TetraDiffusion: Tetrahedral Diffusion Models for 3D Shape Generation

Supplementary Material

A. Dataset preprocessing

We employ an isosurface stuffing algorithm with the A15 tile [10] to tessellate a cube $[-1, 1]^3$ at the necessary grid resolutions using Quartet https://github.com/crawforddoran/quartet. Since our algorithm can handle arbitrary edge configurations, there is no need for post-processing non-symmetrical boundaries. In our experiments, we generate grids with tetrahedral resolutions of 32, 48, 64, 96, 128, and 192.

Our network requires SDF values, deformation vectors and color vectors. In order to convert meshes with possible texture maps, we adapt the optimization pipeline of [30] (*nvdiffrec* https://github.com/NVlabs/nvdiffrec/). In particular, we include per vertex color information and extend the marching tetrahedra algorithm to extract the corresponding surface color (see Sec. C.2). Originally, *nvdiffrec* jointly optimizes geometry and materials, stored in 2D textures, from multi-view object renderings. In our case, material information must be stored in the vertex features to jointly diffuse geometry and texture in a single U-Net, *i.e.* we implement a custom render function that additionally rasters and interpolates the vertex color information extracted from our extended marching tetrahedra. We iteratively optimize SDF value, deformation and color, starting from a random initialization, rendering random views uniformly around a sphere in each step. The vertex features are updated by backward propagation of different 2D and 3D loss functions, making use of the differentiable marching algorithm and differentiable rendering. In particular, we define the following total loss with respect to an optimizable mesh M_{Opt} extracted from the updated SDF, deformations and colors and the ground truth mesh M_{Gt}.

$$\mathcal{L}(M_{\text{Opt}}, M_{\text{Gt}}) = \lambda_1 \mathcal{L}_{\text{Img}} + \lambda_2 \mathcal{L}_{\text{Normal}} + \lambda_3 \mathcal{L}_{\text{Depth}} + \lambda_4 \mathcal{L}_{\text{Mask}} + \lambda_5 \mathbb{1}_{2^{nd}} \mathcal{L}_{\text{Laplace}} + \lambda_6 \mathcal{L}_{\text{Reg}}. \tag{7}$$

Similar to [26], the vertex features are updated in a two-stage procedure, each consisting of 5000 iterations in total. In a first step, we limit the deformation vector to a range of 0.45 of the grid resolution, which results in an increase in the number of vertices for the extracted mesh. A higher number of vertices allows better alignment of the ground truth shape in the second step, where we fix all SDF values to their respective sign, *i.e.* SDF $\in \{-1, 1\}$. We compensate for the restriction by increasing the range of the deformation vector to twice the grid resolution in the second step.

$\boldsymbol{\mathcal{L}_{\text{Img}}(M_{\text{Opt}}, M_{\text{Gt}})}$: We extract the two closest surfaces for each pixel in a raster using depth peeling, which allows us to optimize inner structure as well (see Sec. B.1). After interpolation and antialiasing, this gives us two pairs of RGB images $\{(\text{Img}_1^{\text{gt}}, \text{Img}_1^{\text{opt}}), (\text{Img}_2^{\text{gt}}, \text{Img}_2^{\text{opt}})\}$ For which we compute gradients w.r.t. an L1-loss and $\lambda_1 = 10$.

$\boldsymbol{\mathcal{L}_{\text{Normal}}(M_{\text{Opt}}, M_{\text{Gt}})}$: Similar to the RGB images, we extract vertex (smooth shading) and face normals (flat shading) For both depth peels and compute gradients w.r.t. an L1-loss of ground truth and optimizable normals and $\lambda_2 = 10$ For the first layer and $\lambda_2 = 0.1$ For the second layer with slight abuse of notation. Smooth vertex normals are computed as the normalized average of the involved surface normals.

$\boldsymbol{\mathcal{L}_{\text{Depth}}(M_{\text{Opt}}, M_{\text{Gt}})}$: We compute normalized depth maps For both raster layers and compute the corresponding L1-loss. We weight the depth map with $\lambda_3 = 100$.

$\boldsymbol{\mathcal{L}_{\text{Mask}}(M_{\text{Opt}}, M_{\text{Gt}})}$: We render the silhouette of our shape For the first depth peel only, since the silhouette of the second layer does not provide much more information in general. Again, we use the L1-loss.

$\boldsymbol{\mathcal{L}_{\text{Chamfer}}(M_{\text{Opt}}, M_{\text{Gt}})}$: We compute the Chamfer distance w.r.t. to M_{Gt} and M_{Opt} and weight it with $\lambda_4 = 1$.

$\boldsymbol{\mathcal{L}_{\text{Laplace}}(M_{\text{Opt}}, M_{\text{Gt}})}$: To regularize the extracted triangular mesh, we add Laplacian smoothing using the weighted umbrella operator in the second stage of our optimization. The loss is attenuated over time.

$\boldsymbol{\mathcal{L}_{\text{Reg}}(M_{\text{Opt}}, M_{\text{Gt}})}$: To diminish floaters, we penalize sign flips in the SDF values with a cross-entropy loss defined over neighboring vertices, following [13, 30].

Additional to the aforementioned regularization loss, we encountered small floaters far away from the actual surface every now and then. In order to remove them, we simply compute the convex hull of our ground truth mesh and enforce all SDF values outside the hull to be positive.

B. Additional qualitative results

B.1. High resolution and inner structure

Our efficient architecture is capable of generating meshes in unprecedented resolution, capturing intricate details such as chains, fine rims, brake disks and brake handles. Importantly, these details are represented in the geometry of the mesh and not in texture maps. Diffusing per vertex texture information enforces the strong geometry. This capability is demonstrated in detail in Figure 7, where the visualizations showcase the model's proficiency in reproducing realistic and fine-grained elements, contributing to the overall realism and quality of the generated shapes.

