TetraDiffusion: Tetrahedral Diffusion Models for 3D Shape Generation

Nikolai Kalischek* Torben Peters* Jan D. Wegner Konrad Schindler
Photogrammetry and Remote Sensing, ETH Zürich
{nkalischek,tpeters,jwegner,schindler}@ethz.ch

https://tetradiffusion.github.io

Figure 1. *TetraDiffusion* is a 3D denoising diffusion model that operates on a tetrahedral grid to enable the generation of high-resolution 3D shapes in seconds. All depicted meshes are shown without any postprocessing, hole-filling or smoothing.

Abstract

Probabilistic denoising diffusion models (DDMs) have set a new standard for 2D image generation. Extending DDMs for 3D content creation is an active field of research. Here, we propose TetraDiffusion*, a diffusion model that operates on a tetrahedral partitioning of 3D space to enable efficient, high-resolution 3D shape generation. Our model introduces operators for convolution and transpose convolution that act directly on the tetrahedral partition, and seamlessly includes additional attributes such as color. Remarkably,* TetraDiffusion *enables rapid sampling of detailed 3D objects in nearly real-time with unprecedented resolution. It's also adaptable for generating 3D shapes conditioned on 2D images. Compared to existing 3D mesh diffusion techniques, our method is up to 200 times faster in inference speed, works on standard consumer hardware, and delivers superior results.*

1. Introduction

The growing demand for virtual content has sparked a wave of research to automate the laborious and costly generation of 3D assets. At the heart of this endeavor lies the search for powerful and flexible 3D representations to encode, store and manipulate the geometry and topology of 3-dimensional objects. In this work, we describe a novel, highly efficient generative model that is able to produce high-quality surface meshes in seconds.

Following the rise of probabilistic denoising diffusion models (DDMs) for image generation [9, 19, 36, 37, 43], there have been several attempts to extend their generative capabilities to 3D [15, 26, 28, 53, 54]. It is relatively straightforward to transfer the DDM principle – gradual per-point perturbations with Gaussian noise – to voxels or points in 3D space, and this has already lead to interesting results [28, 53, 54]. However, these representations also come with their own disadvantages. Voxels are a natural extension of 2D pixels and amenable to well-established neural architectures based on discrete 3D convolutional operators, yet they are notoriously memory-hungry and therefore limited in terms of resolution. Point clouds, on the other hand, are sampled irregularly and avoid unnecessary discretization of empty space, but they lack connectivity information and have no direct notion of the underlying surfaces. Both representations face additional challenges when converting them to a surface mesh, as one must trade off smoothing and loss of detail against surface noise and topological artifacts. An alternative could be to directly work

* Equal contributions.

with meshes, but handling explicit meshes is cumbersome and restricted by the fixed surface topology.

Instead, we turn to a hybrid representation that combines the advantages of both worlds, namely a tetrahedral decomposition of 3D space. We develop neural operators that directly act on the tetrahedral representation, and thus allow for fast and memory-efficient learning. This is in contrast to methods that embed the tetrahedral representation in a voxel grid of much higher resolution, thus inheriting the limitations of voxel models [26]. Tetrahedral decompositions of 3D space have their origin in engineering and physics simulation and have also been used for volumetric modelling in graphics [21, 32], but they have only recently been adopted in the context of deep learning [12, 14, 41]. By combining the flexibility and structure of the tetrahedral grid with the generative power of DDMs, our model overcomes some of the limitations of existing 3D diffusion frameworks. A key ingredient of our method is a carefully predefined neighborhood topology of the space-filling tetrahedral decomposition [12] that enables well-defined convolutional operators and makes it easy to extract a surface mesh with the help of a differentiable Marching Tetrahedra scheme [14].

In the spirit of KPConv [45] and graph convolutions [3, 23, 48], we equip that representation with convolution (and transposed convolution) kernels that operate directly on the deformable tetrahedral structure. These operations make it possible to construct a U-Net architecture [38] in the tetrahedralized space, which, in turn, is the computational core of a denoising diffusion model. Our DDM learns to transform random noise into a 3D object shape by predicting both a per-vertex signed distance field and an individual per-vertex displacement, depicted in Fig. 2.

Our formulation has several benefits. Like in a point cloud, the vertices of the tetrahedral grid can be moved around to align with a desired surface. But at the same time they retain a uniquely defined neighborhood connectivity, which ensures that one can readily extract a topologically sound surface mesh, and obviates the need to train an additional surface reconstruction network [33]. Moreover, our network can be deployed on a multi-resolution hierarchy of tetrahedral decompositions, which makes both training and inference memory-efficient and computationally affordable. Empowered by the modest computational cost we are able to run diffusion at a higher native resolution (in our experiments $>$5 million tetrahedra, see Sec. 3) and to reconstruct 3D shapes with unprecedented detail. To further reduce training time, our proposed tetrahedral convolution layers make it possible to exploit sparsity by pruning tetrahedra in unoccupied regions of 3D space. Going beyond purely geometric object properties (captured by signed distance values and vertex offsets), we extend the tetrahedral representation and the associated Marching Tetrahedra scheme [41] to unrestricted feature vectors, such that it becomes possible to diffuse and extract further attributes like color. The extended Marching Tetrahedra algorithm remains fully differentiable. Consequently one can guide the diffusion during inference, *e.g*. towards smoother shapes, larger or smaller volume, or specific colors. Moreover one can drive the sampled shapes to resemble existing examples, renderings or textual descriptions, similar to classifier-free guidance [18] and regularization [37]. In summary, our key contributions are:

1. To the best of our knowledge, we propose the first 3D denoising diffusion model that operates entirely on a tetrahedral representation.
2. We design convolution operators and up- and downsampling kernels on the tetrahedral grid.
3. We show that *TetraDiffusion* enables efficient training and near real-time inference at unprecedented resolution, on consumer hardware.
4. We extend our tetrahedral DDM to include color, and to allow for 3D shape generation guided by images scraped from the internet.

