# SCORE-BASED GENERATIVE 3D MESH MODELING WITH DEFORMABLE TETRAHEDRA

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

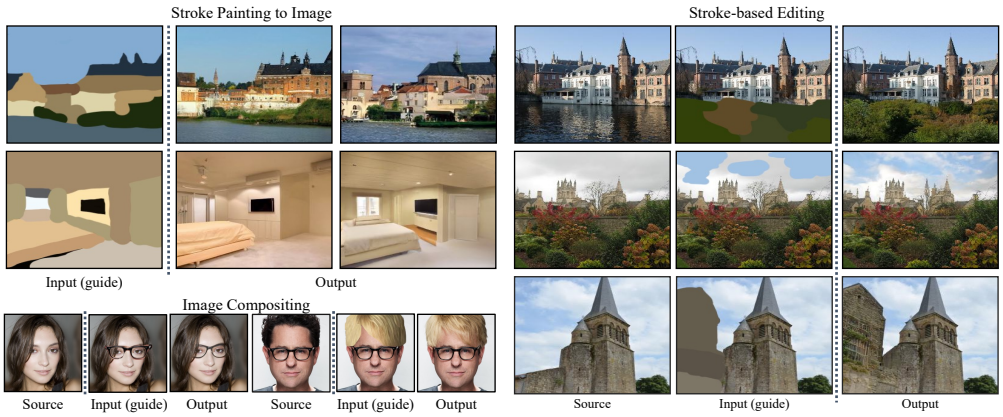


Figure 1: Placeholder for teaser

## 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 address this issue, we propose a novel approach to directly generate 3D meshes for general objects. Specifically, we take advantage of a deformable tetrahedral grid representation and then learn a diffusion model on top of it to generate high-quality 3D meshes. To our knowledge, this is one of the first unconditional generative models which produces a direct parametrization of 3D meshes of arbitrary topology in a scalable way. We demonstrate, both qualitatively and quantitatively, 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 [8]. Traditional methods for generative 3D shape modeling are usually built upon representations of voxels [29] 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. 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 [19] to extract the zero-surfaces and thus 3D meshes. However, SDFs are typically hard to learn as it requires a carefully designed sampling strategy and regularization. As SDFs are usually parametrized 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.

Another issue with SDF-based surface generation methods is the lack of a clear training objective and a suitable inductive bias. [Weiyang: not sure whether this point is well grounded] Because SDFs (parametrized by MLPs) are typically continuous and smooth, a suitable architecture and the corresponding training objective should be able to capture the surface reconstructed at every point in $\mathbb{R}^3$. As a result, complicated sampling strategies, feature designs and/or surface regularization methods are necessary. [Weiyang: Do you want to say things about SDF's property of interpolation between points?]

We instead aim to generate 3D shapes by directly producing 3D meshes, by which 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, because the surface normal of any point on the shape surface is simply a nearest neighbor or some local linear combination. Such structure thus enables us to design models which are more geometry-aware. However, it poses two problems: first, the numbers of vertices and faces are indefinite for general object categories, and second, the underlying topology varies wildly.

Inspired by recent papers on deformable tetrahedral grids [22, 24] – a hybrid representation between SDFs and meshes, we propose to train a diffusion model on this hybrid representation. More specifically, by assuming the SDFs of a point in the 3D space are linear combinations of those of the vertices of the enclosing tetrahedron (known as *marching tetrahedra* algorithm), we can explicitly parametrize a family of 3D meshes with a deformable tetrahedral grid. Due to the regular structure of (uniform) tetrahedral grids, we are able to train a diffusion model with this representation, in which a $L_2$ loss of tetrahedral vertex attributes directly corresponds to a loss of the triangular mesh vertex positions. With such an approach, we solve the above problems of mesh representation. Another benefit of our approach is that with the explicit probablistic modelling using the $L_2$ denoising objective of diffusion models, we introduce an explicit geometric prior into shape generation. Furthermore, with a differentiable rasterizer, such a representation enables efficient differentiable inverse graphics from RGB images and therefore efficient model training and conditional generation.

We demonstrate that our method, dubbed *Mesh Diffusion*, is able to produce high-quality meshes in a memory-efficient way and enables conditional generation with a differentable rasterizer. We validate the visual quality of our generated samples qualitatively with different rendered views and quantitatively by proxy metrics such as FID. We further conduct ablation studies to show that our design choices are necessary and best suited for the task of 3D mesh generation.

In summary, our contributions are:

- To our knowledge, we are the first to unconditionally generate a direct parametrization of 3D meshes of high quality in a scalable way.
- We show that a diffusion model, one of the most promising and powerful generative models, is well suited for 3D geometry generation with deformable tetrahedral grids.
- We create a dataset of shapes parametrized by deformable tetrahedral grids for future work on generative models.
- We qualitatively and quantitatively evaluate our generative model on different tasks, including unconditional generation, conditional generation and interpolation, and show that geometric details can be generated by our model.