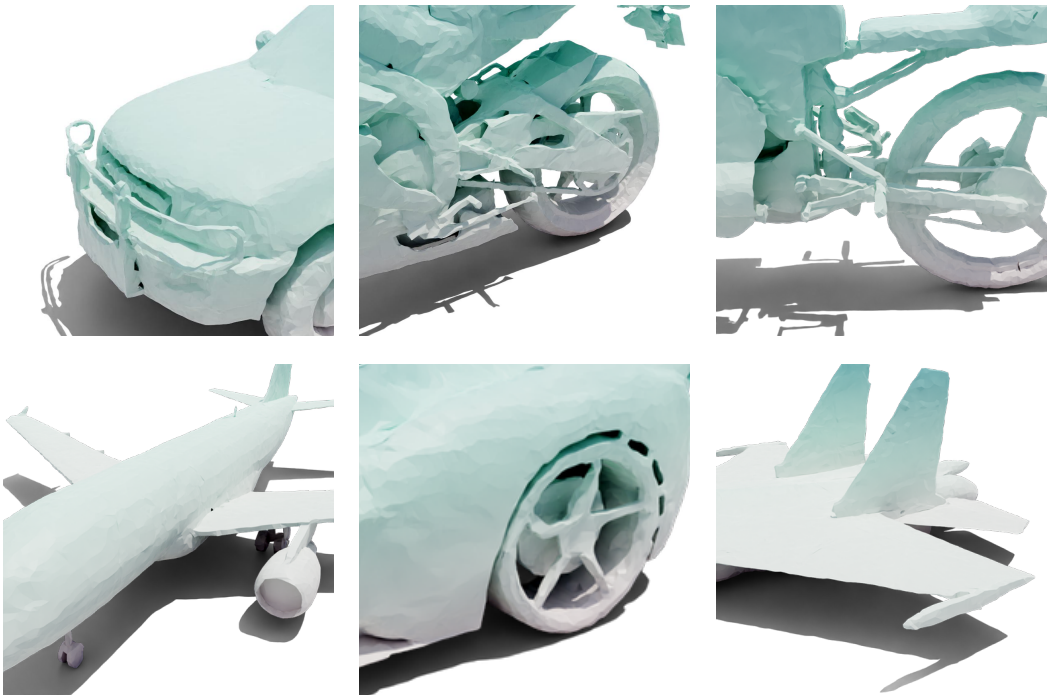

Figure 7. Details of generated meshes in high resolution

Additionally, our ground truth contains inner structure, as explained in Sec. A. Our models are able to generate consistent inner structure, *e.g.* seats, steering wheels or engines. Exemplary shapes are showcased in Figure 8.

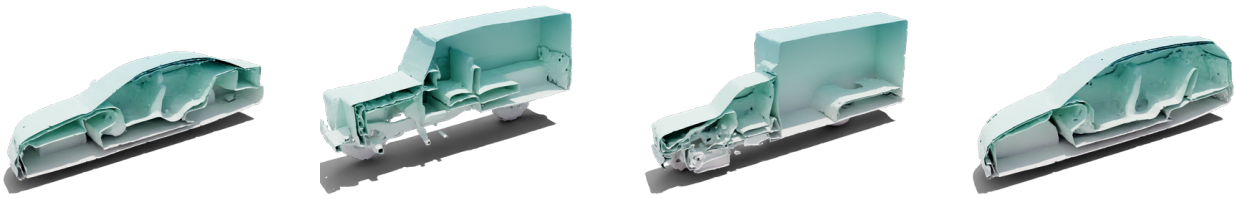

Figure 8. Examples of inner structure of our generated car meshes in high resolution

Figure 9. Shapes generated from a single view using our conditional model that was trained simultaneously on airplanes, bikes, cars, chairs and tables.

B.2. Conditional generation

While our primary goal was not to develop a strong single-view reconstruction (SVR) model, TetraDiffusion can be easily extended to conditional generation. During training, we condition the U-Net on CLIP embeddings from multiple rendered views of ShapeNet meshes and use the CLIP feature of a single novel view during inference. Our approach requires only RGB images and offers flexibility with text embeddings. Note that we do not incorporate additional test time guidance. We depict samples in Figure 9.

B.3. Test time guidance

Similar to classifier and classifier-free guidance [18, 37], we incorporate test time guidance, *i.e.* altering the mean prediction of the reverse diffusion process (*cf*. Equation (3)) with the gradient of a guiding task. *e.g.* In image diffusion, the guiding task can be a classifier $p(y|\mathbf{z_t})$ trained to predict the correct class label y given the latent variable $\mathbf{z_t}$. Instead of training an extra model or training a classifier-free conditional model, we explore the robustness of non-parametric guidance tasks, in the sense that no additional training is required, besides the unconditional model. In particular, we experiment with color and volumetric gradients making use of our fully differentiable mesh and rendering pipeline.

In order to guide the diffusion towards a specific colorized texture, we project the color in text form to its CLIP embedding and compare the embedding to image embeddings obtained by rendering multiple views of $\hat{\mathbf{x}}_\theta(\mathbf{z_t}; t)$ in every step. The loss is computed as a spherical distance loss following [8]:

$$\mathcal{L}(e_{\text{color}}, e_{\text{render}}) = 2 \cdot \left[\arcsin\left(\frac{\|e_{\text{color}} - e_{\text{render}}\|}{2} \right) \right]^2, \tag{8}$$

where $e_{\text{color}} = \text{CLIP}(\text{“color”})$ is the CLIP embedding of a color in text form and $e_{\text{render}} = \Phi(\theta(\mathbf{z_t}; t))$ is a rendered view of the mesh extracted from $\hat{\mathbf{x}}_\theta(\mathbf{z_t}; t)$ with our extended marching tetrahedra algorithm.