2. Related work

3D Generative Models. Compared to their 2D counterparts, 3D generative models have to chose among a wider range of data representations. Most dominant are voxels and point clouds, for which it is straightforward to adapt the 2D formulations [1, 51, 52]. Early work directly treats point clouds as matrices to make them amenable to standard neural architectures [11], or rasterizes 3D data into a voxel grid in order to apply conventional 3D convolutions [2, 50]. To overcome fixed structures and to induce permutation invariance, PointFlow [51] first samples from a shape distribution and in a second step samples from the distribution of points given the corresponding shape prior.

Another prominent stream of work relies on implicit representations. The authors of [7] introduce an implicit field decoder to learn a signed distance function. The network is trained in adversarial fashion to predict the distance from the surface when presented a point coordinate and its associated feature encoding. In [4], shapes are represented as gradient fields over the logarithmic surface density, taking advantage of score-based generative models [43]. Similarly, [1] learn a shape prior with an auto-encoder and GAN in latent space. Our model does not rely on adversarial training or latent interpolation, rather we train directly on input encodings.

A combination of explicit and implicit representations is explored in GET3D [13]. Similar to our work, 3D shapes are described via signed distances and deformation vectors on a tetrahedral grid with fixed topology. Their tetrahedral representation is encoded in separate triplane representations for geometry and texture that are trained in an adversarial manner with rendering losses, making use of the dif-

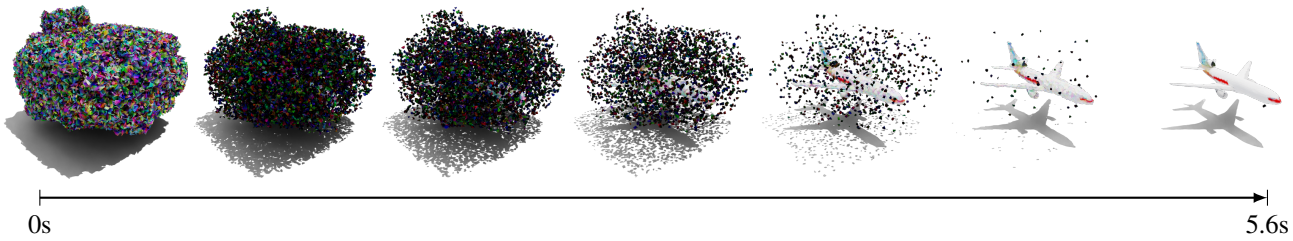

Figure 2. Exemplary reverse diffusion sequence starting from a noisy tetrahedral grid to a final texturized mesh in a few seconds.

ferentiable Marching Tetrahedra [41]. Instead of projecting into triplanes, TetGAN [14] directly operates on the tetrahedralized cuboid with an auto-encoder, supervised with ground truth features as well as global and local adversarial losses. Somewhat similar to our work they define convolutions and up-/ down-sampling operators on entire tetrahedra, which in their setting is straightforward as each tetrahedron has exactly four neighbours. In contrast, we extend those operations to act on vertices directly so as to allow for displacements, in the spirit of DMTet [41]. Our more general formulation naturally offers more flexibility and representation power, since every vertex can be aligned individually in contrast to the tetrahedral occupancy field in TetGAN, which only allows aligning a tetrahedron as a whole.

3D Diffusion Models. With the rise of DDMs, various 3D representations have been transferred to the diffusion setting [28, 29, 53, 54]. Point-voxel diffusion incorporates a point-voxel CNN [54] to directly apply diffusion on the hybrid representation. Similarly, [28] directly diffuse point clouds conditioned on a latent shape representation generated with a normalizing flow. LION [53] uses DDMs in latent space and maps the latent encoding back to a point cloud with a VAE. By itself this generates noisy point clouds, which is why they must train an additional network [33] to turn those into smooth meshes. To circumvent that extra surface extractor, 3DGen [15] introduces a two-stage training pipeline consisting of a triplane VAE and a separately trained latent diffusion model in triplane space. Somewhat similarly, [42] first train a decoder that maps triplane features to occupancy grids, then train a 2D diffusion model to generate those triplanes. In contrast, our method only requires *single-stage training* and no separate decoder, by natively operating on the tetrahedral 3D representation.

Perhaps the closest work to ours is MeshDiffusion [26], which also maps meshes to a deformable tetrahedral grid and performs diffusion on that representation. However, to circumvent the lack of convolutional operators for that grid they embed the tetrahedra in a higher-resolution regular voxel grid. This makes it possible to employ conventional 3D convolutions, but largely sacrifices the benefit of the tetrahedral representation: the voxelization is extremely inefficient, as it leads to a cubic increase in memory footprint and computation without adding any information. By defining convolutions directly on the spatially sparse vertices of the tetrahedral grid our model avoids that large overhead and allows for a finer tetrahedralization that captures higher-resolution details.

