TetraDiffusion: Tetrahedral Diffusion Models for 3D Shape Generation

Nikolai Kalischek*[1], Torben Peters*[1], Jan D. Wegner[2], and Konrad Schindler[1]
* equal contribution

[1] ETH Zurich [2] University of Zurich
{nkalischek,tpeters,schindler}@ethz.ch
jandirk.wegner@uzh.ch

Fig. 1: TetraDiffusion is a 3D denoising diffusion model that operates on a tetrahedral grid to generate 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 like color. Our design generates mesh geometry much more efficiently: Compared to existing mesh diffusion techniques, *TetraDiffusion* is up to 200× faster. At the same time, it reduces memory consumption and can operate at substantially higher resolution than existing mesh generators. Using only standard consumer hardware, it sets a new standard in terms of spatial detail and outperforms other mesh generators across a range of quality metrics. For additional results and code see our project page `tetradiffusion.github.io`.

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.

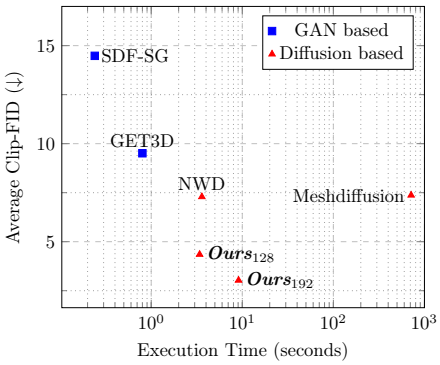

Fig. 2: Generation time per sample vs. mesh quality for different methods. Clip-FID is averaged over the ShapeNet classes airplane, car and motorbike.

Following the rise of probabilistic denoising diffusion models (DDMs) for image generation [8, 16, 29, 30, 34], there have been several attempts to extend their generative capabilities to 3D [14, 23, 24, 41, 42, 45]. 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 [24, 41, 43, 45]. 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 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 [23]. Tetrahedral decompositions of 3D space have their origin in engineering and physics simulation and have also been used for volumetric modelling in graphics [20, 27], but they have only recently been adopted in the context of deep learning [11, 13, 32]. 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 [11] that enables well-defined convolutional operators and makes it easy to extract a surface mesh with the help of a differentiable Marching Tetrahedra scheme [13].

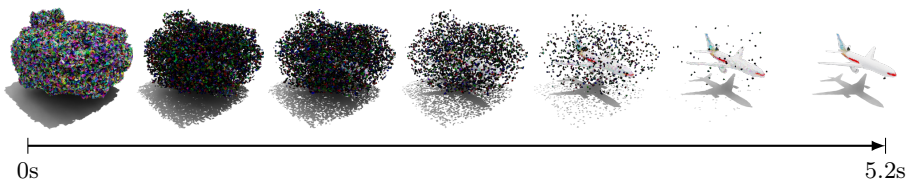

Fig. 3: Reverse tetrahedral diffusion sequence. Starting from a noisy tetrahedral grid, the model recovers a textured mesh in a few seconds. Note the shape of the initial grid: our method allows more targeted grid pruning than prior art.

In the spirit of KPConv [35] and graph convolutions [4, 21, 36], 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 [31] in the tetrahedralized space, which, in turn, is the computational core of a denoising diffusion model. That DDM transforms random noise into a 3D object shape (Fig. 3) by predicting both a per-vertex signed distance field and an individual per-vertex displacement.

Our formulation has several benefits. Like in a point cloud, the vertices of the tetrahedral grid can be moved around to align with object surfaces. 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 [28]. 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, *cf*. Fig. 2. Empowered by the modest computational cost we are able to run diffusion at higher native resolution (in our experiments $> 5 \cdot 10^6$ tetrahedra) and to reconstruct 3D shapes with unprecedented detail. Moreover, our proposed tetrahedral convolution layers offer the possibility to exploit sparsity and prune tetrahedra in unoccupied regions of 3D space (Fig. 3), further reducing training time.

