3DILG: Irregular Latent Grids for 3D Generative Modeling

Biao Zhang
KAUST
Matthias Nießner
TUM
Peter Wonka
KAUST

Abstract

We propose a new representation for encoding 3D shapes as neural fields. The representation is designed to be compatible with the transformer architecture and to benefit both shape reconstruction and shape generation. Existing works on neural fields are grid-based representations with latents defined on a regular grid. In contrast, we define latents on irregular grids, enabling our representation to be sparse and adaptive. In the context of shape reconstruction from point clouds, our shape representation built on irregular grids improves upon grid-based methods in terms of reconstruction accuracy. For shape generation, our representation promotes high-quality shape generation using auto-regressive probabilistic models. We show different applications that improve over the current state of the art. First, we show results for probabilistic shape reconstruction from a single higher resolution image. Second, we train a probabilistic model conditioned on very low resolution images. Third, we apply our model to category-conditioned generation. All probabilistic experiments confirm that we are able to generate detailed and high quality shapes to yield the new state of the art in generative 3D shape modeling.

1 Introduction

Neural fields for 3D shapes are popular in machine learning because they are generally easier to process with neural networks than other alternative representations, e.g., triangle meshes or spline surfaces. Earlier works represent a shape with a single global latent [23, 11, 6, 30, 53]. This already gives promising results in shape autoencoding and shape reconstruction. However, shape details are often missing and hard to recover from a global latent. Later, local latent grids for shapes were proposed [33, 20, 2]. A local latent mainly influences the shape (surface) in a local 3D neighborhood, thus perceiving shape details. However, in contrast to global latents, local latents require positional information about their location, e.g., as implicitly defined by a regular grid. Furthermore, multiscale
latent pyramid grids [8, 5] provide some performance improvement over basic grids. We illustrate the three current ways of modeling neural fields in Fig. 2.

In this paper, we set out to study irregular grids as 3D shape representation for neural fields. This extends previous grid-based representations, but allows each latent to have an arbitrary position in space, rather than a position that is pre-determined on a regular grid. We do not want to give up the advantages of a fixed length representation, and therefore propose to encode shapes as a fixed length sequence of tuples \((x_i, z_i)\) \(\in M\), where \(x_i\) are the 3D positions and \(z_i\) are the latents. The overall advantages of this representation are that it is fixed length, works well with transformer-based architectures using established building blocks, and that it can adapt to the 3D shape it encodes by placing latents at the most important positions. The representation also scales better than grid-based (and especially pyramid-based) representations to a larger number of latents. The number of latents in a representation is a factor that greatly influences the training time of transformer architectures, as the computation time is quadratic in the number of latents.

While our proposed representation brings some improvement for 3D shape reconstruction from point clouds, the improvement is very significant for 3D generative models that have been less explored. We believe that the most promising avenue for 3D shape generation is to follow recent image generative methods based on vector quantization and auto-regressive models using transformer, e.g. VQGAN [14]. These models operate on discrete latent vectors represented by indices [44]. During training, an autoregressive probabilistic model is trained to predict discrete indices. When doing inference (sampling), discrete indices are sampled from the autoregressive model and are decoded to images with a learned decoder. These models work well in the image domain, because a single image can comfortably be represented by medium size, e.g. 16 x 16 latent grids of 256 latents [14]. This does not directly scale to 3D shapes, as a 16 x 16 x 16 grid is too large to be comfortably trained on 4-8 GPUs due to the quadratic complexity of the transformer architecture [46]. As a result, generative models based on low-resolution regular grids lead to artifacts in the details, whereas our representation yields a nice and clean surface (e.g. Fig. 9).

We show the following applications of our proposed representations (see Fig. 1). For 3D shape reconstruction, we show results for 3D reconstruction from a point cloud. For generative modeling, we show results for image-conditioned 3D shape generation, object category-conditioned 3D shape generation, and point-cloud-conditioned 3D shape generation. The generative model is probabilistic and can generate multiple different 3D shapes for the same conditioning information.