3. Background

Tetrahedral Grid. Following [41], we represent shapes within a given cuboid $\mathcal{T}$ with a signed distance field and a displacement field that are both defined on the vertices of the same, space-filling tetrahedral decomposition of $\mathcal{T}$. We refer to that structure as the *tetrahedral grid*, with vertices $V_\mathcal{T} \in \mathbb{R}^{N\times 3}$ and tetrahedra $T \in \mathbb{N}^{K\times 4}$. An nearly regular decomposition can be found via close-packed tetrahedral tiling with the A15 lattice [10]. Tetrahedral grid resolution refers to the grid spacing used in the iso-surface stuffing algorithm. A tetrahedralized cube of resolution R contains $0.72 \cdot R^3$ tetrahedra. Note that a given number of tetrahedra can capture more detail than the same number of voxels, as they can be deformed to better follow the surface. Each tetrahedron T_k consists of four vertices $\{v_{k1}, v_{k2}, v_{k3}, v_{k4}\}$ and corresponding edges to form a simplex, *i.e.*, the connectivity is predefined and fixed during training and inference. Each vertex is assigned a displacement Δ_v and a signed distance value s_v. The SDF provides an implicit surface representation, whereas the displacements serve to precisely align it with the actual object surface. Conveniently, one can extract a surface *mesh* from $\mathcal{T}$ with Deep Marching Tetrahedra (DMTet, [41]). In Section 4, we extend the tetrahedral representation to arbitrary feature vectors, such that additional vertex attributes (*e.g.*, color) can be propagated to the final mesh.

Diffusion. Denoising diffusion models can be seen as restricted, hierarchical Markovian VAEs [27] that learn to approximate the data distribution $p(\mathbf{x})$ with a sequence of steps. In the variational formulation [20, 22] the steps are indexed by a continuous time variable $t \in [0, 1]$. At the final time step $T = 1$ the latent variables $\mathbf{z_t}$ should be normally distributed, $q(\mathbf{z}_T) = \mathcal{N}(\mathbf{z}_T; \mathbf{0}, \mathbf{I})$. The forward process of

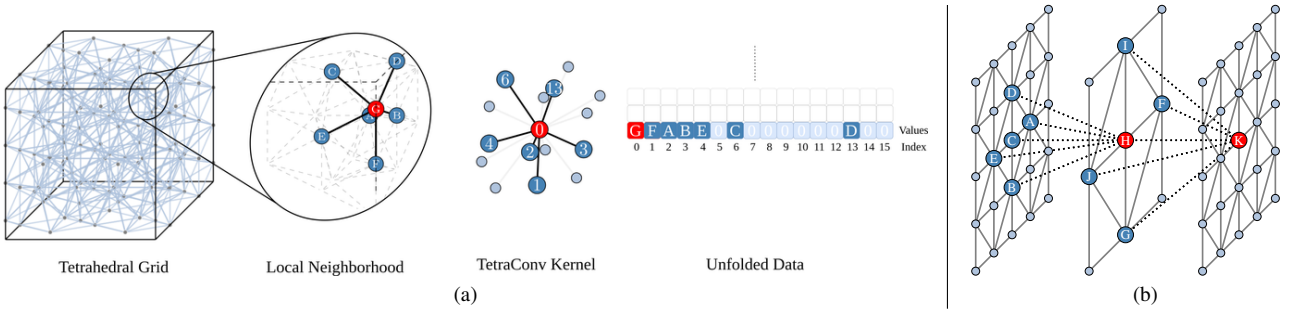

Figure 3. *(a)* Tetrahedral convolution for kernel size $m = 16$. The TetraConv kernel defines the ordering and padding for every local neighborhood. The center vertex, where the convolution aggregates, is marked in red and the neighborhood in dark blue. Missing vertices (light blue) are padded with zeros. *(b)* Strided convolution $H = \text{TetraConv}(A, B, C, D, E)$ and transposed convolution $K = \text{TetraConv}(F, G, H, I, J)$ accumulate information from the nearest neighbors in their preceding layers. Operators are depicted in 2D.

the marginals $q(\mathbf{z_t}|\mathbf{x})$ is Gaussian and given by

$$q(\mathbf{z_t}|\mathbf{x}) = \mathcal{N}(\mathbf{z_t}; \alpha_t \mathbf{x}, \sigma_t^2 \mathbf{I}), \tag{1}$$

where $\text{SNR}(t) = \frac{\alpha_t^2}{\sigma_t^2}$ is the signal-to-noise ratio, assumed to decrease strictly monotonically in time, and α_t and σ_t^2 are strictly positive for all t. Consequently, $\mathbf{z_t}$ will be increasingly noisy over time. We fix $\alpha_t^2 = 1 - \sigma_t^2$, corresponding to a variance-preserving process [44]. Under the Markov assumption the forward transition kernels for $t > s$ are also Gaussian and given by:

$$q(\mathbf{z_t}|\mathbf{z_s}) = \mathcal{N}(\mathbf{z_t}; \alpha_{t|s} \mathbf{z_s}, \sigma_{t|s}^2 \mathbf{I}), \tag{2}$$

where $\alpha_{t|s} = \frac{\alpha_t}{\alpha_s}$ and $\sigma_{t|s}^2 = \sigma_t^2 - \alpha_{t|s}^2 \sigma_s^2$. A common noise schedule is $\alpha_t = \cos(\pi t/2)$, which under variance preservation leads to a signal-to-noise ratio of $\text{SNR}(t) = \frac{1}{\tan(\pi t/2)}$.