Going beyond purely geometric object properties, as captured by signed distance values and vertex offsets, we extend the tetrahedral representation and the associated Marching Tetrahedra scheme [32] to unrestricted feature vectors, so as to diffuse and extract further attributes like color. The extended Marching Tetrahedra algorithm remains fully differentiable. Consequently, the model is fully compatible with guided and conditional diffusion schemes, as we show in the supplementary material, e.g. by guiding the sampled shapes in size, volume or specific colors. Moreover one can drive the shapes to resemble existing examples or renderings, similar to classifier-free guidance [17] and regularization [30]. In summary, our 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 extend our tetrahedral DDM to include color, which enables the generation of textured assets.
4. We show that *TetraDiffusion* enables efficient training and fast inference at unprecedented resolution, on consumer hardware.

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 [1,39,40]. Early work directly treats point clouds as matrices to make them amenable to standard neural architectures [10], or rasterizes 3D data into a voxel grid in order to apply conventional 3D convolutions [3,38]. To overcome fixed structures and to induce permutation invariance, PointFlow [39] 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 [5], shapes are represented as gradient fields over the logarithmic surface density, taking advantage of score-based generative models [34]. 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.

GET3D [12] explores a combination of explicit and implicit representations. 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 differentiable Marching Tetrahedra [32]. Instead of projecting into triplanes, TetGAN [13] 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 [32]. 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 [24,25,41,45]. Point-voxel diffusion incorporates a point-voxel CNN [45] to directly apply diffusion on the hybrid

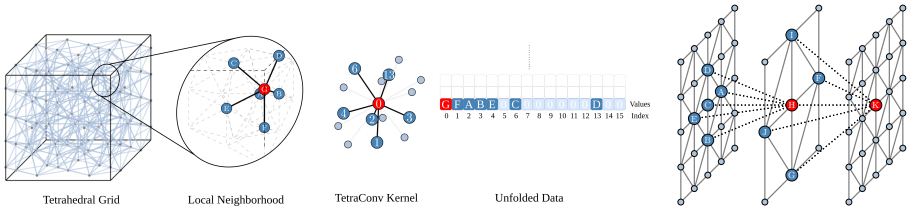

Fig. 4: (a) Tetrahedral convolution for kernel size $m = 16$**.** The TetraConv kernel defines the ordering and padding for every local neighborhood. The center vertex, where the convolution aggregates, is marked in red and the neighborhood in dark blue. Missing vertices (light blue) are padded with zeros. **(b) Strided convolution** $H = \textbf{TetraConv}(A, B, C, D, E)$ **and transposed convolution** $K = \textbf{TetraConv}(F, G, H, I, J)$**.** Both operators accumulate information from the nearest neighbors in their preceding layers. Operators are depicted in 2D.

representation. Similarly, [24] directly diffuse point clouds conditioned on a latent shape representation generated with a normalizing flow. On the other hand, PolyDiff [2] runs diffusion directly on a soup of (quantized) triangles. LION [41] 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 [28] to turn those into smooth meshes. To circumvent that extra surface extractor, 3DGen [14] introduces a two-stage training pipeline consisting of a triplane VAE and a separately trained latent diffusion model in triplane space. Somewhat similarly, [33] 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 [23], 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 at much lower memory footprint (*cf*. Fig. 2 and Tab. 3).

3 Tetrahedral Representation Learning

Tetrahedral Grid. Following [32], 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 $T \in \mathbb{N}^{K\times 4}$. A nearly regular decomposition can be found via close-packed tetrahedral tiling with the A15 lattice [9]. 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, [32]). In Section 3, we extend the tetrahedral representation to arbitrary feature vectors, such that additional vertex attributes (*e.g.*, color) can be propagated to the final mesh.

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 permits 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 ignore the spatial configuration of the neighborhood, but still eliminates the need to search for the relevant points under the kernel and apply distance weighting, as in KPConv [35]. 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 that clustering always converges to a basis of at most $m+1$ reference directions, where m is the maximal number of neighbors in the tetrahedral grid. Compared to 3D voxel convolutions, the resulting kernel size is more than 40% lower (*e.g.*, 15 compared to 26), a key property for efficient learning. A detailed description of our ordering scheme is given in the Appendix.