We summarize the contributions of our work as follows: 1) We propose irregular latent grids as 3D shape representation for neural fields. Our shape representation thereby extends existing works using a global latent, a local latent grid or a multi-scale grid. 2) We improve upon grid-based SOTA methods for 3D shape reconstruction for point-clouds. 3) We improve state-of-the art generative models for 3D shape generation, including image-conditioned generation, object category-conditioned generation and point cloud-conditioned generation.

2 Related Work

2.1 Neural shape representations

Shape analysis with neural networks processes shapes in different representations. Common representations include voxels [9, 10, 37] and point clouds [34, 35, 48, 49, 42]. More recently, researchers study shapes represented with neural fields [50], e.g., signed distance functions (SDFs) or occupancy (indicator) functions of shapes modeled by neural networks. Subsequently, meshes can be extracted by contouring methods such as marching cubes [22]. The methods have been called neural implicit
representations [23, 24, 30, 15, 11, 6, 53] or coordinate-based representations [40]. We decided to use the term neural fields in this paper [50].

**Point cloud processing.** Earlier works for point cloud processing rely on multilayer perceptrons (MLPs), e.g., PointNet [34]. PointNet++ [35] and DGCNN [48] extended the idea by employing a hierarchical structure to capture local information. Inspired by ViT [13], which treats images as a set of patches, PCT [17] and PT [54] propose a transformer backbone for point cloud processing. Both works introduce extra modules which are not standard transformers [46] anymore. Since our main goal is not to develop a general backbone for point clouds, we simply work with a standard transformer for shape autoencoding (first-stage training). Similarly, both PointBERT[52] and PointMAE [28] use standard transformers for point cloud self-supervised learning.

**Neural fields for 3D shapes.** One possible approach is to represent a shape with a single global latent, e.g., OccNet [23], CvxNet [11] and Zhang et al. [53]. While this method is very simple, it is not suitable to encode surface details or shapes with complex topology. Later works studied the use of local latents. Compared to global latent, these representations represent shapes with multiple local latents. Each latent is responsible for reconstruction in a local region. ConvONet [33], as a follow-up work of OccNet, learns latents on a grid (2d or 3d). IF-Net [8], trains a model which outputs 3d latent grids of several different resolutions. The grids are then concatenated into a final representation.

**2.2 Generative models**

Generative models include generative adversarial networks (GANs) [16], variational autoencoders (VAEs) [21], normalizing flows (NFs) [36, 12] and auto-regressive models (ARs) [45]. Our shape representation is designed to be compatible with auto-regressive generative models. Thus we will mainly discuss related work in this area.

In the image domain, earlier AR works generate pixels one-by-one, e.g., PixelRNN [45] and its follow-up works [43, 38]. Combining this idea with autoregressive transformers, similar approaches are applied to 3D data, e.g., PointGrow [39] for point cloud generation and PolyGen [27] for polygonal mesh generation. Other use cases include floor plan generation [29] and indoor scene generation [47, 32].

An important ingredient of many auto-regressive models is vector quantization (VQ) that was originally integrated into VAEs. VQVAE [44] adopts a two-stage training strategy. The first stage is an autoencoder with vector quantization, where bottleneck latents are quantized to discrete indices. In the second stage, an autoregressive model is trained to predict the discrete indices. This idea is further developed in VQGAN [14] which builds the autoregressive model with a transformer [46]. Vector quantization, has been shown to be an efficient way for generating high resolution images. We study how VQ can be used in the 3D domain, specifically, neural fields for 3D shapes.

There are some works adopting VQVAE to the 3D domain. Canonical Mapping [7] proposes point cloud generation models with VQVAE. In concurrent work, AutoSDF [26] generalizes VQVAE to 3D voxels. However, the bottleneck latent grid resolution is low and it is difficult to capture details as in ConvONet and IF-Net. Increasing the resolution is the key for detailed 3D shape reconstruction. On the other hand, higher resolution leads to more difficult training of autoregressive transformers. Another concurrent work, ShapeFormer [51], shows a solution by sparsifying latent grids. However, ShapeFormer still only accepts voxel grids as input (point clouds need to be voxelized). Additionally, the sequence length varies for different shapes, requiring the method to go through a whole dataset to find a maximum sequence length. This relies on dataset-dependent maximum sequence lengths. In contrast to existing works, our method processes point clouds directly, learns fixed-size latents and outputs neural fields. A comparison summary can be found in Table 1.