We are interested in learning the reverse diffusion process. While $q(\mathbf{z_s}|\mathbf{z_t})$ is in general intractable as it requires integration over the whole dataset, conditioning it on a data sample x gives rise to a closed-form solution:

$$q(\mathbf{z_s}|\mathbf{z_t}, \mathbf{x}) = \mathcal{N}(\mathbf{z_s}; \mu_{s,t}(\mathbf{z_t}, \mathbf{x}), \sigma_{s,t}^2 \mathbf{I}), \tag{3}$$

where $\mu_{s,t}(\mathbf{z_t}, \mathbf{x}) = \frac{\alpha_{t|s}\sigma_s^2}{\sigma_t^2}\mathbf{z_t} + \frac{\alpha_s \sigma_{t|s}^2}{\sigma_t^2}\mathbf{x}$ and $\sigma_{s,t}^2 = \sigma_{t|s}^2 \frac{\sigma_s^2}{\sigma_t^2}$. As $\mathbf{x}$ is only available during training, it is replaced by a neural network prediction $\hat{\mathbf{x}}_\theta(\mathbf{z_t}; t) \approx \mathbf{x}$. An equivalent interpretation of the denoising model is as a score model, which in the infinite data limit coverges to the marginal distribution $q(\mathbf{z_t})$ [43].

To train $\hat{\mathbf{x}}_\theta(\mathbf{z_t}; t)$ one optimizes the variational lower bound of the marginal log-likelihood

$$\mathcal{L}(\mathbf{x}) = -\frac{1}{2}\mathbb{E}_{\substack{\epsilon\sim\mathcal{N}(0,\mathbf{I}),\\ t\sim\mathcal{N}(0,\mathbf{I})}}\left[\text{SNR}'(t)\left\|\mathbf{x} - \hat{\mathbf{x}}_\theta(\mathbf{z_t}; t)\right\|_2^2\right], \tag{4}$$

where $\text{SNR}'(t) = d\text{SNR}/dt$ and $\mathbf{z_t} = \alpha_t \mathbf{x} + \sigma_t \epsilon$. Instead of parametrizing the model to directly recover $\mathbf{x}$ from its corrupted version $\mathbf{z_t}$, one can predict the noise $\hat{\epsilon}$ and recover $\hat{\mathbf{x}}$ from $\hat{\mathbf{x}} = \frac{\mathbf{z_t}}{\alpha_t} - \sigma_t \frac{\hat{\epsilon}_t}{\alpha_t}$. Since that objective tends to destabilize training near $t = 1$, we use the more robust v-parametrization [40], $\hat{\mathbf{v}}_t = \alpha_t \hat{\epsilon}_t - \sigma_t \hat{\mathbf{x}}$. Note that $\hat{\mathbf{x}} = \alpha_t \mathbf{z_t} - \sigma_t \hat{\mathbf{v}}_\mathbf{t}$. Once trained, we can sample from our data distribution with ancestral sampling. Setting $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we starting at $\mathbf{z_1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and iteratively denoise it according to

$$\mathbf{z_s} = \frac{\alpha_t \sigma_s^2}{\alpha_s \sigma_t^2}\mathbf{z_t} + \frac{\alpha_s \sigma_{t|s}^2}{\sigma_t^s}\hat{\mathbf{x}}_\theta(\mathbf{z_t}; t) + \sqrt{\frac{\sigma_{t|s}^2 \sigma_s^2}{\sigma_t^2}}\epsilon. \tag{5}$$

4. Tetrahedral Representation Learning

Tetra Convolution. Our displacement and signed distance fields are not directly amenable to regular 3D convolutions due to varying neighborhoods and connectivity in the grid. However, the superimposed A15 lattice allows a collision-free spatial ordering of vertices within each neighborhood, and as such a well-defined, rigorous tetrahedral convolution. This is in contrast to graph convolutions that are invariant to the spatial configuration of the neighborhood, and unlike KPConv [45], our approach eliminates the need to search for the relevant points under the kernel and subsequent distance weighting. We define our unique ordering, a discrete binning of the local edge orientations, by iteratively clustering the set of all outgoing edges with k-means until we find a collision-free ordering, *i.e.* no edges within a neighborhood fall within the same bin. Interestingly, in practice our clustering approach always converges to a basis of at most $m + 1$ reference directions, where m is the maximal neighborhood in the tetrahedral grid. In contrast to 3D voxel convolutions, the total number of basis directions is almost half the size (*e.g.* 15 compared to 26), highlighting one key ingredient to efficient learning. A concise description of our approach can be found in Sec. C.1.