Additionally, we may influence the volume of the diffused mesh by simply pushing the bounding box of the extracted mesh towards smaller or larger extents.

Of course these guidance tasks are orthogonal to each other and can be combined to change volume and color at the same time due to our somewhat disentangled representation. We depict guided samples in Figure 10. As pointed out by [31], this

simple approach may result in slightly deteriorate results as the renderings of noisy meshes are out of distribution For the CLIP model.

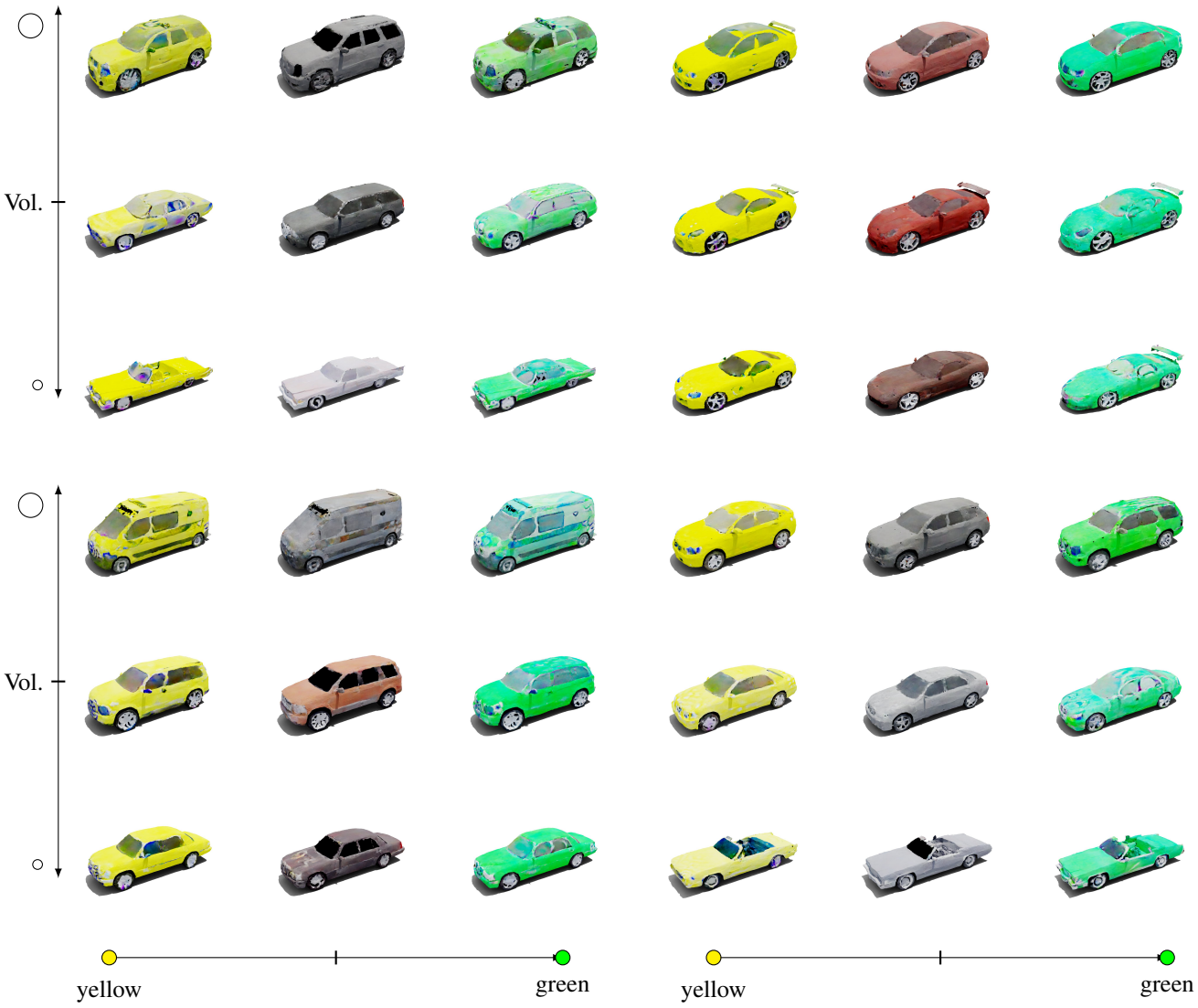

Figure 10. Exemplary results using text and volume guidance using only the unconditional model.

B.4. Unconditional generation

Additional renderings of shapes generated with TetraDiffusion can be found in Fig. 14, Fig. 16, Fig. 12, Fig. 17, Fig. 13, Fig. 15 and Fig. 11. TetraDiffusion consistently produces high-quality results with substantial diversity. Notably, even in low data regimes such as the ShapeNet bike class, our model learns a meaningful manifold and reliably interpolates novel shapes.

Figure 11. A random selection of cars generated in high resolution.

Figure 12. A random selection of cars generated in our standard resolution.

Figure 13. A random selection of airplanes generated in high resolution.

Figure 14. A random selection of airplanes generated in our standard resolution.

Figure 15. A random selection of motorbikes generated in high resolution.

Figure 16. A random selection of motorbikes generated in standard resolution.

Figure 17. A random selection of chairs generated in standard resolution.

B.5. Interpolation

Here, we perform shape interpolation, a common task in generative models. We smoothly transform one shape to another by interpolating along the noise vectors. Let the noise vector of shape i be $z_{i,t}$ at time step t. Common in 2D diffusion is to simply linearly interpolate between two noise vectors $z_{1,t}$ and $z_{2,t}$. However, similar to [53], we find that a simple convex combination of the noise vectors leads to poor results in 3D, *e.g.* holes, deformations or corrupted colors. The authors of [53] provide a precise explanation: due to the high dimensionality of the input space, the noise samples almost certainly lie on a thin spherical shell according to the Gaussian annulus theorem. Therefore, they propose to use a square root-based interpolation, in order to stay within the training set learned by the model.

