# SCORE-BASED GENERATIVE 3D MESH MODELING

**Anonymous authors**
Paper under double-blind review

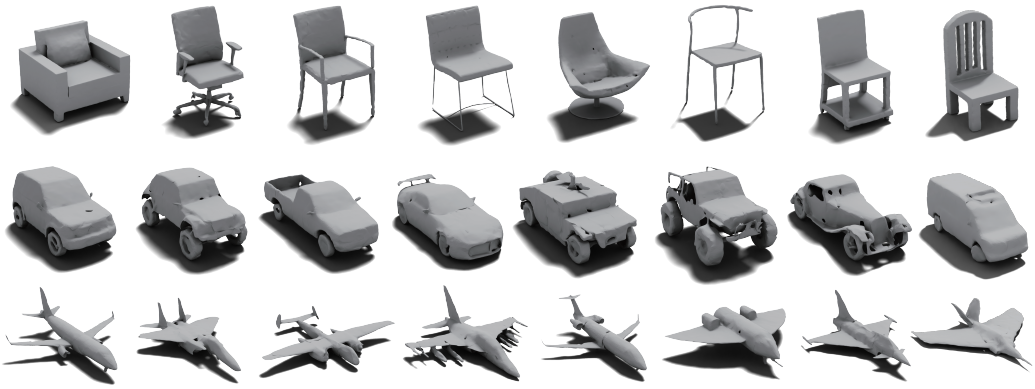


Figure 1: Unconditionally generated 3D mesh samples randomly selected from the proposed *MeshDiffusion*, a simple diffusion model trained on a direct parametrization of 3D meshes without bells and whistles. *MeshDiffusion* produces highly realistic and fine-grained geometric details while being extremely easy and stable to train.

## ABSTRACT

We consider the task of generating realistic 3D shapes, which is useful for a variety of applications such as automatic scene generation and physical simulation. Compared to other 3D representations like voxels and point clouds, meshes are more desirable in practice, because (1) they enable easy and arbitrary manipulation of shapes for relighting and simulation, and (2) they can fully leverage the power of modern graphics pipelines which are mostly optimized for meshes. Existing scalable methods for generating meshes typically rely on sub-optimal post-processing, and they tend to produce overly-smooth or noisy surfaces without fine-grained geometric details. To overcome these shortcomings, we take advantage of the regular graph structure of meshes and use a simple yet very effective generative modeling method to generate 3D meshes. Specifically, we represent meshes in a deformable tetrahedral grid, and then train a diffusion model on this direct parameterization. We demonstrate the effectiveness of our model on multiple generative tasks.

## 1 INTRODUCTION

As one of the most challenging tasks in computer vision and graphics, generative modeling of high-quality 3D shapes is of great significance in many applications such as virtual reality and metaverse [11]. Traditional methods for generative 3D shape modeling are usually built upon representations of voxels [41] or point clouds [1], mostly because ground truth data of these representations are relatively easy to obtain and also convenient to process. Both representations, however, do not produce fine-level surface geometry and therefore cannot be used for photorealistic rendering of shapes of different materials in different lighting conditions. And despite being convenient to process on computers, both voxels and point clouds are relatively hard for artists to edit, especially when the generated 3D shapes are complex and of low quality. Moreover, modern graphics pipelines are built and optimized for explicit geometry representations like meshes, making meshes one of the most desirable final 3D shape representation. While it is still possible to use methods like Poisson reconstruction to obtain surfaces from voxels and points clouds, the resulted surfaces are generally noisy and contain many topological artifacts, even with carefully tuned hyperparameters.

To improve the representation flexibility, sign distance fields (SDFs) have been adopted to model shape surfaces, which enables us to use marching cubes [23] to extract the zero-surfaces and thus 3D

meshes. However, SDFs are typically harder to learn as it requires a carefully designed sampling strategy and regularization. Because SDFs are usually parameterized with multi-layer perceptrons (MLPs) in which a smoothness prior is implicitly embedded, the generated shapes tend to be so smooth that sharp edges and important (and potentially semantic) details are lost. Moreover, SDFs are costly to render and therefore less suitable for downstream tasks like conditional generation with RGB images, which require an efficient differentiable renderer during inference.

We instead aim to generate 3D shapes by directly producing 3D meshes, where surfaces are represented as a graph of triangular or polygon faces. With 3D meshes, all local surface information is completely included in the mesh vertices (along with the vertex connectivity), because the surface normal of any point on the shape surface is simply a nearest neighbor or some local linear combination. Such a regular structure with rich geometric details enables us to better model the data distribution and learn generative models that are more geometry-aware. In light of recent advances in score-based generative modeling [14, 38] where powerful generative performance and effortless training are demonstrated, we propose to train diffusion models on these vertices to generate meshes. However, it is by no means a trivial task and poses two critical problems: (1) the numbers of vertices and faces are indefinite for general object categories, and (2) the underlying topology varies wildly and edges have to be generated at the same time.

A natural solution is to enclose meshes with another structure such that the space of mesh topology and spatial configuration is constrained. One common approach is to discretize the 3D space and encapsulate each mesh in a tiny cell, and it is proven useful in simulation [30] and human surface modeling [29]. Observing that this sort of mesh modeling is viable in recent differentiable geometry modeling literature [26, 31, 34], we instead train diffusion models on a discretized and uniform tetrahedral grid structure which parameterizes a small but representative family of meshes. With such a grid representation, topological change is subsumed into the SDF values and the inputs to the diffusion model now assume a fixed and identical size. More importantly, since SDF values are now independent scalars instead of the outputs of an MLP, the parameterized shapes are no longer biased towards smooth surfaces. Indeed, by such explicit probabilistic modeling we introduce an explicit geometric prior into shape generation, because a score matching loss of diffusion models on the grid vertices has direct and simple correspondence to a loss of the triangular mesh vertex positions.

We demonstrate that our method, dubbed *MeshDiffusion*, is able to produce high-quality meshes and enables conditional generation with a differentiable renderer. MeshDiffusion is also very stable to train without bells and whistles. We validate the superiority of the visual quality of our generated samples qualitatively with different rendered views and quantitatively by proxy metrics. We further conduct ablation studies to show that our design choices are necessary and well suited for the task of 3D mesh generation. Our contributions are summarized below:

- To our knowledge, we are the first to apply diffusion model for unconditionally generating 3D high-quality meshes and to show that diffusion models are well suited for 3D geometry.
- Taking advantage of the deformable tetrahedral grid parameterization of 3D mesh shapes, we propose a simple and effortless way to train a diffusion model to generate 3D meshes.
- We qualitatively and quantitatively demonstrate the supperiority of MeshDiffusion on different tasks, including unconditional generation, conditional generation and interpolation.

## 2 Related Work

**3D Shape Generation**. 3D shape generation is commonly done by using generative models on voxels [41] and point clouds [42], due to their simplicity and accessibility. These resulted (often noisy) voxels or point clouds, however, do not explicitly encode surface information and therefore has to be processed with surface reconstruction methods in order to be used in applications like relighting and simulation. Recent advances in implicit representations [25] lead to a series of generative models on neural fields [5, 6, 18, 33]. Based on these representations, one can learn 3D shapes directly from 2D images [5, 6, 33] with differentiable rendering in the generator. Most of these implicit representations, despite representing shapes in a more photorealistic way, require additional post-processing steps to extract explicit meshes, and are more time-consuming steps to render images. If the underlying topology is known, one may directly generate the vertices of a mesh [24, 39]. [27] extends this approach to the topology-varying case with autoregressive models, sequentially generating vertices and edges, but it is hardly scalable and yields unsatisfactory results on complex geometry.