With fixed ordering, it is straightforward to define the result of a tetrahedral convolution layer l at a given vertex

Figure 4. 3D shape interpolation. We first generate the leftmost and rightmost samples and save the noise applied in every diffusion step. Intermediate shapes are then generated by spherical linear interpolation of the noise.

v_k as a weighted sum over itself and its neighborhood $\mathcal{N}_{v_k}$ in layer $(l-1)$:

$$\Phi^l_{v_k} = W_0 \Phi^{l-1}_{v_k} + \sum_{j=1}^{m+1} \mathbb{1}_{j \in \mathcal{N}_{v_k}} W_j \Phi^{l-1}_{v_j}, \qquad (6)$$

where W_j are the weights of the kernel and $\Phi^l_{v_i}$ the feature vector of v_i in layer l. As illustrated in Fig. 3a, the varying size of the neighborhood simply corresponds to zero padding the corresponding kernel. This layer is referred to as TetraConv. To fully exploit the power of convolutional learning, we need to define down- and upsampling operations on the tetrahedral grid. We construct tetrahedral tesselations of our cuboid $\mathcal{T}$ of varying resolution and establish parent-child relations between vertices in adjacent hierarchy levels via k-nearest neighbor relations. As shown in Figure 3b, this enables flexible down- and upsampling rates corresponding to strided convolution and transposed convolution. In particular, the neighborhood in Eq. (6) gets replaced by the *knn*-neighborhood of the lower or higher tetrahedral resolution.

Grid Pruning. Like any volumetric tesselation, the tetrahedral grid is in most cases sparsely populated, reflecting the fact that only a small part of 3D space lies near an object surface. Unlike voxel-based convolutions, where only axis-parallel cropping is efficient (in particular limited to the bounding box), our tetrahedral formalism facilitates pruning in a more targeted manner. Deleting all unoccupied vertices and their corresponding connections translates to two simple operations: (1) unused vertices are completely removed by deleting the corresponding row in the unfolded kernel-data matrix and (2) removed connections lead to additional zero padding in affected rows. As a result, we can truncate our grid to the convex hull of all data used in a lossless manner. Additionally, training and inference speed and memory consumption is easily further enhanced by (lossy) pruning vertices that are occupied only up to a user-defined number of times. A full description of the pruning methodology can be found in Sec. C.4.

Tetrahedral Diffusion. With all necessary building blocks at hand, we can now introduce our tetrahedral diffusion. We directly diffuse in deformable tetrahedral space, *i.e.* our basic input consists of N vertices with features $\{s_v, \Delta_v\}$, and the associated neighborhood relations. However, we can seamlessly extend the hybrid tetrahedral representation by any vertex feature vector, in particular color and texture information and without loosing the differentiability of the marching tetrahedra algorithm. In particular, surface information like color is a convex combination of the corresponding tetrahedral vertex features. In our experiments we therefore directly diffuse SDF values, deformation vectors and color vectors per vertex, *i.e.* $\{s_v, \Delta_v, c_v\}$.

Our network largely adheres to the standard U-Vit architecture of many diffusion models [20], of course with the tetrahedral convolution operators introduced above, *i.e.*, residual convolution layers, with group normalization [49], SiLU activations [17] and attention layers in between. The entire network, hyperparameters and our tetrahedral marching extension are described in Sec. C.3.

5. Experiments

We now experimentally demonstrate the capabilities of TetraDiffusion. We train and evaluate our method on the classes *airplane*, *bike*, *car* and *chair* from the ShapeNet [5] dataset, using the official ShapeNet train/val/test split. Each shape is individually normalized to lie in $[-1, 1]$. Our network requires SDFs and displacement fields on a tetrahedral grid, *i.e.*, ShapeNet meshes have to be converted to that format. We adapt the rendering pipeline of [30] and fit each shape individually into the grid with a combination of rendering and volumetric losses. Similar to [26], the ground

Category	Method	1-NNA ↓		MMD ↓		COV ↑	
		CD	EMD	CD	EMD	CD	EMD
Airplane	GET3D [13]	93.1 ± 0.6	75.9 ± 1.6	0.42 ± 0.01	0.52 ± 0.01	37.8 ± 1.2	44.0 ± 0.9
	MD [26]	89.2 ± 0.8	89.2 ± 0.7	0.64 ± 0.04	0.75 ± 0.01	32.5 ± 1.7	35.8 ± 1.0
	Ours	**71.9** ± 1.5	**68.3** ± 1.4	**0.30** ± 0.01	**0.49** ± 0.01	**48.2** ± 0.9	**47.8** ± 1.2
	Ours_{hr}	73.1 ± 1.1	73.4 ± 1.4	0.34 ± 0.01	0.52 ± 0.01	44.5 ± 1.7	46.5 ± 1.2
Bike	GET3D [13]	72.7 ± 3.0	68.8 ± 6.1	1.65 ± 0.04	1.18 ± 0.05	40.7 ± 4.8	47.8 ± 4.8
	MD [26]	**61.7** ± 5.3	**65.2** ± 6.0	**1.62** ± 0.13	1.18 ± 0.07	42.2 ± 5.3	49.3 ± 5.9
	Ours	62.6 ± 4.7	65.7 ± 5.6	1.71 ± 0.07	**1.17** ± 0.05	45.7 ± 5.4	52.7 ± 5.9
	Ours_{hr}	65.9 ± 5.7	68.4 ± 5.1	1.82 ± 0.11	1.22 ± 0.06	**47.7** ± 4.9	**54.5** ± 5.1
Car	GET3D [13]	87.3 ± 0.7	73.6 ± 0.8	1.03 ± 0.01	0.72 ± 0.01	20.9 ± 1.4	34.2 ± 1.1
	MD [26]	68.6 ± 1.6	66.7 ± 2.5	0.93 ± 0.01	0.69 ± 0.01	35.1 ± 1.2	**42.3** ± 1.1
	Ours	68.5 ± 0.9	**61.1** ± 1.4	**0.90** ± 0.01	**0.64** ± 0.01	**36.1** ± 1.6	41.3 ± 1.6
	Ours_{hr}	**67.1** ± 1.1	61.4 ± 1.3	0.91 ± 0.01	0.65 ± 0.01	35.6 ± 1.5	39.7 ± 1.3
Chair	GET3D [13]	76.4 ± 0.6	72.9 ± 0.8	5.31 ± 0.04	2.56 ± 0.01	31.8 ± 0.6	34.0 ± 0.4
	MD [26]	73.0 ± 1.0	79.8 ± 0.9	5.18 ± 0.12	2.79 ± 0.02	38.2 ± 1.0	35.2 ± 0.9
	Ours	**62.0** ± 0.6	**61.2** ± 0.9	**4.94** ± 0.11	**2.47** ± 0.03	**46.2** ± 0.6	**47.5** ± 0.8

