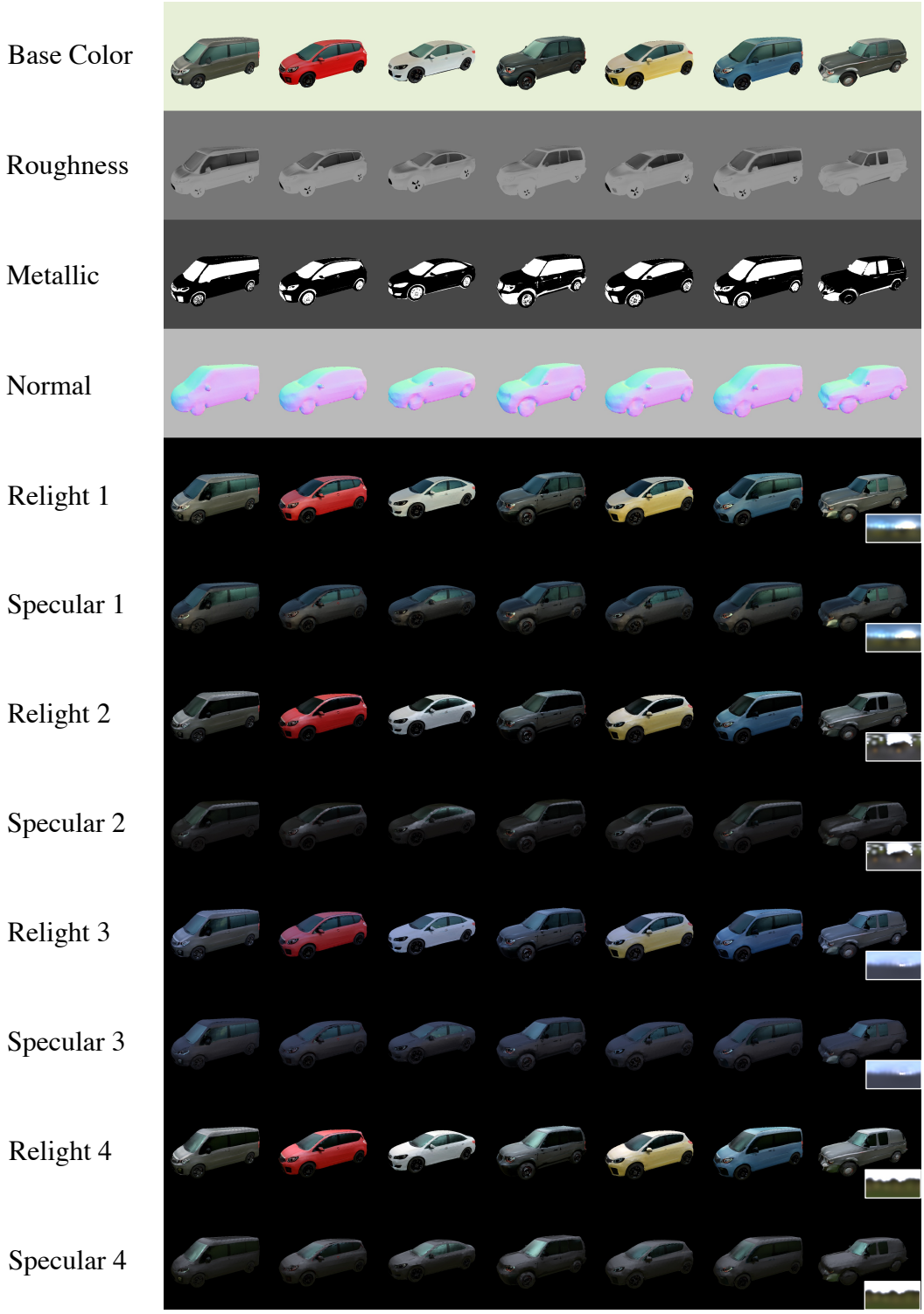


Figure P: **Material generation and relighting.** We visualize seven generated cars' material properties and relight with four different lighting conditions.