**Mesh Reconstruction**. To obtain meshes for downstream tasks, generated non-mesh representations have to be processed with some mesh reconstruction method. One of the most popular method is marching cubes [23] which assumes that a surface is represented by the zero level set of some continuous field, and this continuous field can be well approximated by linear interpolation of discrete grid points. It is possible to avoid implicit representations and directly construct meshes. For instance, by assuming the points lie on a surface, one can build triangular meshes by connecting these points, a process known as Delaunay triangulation [3, 10, 13, 17]. It is relatively rare because of its strong assumption on data. With known mesh topology, one can deform a mesh template to fit the given representation. This approach can be easily used to fit multiview images of shapes with known topology (*e.g.*, human faces) using a differentiable renderer [20]. It is recently shown that it is possible to parametrize meshes of varying topology [31, 34] and optimize them using differentiable renderers [26], which our work leverages.

**Score-based Generative Models**. The recent years witness a surge in modeling data distributions with score-based models [14, 37, 38, 38], which parametrizes the logarithm of the gradient of the probability, known as the score function. Different from energy-based models, the intractable normalization constant is avoided and therefore training can be done by score matching. It has been shown that, by using multi-scale models [14, 36] and denoising score matching [40], score-based models can generate very realistic images and perform high-quality image inpainting [14, 32, 38].

## 3 PRELIMINARIES

**Deep marching tetrahedra** (DMTet) [34] is a method to parametrize and optimize meshes of arbitrary topology in a differentiable way. The 3D space is discretized with a deformable tetrahedral grid, in which each vertex possesses a SDF value. The SDF of each 3D position in the space is computed by marching tetrahedra [12], which assumes SDF values to be a barycentric interpolation of the SDF values of the vertices of the enclosing tetrahedra. More specifically, for a point $x_q$ inside a tetrahedron of vertices $x_1, x_2, x_3, x_4$ with SDF values being $s_1, s_2, s_3, s_4$, we obtain its unique barycentric coordinate $(a_1, a_2, a_3, a_4)$ such that $x_q = \sum_i a_i x_i$ ($a_i \in [0, 1]$). The SDF value $s_p$ of $x_q$ is then computed as $s_q = \sum_i a_i s_i$. As a result, if there exists a triangular mesh in the tetrahedron (*i.e.*, $s_1, s_2, s_3, s_4$ are not of the same sign), the triangular mesh is exactly the zero surface in the tetrahedron. The triangular mesh vertex $v_p$ on a tetrahedron edge $(v_a, v_b)$ is $v_p = (v_a s_b - v_b s_a)/(s_b - s_a)$.

The advantage of using DMTet, compared to variants of deep marching cubes [21, 31] which relies on the marching cubes algorithm [23], is that the grids are deformable and thus more able to capture some finer geometric details like very thin handles of chairs. While the cubic grid used for deep marching cubes can be deformable as well, the deformed cubes are worse objects to deal with, compared to tetrahedra which remain tetrahedra after deformation. Notably, DMTet can be fitted with a differentiable renderer by jointly optimizing for the geometry, the texture and the lighting of a 3D shape given multi-view RGB images within reasonable time and memory budget [26].

**Diffusion models** are a set of score-based generative models which learn and infer with the ground-truth time-dependent score function defined by a forward diffusion process [14]. Under the stochastic differential equation (SDE) framework proposed in [38], diffusion models construct the probability of each point in a space by diffusing all data points $x_0 \in \mathcal{D}$ to a target distribution at time $T$ (normally $p_T(x_T) = \mathcal{N}(x_T; 0, I)$) in a time-variant manner: $p_t(x_t|\mathcal{D}) = \mathbb{E}_{x_0 \in \mathcal{D}}[p_t(x_t|x_0)]$, where $p_t(x_t|x_0)$ is modeled by the following Itô process, with $w(t)$ being a standard Wiener process: $dx = F(x, t)dt + G(t)dw$. The training objective is to fit the score function $\nabla_x \log p_t(x_t|x_0)$ with a neural net $s_\theta(x_t, t)$, with $\lambda(t)$ being a scalar weight subject to model design:

$$L(\theta; \mathcal{D}) = \mathop{\mathbb{E}}_{t \in [T], x_0 \in \mathcal{D}} \lambda(t) \left\| s_\theta(x_t, t) - \nabla_x \log p_t(x_t|x_0) \right\|_2^2 . \tag{1}$$

With the trained score function predictor, samples can be generated by solving the reverse SDE [38] with the initial distribution $p_T(x_T)$ using solvers like Euler-Maruyama method [16]:

$$dx = [F(x, t) - G^2(x) \nabla_x \log p_t(x_t|x_0)]dt + G(x)dw. \tag{2}$$

A popular diffusion model DDPM [14], which utilizes a discrete diffusion process $p(x_t|x_{t-1}) = \mathcal{N}(x_t; (1 - \beta_t)x_{t-1}, \beta_t I)$, can be formulated within this SDE framework as [38]: $dx = -\beta_t x dt + \sqrt{\beta_t} dw$. The parameters $\beta_t$ are chosen such that $p_T(x_T)$ is close to a standard Gaussian distribution.

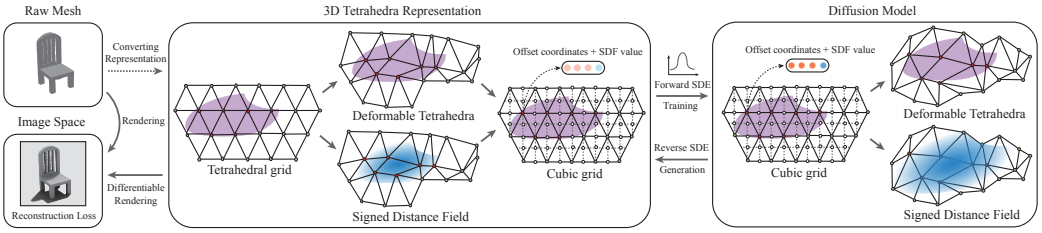



Figure 2: The overview of the proposed MeshDiffusion model.

## 4 *MeshDiffusion*: Diffusion Model on Meshes

With DMTet, 3D meshes can be approximated and parametrized with tetrahedral grids which are regular graphs. Moreover, by initializing the tetrahedral grid using body-centered cubic (BCC) tiling of tetrahedra [19] and treating the deformation of each vertex only as an attribute, we obtain a uniform tetrahedral grid in a 3D space in which translational symmetry is preserved. Such nice properties enable us to train a diffusion model for generating 3D meshes. For now on, we always assume that the undeformed tetrahedral grid is *uniformly* initialized and predict the deformation of tetrahedral grid vertices from their initial positions. As a result, our model takes as input a uniform cubic grid with 4-dimensional attributes (1 dimension for SDF and 3 for deformation).

### 4.1 3D CNN for Deformable Tetrahedral Grids

While it is natural to use graph neural nets (GNNs) for diffusion model on tetrahedral grids, it is better use CNNs which generally have more model capacity than GNNs due to the embedded spatial information in the convolution weights. Specifically, the set of vertex positions of a *uniform* tetrahedral grid, created in the way described in [19], is a subset of the vertex position set of a uniform cubic lattice. A standard voxel-like representation can therefore be easily created by infilling the "missing" sites of a tetrahedral grid (illustrated in Fig. 2). We follow typical practices in diffusion models [38] and use a 3D U-Net for the score function predictor. Details are in the appendix.