Table 1. Generation metrics on ShapeNet classes airplane, bike, car and chair. MMD-CD is multiplied by $\times 10^3$, EMD by $\times 10^2$.

truth is created in two step procedure, fixing SDF values to $\{-1, 1\}$. For a comprehensive overview of the entire preprocessing protocol, please refer to the supplementary material.

5.1. Qualitative results

Unconditional generation. Figure 1 showcases a diverse array of randomly generated 3D shapes, demonstrating the high quality and level of detail of our model. Note the diversity of shapes within each class, and the intricate details like chains and brake discs on motorbikes or propellers and wing appendages on airplanes. The motorbikes nicely illustrate the model's generative abilities: although the training set is small (≈ 200 examples), the model goes beyond memorization and naive interpolation. It seamlessly blends parts of different training samples into plausible assemblies, and comes up with curious designs clearly not observed during training. It is noteworthy that, although further post-processing is a common practice in the context of 3D generative models, our generated shapes are not post-processed or cleaned in any way, as *e.g.* smoothing would remove fine structures and high-frequency details correctly synthesized by our model.

Qualitative comparison. Figure 5 compares TetraDiffusion against other recent mesh generators, namely GET3D [13] and MeshDiffusion (MD) [26]. We train all methods with resolution $R = 128$ on the ShapeNet training split. Additionally, we also train a hi-res version of our model with $R = 192$ denoted by $\textit{Ours}_{hr}$, which is not feasible for MD because it would exceed current hardware limitations on a single GPU (see also Tab. 2). The figure depicts examples from a large pool of generated shapes, matched across methods by finding the nearest neighbors to a given ground truth example from the test set. As general trends, we observe that *(i)* the GAN-based GET3D produces noisier shapes and has a tendency to hallucinate implausible shape details; *(ii)* TetraDiffusion tends to generate cleaner and more detailed shapes already at resolution $R = 128$: crease edges and small structures are crisper, while smooth surfaces have fewer bumps and holes; *(iii)* our hi-res version with $R = 192$ clearly improves shape quality compared to the standard $R = 128$, supporting our claim that resolution is still a bottleneck: current 3D diffusion models run into hardware limits, and efficient use of memory and compute matters.

Interpolation. In Figure 4, we explore the latent shape space learned by our model. We can directly interpolate between two different shape instances using spherical interpolation to blend from start to end noise. Then, we feed the intermediate versions into the diffusion model to generate the corresponding meshes. The geometric integrity of the intermediate samples and the plausible, gradual transitions from start to end suggest that the model has indeed learned to disentangle objects along functionally meaningful dimensions,

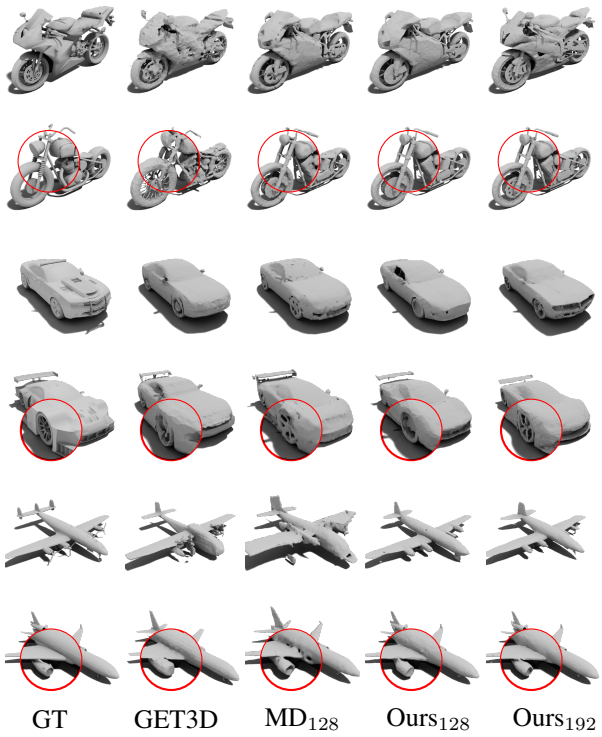

Figure 5. Comparison of different methods, red circles show enlarged highlights. Samples were collected by searching for the nearest neighbour to the ground truth. Ground truth samples are chosen from the test split. Best viewed when zoomed in.

and to represent them in terms of those components.

Conditional Generation. Our network can be seamlessly extended to conditional generation. We render multiple views for each ShapeNet shape and embed the images into CLIP space [35]. We expand our U-Net with additional cross-attention layers [6] and add the result to the output of the corresponding self-attention layer. We train our model conditioning on the shapes' CLIP embeddings. At test time, we can then reconstruct faithful meshes from novel, unseen images by simply conditioning on the corresponding embedding. This effectively allows single-view reconstruction (SVR). We qualitatively show SVR results from RGB data in Fig. 6. Notably, the conditional model of TetraDiffusion, which is trained across all classes together, effectively discerns between diverse shapes and classes, and exhibits a strong adaptability to color variations.