Given a vertex v_k and its neighbors $\{v_j, j \in \mathcal{U}_{v_k}\}$ in a fixed ordering, it is straightforward to define the result of a tetrahedral convolution layer l as a weighted sum over their associated feature values on the previous layer, $\Phi^{l-1}_{v_j}$:

$$\Phi^{l}_{v_k} = W_0 \Phi^{l-1}_{v_k} + \sum_{j=1}^{m+1} \mathbb{1}_{j\in\mathcal{U}_{v_k}} W_j \Phi^{l-1}_{v_j}, \tag{1}$$

where W_j are the kernel weights. We refer to this operation as *TetraConv*. As illustrated in Fig. 4a, the varying size of the neighborhood simply corresponds to appropriately zero-padding the kernel. 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 $\mathcal{T}$ at multiple resolutions and establish parent-child relations between vertices in adjacent hierarchy levels via k-nearest neighbor search (using the Euclidean distance). As

shown in Figure 4b, this enables flexible down- and upsampling rates ; and, with appropriate book-keeping about the neighborhood indices $\mathcal{U}_{v,k}$ in Eq. (1), strided convolutions and transposed convolutions, thus going beyond naive upsampling.

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 training data in a lossless manner. Additionally, training and inference speed and memory consumption are easily reduced further by (lossy) pruning of vertices that are occupied by fewer than a user-defined number of instances.

Tetrahedral Diffusion. With all necessary building blocks at hand, we can now assemble our tetrahedral diffusion model. We directly diffuse in deformable tetrahedral space, *i.e.*, our basic input consists of N vertices with features $\{s_v, \Delta_v\}$, plus their associated neighborhood relations. However, we can seamlessly extend the feature vector with further vertex propoerties without losing 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 [18], but with the tetrahedral convolution operators introduced above. *I.e.*, residual convolution layers with group normalization [37] and SiLU activations [15], alternating with attention layers. The network, hyperparameters and extended marching tetrahedra scheme are further described in the Appendix.

4 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 [6] 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 [26] and fit each shape individually into the grid with a combination of rendering and volumetric losses. Similar to [23], the ground truth is created with a two-step procedure, fixing SDF values to $\{-1, 1\}$. For a comprehensive overview of the pre-processing protocol, please refer to the supplementary material.

4.1 Qualitative results

Fig. 5: Column-wise qualitative comparison between *TetraDiffusion* and existing generative mesh models. Our method generates the finest geometry while seamlessly integrating texture. Please zoom in to appreciate the high resolution quality.

Figures 1 and Fig. 5 showcase randomly generated 3D shapes, demonstrating the high quality and level of detail achievable with 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 ($\approx$ 200 examples), the results go beyond memorization and naive interpolation. Parts of different training samples are seamlessly blended into plausible assemblies, and curious designs emerge that clearly were not observed during training. We emphasize that the presented shapes have *not* been post-processed in any way. Cleaning up the output of 3D generative models in post-processing is a common practice, but has side effects. *E.g.*, smoothing would remove fine structures and high-frequency details that the model may have correctly synthesized.

Geometry comparison. Figure 6 compares TetraDiffusion against other recent mesh generators, namely GET3D [12], MeshDiffision (MD) [23], Neural Field Generation (NFD) [33], Neural Wavelet Diffusion (NWD) [19] and SDF- StyleGAN (SDF-SG) [44]. The figure depicts examples from a large pool of generated shapes, matched across methods by finding the nearest neighbors to the same example from the test set. As general trends, we observe that *(i)* the GAN-based methods GET3D and SDF-SG produce noisier, less intricate shapes and have a tendency to hallucinate implausible shape details; *(ii)* TetraDiffusion excels in

generating more precise and complex shapes, with sharper creases, detailed small structures, and smoother surfaces with fewer defects, compared to oversmoothed surfaces produced by e.g. NFD and NWD. *(iii)* our hi-res version with $R = 192$ clearly improves shape quality compared to MD or GET3D with $R = 128$, supporting our claim that *resolution limits has so far been a bottleneck*: existing 3D diffusion models run into hardware limits, and efficient use of memory and compute matters.

Fig. 6: Detailed comparison of geometry among the different methods. Every second row is a zoomed in crop out of the previous row. Samples are gathered based on nearest neighbor to a reference shape from the validation set.