Figure Q: **Generated 3D Geometry.** Additional qualitative comparison with baseline methods on generated 3D geometry

Figure R: **Generated Image.** Additional qualitative comparison with baseline methods on generated 2D images.

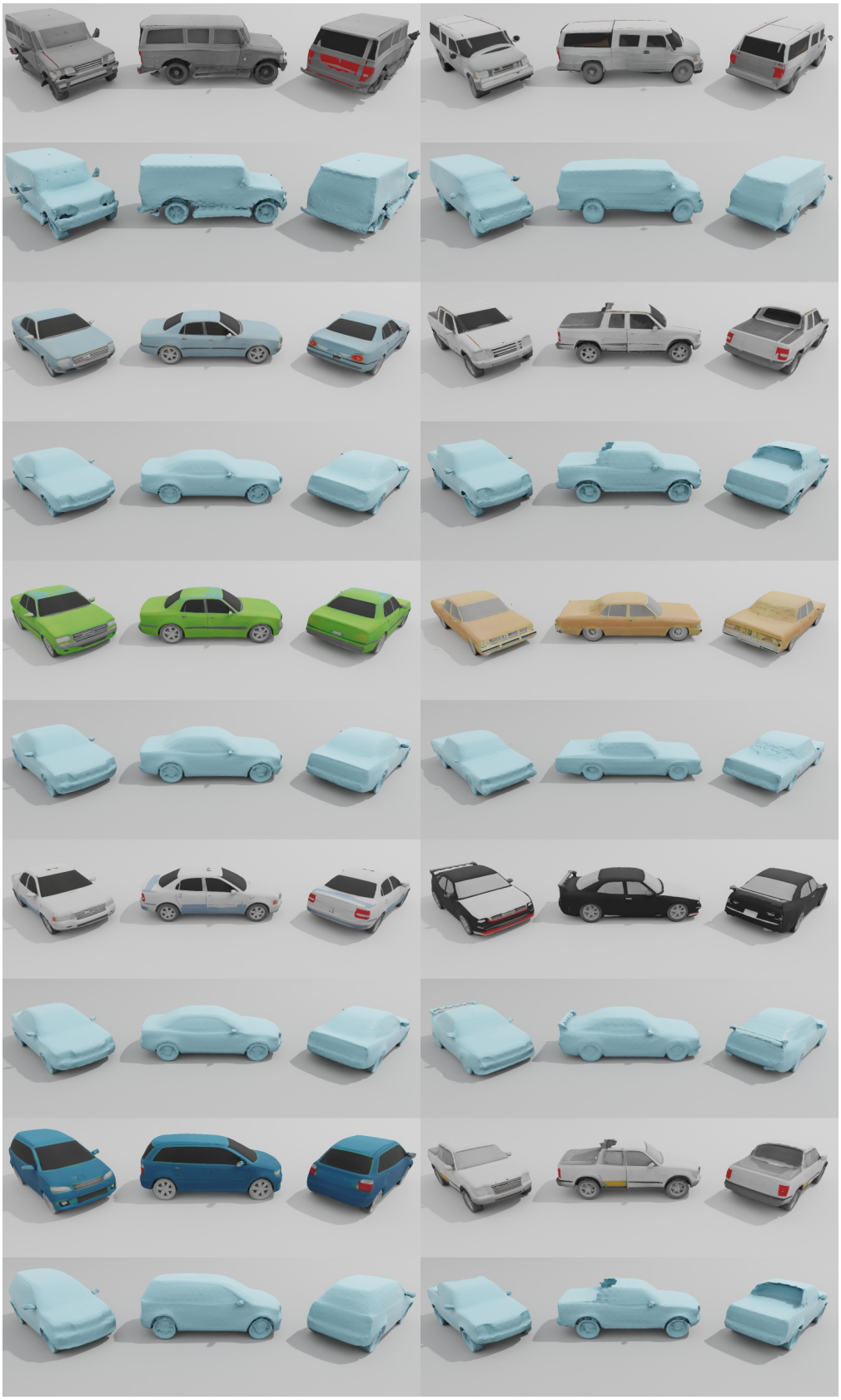


Figure S: Qualitative results on ShapeNet cars.

Figure T: Qualitative results on ShapeNet chairs.