### 4.2 Training Objective

Since the tetrahedral grid representation of each object in the datasets is not given, our objective involves the following constrained optimization:

$$\min_{\theta,\phi} \mathop{\mathbb{E}}_{\substack{t\in[T], y_0\in\mathcal{D} \\ x_t\sim p(x_t|x_0=g_{\phi^*}(y_0))}} \lambda(t) \left\| s_\theta(x_t,t) - \nabla_x \log p_t(x_t|g_{\phi^*}(y_0)) \right\|_2^2, \quad \text{s.t. } \phi^* = \arg\min_{\phi} L_{\text{Render}}(y_0, g_\phi(y_0)),$$

in which $g_\phi(y_0)$ is the mapping from 2D views of a 3D shape to its tetrahedral grid representation and $L_{\text{Render}}$ is the rendering (and relevant regularization) loss used in [26].

While in theory it is possible to train $g_\phi$ such that $s_\theta$ can be better learned, we instead take the simplest two-stage and non-amortized approach as detailed optimization and encoder architecture design to use is beyond our scope. Specifically, we first solve for the constraint $L_{\text{Render}}$ and create a dataset $\mathcal{D}_x$ by fitting $x_0$ for each $y_0 \sim \mathcal{D}$ in a non-amortized way. The second stage is simply optimize for the main objective, a $L_2$ denoising loss of diffusion model, on $\mathcal{D}_x$.

In principle, the first data fitting stage can be trained with multiview RGB images by following the procedure in [26]. However, we notice that with RGB images only, it fails to learn some complex geometries, especially when surface materials are highly specular. To demonstrate our idea while not delving too deep into the rendering details, we instead assume that we have access to 3D mesh datasets (in our experiments, ShapeNet datasets [7]) so that RGBD images can be rendered with random lighting and simple surface materials. More specifically, we use a single default material with diffuse components only for all ground truth meshes, and render multiview RGBD images with some known but randomly rotated environment light (represented as a cubemap [35]). With the additional depth information, our reconstruction objective is $L_{\text{Render}} = \alpha_{\text{image}} L_{\text{image}} + \alpha_{\text{depth}} L_{\text{depth}} + \alpha_{\text{chamfer}} L_{\text{chamfer}} + \alpha_{\text{penalty}} L_{\text{penalty}}$, in which $L_{\text{image}}$ and $L_{\text{depth}}$ are the image loss (RGB and silhouette) and the depth loss between the rendered views and the ground truth views, respectively; $L_{\text{chamfer}}$ is the Chamfer distance [1] between a sampled point cloud from the predicted mesh and one from the ground truth mesh; $L_{\text{penalty}}$ is the regularization terms to move reconstructed meshes out from bad local minima. Details of these losses are included in the appendix.

To remove absolute scales from SDFs (Sec. 4.3), we solve the optimization objective twice, where during the second pass we fix the SDFs to be the signs (*i.e.*, $\pm 1$) of the resulted SDF values from the first pass.

### 4.3 Reducing Noise-Sensitivity of Marching Tetrahedra

It is tempting to train diffusion models on arbitrary $\mathcal{D}_x$ with low $L_{\text{Render}}$. However, the naively trained models generate highly non-smooth surfaces (Fig. 3) due to a mismatch between the $L_2$ denoising objective on the 4-dimensional inputs (SDF and deformation) and the ideal $L_2$ reconstruction objective on mesh positions. Recall that the triangular mesh vertex position $v_p$ on a single tetrahedron edge $e = (a, b)$ is computed by linear interpolation $v_p = (v_a s_b - v_b s_a)/(s_b - s_a)$, in which $v_a, v_b$ are the positions of tetrahedron vertices $A$ and $B$, and $s_a, s_b$ are the corresponding SDF values. With perfectly fit $v_a, v_b$ and $s_b$ but a small noise $\epsilon$ on $s_a$ and $s_b$, the triangular mesh vertex is subject to a perturbation inversely proportional to $s_b - s_a$:

$$v_{p,\text{noisy}} = \frac{v_a(s_b + \epsilon) - v_b(s_a + \epsilon)}{(s_b + \epsilon) - (s_a + \epsilon)} = v_p + \frac{v_a - v_b}{s_b - s_a}\epsilon. \tag{3}$$

An arbitrarily fitted dataset $\mathcal{D}_x$ does not provide any guarantee on the scale of $|s_b - s_a|$ and clearly $\|v_{p,\text{noisy}} - v_p\|$ can be unbounded in the worst case, even when the $L_2$ denoising loss is low on $\mathcal{D}_x$.

A similar issue arises when we consider the topological change due to a small noise in the SDF values. For illustration, suppose for some ground truth edge in a tetrahedral grid, $s_a = 1$ and $-C < s_b < 0$ where $C$ is a tiny positive scalar. With a small noise on $s_b$, the $L_2$ denoising loss is small but there can be a high probability of local topological change in the resulted 3D meshes.

Due to these undesired behaviors, we adopt the two-pass optimization scheme described in Sec. 4.2 so that $s_a, s_b \in \{+1, -1\}$ and $|s_b - s_a| = 2$ for all mesh-generating tedrahedron edges.

With discretized SDFs values and continuous deformation vectors, it is natural train a hybrid diffusion model in which the discrete part is modeled like in D3PM [2]. However, we empirically find that it suffices to simply treat the SDF signs (*i.e.*, $\pm 1$) as float numbers and train a standard Gaussian diffusion model. The inference thus requires a final discretization operation to round float SDF values to $\pm 1$ according to their signs.

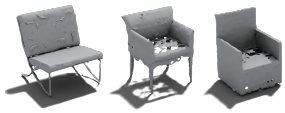


Figure 3: Undesirable artifacts produced by naïve DDPM due to the sensitivity of marching tetrahedra to SDF noises. Better viewed by zooming in.

Because the generative model is trained under the assumption that SDF values are float numbers and will not be clipped to $\pm 1$, a further refinement step can be applied by setting the generated (and discretized) SDFs as conditional inputs and run a conditional generation step on the deformation vectors only.

### 4.4 Conditional Generation with Single-view Images

In many cases, we would like to generate 3D objects given a (possibly partial) single view RGBD image. Such task is simple with our mesh generative model by a two-stage process, first to fit a tetrahedral grid with the given single RGBD view and second to use diffusion models to (stochastically) correct wrong parts. We follow the following procedure shown in Alg. 2. For the conditional inference process, we consider the simplest conditional generation methods [14] (called replacement method in [15]). Specifically, we sample $\hat{x}_{t-1}$ from $x_t$ and replace $\hat{x}^b_{t-1}$ by $p_{t-1}(x^b_{t-1}|x^a_0)$ which is defined by the forward diffusion. More details are included in the appendix.

## 5 Experiments

### 5.1 Settings

**Data Fitting**. We fit tetrahedral grids on five ShapeNet subcategories: Chair, Airplane, Car, Rifle and Table, the same set of categories used in [9, 43]. Fitting each object takes roughly 20-30 minutes on a single Quadro RTX 6000 GPU. We use the same train/test split in [43]. The detailed architecture, training and hyperparameter settings are explained in the appendix.

**Generative Model**. To show the effectiveness and universality of our approach of training diffusion models on 3D meshess, we use an extremely simple 3D CNN architecture for the 3D U-Net in the

**Algorithm 1** Training and Inference

**Training:**

1: Fitting $x'$ for each $y \in \mathcal{D}$, and discretize the SDF values to $\pm 1$, resulting the dataset $(x', y) \in \mathcal{D}'$.
2: Fitting $x$ for each $(x', y) \in \mathcal{D}'$ by conditioning the SDF values on those of $x'$.
3: Train a diffusion model $s_\theta(x, t)$ on $\mathcal{D}'$ by treating the discretized SDF values as float numbers.