Another, simpler reason is that the linear interpolation of two standard Gaussians is not a standard Gaussian anymore. For $i = 1, 2$ let $X_i \sim \mathcal{N}(0, I)$ be independent Gaussian random variables and $\lambda_1 = 1 - \lambda_2$, $\lambda_i \in [0, 1]$. Then $X \coloneqq \lambda_1 X_1 + \lambda_2 X_2 \sim \mathcal{N}(0, \sigma^2 \coloneqq \lambda_1^2 + \lambda_2^2)$, *i.e.* in particular $\sigma^2 = 1 \iff \lambda_i = 1$ (see [47] For a more general proof). In particular, we end up with a non-isotropic Gaussian between the end values. To have have isotropic Gaussian samples in between, one needs to take the square root of the interpolation weights, as proposed by [53].

Additionally, we found that morphing with the square root-based interpolation resulted in sudden jumps between shapes. Instead, we have used spherical interpolation (Slerp) as defined in the following:

$$\hat{z}_{t,k} \coloneqq \text{Slerp}(z_{t,0}, z_{t,1}; k) = \frac{\sin[(1-k)\Omega]}{\sin \Omega} z_{t,0} + \frac{\sin[k\Omega]}{\sin \Omega} z_{t,1}, \tag{9}$$

where Ω is the subtended angle between the arc that is spanned by each $z_{t,0}$ and $z_{t,1}$ on the sphere and $k \in [0, 1]$. In order to interpolate between two shapes we interpolate all noise vectors using equation Eq. (9) along the trajectory, resulting in smooth interpolations as we are explicitly moving along the arc on the high dimensional sphere. See Figure 18 for various interpolations of airplanes, cars and chairs.

Figure 18. Exemplary 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.

C. Tetrahedral Diffusion

C.1. Optimized Edge Binning Algorithm

Algorithm 1 Optimized Binning for Tetrahedral Grid Convolution

Require: Grid vertices V, cluster count $k = m + 1$, max number of iterations n
Ensure: Ordered edge sets for each vertex

```
Initialize k-means with k clusters
for each vertex v ∈ V do
    Collect outgoing edges E_v centered at v
end for
Cluster all edges E = ⋃_v E_v into k groups
for each vertex v ∈ V do
    for each edge e ∈ E_v do
        Assign e to closest cluster based on cosine similarity
    end for
    Sort assigned edges of v based on cluster order
end for
Check for collisions in edge assignments
if collisions exist then
    if current iteration < n then
        repeat clustering and sorting
    else
        Increment k and repeat clustering and sorting
    end if
end if
```

As already outlined in the main paper, the hybrid tetrahedral representation cannot be manipulated with regular grid-based convolution, since the neighborhood of a vertex in the grid can vary. However, even though the tetrahedral grid initialized with Quartet is relatively regular, it is not straightforward to establish a discrete binning of the local edge orientations that would define a unique, collision-free ordering. To resolve this we propose to globally optimize the binning in such a way that a unique ordering is ensured, see Alg. 19

1. Collect the set of outgoing edges at all vertices of the (undeformed) grid, center them in one point, and cluster them into $k = m + 1$ groups with k-means. The cluster centers serve as a global basis of reference directions.
2. Arrange the basis vectors in a fixed order. At every vertex, assign the outgoing edges to basis vectors according to the cosine similarity to obtain a spatially meaningful ordering For convolution.
3. If, at any vertex, the binning would lead to a collision, increment k to expand the number of basis vectors and repeat clustering and assignment.

In practice, we have never seen a case where a collision-free basis needed more than $m + 1$ reference directions.

C.2. Tetrahedral marching extension

The differentiable Marching Tetrahedra algorithm has been popularized by [41] and utilized in various works [13–15, 30]. Similar to the Marching Cube scheme, that algorithm individually inspects every tetrahedron to determine the surface topology. Whenever the sign of the SDF changes within a tetrahedron, the surface must pass through it and one can extract the corresponding triangle by interpolating the vertex positions:

$$v_{ab} = \frac{(v_a + \Delta_{v_a}) \cdot s_b - (v_b + \Delta_{v_b}) \cdot s_a}{s_b - s_a}, \tag{10}$$

where s_x is the SDF value of vertex x and Δ_{v_x} its corresponding deformation vector, $x \in \{a, b\}$.

In our case, we not only want to diffuse untextured meshes, but include texture information without using multiple networks. Luckily, extending the Marching Tetrahedra algorithm to extract any mesh vertex feature is as simple as adding features in the parenthesis in Eq. (10), *i.e.* $(v_x + \Delta_{v_x} + c_{v_x} + ... + *_{v_a})$, where c_{v_x} is *e.g.* color information.