# 2 RELATED WORK

## 2.1 3D SHAPE GENERATION

3D shape generation or synthesis has a long history in computer vision and graphics. The most classical way is to template-based methods by retrieving a library of parts and geometries and perform shape assembly. [Zhen: cite papers; see related work in https://proceedings.neurips.cc/paper/2016/file/44f683a84163b3523afe57c2e008bc8c-Paper.pdf]

Among all 3D representations, Voxel is the earliest to be explored in template-free 3D shape generation due to its simplicity []. A GAN, VAE or some combination is typically used [] in a way similar to 2D image generation but by replacing 2D convolutions to 3D ones. However, voxels do not model geometry well given reasonable memory constraint.

Point clouds, on the other hand, directly encode the continuous coordinates. Typically, variants of the PointNet architecture [] are used to build a generative model on point clouds using GANs [2, 28], VAEs [], normalizing flows [30] and diffusion models [20].

Recent papers have been exploring the use of neural rendering [] in 3D shape generation. By using a differentiable renderer in the decoder, one can train a GAN on 2D images with an intermediate 3D representation as a by-product. For instance, a generator may produce a 3D volume of densities and RGB values from which one may produces 3D-consistent novel views with neural volume rendering techiniques [].

Similarly, some papers explore 3D shape generation through synthesizing signed distance fields (SDFs). By encoding the SDFs with a grid [] or latent codes using neural fields [], they manage to generate meshes by using algorithms like marching cubes [19] as a post-processing step. These methods however do not take this post-processing step into end-to-end training and generate overly-smooth 3D shapes.

Few papers attempt to generate meshes in an autoregressive manner. For instance, in [] a policy is trained to generate meshes by iteratively adding new vertices and edges. This method, however, is unlikely to scale to meshes with a moderate number of vertices and relatively fine details.

## 2.2 MESH RECONSTRUCTION

**Generic methods.** The process of constructing meshes, typically called mesh reconstruction or generation (not to be confused with mesh generative models), is mostly done with variants of these two approaches: marching cubes [19] and Delaunay triangulation [4, 7, 10, 15]. In marching cube, the algorithm discretizes the input SDF using a cubic grid and enumerate all cubes with vertices of different SDF signs. By assuming that the SDF values in a cube are trilinear interpolation of the SDF values of the 8 vertices, the isosurfaces (*i.e.*, the zero levelset) can be easily computed and represented as triangular meshes. In Delaunay triangulation, we assume all given points lie on the surface and construct a triangular mesh such that these points are the vertices in the mesh and satisfy some desired properties.

**Implicit surface reconstruction.** To use marching-cube-like methods on voxels or point clouds, one needs to first convert them into implicit representation of surfaces. While for voxels one could take the occupancy [21] (whether one cube is occupied or not) as the input to the marching cubes algorithm, it is harder to do the same for point clouds because their very discrete nature. One way to reconstruct implicit surface from point clouds is by the so-called Poisson reconstruction [13]: knowing all points lie on the surface, one can connect the indicator function (*i.e.*, indicating inside or outside) with these points via a Poisson equation, and solving this PDE yields an implicit surface. An alternative is that one assumes that each point in a point cloud possess a "radius" such that a volume-less point becomes a sphere with volume, from which one may construct implicit surfaces [17].

**Inverse graphics from multiview images.** Besides the aforementioned methods, one can leverage differentiable methods to reconstruct meshes from multiview images. It is commonly done by 1) morphing a mesh template with a learned network [] or 2) optimization with a differentiable renderer [].

# 3 PRELIMINARIES

## 3.1 DEEP MARCHING TETRAHEDRA

Deep marching tetrahedra (DMT) [24] 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 [9], 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 $x_q = \sum_i a_i x_i$ ($a_i \in [0, 1]$). The SDF value $s_p$ of $x_q$ is 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 position of $v_p$ on a tetrahedron edge $(v_A, v_B)$ is thus [Liam: $v_A$ vs $v_a$ be consisted]

$$v_p = \frac{v_a s_b - v_b s_a}{s_b - s_a}.$$

The advantage of using DMT, compared to variants of deep marching cubes [18, 23] which relies on the marching cubes algorithm [19], is that the grids are deformable and thus 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 (*i.e.*, parallelepipeds) are worse objects to deal with compared to tetrahedra, which remain as tetrahedra after deformation.

Note that in [24], DMT is used for a "conditional generative model" with given ground truth 3D point clouds; the adversarial loss is only used for detail refinement instead of obtaining a learned distribution to sample from in the typical definition of generative models. [Zhen: be careful on this point. Can be inaccurate and/or offensive.]