**Inference:**

1: Obtain $x_0$ by solving the reverse SDE with initial distribution $q(x_T)$ and discretize the SDF values of $x_0$ to $\pm 1$.
2: (Optionally) Regenerate $x_0$ by conditioning on the previously generated (and discretized) SDF values.

**Algorithm 2** Conditional Generation

1: Randomly initialize and fit a deformable tetrahedral grid using the given RGBD view.
2: Using rasterization, find our all occluded tetrahedra of which we mask all tetrahedral vertices.
3: Re-initialize the unmasked tetrahedral vertices and run the completion algorithm using the trained diffusion model.
4: (Optionally) Finetune the masked tetrahedral vertices as well near the end of the completion process.

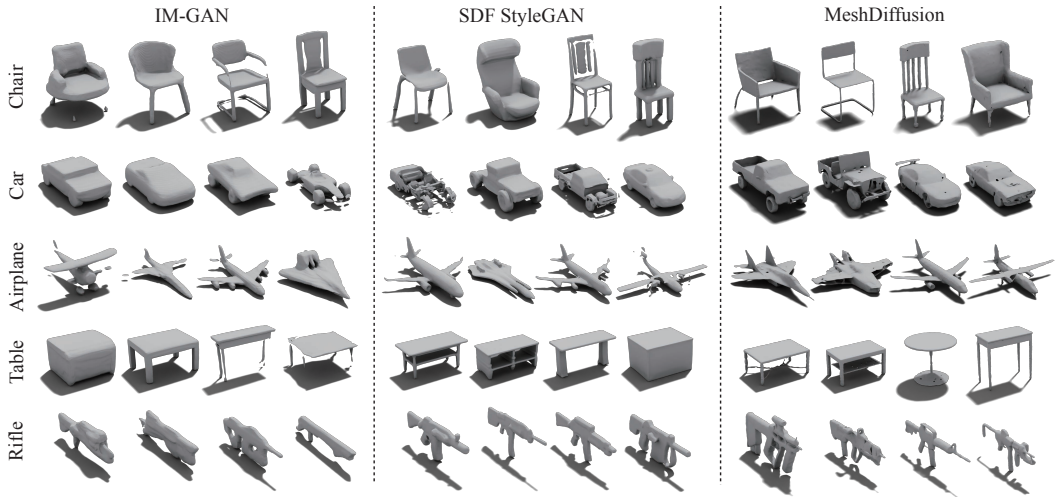



Figure 4: Qualitative model comparison.

diffusion model. Specifically, we adapt the U-Net architecture for DDPM described in [38] by simply replacing the 2D convolution and other 2D modules with their 3D counterparts. To improve model capacity, we provide to the model as a conditional input a mask indicating which lattice sites are artificially augmented. We slightly increase the depths of the networks due to the larger input size. We do not tune the hyperparameters of diffusion SDEs but instead use the same set of hyperparemters as described in [38]. We train discrete-time category-specific diffusion models for all datasets for 90k iterations with batch size 48 on 8 A100-80GB GPUs. The training process takes 2 to 3 days. We use a single 24GB GPU to perform mesh generation with batch size 32.

For ablation on alternative architectures, we train GANs on our datasets with an architecture similar to the one used in [41]. Our SDF-based baselines include IM-GAN [9] and SDF-StyleGAN [43].

### 5.2 Unconditional Generation

We randomly sample shapes from each of our trained generative models for each dataset, and apply the standard Laplace smoothing [28] on the generated meshes (smoothed with $\lambda = 0.25$ with 5 optimization steps). We visualize samples created by our methods and baseline methods in Fig. 4. We note that our method produces the sharpest samples and preserve the most of geometric details, while pure SDF-based methods tend to be too smooth. Part of the reason is that these SDF-based methods assume a very smooth interpolation between points, while our method explicitly model the interpolation in a linear way.

| | Model | MMD (↓) | | | COV (%, ↑) | | | 1-NNA (%, ↓) | | | JSD ($10^{-3}$, ↓) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CD | EMD | LFD | CD | EMD | LFD | CD | EMD | LFD | |
| Chair | IM-GAN | 13.928 | 1.816 | 3615 | **49.64** | 41.96 | **47.79** | 58.59 | 69.05 | 68.58 | 6.298 |
| | SDF-StyleGAN | 15.763 | 1.839 | 3730 | 45.60 | 45.50 | 43.95 | 63.25 | 67.80 | 67.66 | 6.846 |
| | Ours | **13.212** | **1.731** | **3472** | 46.00 | **46.71** | 42.11 | **53.69** | **57.63** | **63.02** | **5.038** |
| Airplane | IM-GAN | 3.736 | 1.110 | 4939 | 44.25 | 37.08 | **45.86** | 79.48 | 82.94 | 79.11 | 21.151 |
| | SDF StyleGAN | 4.558 | 1.180 | 5326 | 40.67 | 32.63 | 38.20 | 85.48 | 87.08 | 84.73 | 26.304 |
| | Ours | **3.612** | **1.042** | **4538** | **47.34** | **42.15** | 45.36 | **66.44** | **76.26** | **67.24** | **11.366** |
| Car | IM-GAN | 5.209 | 1.197 | 2645 | 28.26 | 24.92 | 30.73 | 95.69 | 94.79 | 89.30 | 42.586 |
| | SDF StyleGAN | 5.064 | **1.152** | 2623 | 29.93 | **32.06** | 41.93 | 88.34 | 88.31 | 84.13 | 15.960 |
| | Ours | **4.972** | 1.196 | **2477** | 34.07 | 25.85 | **37.53** | **81.43** | **87.84** | **70.83** | **12.384** |
| Rifle | IM-GAN | 3.550 | 1.058 | 6240 | 46.53 | 37.89 | 42.32 | 70.00 | 72.74 | 69.26 | 25.704 |
| | SDF StyleGAN | 4.100 | 1.069 | 6475 | 46.53 | 40.21 | 41.47 | 73.68 | 73.16 | 76.84 | 33.624 |
| | Ours | **3.124** | **1.018** | **5951** | **52.63** | **42.11** | **48.84** | **57.68** | **67.79** | **55.58** | **19.353** |
| Table | IM-GAN | **11.378** | 1.567 | 3400 | **51.04** | 49.20 | 51.04 | 65.96 | 63.17 | 62.49 | 4.865 |
| | SDF StyleGAN | 13.896 | 1.615 | **3423** | 42.21 | 41.80 | 42.98 | 68.35 | 68.21 | 66.19 | 4.603 |
| | Ours | 11.405 | **1.548** | 3427 | 49.56 | **50.33** | **51.92** | **59.35** | **59.47** | **58.97** | **4.310** |

Table 1: Shape metrics of our model and baseline models.

| Model | Chair | Airplane | Car | Rifle | Table |
|---|---|---|---|---|---|
| IM-GAN [9] | 64.19 | 74.57 | 141.2 | 103.3 | 51.70 |
| SDF-StyleGAN [43] | 36.48 | 65.77 | 128.70 | 65.50 | 42.29 |
| Ours | 39.62 | 64.30 | 190.80 | 54.73 | 48.55 |

Table 2: FID scores averaged across 24 views.

#### 5.2.1 QUANTATITVE METRICS.

In Table 1, we show various point cloud distances [1, 43] and light field distance (LFD) [8] between the test set and the generated samples by sampling point clouds of size 2048 from both ground truth and generated meshes. Our MeshDiffusion better captures the geometric details, and therefore achieve better point cloud metrics in most of the cases. For a more detailed description on the metrics, please refer to the appendix.