In Fig. 6 this is underlined by direct comparison between TetraDiffusion and other approaches. Our results are the only ones that produce detailed, realistic shapes that are neither oversmoothed nor have apparent geometric flaws.

Texture comparison. GET3D is the only other model that generates colorized meshes, opting to predict color maps directly, whereas our approach assigns colors to vertices. The latter could be seen as a disadvantage, but TetraDif-

fusion's ability to produce high-resolution details also results in superior texture, see Fig. 7. Whereas GET3D tends to create blurry textures (2^{nd}, 4^{th}, 5^{5}th columns), misses small features (front of the car, 4^{th} column), and suffers from misaligned color boundaries (1^{st}, 3^{rd}, 5^{th} columns). There are two reasons for this: On the one hand, a higher vertex density naturally leads to crisper color discontinuities such as numbers on a car body in Fig. 7, especially when vertex placement is trained to jointly reproduce geometry and color. Second, color maps are of limited use unless they are markedly sharper than the geometry *and* well-aligned with shape details.

Fig. 7: Comparison of colorized assets between *TetraDiffusion* and GET3D. Our method is able to produce highly consistent textures with abrupt color changes and diverse patterns. Pairs were selected by finding nearest neighbours in the feature space of a pre-trained ResNet18. Best viewed when zoomed in.

Interpolation. In Fig. 8 we explore the latent shape space learned by TetraDiffision. The model allows for direct interpolation between two different shape instances by spherical interpolation of their generating noise patterns. The intermediate noises are fed into the diffusion model to generate interpolated meshes. The geometric integrity of the intermediate samples and the plausible, gradual transitions suggest that the model indeed disentangles functionally meaningful dimensions and represents objects in terms of those components.

Fig. 8: Row-wise interpolation between two reference shapes: We first create the extreme left and right samples, storing the noise at each diffusion step. Then we use spherical linear interpolation of these noises to produce intermediate shapes.

4.2 Quantitative results

Metrics. We quantitatively evaluate our model in terms of *1-nearest neighbor accuracy* (1NNA), as proposed by Yang *et al.* [39], as well as minimum matching distance (MMD) and coverage (COV) (Tab. 1). Like [41], we compute all metrics based on both Earth Mover's Distance (EMD) and Chamfer Distance (CD). 1-NNA evaluates how closely the generated samples match the distribution of real examples by counting how often the nearest neighbor of a generation result is a real, respectively generated one. 1-NNA scores should lie near 50%, with higher scores suggesting underfitting and lower scores indicating overfitting. MMD measures how far generated examples are from real ones, via the average distance to the nearest reference sample. COV measures how well the generated samples represent the diversity of the ground truth, with higher values indicating better coverage of the ground truth's variability. Additionally, by rendering all the meshes, we evaluate the similarity to the original ShapeNet class through the Fréchet Inception Distance (FID), CLIP-FID, and Kernel Inception Distance (KID) of shaded meshes (Tab. 1). This allows us to assess visual simi-

Table 1: Quantitative evaluation of *TetraDiffusion* and current mesh generators. Metrics are computed over ShapeNet classes airplane, bike, car and chair. MMD-CD is multiplied $\times 10^3$, MMD-EMD $\times 10^2$. We use our high resolution model for all experiments except chairs. † NFD could not be retrained as there is no public code, thus training and test split overlap.