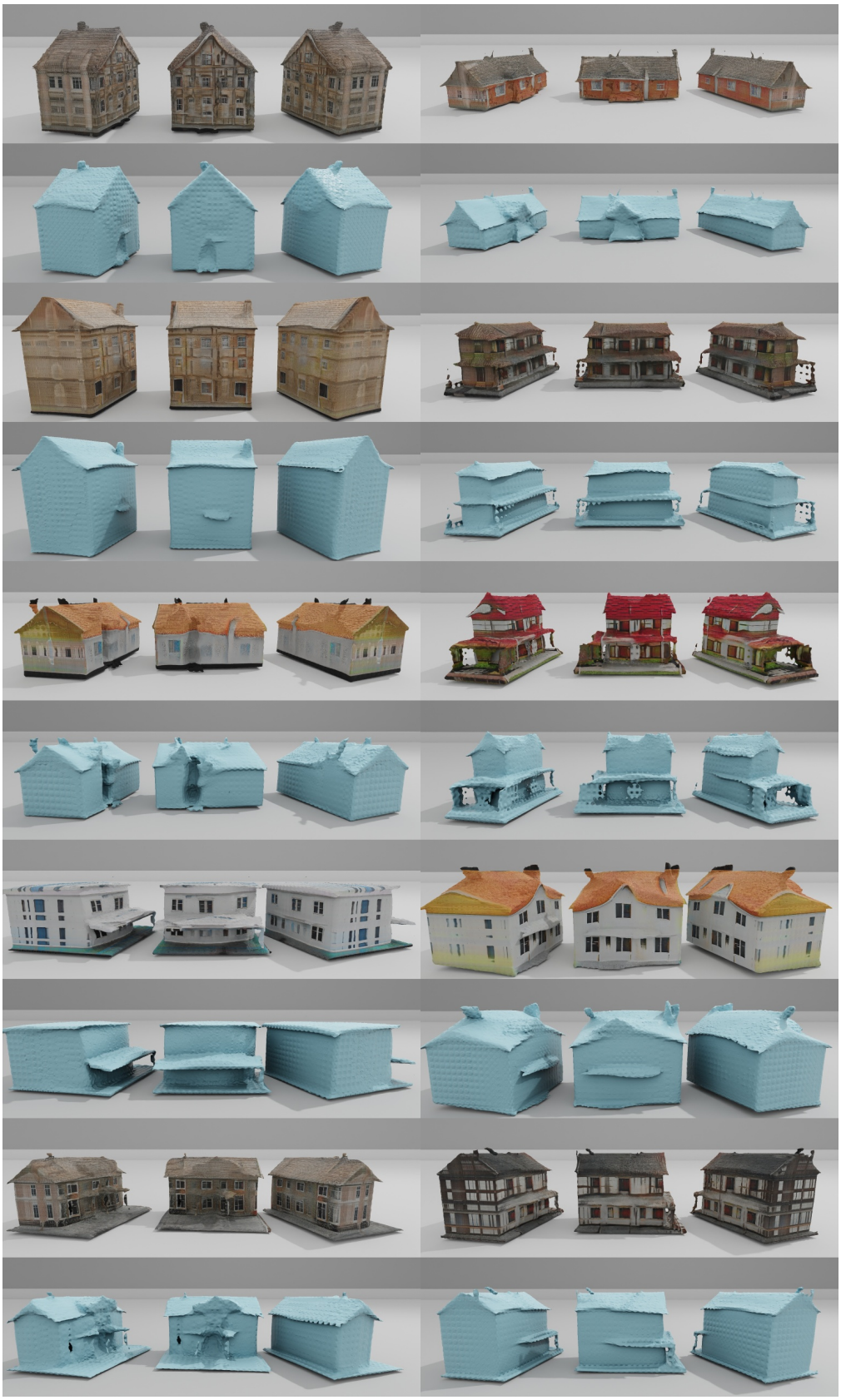


Figure U: Qualitative results on Turbosquid houses.