We also follow [43] and measures Frechet inception distances (FIDs) [4] on rendered views as a proxy of the goodness of surfaces. Same as in [43], the views are rendered with 24 camera poses uniformly distributed on a sphere; 3 fixed point light sources are used to light the meshes; a gray diffuse material is used for all meshes. The FID scores of each view are computed with an ImageNet pretrained model and averaged to obtain the FID scores for 3D shapes. Our MeshDiffusion is able to achieve competitive scores compared to SDF-StyleGAN. We note, however, the computed FID is not a genuine score for 3D shapes since the distribution of rendered views of these meshes is very different from the distribution of natural RGB images.

Additionally, we perform ablation study on the choice of models in Table. 3. It is clear that the our discretization strategy is beneficial for the diffusion model and our diffusion model is better suited for this setting.

### 5.3 CONDITIONAL GENERATION

Our model can generate meshes by conditioning on a single-view RGBD image, as shown in Fig. 5. Because the geometry estimated from the single-view fitting is not perfect even in the given single

| | MMD (↓) | | | COV (%, ↑) | | | 1-NNA (%, ↓) | | | JSD ($10^{-3}$, ↓) |
|---|---|---|---|---|---|---|---|---|---|---|
| | CD | EMD | LFD | CD | EMD | LFD | CD | EMD | LFD | |
| GAN on Tets | 16.116 | 1.537 | 4173 | 45.13 | 46.39 | 41.74 | 72.97 | 72.57 | 85.91 | 5.353 |
| Ours | **13.212** | **1.731** | 3472 | 46.00 | **46.71** | 42.11 | **53.69** | **57.63** | 63.02 | 5.038 |
| Ours, Unsmoothed | 13.885 | 1.772 | 3446 | 43.36 | 45.50 | 43.36 | 60.88 | 62.54 | 62.32 | **4.716** |
| Ours, No Discretization | 14.324 | 1.816 | 3690 | 44.76 | 46.61 | 44.69 | 63.94 | 65.01 | 65.34 | 5.178 |

Table 3: Ablation on the Chair category in ShapeNet.

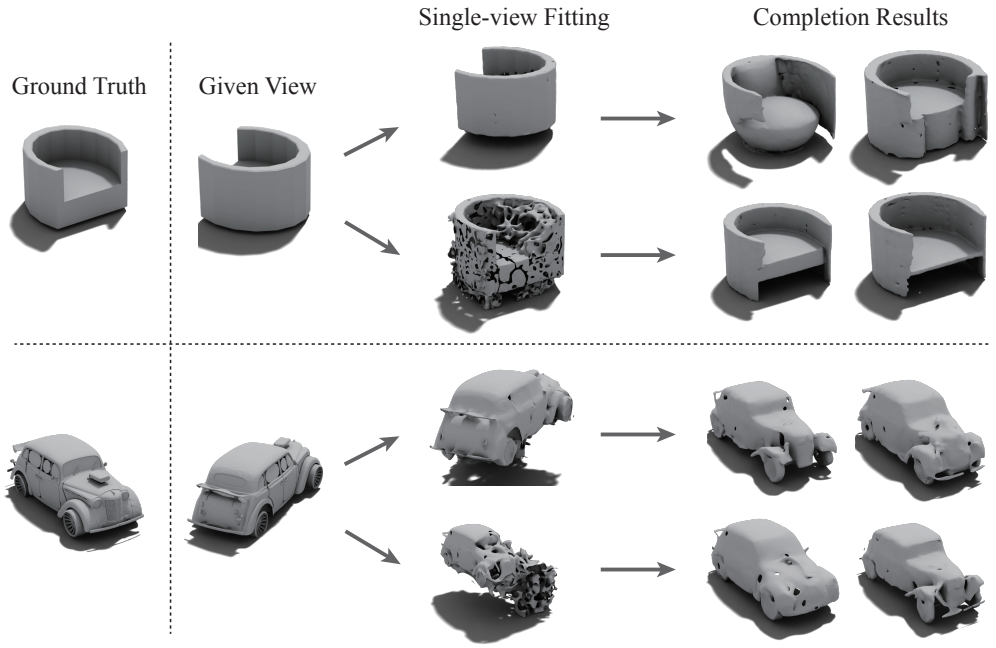



Figure 5: Mesh generation conditioning on a single RGBD view.

view, we allow the originally-fixed tetrahedral vertices to be updated by the diffusion model near the end of the diffusion completion process (in our experiments, $T = 50$; the inference process starts at $T = 1000$ and ends at $T = 0$).

### 5.4 Interpolation

We use DDIM [36] to convert the stochastic sampling process into a deterministic one. With DDIM, the initial random Gaussian noise $x_T$ is treated as the "latent code" of the generated image, and we run the following sampling process:

$$x_{t-1} = \alpha_{t-1} \frac{x_t - (1 - \alpha_t) s_\theta(x_t, t)}{\alpha_t} + (1 - \alpha_{t-1}) s_\theta(x_t, t). \tag{4}$$

Following the settings in DDIM paper [36], we use spherical interpolation for the latent codes. We visualize some of the interpolation sequence in Fig. 6. We set the number of inference steps to 100 for faster inference and use quadratic time spacing as described in [36].

## 6 Discussion

**Optimization issues with DMTet.** While DMTet is capable of fitting geometries, it fails in cases where the underlying topology is complex. Besides, it is not encouraged to learn the true topology of target shapes and may produce invisible topological holes by contracting the neighboring triangular mesh vertices close enough. Furthermore, we observe that the optimization process with differentiable rasterization of 3D meshes may produce floating and isolated meshes, especially when depth supervision is introduced. It is therefore worth designing better optimization techniques, regularization methods and possibly better parametrization of meshes for the purpose of training mesh diffusion models.

**Diffusion model design.** Our experiments demonstrate the effectiveness of the simple design with few parameter changes: a 3D-CNN-based U-Net on augmented cubic grids with DDPM. We note that it is straightforward to utilize advanced diffusion models and switch to more memory-efficient architectures. Especially, since it is possible to regularize SDFs with methods such as Lipschitz constraints [22], it is a promising approach to train diffusion models on the latent space produced by a regularized autoencoder, the same strategy adopted in [32].

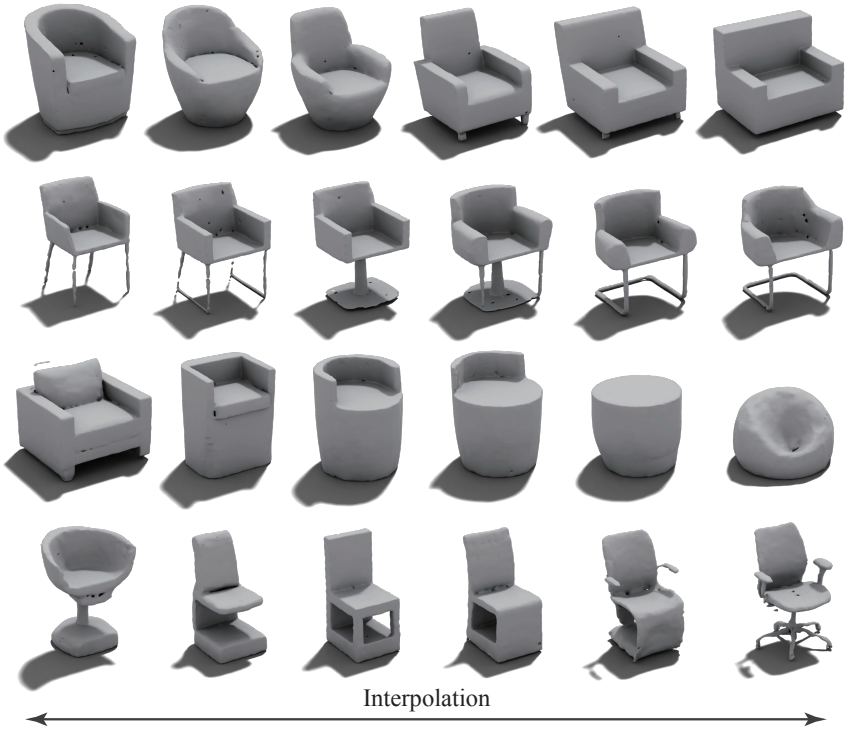



Figure 6: Some examples of our interpolation results.