**3 Shape Reconstruction**

Given an input point cloud \{x_i \in \mathbb{R}^3 \}_{i \in \mathcal{N}} (also in matrix form \mathbf{X} \in \mathbb{R}^{N \times 3}), our goal is to predict the indicator function \mathcal{O}(\cdot) : \mathbb{R}^3 \rightarrow [0, 1] corresponding to the volume represented by the point cloud. Our representation is being generated in three steps (See Fig. 3): patch construction (Sec. 3.1), patch information processing (Sec. 3.2), and reconstruction (Sec. 3.3).
Table 1: **Method comparison.** All methods here take point clouds as input. If the column **Requiring Voxelization** says yes, it means the method takes voxels as input. The numbers shown in parenthesis are alternative choices of hyperparameters.

<table>
<thead>
<tr>
<th>Method</th>
<th>Local Latents</th>
<th>Requiring Voxelization</th>
<th>Output</th>
<th>Latent Resolution</th>
<th>Vector Quantization</th>
<th>Sequence Length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OccNet [23]</td>
<td>✗</td>
<td>✗</td>
<td>Fields</td>
<td>1</td>
<td>✗</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ConvONet [33]</td>
<td>✗</td>
<td>✗</td>
<td>Fields</td>
<td>32^3(64^3)</td>
<td>✗</td>
<td>32^3(64^3)</td>
<td>Multiscale</td>
</tr>
<tr>
<td>IF-Net [8]</td>
<td>✓</td>
<td>✓</td>
<td>Fields</td>
<td>128^3</td>
<td>✗</td>
<td>128^3</td>
<td></td>
</tr>
<tr>
<td>AutoSDF [26]</td>
<td>✓</td>
<td>✓</td>
<td>Voxels</td>
<td>8^3</td>
<td>✓</td>
<td>8^3</td>
<td></td>
</tr>
<tr>
<td>CanMap [7]</td>
<td>✓</td>
<td>✗</td>
<td>Point Cloud</td>
<td>128</td>
<td>✓</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>ShapeFormer [51]</td>
<td>✓</td>
<td>✓</td>
<td>Fields</td>
<td>16^3(32^3)</td>
<td>✓</td>
<td>variable</td>
<td>Sparsification</td>
</tr>
<tr>
<td>3DILG (Ours)</td>
<td>✓</td>
<td>✗</td>
<td>Fields</td>
<td>512</td>
<td>✓</td>
<td>512</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: **Shape reconstruction from point clouds.** Left: the main pipeline. The framework can be used with (Bottom Right) or without (Top Right) vector quantization.

### 3.1 Patch Construction

We sub-sample the input point cloud via Farthest Point Sampling (FPS),

\[
\text{FPS} \{ \{x_i\}_{i \in N}\} = \{x_i\}_{i \in M} = X \in \mathbb{R}^{M \times 3},
\]

where \(M \subset N\) and \(|M| = M\). Next, for each point in the sub-sampled point set \(\{x_i\}_{i \in M}\), we apply the K-nearest neighbor (KNN) algorithm to find \(K - 1\) points to form a point patch of size \(K\),

\[
\forall i \in M, \quad \text{KNN} \{x_i\} = \{x_j\}_{j \in N_i},
\]

where \(N_i\) is the neighbor index set for point \(x_i\) and \(|N_i| = K - 1\). Thus we have a collection of point patches, \(\{\{x_i, \{x_j\}_{j \in N_i}\} : |N_i| = K - 1\}_{i \in M} = X_{M,K} \in \mathbb{R}^{M \times K \times 3}\). We project each patch with a mini-PointNet-like [34] module to an embedding vector

\[
\forall i \in M, \quad \text{PointNet} (x_i, \{x_j\}_{j \in N_i}) = e_i \in \mathbb{R}^C,
\]

where \(C\) is the embedding dimension for patches.