C.3. Network

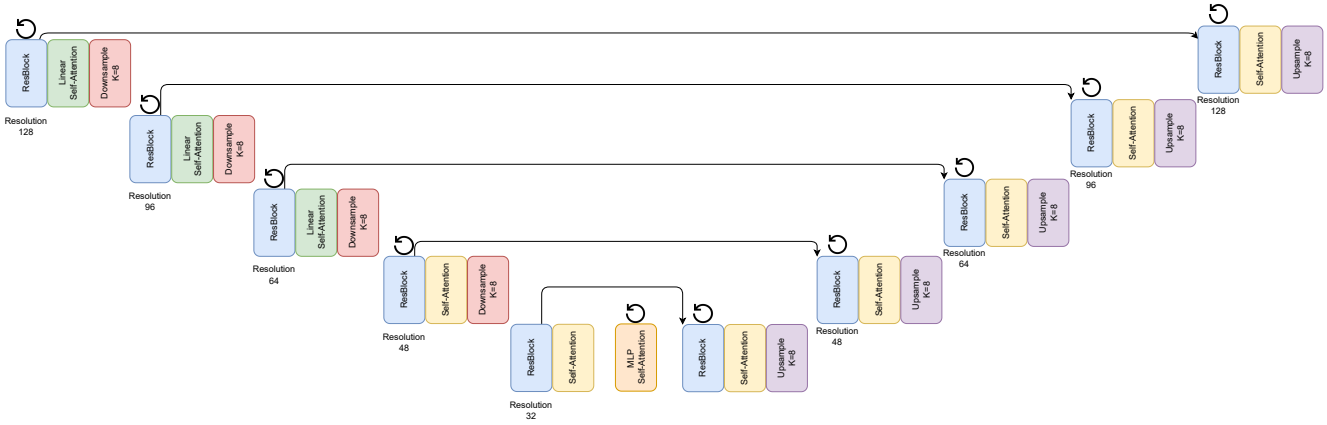

Figure 19. Schematic of our U-Vit with 128 Resolution. Every ResBlock, Downsample and Upsample contain TetraConv blocks.

The general structure of our network is depicted in Fig 19. Our standard network (resolution 128) has five encoder stages with $\{128, 128, 128, 512, 1024\}$ channels. Each stage consists of two residual blocks with tetrahedral convolutions followed by a linear attention layer [46]. In the last two stages the linear attention layers are substituted with regular self-attention layers. Our bottleneck comprises four Transformer blocks, each with self-attention followed by a Multilayer Perceptron (MLP). The time parameter t is embedded via feature modulation [34]. Our downsampling rate is 0.75, *e.g.* going from a 128-cube to a 96-cube. The lowest resolution is 32. In the following, we list the most important training parameters.

```
cube_res: [128, 96, 64, 48, 32]
first_channel: 128
vit_depth: 4
num_blocks: [2, 2, 2, 2, 2]
channel_multiplier: [1, 1, 1, 4, 8]
dropout: [0.0, 0.0, 0.0, 0.1, 0.1]
linear_attentions: [True, True, True, True, False]
attentions: [False, False, False, False, True]
group_norm_groups: 32
knn_up_downsample: 8
ema_decay: 0.995
lr: 1.e-4
```

Differently, for our high resolution network, *i.e.* a resolution of 192, we directly downsample from 192 to 128, as to reduce memory requirements. However, to increase information flow, we use a nearest neighborhood size of 16 instead of 8 when defining the downsampling operation. Additionally, we only use linear attention layers to accelerate sampling. We have not encountered a loss in performance using the higher downsampling rate and only linear attention layers. We only list parameters that have changed below.

```
cube_res: [192,128,64,48,32]
channel_multiplier: [1, 1, 2, 4, 6]
linear_attentions: [True, True, True, True, True]
attentions: [False, False, False, False, False]
knn_up_downsample: 16
```

We have found noise shift to be an important parameter when diffusing high resolution shapes. Following [20], we shift the noise scheduler by a factor of $\frac{128}{32} = 4$, which increases the noise level throughout the diffusion processes. We hypothesize

that the combination of noise shift, ν-parametrization and the time-continuous formulation allows us to sample high quality shapes within 24 steps.

C.4. Pruning

Similar to other volumetric representations, the tetrahedral grid is typically sparsely populated. This reflects the observation that only a small portion of 3D space is in close proximity to an object's surface. However, our fully tetrahedral formulation allows us to alleviate this disadvantage, boosting computational efficiency. To do so, we systematically remove sections of the grid that are infrequently or never used. Unlike regular 3D convolutions, which permit only axis-parallel cropping, our tetrahedral convolutions enable more precise and efficient elimination. In data space, *i.e.* in the first layer of our architecture, unused vertices are directly removed from the input grid, with subsequent updating of the vertices' neighborhoods. On the contrary, this procedure becomes notably intricate in the lower layers. Here, not only must we update the vertices, neighborhoods, and tetrahedrons of the remaining vertices from the preceding layer, but we also need to ensure that vertices within the current layer's receptive field from those layers are preserved as downsampling is defined over nearest neighbors. Our proposed methodology initiates at the data level, applying a global occupancy mask (indicating vertex usage with 1 or 0). We propagate this mask to the lower resolution grids using our TetraConv and downsampling layers. Then, we iteratively process each level, removing unused connections (responses are all zero) and updating affected vertices, including their neighborhoods and tetrahedral indices. Conceptually, deleting a vertex corresponds to removing a row, and removing an edge equates to inserting a zero in a cell within the unfolded data matrix, as depicted in Figure 3. This entire refinement process is executed once, requiring only a few seconds, and is depending on the data set and the specific network configuration to be trained. It's important to note that the pruning of vertices and connections in our methodology is non-lossy, meaning it neither compromises network performance nor results in data loss. This technique is particularly beneficial for high-resolution data, which is often sparsely populated. We can significantly accelerate both training and inference times without sacrificing information. In our experiments, the proportion of utilized data ranged from 8% (when trained on a single class) to 82% (when training across all available classes, such as airplanes, bikes, cars, and chairs).

D. Quantitative evaluation

D.1. Metrics

We evaluate our model in terms of 1-nearest neighbor accuracy (1NNA), minimum matching distance (MMD) and coverage (COV). All metrics compare a set of reference point clouds S_r to a generated set of point clouds S_g. Importantly, the sets should be of same size, *i.e.* $|S_r| = |S_g|$. In all our experiments, the individual point clouds are of size 2048.