**Limitations.** Diffusion model typically assumes a known dataset in the input modality (augmented tetrahedral grids in our case), but to efficiently train diffusion models on 2D images, we need a better way to amortize the costs and fully leverage the power of differentiable rendering. Our paper avoids this important aspect but instead adopts the two-stage approach of "reconstruction-then-generation". Moreover, in our formulation, the differentiable renderer is useful only in obtaining the tetrahedral grid dataset, while in principle we believe there can be ways to incorporate the differentiable render in the training and inference process of diffusion models. Finally, our diffusion model is built with very naïve architecture, thus limiting the resolution of input tetrahedral grids. However, we notice that some of the fine details cannot be fully captured with the current resolution of 64 during the data fitting stage. With better architecture design or adaptive resolution techniques (as in [34]), we may greatly increase the resolution and general a more diverse set of fine-level geometric details.

## 7 Conclusion

We demonstrate that our *MeshDiffusion*, a diffusion model on 3D meshes parametrized by tetrahedral grids, is able to generate fine-details of 3D shapes with arbitrary topology. Such model is minimally designed but still outperforms the baselines. We demonstrate that, despite being trained and performing inference in the 3D space, can be easily used when only 2.5D information is available. We believe that such a model can potentially shed some light on future studies of score-based models for learning and generating photorealistic shapes.

## REFERENCES


[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, 2018.

[2] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, 2021.

[3] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics (TOG)*, 3(4):266–286, 1984.

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

[5] 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.

[6] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021.

[7] 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.

[8] 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.

[9] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019.

[10] Tamal K Dey, Joachim Giesen, and James Hudson. Delaunay based shape reconstruction from large data. In *PVG*, 2001.

[11] John David N Dionisio, William G Burns III, and Richard Gilbert. 3d virtual worlds and the metaverse: Current status and future possibilities. *ACM Computing Surveys (CSUR)*, 45(3):1–38, 2013.

[12] 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.

[13] Meenakshisundaram Gopi, Shankar Krishnan, and Cláudio T Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. In *Computer Graphics Forum*, volume 19, pages 467–478, 2000.

[14] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

[15] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv preprint arXiv:2204.03458*, 2022.

[16] P.E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Springer Berlin Heidelberg, 2011.

[17] Ravikrishna Kolluri, Jonathan Richard Shewchuk, and James F O'Brien. Spectral surface reconstruction from noisy point clouds. In *SGP*, 2004.

[18] Adam R Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Sona Mokrá, and Danilo Jimenez Rezende. Nerf-vae: A geometry aware 3d scene generative model. In *International Conference on Machine Learning*, pages 5742–5752. PMLR, 2021.

[19] François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. In *SIGGRAPH*. 2007.

[20] Tianye Li, Shichen Liu, Timo Bolkart, Jiayi Liu, Hao Li, and Yajie Zhao. Topologically consistent multi-view face inference using volumetric sampling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3824–3834, 2021.

[21] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *CVPR*, 2018.

[22] Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. Learning smooth neural functions via lipschitz regularization, 2022.

[23] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, 1987.

[24] Qianli Ma, Jinlong Yang, Anurag Ranjan, Sergi Pujades, Gerard Pons-Moll, Siyu Tang, and Michael J Black. Learning to dress 3d people in generative clothing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6469–6478, 2020.

[25] 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 *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, pages 405–421, 2020.

[26] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Mueller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. *arXiv:2111.12503*, 2021.

[27] Charlie Nash, Yaroslav Ganin, SM Ali Eslami, and Peter Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International conference on machine learning*, pages 7220–7229. PMLR, 2020.

[28] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Laplacian mesh optimization. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 381–389, 2006.

[29] Hayato Onizuka, Zehra Hayirci, Diego Thomas, Akihiro Sugimoto, Hideaki Uchiyama, and Rin-ichiro Taniguchi. Tetratsdf: 3d human reconstruction from a single image with a tetrahedral outer shell. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6011–6020, 2020.

[30] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2020.

[31] Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoît Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. Meshsdf: Differentiable iso-surface extraction. In *NeurIPS*, 2020.

[32] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[33] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. In *NeurIPS*, 2020.

[34] 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 *NeurIPS*, 2021.

[35] Peter Shirley, Michael Ashikhmin, and Steve Marschner. *Fundamentals of computer graphics*. AK Peters/CRC Press, 2009.

[36] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.

[37] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in Artificial Intelligence*, pages 574–584. PMLR, 2020.

[38] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2021.

[39] Qingyang Tan, Lin Gao, Yu-Kun Lai, and Shihong Xia. Variational autoencoders for deforming 3d mesh models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5841–5850, 2018.

[40] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

[41] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NIPS*, 2016.

[42] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *ICCV*, 2019.

[43] Xin-Yang Zheng, Yang Liu, Peng-Shuai Wang, and Xin Tong. Sdf-stylegan: Implicit sdf-based stylegan for 3d shape generation. In *SGP*, 2022.

# A DETAILS ON DATASET PREPARATION

## A.1 ARCHITECTURE AND LOSSES

**RGB and silhouette loss.** By interpolating between a black background image and the binary mask (produced by rasterization to indicate the existence of rasterized meshes on each pixel), we obtain a smoothed binary mask $M$ and $M_{\text{gt}}$, for the fitted meshes and predicted meshes respectively. We then compute the silhouette loss:

$$L_{\text{silhouette}} = \|M - M_{\text{gt}}\|_2^2 .$$

We compute the averaged pixel-wise log-$L_1$ loss between rendered images and ground truth images:

$$L_{\text{RGB}} = \mathop{\mathbb{E}}_{x \in \mathcal{D}, Q \in \mathcal{D}_{\text{pose}}} \log(\|I_{\text{gt}} \odot M_{\text{gt}} - \text{Render}(x, Q) \odot M\|)$$

in which $\mathcal{D}_{\text{pose}}$ is the set of random camera poses which always look at the center of the object, and $x$ is the sampled tetrahedral grid from the dataset $\mathcal{D}$.

**Depth loss.** We use a mixed loss for depth: when the depth is greater than a threshold (set to $1.0$), we use a $L_2$ loss; otherwise, we switch to $L_1$ loss as we find it producing much better surfaces.

$$L_{\text{depth}} = \mathop{\mathbb{E}}_{x \in \mathcal{D}, Q \in \mathcal{D}_{\text{pose}}} \|d_{\text{gt}} - \text{Render}_{\text{depth}}(x, Q)\|_{\text{mixed}} .$$

The depth images are computed by barycentric interpolation following rasterization, in which the background is assumed to have a large default depth of $20$, which is much greater than the object depth (constrained in $[-1, 1]$).