### 3.2 Transformer

Furthermore, we build a transformer to learn local latents \(\{z_i\}_{i \in M}\). The point coordinates \(\{x_i\}_{i \in M}\) are converted to positional embeddings (PEs) [25], \(p_i = \text{PE}(x_i) \in \mathbb{R}^C\). Our transformer takes as input the sequence of patch-PE pairs,

\[
\text{Transformer} \{\{e_i, p_i\}_{i \in M}\} = \{z_i \in \mathbb{R}^C\}_{i \in M}.
\]

The transformer is composed of \(L\) blocks. We denote the intermediate outputs of all blocks as \(z_i^{(0)}, z_i^{(1)}, \ldots, z_i^{(l)}, \ldots, z_i^{(L)}\), where \(z_i^{(0)}\) is the input \(e_i\) and \(z_i^{(L)}\) is the output \(z_i\).

**Vector Quantization.** This model can be used with vector quantization [44]. We simply replace the intermediate output at block \(l\) with its closest vector in a dictionary \(\mathcal{D}\),

\[
\forall i \in M, \quad \arg\min_{\hat{z}^{(l)}_i \in \mathcal{D}} \|z_i^{(l)} - \hat{z}^{(l)}_i\|.
\]

The dictionary \(\mathcal{D}\) contains \(|\mathcal{D}| = D\) vectors. As in VQVAE [44], the gradient with respect to \(z_i^{(l)}\) is approximated with the straight-through estimator [1].
With Vector Quantization (Eq. 5), we compress the bit size of intermediate latents. We expect that each latent $z_i$ is responsible for shape reconstruction near the point $x_i$. However, our goal is to estimate $\hat{O}(\cdot)$ for an arbitrary $x \in \mathbb{R}^3$. We interpolate latent $z_x$ for point $x$ with the Nadaraya-Watson estimator, $z_x = \frac{\sum_{i \in M} \exp(-\beta \|x-x_i\|^2) z_i}{\sum_{i \in M} \exp(-\beta \|x-x_i\|^2)}$, where $\beta$ controls the smoothness of interpolation and can be fixed or learned. The final indicator is an MLP with a sigmoid activation, $\hat{O}(x) = \text{Sigmoid}(\text{MLP}(x, z_x))$.

**Loss Function.** We optimize the estimated $\hat{O}(\cdot)$ by comparing to ground-truth $O(\cdot)$ via a binary cross entropy loss (BCE) and an additional commitment loss term [44],

$$L = L_{\text{recon}} + \lambda L_{\text{comm}} = \mathbb{E}_{x \in \mathbb{R}^3} \left[ \text{BCE}(\hat{O}(x), O(x)) \right] + \lambda \mathbb{E}_{x \in \mathbb{R}^3} \left[ \mathbb{E}_{i \in M} \left\| \text{sg}(\hat{z}_i^{(l)}) - z_i^{(l)} \right\|^2 \right], \quad (6)$$

where $\text{sg}(\cdot)$ is the stop-gradient operation. When $\lambda = 0$, it means we train the model without vector quantization.

### 3.3 Reconstruction

We expect that each latent $z_i$ is responsible for shape reconstruction near the point $x_i$. However, our goal is to estimate $\hat{O}(\cdot)$ for an arbitrary $x \in \mathbb{R}^3$. We interpolate latent $z_x$ for point $x$ with the Nadaraya-Watson estimator, $z_x = \frac{\sum_{i \in M} \exp(-\beta \|x-x_i\|^2) z_i}{\sum_{i \in M} \exp(-\beta \|x-x_i\|^2)}$, where $\beta$ controls the smoothness of interpolation and can be fixed or learned. The final indicator is an MLP with a sigmoid activation, $\hat{O}(x) = \text{Sigmoid}(\text{MLP}(x, z_x))$.

**Loss Function.** We optimize the estimated $\hat{O}(\cdot)$ by comparing to ground-truth $O(\cdot)$ via a binary cross entropy loss (BCE) and an additional commitment loss term [44],

$$L = L_{\text{recon}} + \lambda L_{\text{comm}} = \mathbb{E}_{x \in \mathbb{R}^3} \left[ \text{BCE}(\hat{O}(x), O(x)) \right] + \lambda \mathbb{E}_{x \in \mathbb{R}^3} \left[ \mathbb{E}_{i \in M} \left\| \text{sg}(\hat{z}_i^{(l)}) - z_i^{(l)} \right\|^2 \right], \quad (6)$$

where $\text{sg}(\cdot)$ is the stop-gradient operation. When $\lambda = 0$, it means we train the model without vector quantization.