5.2. Quantitative results

Metrics. We quantitatively evaluate our model in terms of *1-nearest neighbor accuracy* (1NNA), as proposed by Yang *et al*. [51], as well as minimum matching distance (MMD) and coverage (COV). All three latter metrics are computed between the ground truth test set[1] and an equally large set of generated samples, and are based on pairwise distances between shapes. Since those distances are computed from point sets, we uniformly resample all (generated and ground truth) meshes into point clouds of size 2048 using the outer object surface only. Like [53], we compute all metrics based on both Earth Mover's Distance (EMD) and Chamfer Distance (CD).

1-NNA measures the distributional discrepancy between two sets of point clouds: it finds the nearest neighbor for every generated sample and counts how often that nearest neighbor is another generated sample, respectively a ground truth sample. A value close to 50% is considered optimal: higher values mean that generated samples are far from the ground truth, indicating underfitting; lower values mean that the model mostly replicates ground truth samples, indicating overfitting. **MMD** measures the average distance from a generated sample to the nearest reference sample, as a proxy for fidelity to the ground truth shape space. **COV** quantifies what portion of reference samples are nearest neighbors to some generated sample, i.e., higher values mean that a larger part of the ground truth variability is covered by the generator.

We benchmark our model against the state-of-the-art 3D mesh generators, GET3D [13] and MeshDiffusion (MD) [26]. To get a meaningful and consistent comparison, we make sure that all models are evaluated with the same train/test split. To that end, we retrained all methods on the official ShapeNet training part. We sample 1000 shapes for each category and average ten runs with random splits equal the test set size. As demonstrated in Table 1, our method fares very well in the direct comparison. It achieves the best 1-NNA scores (i.e., it comes closest to 50) for 3 out of 4 object classes in terms of EMD and CD. We also achieve the highest COV scores in all cases, and the lowest MMD in all cases but one (CD for motorbikes).

	Training		Inference	
Method	GPU (GB)	Speed (it/s)	GPU (GB)	Speed (s/shape)
GET3D	13.3	0.10	11.3	0.83
MD	76.6	0.5	29.2 (22.6†)	714.3 (526.3†)
Ours*	12.0	2.8	7.4	3.4
Ours	20.8	1.0	9.7	11.2
Ours_{hr}*	20.9	1.2	11.7	9.1
Ours_{hr}	78.2	0.3	42.1	33.3

Table 2. Memory consumption and computing time of different generative shape models. † We implement 16-bit inference in MD for fair comparison. Our methods labelled with a * are pruned versions, highlighting the efficiency boost of our tetrahedral formulation. Full details in Appendix.

[1] We follow literature [51, 53] and use the validation split of ShapeNet as the test set, which has not been used during training.

Figure 6. Conditional generation of 3D shapes. *TetraDiffusion* is conditioned on the CLIP embeddings of images in the wild during inference, while only trained on embeddings from rendered ShapeNet views. The shapes are displayed in matched pairs, with the upper image illustrating the condition and the lower image showcasing the corresponding generated mesh.

5.3. Efficiency

A central aspect of our approach is its efficiency. We argue that, despite the rapid development of GPU hardware, 3D generative models are at present held back by hardware limits, especially GPU memory (but also training time could become prohibitive when scaling current models up to industrial scale). In other words, a lighter and more efficient design makes 3D generative models not only faster and cheaper, but also yields results of higher quality.

By performing convolution natively on the tetrahedral grid, we can avoid overlaying a higher-resolution voxel grid as done, for instance, in MeshDiffusion. What is more, the voxel grid implies convolution kernels with at least $3 \times 3 \times 3 = 27$, whereas our operator works well with only 16 neighbors. This, in turn, makes it possible to increase the network capacity under the same hardware constraints. For instance TetraDiffusion can, due to its maller memory footprint, use 1028 feature channels in the bottleneck, and can also be run at higher resolution (see above).

Table 2 compares the computational demands of different 3D generative models. Our peak memory consumption at resolution $R = 128$ (with batch size 1) is 20.8 GB, in contrast to MeshDiffusion's 76.6 GB. Pruning the data cube as described in Section 4 further reduces memory usage to 12GB, without compromising prediction quality. Even at high resolution $R = 192$, and assuming that shape variability is so large that no significant pruning is possible, our model consumes less than 80 GB and can be run on a single high-end GPU. As a consequence, TetraDiffusion is also significantly faster. Inference time lies at about one shape per three seconds, getting close to GAN-based inference time and about 200× faster than MeshDiffusion on the same hardware.

Furthermore, the time-continuous formulation of our diffusion model provides the flexibility to vary sampling steps during inference. In other words, we are not constrained to a fixed number of steps, such as sampling with 1000 time steps. In our experimentation, detailed in the supplementary material, we explore the range of steps and demonstrate that we can successfully diffuse meshes within 32 steps without compromising quality.

6. Conclusion

We present *TetraDiffusion*, an innovative 3D diffusion framework capable of generating high-resolution, colorized meshes with arbitrary topology and unprecedented resolution. To our knowledge, our work is the first 3D diffusion model that fully operates on a tetrahedral data structure, enabling highly efficient training and sampling processes. Through our experiments, we have showcased the ability of diffusion on the tetrahedral grid to synthesize diverse 3D shapes, incorporating attributes such as color and conditions based on CLIP features.