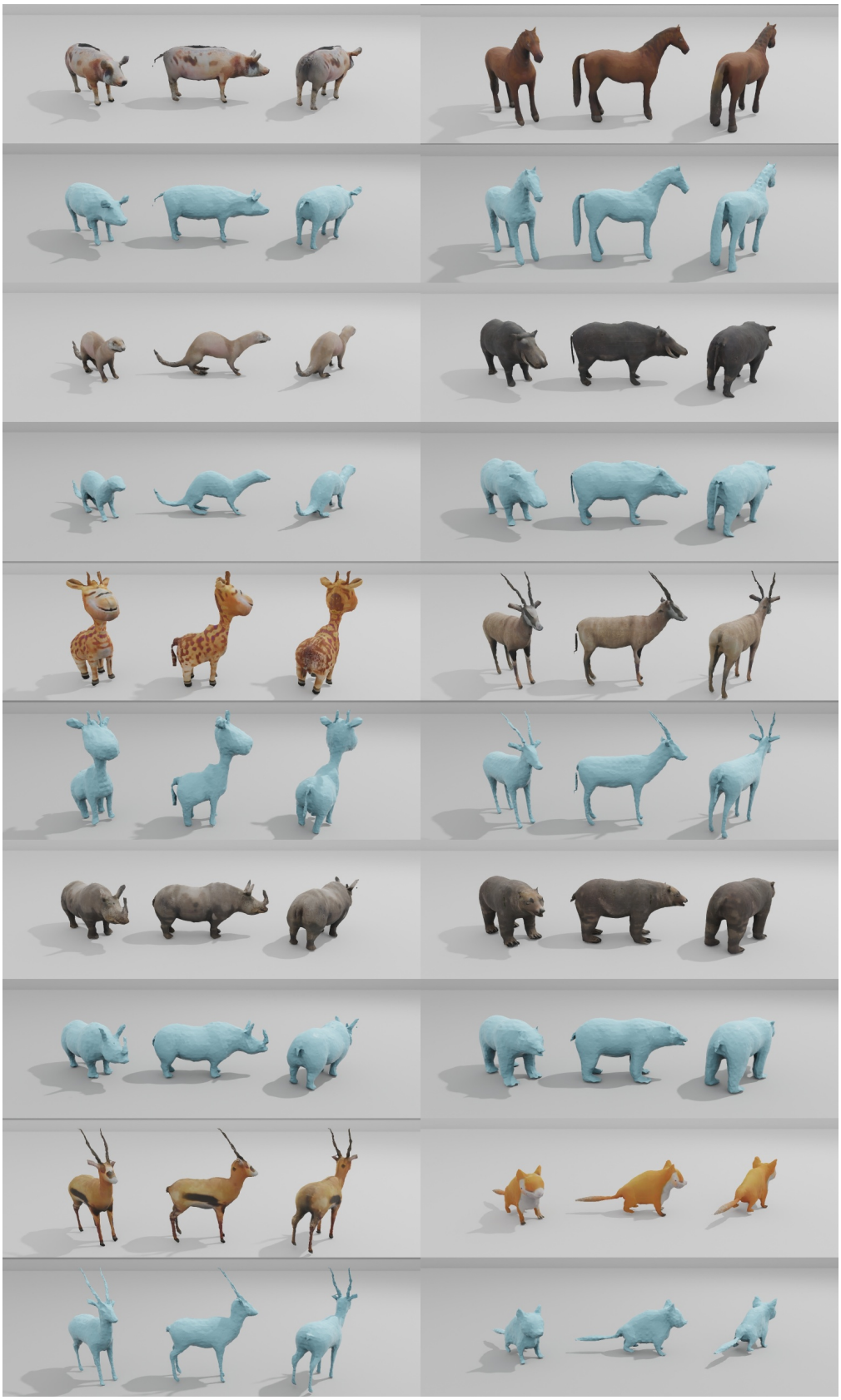


Figure V: Qualitative results on Turbosquid animals.

Figure W: Qualitative results on ShapeNet motorbikes.

Figure X: Qualitative results on Renderpeople.

w/o volume subdivision w/ volume subdivision

Figure Y: We compare results with and without applying volume subdivision on ShapeNet motorbikes. With volume subdivision, our model can generate finer geometric details like handle and steel wire.