### 4 Autoregressive Generative Modeling

With Vector Quantization (Eq. 5), we compress the bit size of intermediate latents $\{\hat{z}_i^{(l)}\}_{i \in M}$ to $\log_2 D$ where $D$ is the size of the dictionary $D$. We denote the compressed index as $z_i \in \{0, 1, \ldots, D - 1\}$. We also quantize point coordinates $x_i$ to $(x_{i,1}, x_{i,2}, x_{i,3})$ where each entry is an 8-bit integer $\{0, 1, \ldots, 255\}$. As a result, we obtain a discrete representation of the 3D shape $\{(x_{i,1}, x_{i,2}, x_{i,3}, z_i)\}_{i \in M}$.

#### 4.1 Unidirectional Transformer

Autoregressive generation with unidirectional transformer is a more classical approach. We often need to sequentialize an unordered set if the order is not defined yet. Specifically, we re-order the representations in ascending order by the first component $x_{i,1}$, then by the second component $x_{i,2}$, and finally by the third component $x_{i,3}$,

$$S = \{(x_{0,1}, x_{0,2}, x_{0,3}, z_0), (x_{1,1}, x_{1,2}, x_{1,3}, z_1), \ldots, (x_{M-1,1}, x_{M-1,2}, x_{M-1,3}, z_{M-1})\}. \quad (7)$$

Our goal to predict sequence elements one-by-one. A common way is to flatten the sequence $S$ by concatenating the quadruplets, for examples, PointGrow [39] and PolyGen [27]. However, here we...
We set the size of the input point cloud to $N = 2048$. The number and the size of point patches are $M = 512$ and $K = 32$, respectively. In the case of Vector Quantization, there are $D = 1024$ vectors in the dictionary $\mathcal{D}$. Other details of the implementation can be found in the Appendix. We use the datset ShapeNet-v2 [3] for shape reconstruction. We split samples into train/val/test (48597/1283/2592) set. Following the evaluation protocol of [11, 53], we include three metrics, volumetric IoU, the Chamfer-L1 distance, and F-Score [41]. An evaluation on scene level reconstruction can be found in the appendix.

### 4.2 Bidirectional Transformer

Bidirectional transformer for autoregressive generation is recently proposed by MaskGIT [4]. Here we show our model can also be combined with bidirectional transformer. However, here we consider a different task. We generate $\{z_i\}_{i \in \mathcal{M}}$ conditioned on $\{(x_{i,1}, x_{i,2}, x_{i,3})\}_{i \in \mathcal{M}}$. In training phase, we sample a subset $\{z_i\}_{i \in \mathcal{V} \subset \mathcal{M}}$ of $\{z_i\}_{i \in \mathcal{M}}$ as the input of the bidirectional transformer. We aim to predict $\{z_i\}_{i \in \mathcal{M} \setminus \mathcal{V}}$. The coordinates $\{(x_{i,1}, x_{i,2}, x_{i,3})\}_{i \in \mathcal{M}}$ are converted to positional embeddings (either learned or fixed) as condition. The likelihood of generating $\mathcal{M} \setminus \mathcal{V}$ is as follows,

$$p(\{z_i\}_{i \in \mathcal{M} \setminus \mathcal{V}} \mid \{z_i\}_{i \in \mathcal{V}}, \{(x_{i,1}, x_{i,2}, x_{i,3})\}_{i \in \mathcal{M}}).$$

In practice, the bidirectional transformer takes as input all tokens except that $\{z_i\}_{i \in \mathcal{M} \setminus \mathcal{V}}$ are replaced by a special masked token $[m]$. When decoding (inference), we iteratively predict multiple tokens at the same time. Tokens are sampled based on their probabilities (transformer output). See [4] for a detailed explanation. We show visualization of decoding steps in Fig. 5.