Notably, DMT 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 [22].

## 3.2 DIFFUSION MODEL

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 [11]. Under the stochastic differential equation (SDE) framework proposed in [27], 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:

$$p_t(x_t|\mathcal{D}) = \mathop{\mathbb{E}}_{x_0 \in \mathcal{D}}[p_t(x_t|x_0)],$$

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:

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

With the trained score function predictor, samples can be generated by solving the corresponding reverse SDE [27] with the initial distribution $p_T(x_T)$ using solvers like Euler-Maruyama method [14]:

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

A popular diffusion model called DDPM [11], 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 [27]:

$$dx = -\beta_t x dt + \sqrt{\beta_t} dw.$$

The parameters $\beta_t$ are chosen such that $p_T(x_T)$ is close to a standard Gaussian distribution.

# 4 Mesh Diffusion: Diffusion Model on Deformable Tetrahedra

## 4.1 3D CNN for Deformable Tetrahedra

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 [16], 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. **??**). To this end, 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). We follow typical practices in diffusion models [27] and use a 3D U-Net for the score function predictor. For details, please refer to the appendix.

[Zhen: A figure here to show how the "infilling" is completed]

## 4.2 Training Objective

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

$$L(\theta, \phi; \mathcal{D}) = \mathop{\mathbb{E}}_{\substack{t\in[T], y_0\in\mathcal{D} \\ x_t\sim p(x_t|x_0=g_\phi(y_0))}} \lambda(t) \left\| s_\theta(x_t, t) - \nabla_x \log p_t(x_t|g_\phi(y_0)) \right\|_2^2 \quad s.t. \quad L_{\text{Render}}(y_0, g_\phi(y_0)) = 0. \tag{1}$$

, 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 [22].

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 [22]. 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 [5]) 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 [25]). With the additional depth information, our reconstruction objective is

$$L_{\text{Render}} = \alpha_{\text{RGB}} L_{\text{RGB}} + \alpha_{\text{depth}} L_{\text{depth}} + \alpha_{\text{chamfer}} L_{\text{chamfer}} + \alpha_{\text{penalty}} L_{\text{penalty}},$$

in which $L_{\text{RGB}}$ and $L_{\text{depth}}$ are the RGB loss 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 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 Removing Nonlinearity 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. **??**) 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

$$v_p = \frac{v_a s_b - v_b s_a}{s_b - s_a},$$

in which $v_a, v_b$ are the positions of tetrahedron vertice $A$ 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 a perturbation inversely proportional to $s_b - s_a$:

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

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 $\mathcal{D}_x$. Due to this undesired behavior, 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.

[Zhen: An illustration here to explain the phenomenon]

[Zhen: A figure here to show the failure cases (generated samples)]

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 [3]. 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.

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 and more details are included in the appendix.

We follow the work [12] and consider two conditional generation methods: replacement method and reconstruction-guided method. For illustration, we denote $x_t^a$ the unconditional part $x_t^b$ the conditional part and $x^b$ the given condition. In replacement method, we sample $\hat{x}_{t-1}$ from $x_t$ and replace $\hat{x}_{t-1}^b$ by $p_{t-1}(x_{t-1}^b | x_0^a)$ which is defined by the forward diffusion. In reconstruction-guided method, we correct the generated samples $x_{t-1}^a$ by taking into account the fact that $x_{t-1}^b$ does not depend on $x_t^b$. The resulted approximate update is $x_{t-1}^a = \hat{x}_{t-1}^a - \frac{\alpha_t}{2} \nabla_{x^a} \left\| \hat{x}_{t-1}^b - x^b \right\|$.

**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.

# 5 Experiments

## 5.1 Settings

*Data Fitting.*

We fit tetrahedral grids on three ShapeNet subcategories: Chair, Airplane and Car. Fitting each object takes roughly 30-40 minutes on a single Quadro RTX 6000 GPU. We use the same train/test split in [31]. [Zhen: more details, or just leave them to the appendix?]

*Generative Model.*

For our diffusion model, we use an adapted version of the U-Net architecture for DDPM++ as described in [27] by changing the 2D convolution and other 2D modules to their 3D counterparts. To improve model capacity, we provide to the model a mask indicating which lattice sites are void as a conditional input. We train diffusion models for all datasets for 70k 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 16.

For alternative architectures, we also train GANs on our datasets, with an architecture similar to the one used in [29]. Our SDF-based baselines include IM-GAN [6] and SDF-StyleGAN [31]. In addition, we consider point cloud diffusion models [20] and voxel-based models [29] with the same resolution of 64.