1NNA measures diversity and quality of the generated samples. It is defined as the leave-one-out accuracy of the 1-nearest neighbor classifier over the union of the two sets. In particular, a value of 50% is considered optimal: half of the generated samples are closest to a reference sample and the other half is closest to a generated sample. Values above 50% correlate with underfitting - generated samples are further away from the ground truth - and values below 50% coincide with overfitting. The score can be computed with the following equation:

$$\text{1NNA}(S_g, S_r) = \frac{\sum_{x \in S_g} \mathbb{1}[\mathcal{N}_x \in S_g] + \sum_{y \in S_r} \mathbb{1}[\mathcal{N}_y \in S_r]}{|S_r| + |S_g|}, \tag{11}$$

where $\mathcal{N}_x$ is the nearest neighbor of x in the set $S_r \cup S_g - \{x\}$. The nearest neighbor can be computed in terms of Chamfer Distance (CD) or Earth Mover Distance (EMD).

MMD computes the minimal distance between each reference sample and the nearest generated sample using CD or EMD and averages the score over the reference set:

$$\text{MMD}(S_g, S_r) = \frac{1}{|S_r|} \sum_{y \in S_r} \min_{x \in S_g} D(x, y), \tag{12}$$

where $D(\cdot, \cdot)$ is either CD or EMD.

COV quantifies the portion of reference samples that are matched to a generated samples. Higher values indicate diversity in the generative model. Mathematically, this translates to

$$\text{COV}(S_g, S_r) = \frac{1}{|S_g|} |\{\arg\min_{y \in S_r} D(x, y) | x \in S_g\}|, \tag{13}$$

where $D(\cdot, \cdot)$ is either CD or EMD.

D.2. Quantitative results

We provide additional quantitative comparisons using different sampling strategies and reference datasets, namely sampling point clouds with inner structure (Tab. 5) and a reference set sampled from our ground truth meshes with inner (Tab. 3) and only outer surface (Tab. 4).

Category	Method	1-NNA ↓		MMD ↓		COV ↑	
		CD	EMD	CD	EMD	CD	EMD
Airplane	GET3D [13]	93.51 ± 0.48	76.31 ± 0.86	0.43 ± 0.00	0.57 ± 0.00	34.86 ± 0.89	41.14 ± 1.17
	MD [26]	89.6 ± 0.98	87.56 ± 1.25	0.64 ± 0.03	0.81 ± 0.02	33.01 ± 1.67	35.48 ± 1.70
	Ours	**73.38** ± 1.21	**63.51** ± 1.89	**0.31** ± 0.01	**0.51** ± 0.01	**46.22** ± 1.41	**51.75** ± 1.07
Bike	GET3D [13]	72.98 ± 3.01	59.85 ± 5.58	1.48 ± 0.04	**1.19** ± 0.04	41.01 ± 5.26	48.18 ± 5.92
	MD [26]	**65.25** ± 4.92	61.31 ± 6.19	**1.44** ± 0.11	1.22 ± 0.07	42.63 ± 5.21	**55.35** ± 5.39
	Ours	65.96 ± 4.53	**57.22** ± 5.71	1.55 ± 0.06	1.23 ± 0.06	**46.16** ± 5.20	53.74 ± 4.90
Car	GET3D [13]	96.96 ± 0.36	90.21 ± 0.76	1.50 ± 0.01	1.08 ± 0.02	9.66 ± 0.52	19.0 ± 0.90
	MD [26]	74.8 ± 1.59	67.39 ± 1.09	1.14 ± 0.01	0.80 ± 0.01	34.01 ± 0.59	41.12 ± 1.47
	Ours	**69.08** ± 1.37	**59.00** ± 1.91	**1.08** ± 0.01	**0.74** ± 0.01	**36.68** ± 1.45	**47.59** ± 1.80
Chair	GET3D [13]	78.22 ± 0.49	72.35 ± 0.67	5.13 ± 0.03	2.59 ± 0.01	31.59 ± 0.46	37.04 ± 0.47
	MD [26]	73.21 ± 0.89	72.85 ± 0.70	5.01 ± 0.09	2.77 ± 0.05	38.95 ± 0.89	41.71 ± 1.20
	Ours	**68.18** ± 0.86	**65.52** ± 1.14	**4.72** ± 0.09	**2.53** ± 0.02	**42.08** ± 0.84	**47.64** ± 0.98

Table 3. Generation metrics on ShapeNet classes airplane, bike, car and chair, sampling inner and outer structure. Reference dataset was sampled from our ground truth. MMD-CD is multiplied by $\times 10^3$, EMD by $\times 10^2$.

Category	Method	1-NNA ↓		MMD ↓		COV ↑	
		CD	EMD	CD	EMD	CD	EMD
Airplane	GET3D [13]	93.44 ± 0.80	70.51 ± 1.51	0.43 ± 0.01	0.49 ± 0.00	36.81 ± 1.29	47.60 ± 1.17
	MD [26]	89.84 ± 0.70	87.04 ± 0.88	0.63 ± 0.04	0.71 ± 0.02	33.25 ± 1.23	34.02 ± 1.48
	Ours	**72.26** ± 1.76	**63.27** ± 1.61	**0.30** ± 0.00	**0.46** ± 0.00	**47.75** ± 1.00	**48.94** ± 0.97
Bike	GET3D [13]	73.79 ± 3.63	66.87 ± 5.48	1.66 ± 0.03	**1.10** ± 0.04	39.29 ± 4.81	50.10 ± 4.89
	MD [26]	**63.43** ± 3.46	**61.92** ± 6.82	**1.59** ± 0.15	1.11 ± 0.07	42.42 ± 5.51	52.12 ± 6.86
	Ours	64.95 ± 5.45	62.47 ± 6.75	1.72 ± 0.07	1.13 ± 0.06	**45.35** ± 6.29	**54.34** ± 5.62
Car	GET3D [13]	86.50 ± 0.44	70.57 ± 1.61	1.02 ± 0.00	0.59 ± 0.00	20.41 ± 0.98	38.05 ± 1.68
	MD [26]	68.40 ± 1.36	60.97 ± 1.58	0.93 ± 0.01	0.57 ± 0.00	34.24 ± 1.43	44.53 ± 1.60
	Ours	**65.93** ± 1.13	**53.40** ± 1.02	**0.89** ± 0.01	**0.55** ± 0.00	**35.36** ± 0.93	**45.07** ± 1.05
Chair	GET3D [13]	76.89 ± 0.75	70.38 ± 0.66	5.29 ± 0.02	2.37 ± 0.01	31.90 ± 0.46	37.55 ± 0.70
	MD [26]	72.22 ± 0.86	80.09 ± 0.62	5.12 ± 0.08	2.64 ± 0.04	38.82 ± 1.20	38.02 ± 0.58
	Ours	**61.28** ± 0.95	**59.58** ± 0.77	**4.93** ± 0.10	**2.36** ± 0.03	**46.30** ± 0.41	**48.83** ± 0.90