### 5 Reconstruction Experiments

We train all models on ShapeNet-v2 (55 categories). All baseline methods are trained with the corresponding officially released code. For our model, we set $N = 2048$ and $K = 32$. We select 7 categories to show. These categories have largest training set among 55 categories. We also show averaged metrics over all categories. The numbers shown in parenthesis are results of vector quantization.

<table>
<thead>
<tr>
<th>Metric</th>
<th>OccNet</th>
<th>ConvONet</th>
<th>IF-Net</th>
<th>3DILG (Ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoU↑</td>
<td>0.822</td>
<td>0.881</td>
<td>0.929</td>
<td>0.922(0.904)</td>
</tr>
<tr>
<td></td>
<td>mean (all)</td>
<td>0.825</td>
<td>0.888</td>
<td>0.934</td>
</tr>
<tr>
<td>Chamfer↓</td>
<td>0.058</td>
<td>0.040</td>
<td>0.034</td>
<td>0.038(0.038)</td>
</tr>
<tr>
<td></td>
<td>mean (all)</td>
<td>0.072</td>
<td>0.052</td>
<td>0.041</td>
</tr>
<tr>
<td>F-Score↑</td>
<td>0.898</td>
<td>0.951</td>
<td>0.975</td>
<td>0.959(0.948)</td>
</tr>
<tr>
<td></td>
<td>mean (all)</td>
<td>0.858</td>
<td>0.933</td>
<td>0.967</td>
</tr>
</tbody>
</table>

Consider an approach generating quadruplet-by-quadruplet. We write $o_i = (x_{i,1}, x_{i,2}, x_{i,3}, z_i)$. The likelihood of generating $o_i$ autoregressively is

$$p(\mathcal{S} \mid \mathcal{C}) = \prod_{i=0}^{i=M-1} p(o_i \mid o_{<i}, \mathcal{C}),$$

where $\mathcal{C}$ is conditioned context. Similar to ATISS [32], we predict components of $o_i = (x_{i,1}, x_{i,2}, x_{i,3}, z_i)$ autoregressively:

$$p(o_i \mid o_{<i}) = p(x_{i,1} \mid o_{<i})p(x_{i,2} \mid x_{i,1}, o_{<i})p(x_{i,3} \mid x_{i,2}, x_{i,1}, o_{<i})p(z_i \mid x_{i,1}, x_{i,2}, x_{i,1}, o_{<i}),$$

where the context $\mathcal{C}$ is omitted to avoid clutter. The model is shown in Fig. 4. Different from ATISS [32] which uses MLPs to decode different components, instead we continue to apply transformer blocks for component decoding.
Results

We compare our method with three existing works, OccNet [23], ConvONet [33] and IF-Net [8]. The results can be found in Table 2. We select 7 categories among 55 categories with largest training set (table, car, chair, airplane, sofa, rifle, lamp). Detailed results can be found in the Appendix. We show different choices of $M$. As we can see in this table, increasing $M$ from 64 to 512 gives a performance boost. Even with the simplest model $M = 64$, our results outperform ConvONet. The best results are achieved when setting $M = 512$. In this case, our results lead in most categories when comparing to IF-Net. The results of IF-Net are better than ours in terms of metric F-Score in some categories. However, we argue that in this case the metric F-score is saturated (values are close to 1), which makes it hard to compare. We also show results after introducing vector quantization to our model in the same table. We can see that vector quantization harms the performance slightly.

6 Generative Experiments

We introduce three experiments to show how our model can be combined with auto-regressive transformers for generative modeling. In Sec. 6.1, we show image-conditioned generation as probabilistic shape reconstruction from single image. In Sec. 6.2, we show how we generate samples given a category label (using the 55 ShapeNet categories). In Sec. 6.3, we further show generation conditioned on downsampled point clouds. In contrast to the first two tasks, the point-cloud-conditioned generation is based on a bidirectional transformer described in Sec 4.2.