**Chamfer loss.** At every iteration, we randomly sample $50,000$ points from both ground truth meshes and predicted meshes and compute the standard Chamfer distance:

$$L_{\text{Chamfer}} = \sum_{x \in P_{\text{fitted}}} \min_{y \in P_{\text{gt}}} \|x - y\|_2^2 + \sum_{y \in P_{\text{gt}}} \min_{x \in P_{\text{fitted}}} \|x - y\|_2^2$$

**SDF regularization loss.** Following [26], we use the same $L_2$ regularization loss and penalize the difference of SDFs of two neighboring tetrahedral vertices (*i.e.*, $E_{tet}$ the set of all edges in the tetrahedral grid) as in [26] so that the occluded regions do not produce complex inner geometry. It is also helpful for optimizing tetrahedral grids.

$$L_{\text{SDF}} = \sum_{(u,v) \in E_{tet}} \|\text{SDF}(u) - \text{SDF}(v)\|^2$$

**Hyperparameter setting.** We set $\alpha_{\text{image}} = \alpha_{\text{Chamfer}} = 1.0$ and $\alpha_{\text{depth}} = 100.0$. We set $\alpha_{\text{SDF}}$ to $0.2$ and use a linear decay of the scale towards $0.01$. We use an Adam optimizer for all the parameters with a learning rate of $5e-4$ and $(\beta_1, \beta_2) = (0.9, 0.999)$. We train both reconstruction passes with $5000$ iterations.

## A.2 IMPLMENTATION DETAILS

The initial tetrahedral grid is initialized in a cube $[-1, 1]^3$ by a dense body-centered cubic (BCC) tiling of tetrahedra [19]. As some tetrahedral generation packages produce additional tetrahedral vertices on the boundary of the cube which breaks translational symmetry (hence detrimental to the performance of 3D CNN), we remove such symmetry-breaking boundaries if any.

We find that using the clipped deformation vectors is better than using the *tanh*-ed deformation vectors as they remove some nonlinearity. And to ensure that the deformation vectors are always differentiable, we clip the deformation vectors only after each gradient update, but not during the forward computational pass.

| Input |
| --- |
| $2 \times$ ResBlocks, $3 \times 3$ kernel, width 64 |
| Pooling, Stride 2 |
| $3 \times$ ResBlocks, $3 \times 3$ kernel, width 128 |
| Pooling, Stride 2 |
| $3 \times$ ResBlocks, $3 \times 3$ kernel, width 128 |
| Pooling, Stride 2 |
| $3 \times$ ResBlocks, $3 \times 3$ kernel, width 256 |
| Pooling, Stride 2 |
| $3 \times$ ResBlocks, $3 \times 3$ kernel, width 256 |

Table 4: Architecture for the encoder part of the 3D U-Net. The shortcuts to the decoder part are not shown.

We set the range of deformation to be three times of that used in [34] so that the meshes can better capture details in the absence of SDF scales (Sec. 4.3), and we do not observe many pathological artifacts with this setting.

Because we discretize SDFs after the first reconstruction pass, it is more desirable for the underlying shape to be captured mostly by the topology implied by SDF values, not the vertex deformation. To encourage convergence to such solutions, during the first $2000$ iterations, we periodically scale the learned SDF values by $0.4$ for every $300$ iterations.

We observe that depth supervision is not enough to remove some isolated floater meshes in the space. Therefore, we cull these floaters by setting the SDF values of all tetrahedral vertices lying outside the visual hull to positive values (with the assumption that positive values represent ouside-ness). Due to constraints on computational resources, we do not perform culling in a full 3D way, but use rendered shape silhouette to determine floaters visible in the rendered views only.

We notice that some of the meshes in ShapeNet are not watertight while some others have parts which are very thin. Depth supervision on these parts can lead to more topological holes. Therefore, during rasterization we extract the depths (stored in Z-buffers) of the two closet meshes for each pixel and decide if we compute the depth loss on these pixels according to the difference in depth of these two meshes. Specially, depth supervision is not included if the difference is too small or there is only one rasterized mesh (meaning that the ray only intersects with a single mesh).

Motivated by the fact that only SDFs of the mesh-generating tetrahedra (*i.e.*, tetrahedra with vertices of different SDF signs) matter, we set the SDFs of all the non-mesh-generating tetrahedral vertices to $\pm 1$ (signs depending on if vertices inside/outside the shape) to reduce the complexity of the dataset.

For the single-view reconstruction stage in the conditional generation experiments, we observe that the lack of other views leads to many floating artifacts when the target shape consists of a bulk volume. Therefore, during the first $300$ iterations, we periodically (every $10$ iterations) query the nearest neighbor ground truth mesh vertices for every tetrahedral vertex, and determine if the vertex is in front of the camera and not behind the current view (by computing the nearest surface normal, the displacement from the nearest face center and the view direction from camera). If the tetrahedral vertex satisfies the condition, we set its SDF value to $+1$.

## B Details on Diffusion Model

We adapt the network of DDPM in [14] and use a base width of $64$. The encoder in the U-Net is shown in Fig. 4, and the decoder follows the same but reverse pattern.

All the values at the artificially-introduced sites of a cubic lattice are set to zero. We append to the cubic grid in the first and last few layers a binary mask that indicates which vertices are from the tetrahedral grid and which are fake. The score matching loss is also masked accordingly so that the augmented vertices in the predicted cubic grids do not contribute.

To prevent overfitting, we augment the dataset by randomly translate all tetrahedron vertices by the same but tiny amount.