		1-NNA ↓		MMD ↓		COV ↑		shading-FID		
Category	Method	CD	EMD	CD	EMD	CD	EMD	FID ↓	CLIP-FID ↓	KID ↓
Airplane	NFD† [33]	87.3	91.0	0.40	0.79	40.5	23.0	36.9	2.39	2.38
	SDF-SG [44]	98.0	97.3	0.64	1.14	32.8	15.3	80.7	6.66	7.17
	NWD [19]	90.4	90.9	0.44	0.79	36.8	27.9	20.2	2.09	1.35
	GET3D [12]	89.9	93.6	0.49	0.79	35.8	22.7	27.0	5.35	1.56
	MD [23]	92.7	93.7	0.73	1.23	36.1	21.0	82.9	8.78	7.28
	Ours	**64.9**	**85.9**	**0.36**	**0.69**	**44.2**	**33.1**	**15.6**	**1.02**	**0.78**
Car	NFD† [33]	76.3	72.9	1.36	0.80	35.8	**44.6**	143.9	9.60	11.6
	SDF-SG [44]	87.4	82.1	1.47	0.85	29.0	36.1	146.1	13.6	11.4
	NWD [19]	75.9	77.7	1.38	0.82	26.7	31.8	116.3	8.37	9.09
	GET3D [12]	90.9	90.5	1.69	1.02	10.2	19.6	169.9	12.6	15.3
	MD [23]	64.5	77.8	1.29	0.83	**38.1**	38.9	151.5	10.1	13.6
	Ours	**60.1**	**69.9**	**1.26**	**0.78**	34.94	42.61	**107.2**	**5.15**	**8.63**
Chair	SDF-SG [44]	85.2	83.4	4.76	2.82	30.9	29.0	74.1	9.57	4.91
	NWD [19]	65.2	66.0	**3.90**	**2.18**	43.4	45.3	**16.7**	**2.23**	**0.69**
	GET3D [12]	74.7	71.8	5.35	2.68	31.6	36.4	41.3	6.09	2.36
	MD [23]	74.9	70.4	5.28	2.75	38.5	43.3	60.9	9.19	4.33
	Ours	**61.2**	**63.2**	4.90	2.66	**45.6**	**47.2**	39.9	4.17	2.67
Bike	SDF-SG [44]	89.8	88.3	1.77	1.25	20.5	32.1	198.2	23.2	8.41
	NWD [19]	69.1	71.1	1.09	0.96	42.7	47.8	131.0	11.4	10.3
	GET3D [12]	73.7	76.9	1.45	1.04	26.4	38.9	135.8	10.6	10.7
	MD [23]	**54.0**	72.0	**0.99**	**0.93**	**46.6**	44.2	56.0	3.25	4.19
	Ours	58.5	**66.9**	1.17	0.95	40.1	**48.1**	**47.9**	**2.94**	**3.00**

larity between generated meshes (without color) and their respective ShapeNet categories. To assess color fidelity we do the same for colored meshes (Tab. 2).

We compare our model with leading 3D mesh generators, particularly noting GET3D [12] and MeshDiffusion (MD) [23] as direct competitors that also employ tetrahedral grids. Additionally, we benchmark against a GAN-based mesh generator (SDF-SG, [44]), diffusion in the wavelet domain (NWD, [19]) and diffusion on triplanes (NFD, [33]). For NFD we could not retrain and had to use the authors' weights as no training code is available, hence there is an overlap between training and test set. TetraDiffusion and GET3D are the only methods that also generate color.

The results presented in Tab. 1 show that our approach performs very well in a head-to-head comparison. It achieves the top 1-NNA scores (nearing 50) in all cases except for the bikes CD category, where it ranks second. Regarding COV scores, our method outperforms others in 5 out of 8 categories, achieving the highest scores, and takes the second position for Cars EMD. In the evaluation of MMD, our approach leads by registering the smallest distances for both Airplane and Car categories, while it ranks second for Chair and Bike EMD. Additionally, our high-resolution variant, Ours_{192}, outclasses all competing methods in Shading-FID scores (Tab. 1), and our Ours_{128} variant secures the second-highest Shading-FID score in the Chair category. When assessing the texture quality of our generated meshes, TetraDiffusion beats GET3D across all three FID scores for Airplane, Chair, and Bike, as detailed in Tab. 2. We also compared our method to LION and PVD, both of which generate point clouds only. See the supplementary material. TetraDiffusion outperforms both in most of the metrics, too, although the comparison across different native shape representations comes with its own challenges and should be taken with a grain of salt.

Table 2: Quantitative comparison of generative texture quality, evaluated via renderings of colored meshes. We use our high resolution model.