6.1 Probabilistic Shape Reconstruction from a Single Image

We train a uni-directional transformer for this task (see Sec. 4.1). The context $C$ is an image. To train this model, we render 40 images ($224 \times 224$) of different views for each shape in ShapeNet. The implementation of the uni-directional transformer is based on GPT. It contains 24 blocks, where each block has an attention layer with 16 heads and 1024 embedding dimension. When sampling, nucleus sampling [19] with top-p (0.85) and top-k (100) are applied to predicted token probabilities.

High resolution images. We show some generated samples when $C$ are high resolution images in Fig. 7. We compare our results with a deterministic method, OccNet [23]. As we can see in the results, the deterministic method (OccNet) tends to create blurred meshes. However, our probabilistic reconstruction is able to output detailed meshes.

Low resolution images. We also consider another more challenging task. The input images are downsampled to low resolution ($16 \times 16$). In this case, the generative model has more freedom to find multiple plausible interpretations, including variations with different topology (see Fig. 8).
Figure 7: **Image-conditioned generation** (224 × 224). We sample 2 shapes for each input image, and compare them with OccNet.

Figure 8: **Image-conditioned generation** (16 × 16). We sample 8 shapes for each input image.

### 6.2 Category-Conditioned Generation

To generate shapes given a category label, we use the context $C$ to encode the category label. We employ the uni-directional transformer based on GPT as in Sec. 6.1. We compare to our proposed baseline model that encodes shapes as a latent grid of resolution $8^3$. To make a fair comparison, we also extend ViT [13] to the 3D voxel domain in the first stage of training. In the second stage, we use the same uni-directional transformer for sequence prediction. Here the sequence length is $8^3 = 512$, which is the same as in our proposed model. The baseline model is named **Grid-8³**. The comparison to the baseline model is in Fig. 9. We can see that the baseline is unable to generate high quality shapes. We argue that this is because the representation is not expressive enough to capture surface details. Furthermore, we show more generated samples of our model in Fig. 10. The three selected categories are *bookshelf*, *bench* and *chair*. For both the baseline and our model, we render 10 images of predicted shapes, and calculate the Fréchet Inception Distance (FID) [18, 31] between predictions and test sets. The metrics can be found in Table 3. A perceptual study on the quality of generated samples can be found in the appendix.

### 6.3 Point Cloud Conditioned Generation

We train a bidirectional transformer described in Sec 4.2. The model takes as input $\{x_i\}_{i \in \mathcal{M}}$. The results can be found in Fig. 11.

Figure 9: **Comparison of category-conditioned generation**. We compare our results (**Left**) with an $8^3$ latent grid baseline (**Right**).
Table 3: FID ↓ for category-conditioned generation. We compare our results with the baseline Grid-8^3. The 7 categories are the largest categories in ShapeNet.

<table>
<thead>
<tr>
<th>Categories</th>
<th>table</th>
<th>car</th>
<th>chair</th>
<th>airplane</th>
<th>sofa</th>
<th>rifle</th>
<th>lamp</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid-8^3</td>
<td>72.396</td>
<td>95.566</td>
<td>58.649</td>
<td>42.009</td>
<td>58.092</td>
<td>59.456</td>
<td>87.319</td>
<td>67.641</td>
</tr>
<tr>
<td>3DILG (Ours)</td>
<td>68.016</td>
<td>92.597</td>
<td>45.333</td>
<td>30.957</td>
<td>53.244</td>
<td>40.500</td>
<td>72.672</td>
<td>57.617</td>
</tr>
</tbody>
</table>

Figure 10: Category-conditioned generation. We choose 3 categories to show (bookshelf, bench and chair). We show 100 samples for each category.

7 Conclusion

We have studied neural fields for shapes and presented a new representation. In contrast to common approaches which define latents a on a regular grid or multiple regular grids, we position latents on an irregular grid. Comparing to existing works, the representation better scales to larger models, because the irregular grid is sparse and adapts to the underlying 3D shape of the object. In our results, we demonstrated an improvement in 3D shape reconstruction from point clouds and generative modeling conditioned on images, object category, or point-clouds over alternative grid-based methods. In future work, we suggest to explore other applications of our proposed representation, e.g., shape completion, and extensions to textured 3D shapes and 3D scenes.

Figure 11: Point cloud conditioned generation. We show 8 decoding steps.
References