## 5.2 Unconditional Generation

We randomly sample shapes from each of our trained generative models for each dataset and visualize a random subset of samples created by our methods and baseline methods in Fig. [Zhen: show the fig]. We consider [Zhen: list of baseline methods]. For point cloud generative models, we run Poisson reconstruction on the generative point clouds with the best hyperparameters to measure both FID and LFD.

We note that our method produces the sharpest samples, 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. To some extent, our model better captures the geometric priors in shape edges.

We show the failure cases of our model and baseline models in Fig. [Zhen: failure figs + explanation].

### 5.2.1 Quantatitve Metrics.

We first show various point cloud distances between the test set and the generated samples by sampling point clouds from both ground truth and generated meshes.

We note that these point cloud metrics inevitably favor point cloud generative models, not only because they are explicitly trained to model point clouds, but also because point cloud generative models are mostly trained on less than 10k points while other models can have much larger input space dimensions. Moreover, while these metrics roughly measure the 3D occupancy of shapes, they do not measure the quality of the surfaces, as we lose the geometric information by sampling point clouds from meshes. Instead, metrics on rendered views provide us with the geometric information.

Therefore, we follow [Zhen: cite paper here] and use Frechet inception distance (FID) on rendered views. We use the same set of 24 camera poses of which cameras are evenly distributed on a unit sphere and looking into the sphere center. We use an ImageNet pretrained model to compute the FIDs since [Zhen: the paper] shows that such model suffices to produce reasonable comparison between 3D shapes. Additionally, we compute light field distances (LFD), which employs low-level region and contour descriptors instead of semantic features to compute rendered views.

| | Model | MMD (↓) | | COV (%, ↑) | | 1-NNA (%, ↓) | | JSD ($\times 10^{-3}$, ↓) |
|---|---|---|---|---|---|---|---|---|
| | | CD | EMD | CD | EMD | CD | EMD | |
| Chair | IM-GAN | 13.928 | 1.816 | **49.64** | 41.96 | 58.59 | 69.05 | **6.298** |
| | SDF StyleGAN | 15.763 | 1.839 | 45.60 | 45.50 | 63.25 | 67.80 | 6.846 |
| | ddpm | 13.885 | 1.772 | 43.36 | 45.50 | 60.88 | 62.54 | **4.716** |
| | ddpm smoothed | **13.212** | **1.731** | 46.00 | **46.71** | **53.69** | **57.63** | 5.038 |
| Airplane | SDF StyleGAN | 4.57 | 1.180 | 40.67 | 32.63 | 85.48 | 87.08 | 26.30 |
| | ddpm | **3.61** | **1.026** | **47.22** | **45.24** | **63.10** | **72.31** | **10.27** |
| | ddpm smoothed | **3.61** | 1.042 | 47.34 | 42.15 | 66.44 | 76.26 | 11.36 |
| Car | SDF StyleGAN | 0.495 | 1.148 | 31.74 | 32.47 | 85.93 | 86.87 | 17.99 |
| | ddpm | | | | | | | |
| | ddpm smoothed | 10.52 | 41.07 | 1.15 | 2.20 | | | 526 |

| Model | FID on Chair | FID on Airplane |
|---|---|---|
| Pointcloud Diffusion [20] | 285.07 | N/A |
| IM-GAN [6] | 64.19 | 74.57 |
| SDF-StyleGAN [31] | 36.48 | 65.77 |
| rerun SDF-StyleGAN [31] | 35.28 | |
| Ours | 39.62 | 68.92 |

*Generation Sequence.* We visualize the predicted $x_0$.

### 5.3 Conditional Generation

As the geometry estimated from the single-view fitting is not perfect even in the given single view, we allow the masked tetrahedral vertices to change near the end of the diffusion completion process. In our setting, we set this threshold iteration to be 900.

### 5.4 Interpolation

We use DDIM [26] 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:

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

## 6 Conclusion

## REFERENCES


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

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

[3] 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.

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

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *SGP*, 2006.

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

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

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

[17] Christoph Lassner and Michael Zollhöfer. Pulsar: Efficient sphere-based neural rendering. In *CVPR*, 2021.

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

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

[20] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *CVPR*, 2021.

[21] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019.

[22] 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.

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

[24] 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.

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

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

[27] 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*, 2020.

[28] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3d point clouds via graph convolution. In *ICLR*, 2018.

[29] 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.

[30] 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.

[31] 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

## A.2 Losses

**RGB loss.**

**Depth loss.**

**Chamfer loss.**

**SDF regularization loss.**

## A.3 Hyperparameters and Settings

We find that using the clipped deformation vectors is better than using the *tanh*-ed deformation vectors as they remove some nonlinearity.

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.

We double the range of deformation used in [24] so that the meshes can better capture very thin geometries, and we do not observe pathological artifacts with this setting.

We notice 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. 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.

# B Details on Diffusion Models

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 fake 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 small amount. [Zhen: maybe padding + center crop]