Table 4. Generation metrics on ShapeNet classes airplane, bike, car and chair. Only the outer surface was sampled. Reference dataset was sampled from our ground truth. MMD-CD is multiplied by $\times 10^3$, EMD by $\times 10^2$.

Category	Method	1-NNA ↓		MMD ↓		COV ↑	
		CD	EMD	CD	EMD	CD	EMD
Airplane	GET3D [13]	94.36 ± 0.47	87.30 ± 0.89	0.43 ± 0.00	0.71 ± 0.01	37.48 ± 0.82	35.33 ± 1.49
	MD [26]	89.73 ± 1.27	89.4 ± 0.78	0.65 ± 0.04	0.91 ± 0.02	34.25 ± 1.54	35.23 ± 1.53
	Ours	**80.15** ± 1.14	**77.35** ± 1.18	**0.33** ± 0.01	**0.62** ± 0.01	**45.58** ± 1.63	**45.65** ± 1.19
Bike	GET3D [13]	74.95 ± 2.95	68.19 ± 3.77	1.46 ± 0.04	1.37 ± 0.04	35.88 ± 4.91	40.49 ± 4.80
	MD [26]	**65.39** ± 4.76	**61.42** ± 6.48	**1.41** ± 0.11	**1.33** ± 0.08	**41.96** ± 5.40	**47.65** ± 4.91
	Ours	66.57 ± 5.38	64.17 ± 7.04	1.53 ± 0.06	1.37 ± 0.08	**41.96** ± 6.18	44.61 ± 5.47
Car	GET3D [13]	97.16 ± 0.31	90.80 ± 0.69	1.47 ± 0.02	1.13 ± 0.01	10.57 ± 0.44	18.18 ± 0.97
	MD [26]	**73.01** ± 1.72	65.28 ± 1.38	1.12 ± 0.01	0.86 ± 0.01	34.69 ± 1.86	41.82 ± 2.35
	Ours	73.35 ± 1.36	**63.04** ± 1.80	**1.08** ± 0.00	**0.83** ± 0.01	**35.20** ± 1.60	**42.33** ± 1.25
Chair	GET3D [13]	78.26 ± 0.43	73.62 ± 0.65	5.11 ± 0.02	2.64 ± 0.01	32.33 ± 0.56	36.28 ± 0.63
	MD [26]	72.43 ± 0.96	72.13 ± 1.60	4.98 ± 0.08	2.79 ± 0.05	38.60 ± 0.81	41.66 ± 0.90
	Ours	**67.09** ± 0.71	**66.69** ± 0.97	**4.76** ± 0.09	**2.58** ± 0.02	**41.43** ± 0.60	**47.07** ± 1.53

Table 5. Generation metrics on ShapeNet classes airplane, bike, car and chair, sampling inner and outer structure. Reference dataset was sampled from ShapeNet ground truth. MMD-CD is multiplied by $\times 10^3$, EMD by $\times 10^2$.

E. Ablations

E.1. Step size

Treating time as continuous rather than discrete allows us to vary sampling steps during inference without relying on DDIM sampling strategies [43]. Here, we quantitatively evaluate the range of steps needed in *TetraDiffusion* to generate diverse and high quality meshes. The metrics for steps 24, 32, 48, 64, 96 and 128 can be found in Table 6. Overall, the model tends to perform equally well starting from 32 time steps.

		1-NNA ↓		MMD ↓		COV ↑	
Category	Step size	CD	EMD	CD	EMD	CD	EMD
Airplane	24	78.27 ± 1.17	74.37 ± 1.20	0.32 ± 0.01	0.53 ± 0.01	38.96 ± 0.99	40.64 ± 1.41
	32	74.84 ± 0.95	72.17 ± 1.59	0.32 ± 0.01	0.52 ± 0.01	43.01 ± 0.75	43.80 ± 1.70
	48	73.51 ± 0.51	70.89 ± 1.35	0.32 ± 0.01	0.51 ± 0.01	46.96 ± 1.11	44.44 ± 1.99
	64	72.05 ± 1.21	**68.37** ± 1.30	**0.30** ± 0.01	**0.49** ± 0.00	47.56 ± 0.67	46.86 ± 1.07
	96	72.27 ± 1.12	72.89 ± 1.14	0.31 ± 0.01	0.51 ± 0.01	47.06 ± 1.40	46.07 ± 0.62
	128	**71.41** ± 2.07	69.28 ± 1.89	0.31 ± 0.01	0.50 ± 0.01	**48.10** ± 2.43	**47.41** ± 0.68
Car	24	73.31 ± 1.15	68.43 ± 1.75	0.92 ± 0.01	**0.63** ± 0.01	35.14 ± 0.86	**44.4** ± 1.47
	32	67.91 ± 1.03	61.8 ± 1.06	0.91 ± 0.01	**0.63** ± 0.02	**37.71** ± 0.95	44.29 ± 1.32
	48	68.97 ± 1.09	**60.43** ± 1.60	0.90 ± 0.01	**0.63** ± 0.02	36.40 ± 2.16	42.46 ± 1.73
	64	68.20 ± 0.99	61.71 ± 1.90	0.90 ± 0.01	**0.63** ± 0.01	35.26 ± 1.15	42.23 ± 1.75
	96	67.77 ± 1.08	60.89 ± 0.76	0.91 ± 0.01	0.66 ± 0.01	33.77 ± 1.02	42.57 ± 1.83
	128	**66.89** ± 1.22	60.49 ± 1.28	**0.89** ± 0.00	0.65 ± 0.00	35.60 ± 0.52	40.57 ± 2.12
Chair	24	70.85 ± 1.05	69.42 ± 0.68	5.47 ± 0.05	2.69 ± 0.02	40.58 ± 0.48	43.85 ± 0.80
	32	66.65 ± 0.38	63.98 ± 1.30	5.05 ± 0.08	2.56 ± 0.03	43.99 ± 0.64	45.34 ± 0.37
	48	64.40 ± 0.39	62.93 ± 0.46	5.18 ± 0.06	2.58 ± 0.02	44.29 ± 0.32	47.73 ± 0.42
	64	**61.94** ± 0.58	**60.52** ± 0.35	4.96 ± 0.06	2.47 ± 0.02	46.18 ± 0.72	47.64 ± 0.81
	96	63.48 ± 1.21	61.72 ± 1.11	4.89 ± 0.01	**2.46** ± 0.02	**46.36** ± 0.54	47.52 ± 0.97
	128	62.51 ± 0.58	62.51 ± 0.77	**4.85** ± 0.05	2.48 ± 0.02	45.74 ± 0.49	**48.05** ± 1.20