		Texture-FID ↓		
		FID	CLIP-FID	KID
Airplane	GET3D [12]	41.91	11.87	0.023
	Ours	**22.51**	**2.35**	**0.013**
Car	GET3D [12]	**40.30**	**7.0**	**0.023**
	Ours	61.05	8.5	0.041
Chair	GET3D [12]	68.89	18.91	0.043
	Ours	**41.63**	**15.34**	**0.026**
Bike	GET3D [12]	87.79	9.44	0.068
	Ours	**52.24**	**8.65**	**0.029**

4.3 Efficiency

A central asset of TetraDiffusion 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 methods 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 smaller memory footprint, work with 1028 feature channels in the bottleneck, and can also be run at higher resolution.

Table 3: Memory consumption and runtime of generative shape models. For a fair comparison we also implemented 16-bit inference for MD. * denotes TetraDiffusion with pruning. NWD training numbers are for only on the first of two training stages. For NFD there is no public training code.

	Training		Inference	
Method	GPU (GB)	Speed (it/s)	GPU (GB)	Speed (s/obj)
SDF [44]	2.9	2.4	4.0	0.2
NWD [19]	(5.3)	(15.3)	5.6	3.6
NFD [33]	—	—	4.9	23.3
GET3D [12]	13.3	0.1	11.3	0.8
MD(org) [23]	76.6	0.5	29.2	714.3
MD(16 bit) [23]	76.6	0.5	22.6	526.3
Ours$^{*}_{128}$	12.0	2.8	7.4	3.4
Ours$_{128}$	20.8	1.0	9.7	11.2
Ours$^{*}_{192}$	20.9	1.2	11.7	9.1
Ours$_{192}$	78.2	0.3	42.1	33.3

Table 3 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 3 further reduces the memory footprint to 12GB, without compromising prediction quality. Even at high resolution $R = 192$, and assuming that shape variability is so large that no pruning is possible, our model consumes less than 80 GB and can be trained on a single high-end GPU. As a consequence, TetraDiffusion is significantly faster: Inference takes ≈ 3 seconds per sample, $\approx 200\times$ less than MeshDiffusion on the same hardware, and approaching GAN-based methods. Furthermore, the time-continuous formulation of our diffusion model provides the flexibility to use a varying number of sampling steps during inference. In our experiments, described in the supplementary material, we have explored the range of steps and found that we can generate a mesh in 32 diffusion steps without compromising quality.

4.4 Ablations

To demonstrate the importance of the spatial ordering in our tetrahedral convolution, we run control experiments with random ordering of the neighboring nodes, and with graph convolutions that are invariant to the ordering. We define four otherwise identical networks: (1) our standard *Ours*, equipped with TetraConv; (2) *TetraConv$_{Rand}$*, with randomized ordering of the neighbors within each TetraConv layer; (3) *GraphConv* graph convolutions instead of TetraConv; (4) another version of GraphConv, *GraphConv$_{big}$*, with increased number of parameters to match the parameter count of our standard model with TetraConv.

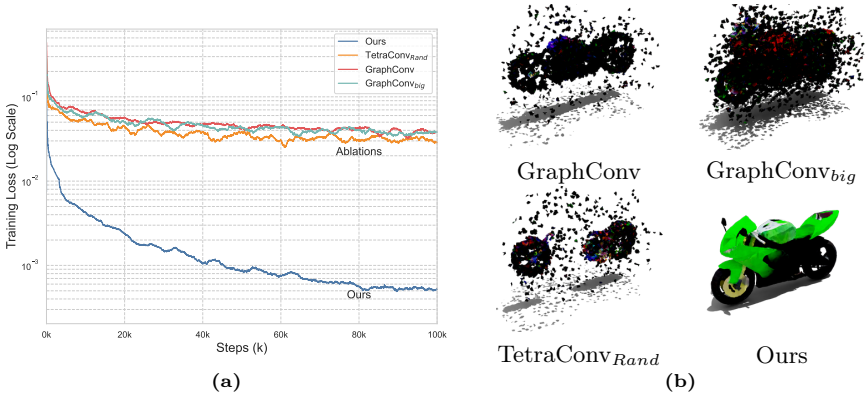

Fig. 9: **(a)** Training loss and **(b)** inference results using graph convolutions (GraphConv and GraphConv$_{big}$), randomized node ordering (TetraConv$_R$) and our spatially correct ordering (Ours).