We believe a promising avenue for further exploration is to better leverage the extended differentiable tetrahedral marching algorithm and differentiable rendering [24, 30]. This approach aims to overcome the constraints of limited 3D assets and seamlessly integrate extensive 2D data into the framework. Moreover, *TetraDiffusion* is presently trained exclusively on objects. Nevertheless, the sparse nature of scenes represents a logical extension for our framework.

References

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

[2] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016. 2

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013. 2

[4] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *ECCV*, 2020. 2

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 5

[6] Chun-Fu Richard Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *ICCV*, pages 357–366, 2021. 7

[7] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019. 2

[8] Katherine Crowson. Clip guided diffusion hq 256x256. *Colab Notebook. URL https://colab. research. google. com/drive/12a_Wrfi2_ gwwAuN3VvMTwVMz9TfqctNj*, 2021. 4

[9] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. In *NeurIPS*, 2021. 1

[10] Crawford Doran, Athena Chang, and Robert Bridson. Isosurface stuffing improved: acute lattices and feature matching. In *ACM SIGGRAPH*, pages 1–1. 2013. 3, 2

[11] Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution tree networks for 3d point cloud processing. In *ECCV*, 2018. 2

[12] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. In *NeurIPS*, 2020. 2

[13] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. *NeurIPS*, 35:31841–31854, 2022. 2, 6, 7, 12, 16

[14] William Gao, April Wang, Gal Metzer, Raymond A Yeh, and Rana Hanocka. TetGAN: A convolutional neural network for tetrahedral mesh generation. *arXiv preprint arXiv:2210.05735*, 2022. 2, 3

[15] Anchit Gupta, Wenhan Xiong, Yixin Nie, Ian Jones, and Barlas Oğuz. 3dgen: Triplane latent diffusion for textured mesh generation. *arXiv preprint arXiv:2303.05371*, 2023. 1, 3, 12

[16] Nicholas Guttenberg. 17

[17] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016. 5

[18] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. 2, 4

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 1

[20] Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. simple diffusion: End-to-end diffusion for high resolution images. *arXiv preprint arXiv:2301.11093*, 2023. 3, 5, 13, 17

[21] Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM ToG*, 30(4):78, 2011. 2

[22] Diederik P Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. In *NeurIPS*, 2021. 3

[23] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2

[24] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):1–14, 2020. 8

[25] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. Common diffusion noise schedules and sample steps are flawed. *arXiv preprint arXiv:2305.08891*, 2023. 17

[26] Zhen Liu, Yao Feng, Michael J. Black, Derek Nowrouzezahrai, Liam Paull, and Weiyang Liu. Meshdiffusion: Score-based generative 3d mesh modeling. In *International Conference on Learning Representations*, 2023. 1, 2, 3, 5, 6, 7, 16

[27] Calvin Luo. Understanding diffusion models: A unified perspective. *arXiv preprint arXiv:2208.11970*, 2022. 3

[28] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *CVPR*, 2021. 1, 3

[29] Shentong Mo, Enze Xie, Ruihang Chu, Lewei Yao, Lanqing Hong, Matthias Nießner, and Zhenguo Li. Dit-3d: Exploring plain diffusion transformers for 3d shape generation. *arXiv preprint arXiv:2307.01831*, 2023. 3

[30] 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 *CVPR*, pages 8280–8290, 2022. 5, 8, 2, 12

[31] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021. 4

[32] Gilles-Philippe Paillé, Nicolas Ray, Pierre Poulin, Alla Sheffer, and Bruno Lévy. Dihedral angle-based maps of tetrahedral meshes. *ACM ToG*, 34(4):1–10, 2015. 2

[33] Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. Shape as points: A differentiable Poisson solver. In *NeurIPS*, 2021. 2, 3

[34] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, 2018. 13

[35] 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 *ICML*, pages 8748–8763. PMLR, 2021. 7

[36] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022. 1

[37] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 1, 2, 4

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 2

[39] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022. 17

[40] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022. 4

[41] 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. 2, 3, 12

[42] J Ryan Shue, Eric Ryan Chan, Ryan Po, Zachary Ankner, Jiajun Wu, and Gordon Wetzstein. 3d neural field generation using triplane diffusion. In *CVPR*, pages 20875–20886, 2023. 3

[43] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021. 1, 2, 4, 17

[44] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020. 4

[45] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. KPConv: Flexible and deformable convolution for point clouds. In *ICCV*, 2019. 2, 4

[46] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. 13

[47] Sven-Ake Wegner. Lecture notes on high-dimensional data. *arXiv preprint arXiv:2101.05841*, 2021. 10

[48] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019. 2

[49] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018. 5

[50] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015. 2

[51] 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. 2, 7

[52] Maciej Zamorski, Maciej Zieba, Piotr Klukowski, Rafał Nowak, Karol Kurach, Wojciech Stokowiec, and Tomasz Trzciński. Adversarial autoencoders for compact representations of 3d point clouds. *CVIU*, 193:102921, 2020. 2

[53] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. LION: Latent point diffusion models for 3d shape generation. *arXiv preprint arXiv:2210.06978*, 2022. 1, 3, 7, 10

[54] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *ICCV*, 2021. 1, 3