## C Quantative Metrics for 3D Meshes

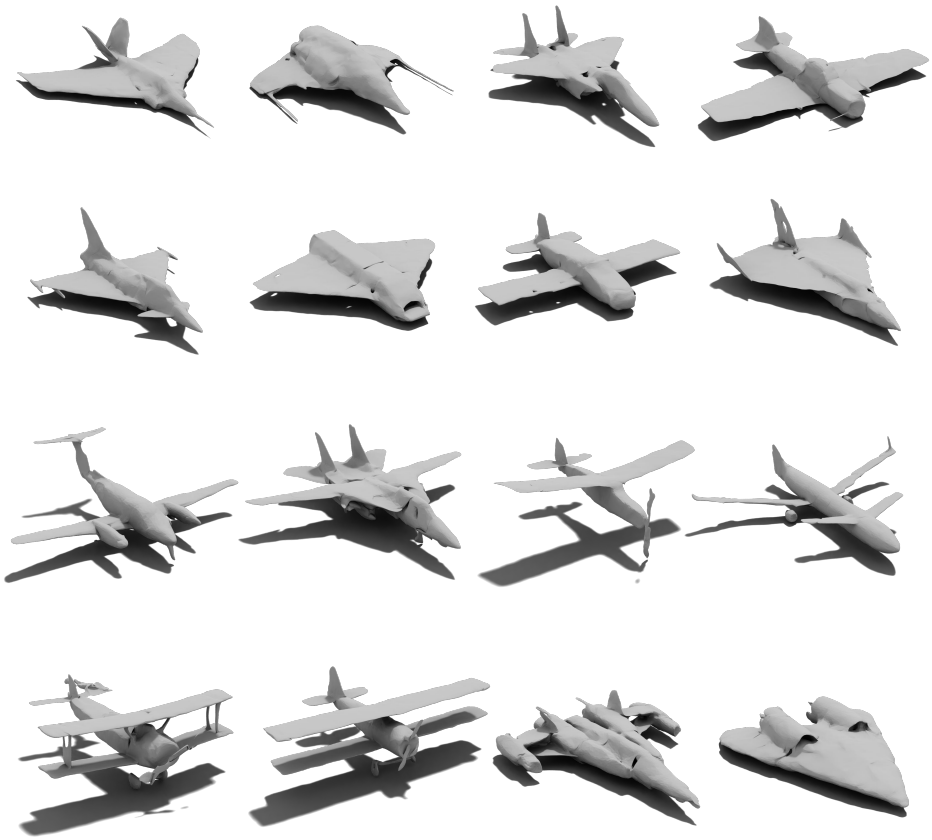
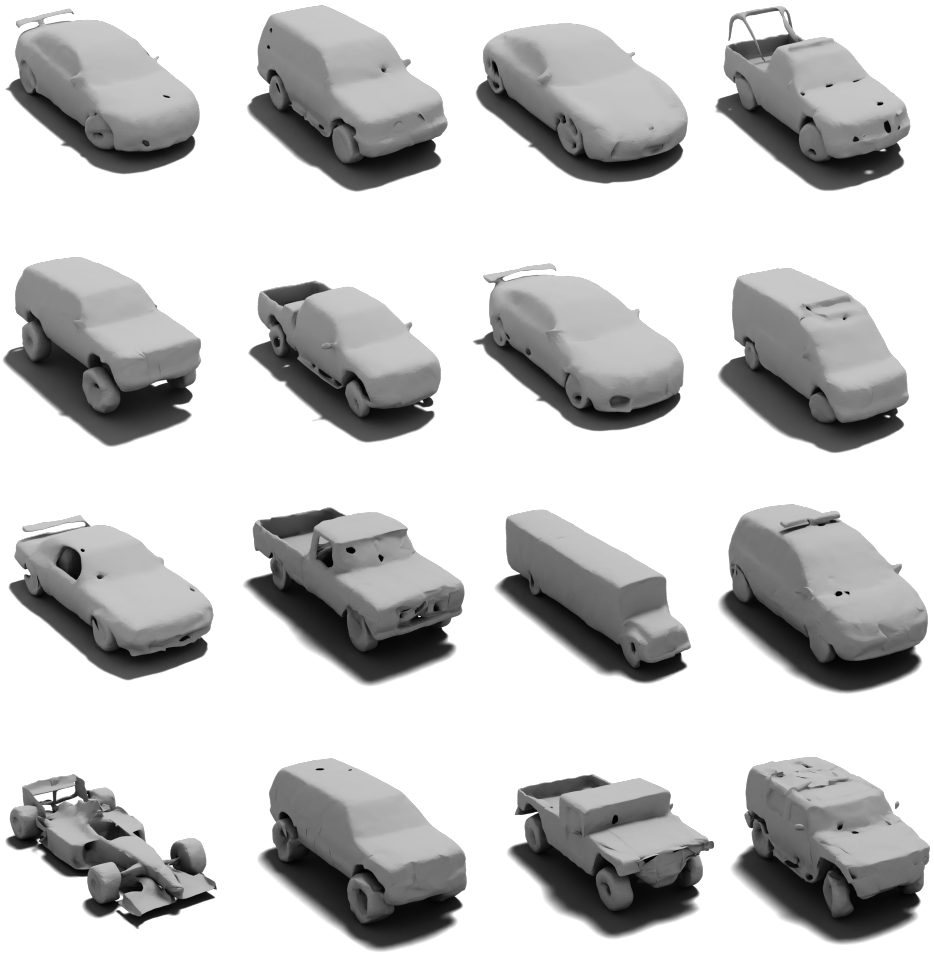
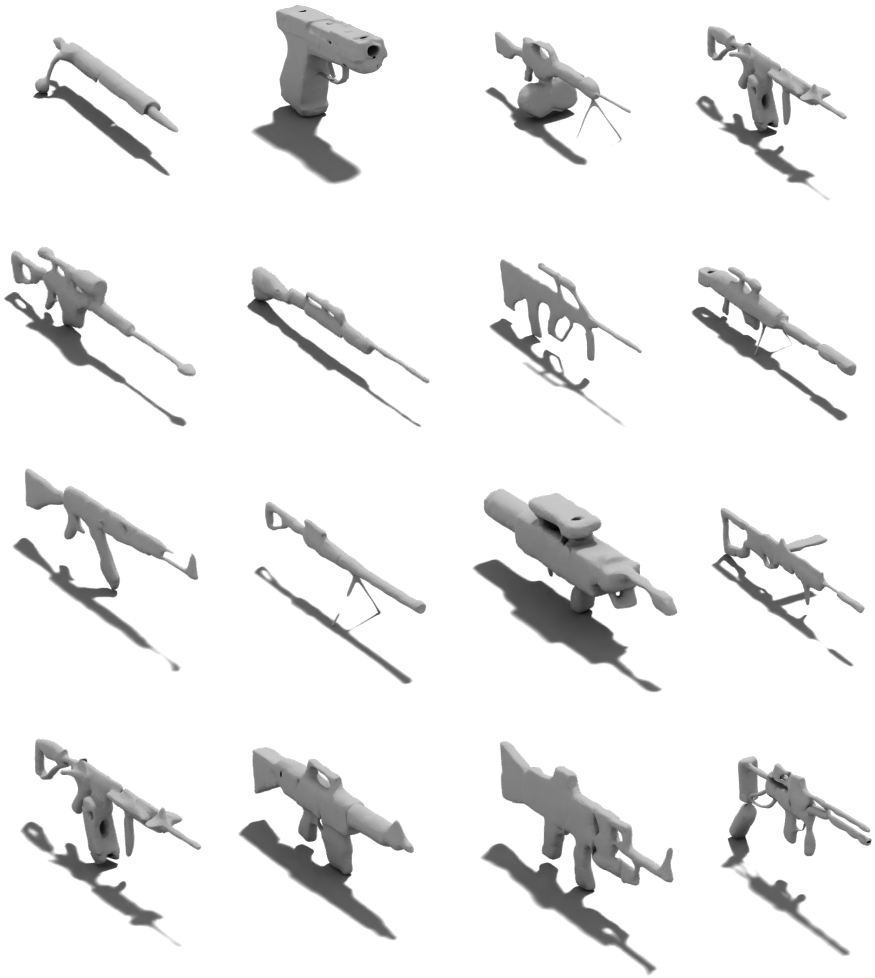
We briefly explain the metrics we used for evaluating the quality of 3D Meshes.

**Minimum Matching Distance (MMD).** MMD measures the average distance to the nearest neighbor of individual points in one point cloud to another point cloud.

**Coverage.** Suppose every point cloud in a set $A$ is approximated by its nearest neighbor in another set $B$. Coverage measures the fraction of elements in $B$ which are used to cover $A$ in the nearest neighbor matching sense. Higher the coverage, Better that $A$ is a representative set of $B$.

**Leave-one-out Accuracy (1-NNA).** As its name indicates, 1-NNA is the average leave-one-out accuracy of the 1-NN classifier fitted on $A$ to classify $B$. It measures if each element in the set $A$ is important in representing another set $B$. A low 1-NNA score means that $A$ well covers $B$.

**Jensen-Shannon Divergence (JSD).** With JSD, we compute the distance between distribution of ground truth point clouds and that of the generated point clouds: $\text{JSD}(P_A, P_B) = \frac{1}{2}\text{KL}(P_A||M) + \frac{1}{2}\text{KL}(P_B||M)$, in which $M = \frac{1}{2}(P_A + P_B)$.

**Light Field Distance (LFD).** LFD leverages the so-called light field descriptor [8], which combines local region-based and contour-based descriptors to measure the silhouettes and thus object shapes.

## D MORE UNCONDITIONAL GENERATION SAMPLES