All variants were trained on the motorbike category, with the same training protocol.

As shown in Fig. 9, TetraConv$_{Rand}$, GraphConv and GraphConv$_{big}$ all fail to converge to meaningful solutions. Only our standard model, which preserves the spatial ordering, is able to produce convincing results. The experiment supports our claim that a convolution operator should (as per its mathematical definition) account for the spatial layout of the function values. We note in particular that graph convolutions, despite their name, are not convolutions in the formal sense, since they combine information from the neighborhood in a permutation-invariant manner, thus discarding the spatial layout.

5 Conclusion

We have presented *TetraDiffusion*, an innovative 3D diffusion framework capable of generating high-resolution, colored triangle meshes with arbitrary topology at unprecedented resolution. To our knowledge, our work is the first 3D diffusion model that natively operates on a tetrahedral data structure, enabling highly efficient training and sampling. Through a series of experiments, we have showcased its ability to synthesize diverse 3D shapes of high geometric quality and the advantages of our tetrahedral convolution over traditional methods such as 3D and graph convolutions.

We believe that a promising avenue for further exploration is to better leverage the extended differentiable tetrahedral marching algorithm and differentiable rendering [22, 26]. This could pave the way towards a seamless integration of large-scale 2D data into the framework. Moreover, *TetraDiffusion* has so far been trained for isolated objects, yet our framework seems a good fit for synthesising entire scenes, with larger extent and often high sparsity.

References

1. Achlioptas, P., Diamanti, O., Mitliagkas, I., Guibas, L.: Learning representations and generative models for 3d point clouds. In: ICML (2018)
2. Alliegro, A., Siddiqui, Y., Tommasi, T., Nießner, M.: Polydiff: Generating 3d polygonal meshes with diffusion models. arXiv preprint arXiv:2312.11417 (2023)
3. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Generative and discriminative voxel modeling with convolutional neural networks. arXiv:1608.04236 (2016)
4. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. arXiv:1312.6203 (2013)
5. Cai, R., Yang, G., Averbuch-Elor, H., Hao, Z., Belongie, S., Snavely, N., Hariharan, B.: Learning gradient fields for shape generation. In: ECCV (2020)
6. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An information-rich 3d model repository. arXiv:1512.03012 (2015)
7. Chen, Z., Zhang, H.: Learning implicit fields for generative shape modeling. In: CVPR (2019)
8. Dhariwal, P., Nichol, A.: Diffusion models beat GANs on image synthesis. In: NeurIPS (2021)
9. Doran, C., Chang, A., Bridson, R.: Isosurface stuffing improved: acute lattices and feature matching. In: ACM SIGGRAPH (2013)
10. Gadelha, M., Wang, R., Maji, S.: Multiresolution tree networks for 3d point cloud processing. In: ECCV (2018)
11. Gao, J., Chen, W., Xiang, T., Jacobson, A., McGuire, M., Fidler, S.: Learning deformable tetrahedral meshes for 3d reconstruction. In: NeurIPS (2020)
12. Gao, J., Shen, T., Wang, Z., Chen, W., Yin, K., Li, D., Litany, O., Gojcic, Z., Fidler, S.: Get3d: A generative model of high quality 3d textured shapes learned from images. NeurIPS (2022)
13. Gao, W., Wang, A., Metzer, G., Yeh, R.A., Hanocka, R.: TetGAN: A convolutional neural network for tetrahedral mesh generation. arXiv:2210.05735 (2022)
14. Gupta, A., Xiong, W., Nie, Y., Jones, I., Oğuz, B.: 3DGen: Triplane latent diffusion for textured mesh generation. arXiv:2303.05371 (2023)
15. Hendrycks, D., Gimpel, K.: Gaussian error linear units (GELUs). arXiv:1606.08415 (2016)
16. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. In: NeurIPS (2020)
17. Ho, J., Salimans, T.: Classifier-free diffusion guidance. arXiv:2207.12598 (2022)
18. Hoogeboom, E., Heek, J., Salimans, T.: Simple diffusion: End-to-end diffusion for high resolution images. arXiv:2301.11093 (2023)
19. Hui, K.H., Li, R., Hu, J., Fu, C.W.: Neural wavelet-domain diffusion for 3d shape generation. In: SIGGRAPH Asia (2022)
20. Jacobson, A., Baran, I., Popovic, J., Sorkine, O.: Bounded biharmonic weights for real-time deformation. ACM ToG **30**(4), 78 (2011)
21. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv:1609.02907 (2016)
22. Laine, S., Hellsten, J., Karras, T., Seol, Y., Lehtinen, J., Aila, T.: Modular primitives for high-performance differentiable rendering. ACM ToG **39**(6), 1–14 (2020)
23. Liu, Z., Feng, Y., Black, M.J., Nowrouzezahrai, D., Paull, L., Liu, W.: MeshDiffusion: Score-based generative 3d mesh modeling. In: ICLR (2023)