Table 6. Ablation over step size on ShapeNet classes airplane, bike, car and chair, sampling only outer structure. Reference dataset was sampled from ShapeNet ground truth. MMD-CD is multiplied by $\times 10^3$, EMD by $\times 10^2$.

E.2. Clipping and offset noise

Additionally, we ablate two orthogonal inference strategies, namely clipping $\hat{\mathbf{x}}_\theta(\mathbf{z_t}; t)$ to the data range along the reverse diffusion chain and varying the offset noise [16]. Clipping the prediction in x-space is often found in codebases but often not emphasized [20, 39]. As $\hat{\mathbf{x}}_\theta(\mathbf{z_t}; t)$ is iteratively used in ancestral sampling, not clipping can be seen as a train-test mismatch and may lead to deteriorate performance [39].

An interesting observation was made in [16, 25]. Diffusion models tend to generate images with medium brightness due to the discrepancy between training and inference signal-to-noise ratio. This becomes even more apparent in the high resolution regime as low frequency features are often not completely removed during the forward process. Reversely, the long wavelength features of the Gaussian noise at the start of the reverse diffusion process are least likely to be destructed, which forces images to respect the average value, the longest wavelength. A simple, yet effective solution is proposed by [16]: randomizing the zero-frequency component of the noise by adding single independent and identically distributed sample over the entire image during training.To evaluate this method effectiveness, we train a model using offset noise and conduct tests both with and without noise offsetting.

While we do not find significant differences using clipping, offset noise or both quantitatively (*cf*. Table 7), we have encountered an increase in details in some classes using clipping. However, the generated meshes generally contained more holes.

			1-NNA ↓		MMD ↓		COV ↑	
Category	Clip	Offset	CD	EMD	CD	EMD	CD	EMD
Airplane	✓		72.79 ± 1.11	69.67 ± 1.34	0.31 ± 0.01	**0.50** ± 0.01	**47.90** ± 1.02	**47.70** ± 1.19
		✓	**71.35** ± 0.96	71.62 ± 1.17	0.31 ± 0.01	0.51 ± 0.00	46.15 ± 0.98	45.43 ± 1.58
	✓	✓	72.98 ± 1.22	**69.43** ± 1.13	0.31 ± 0.01	0.51 ± 0.00	45.65 ± 1.35	46.32 ± 1.54
Bike	✓		64.41 ± 5.27	67.21 ± 6.58	**1.71** ± 0.06	**1.16** ± 0.06	45.88 ± 3.97	**57.35** ± 3.98
		✓	**62.94** ± 6.96	**66.62** ± 6.97	1.72 ± 0.11	1.19 ± 0.07	47.35 ± 5.79	54.71 ± 4.64
	✓	✓	64.12 ± 6.17	69.12 ± 5.46	1.76 ± 0.06	1.18 ± 0.06	**50.00** ± 7.72	54.71 ± 5.41
Car	✓		66.73 ± 0.77	63.17 ± 1.06	0.90 ± 0.01	0.65 ± 0.01	**37.75** ± 1.55	42.29 ± 1.27
		✓	**67.69** ± 0.76	62.29 ± 1.21	**0.89** ± 0.01	**0.65** ± 0.00	35.60 ± 1.48	**43.40** ± 2.11
	✓	✓	67.74 ± 1.17	**61.87** ± 1.36	0.90 ± 0.01	0.66 ± 0.01	34.09 ± 1.26	41.63 ± 0.84
Chair	✓		64.59 ± 0.37	**63.07** ± 0.68	**4.85** ± 0.09	**2.47** ± 0.03	**45.50** ± 0.69	**47.48** ± 0.84
		✓	**62.71** ± 0.43	63.72 ± 0.73	5.01 ± 0.04	2.51 ± 0.02	44.46 ± 0.75	46.56 ± 0.70
	✓	✓	63.66 ± 0.69	63.91 ± 0.78	4.99 ± 0.09	2.52 ± 0.03	42.90 ± 0.95	46.34 ± 0.82

Table 7. Comparison of clipping and offset noise during inference on ShapeNet classes airplane, bike, car and chair. Only the outer surface was sampled. Reference dataset was sampled from ShapeNet ground truth. MMD-CD is multiplied by $\times 10^3$, EMD by $\times 10^2$.