24. Luo, S., Hu, W.: Diffusion probabilistic models for 3d point cloud generation. In: CVPR (2021)
25. Mo, S., Xie, E., Chu, R., Yao, L., Hong, L., Nießner, M., Li, Z.: DiT-3D: Exploring plain diffusion transformers for 3d shape generation. arXiv:2307.01831 (2023)
26. Munkberg, J., Hasselgren, J., Shen, T., Gao, J., Chen, W., Evans, A., Müller, T., Fidler, S.: Extracting triangular 3d models, materials, and lighting from images. In: CVPR (2022)
27. Paillé, G.P., Ray, N., Poulin, P., Sheffer, A., Lévy, B.: Dihedral angle-based maps of tetrahedral meshes. ACM ToG **34**(4), 1–10 (2015)
28. Peng, S., Jiang, C., Liao, Y., Niemeyer, M., Pollefeys, M., Geiger, A.: Shape as points: A differentiable Poisson solver. In: NeurIPS (2021)
29. Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., Chen, M.: Hierarchical text-conditional image generation with CLIP latents. arXiv:2204.06125 (2022)
30. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: CVPR (2022)
31. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: MICCAI (2015)
32. Shen, T., Gao, J., Yin, K., Liu, M.Y., Fidler, S.: Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In: NeurIPS (2021)
33. Shue, J.R., Chan, E.R., Po, R., Ankner, Z., Wu, J., Wetzstein, G.: 3d neural field generation using triplane diffusion. In: CVPR (2023)
34. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. In: ICLR (2021)
35. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: KPConv: Flexible and deformable convolution for point clouds. In: ICCV (2019)
36. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: ICML (2019)
37. Wu, Y., He, K.: Group normalization. In: ECCV (2018)
38. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d ShapeNets: A deep representation for volumetric shapes. In: CVPR (2015)
39. Yang, G., Huang, X., Hao, Z., Liu, M.Y., Belongie, S., Hariharan, B.: PointFlow: 3d point cloud generation with continuous normalizing flows. In: ICCV (2019)
40. Zamorski, M., Zieba, M., Klukowski, P., Nowak, R., Kurach, K., Stokowiec, W., Trzciński, T.: Adversarial autoencoders for compact representations of 3d point clouds. CVIU **193**, 102921 (2020)
41. Zeng, X., Vahdat, A., Williams, F., Gojcic, Z., Litany, O., Fidler, S., Kreis, K.: LION: Latent point diffusion models for 3d shape generation. arXiv:2210.06978 (2022)
42. Zhang, B., Tang, J., Niessner, M., Wonka, P.: 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. ACM Transactions on Graphics (TOG) **42**(4), 1–16 (2023)
43. Zheng, X.Y., Pan, H., Wang, P.S., Tong, X., Liu, Y., Shum, H.Y.: Locally attentional sdf diffusion for controllable 3d shape generation. ACM Transactions on Graphics (ToG) **42**(4), 1–13 (2023)
44. Zheng, X., Liu, Y., Wang, P., Tong, X.: SDF-StyleGAN: Implicit SDF-based StyleGAN for 3d shape generation. In: Computer Graphics Forum. vol. 41, pp. 52–63 (2022)
45. Zhou, L., Du, Y., Wu, J.: 3d shape generation and completion through point-voxel diffusion. In: ICCV (2021)