Learning to Generate 3D Shapes from a Single Example

RUNDI WU, Columbia University, USA
CHANGXI ZHENG, Columbia University, USA

Fig. 1. A gallery of acropolises. Trained on a single acropolis shape (top left), our proposed generative model is able to produce a diverse gallery of acropolises, possibly of different sizes and aspect ratios. The generated shapes depict rich variations (such as the dents and breakage of the columns) across different scales, and at the same time retain the essential structure of the reference shape (such as the layout of the columns on the rectangular base). ©The original 3D acropolis model (top left) by choly kurd under Standard License Editorial Use Only (turbosquid.com).

Existing generative models for 3D shapes are typically trained on a large 3D dataset, often of a specific object category. In this paper, we investigate the deep generative model that learns from only a single reference 3D shape. Specifically, we present a multi-scale GAN-based model designed to capture the input shape’s geometric features across a range of spatial scales. To avoid large memory and computational cost induced by operating on the 3D volume, we build our generator atop the tri-plane hybrid representation, which requires only 2D convolutions. We train our generative model on a voxel pyramid of the reference shape, without the need of any external supervision or manual annotation. Once trained, our model can generate diverse and high-quality 3D shapes possibly of different sizes and aspect ratios. The resulting shapes present variations across different scales, and at the same time retain the global structure of the reference shape. Through extensive evaluation, both qualitative and quantitative, we demonstrate that our model can generate 3D shapes of various types.

1 INTRODUCTION

The user creation of novel 3D digital shapes is nontrivial, requiring technical know-how and a sense of art, often taking time and patience. It is such a laborious process that motivates the development of computer algorithms that create shapes—new, diverse, and high-quality 3D shapes created in a fully automatic fashion. Leveraging recent advance in deep learning, research in this direction has been vibrant [Achlioptas et al. 2018; Chen and Zhang 2019; Jayaraman et al. 2022; Nash et al. 2020; Wu et al. 2016]: the general theme here is to develop a generative model able to learn from a training dataset to generate 3D shapes.

The training dataset for almost all existing 3D generative models must be sufficiently large—often provided in a specific category such as chairs [Chang et al. 2015], mechanical parts [Willis et al. 2021] and terrains [Guérin et al. 2017]. However, collecting a large set of 3D shapes in the first place is by no means an easy task. Unlike images, which can be easily captured by cameras and are widely available online, 3D shapes require significant manual effort to 3D scan or model in modern 3D modeling software. Therefore, to use a learning-based generative model for creating 3D shapes, one must address the chicken and egg problem, unless the generative model is liberated from demanding a large training dataset.

Our work aims for this liberation. We present a deep generative model that learns from just a single 3D shape, without the need of any manual annotation or external data. Our goal is conceptually similar to example-based 2D texture synthesis [Wei et al. 2009]: produce new, diverse, as many as needed samples from a given...
input example. But our model is not merely meant to generate stationary 3D textures. It strives to produce diverse shape variations while preserving the global structure presented in the input shape.

As an example shown in Fig. 1, an acropolis shape is provided to train our generative model. The generated shapes all share the same global structure as the input example: a number of columns are placed on the edge of a rectangular base, leaving the central chamber area open. Meanwhile, they are all different. They have varying local features, such as different dents and breakage of the columns. Our model can further synthesize new shapes that have bounding box sizes and aspect ratios different from the input. We refer the reader to Fig. 6 and the appendix (Fig. 13 and Fig. 14) for a range of example shapes and synthesized results.

Technically, our work is inspired by the recent advance in single image generative adversarial networks (GANs) [Hinz et al. 2021; Shaham et al. 2019], wherein the goal is to learn the distribution of image patches on a single input image. Similar in spirit to those works, our generative model is based on a multi-scale, hierarchical GAN architecture (see Fig. 2), trained on a voxel pyramid of the input 3D shape. The voxel pyramid is responsible for capturing geometric features across multiple scales, from global structures to local details. However, since each level of the voxel pyramid is a 3D grid, it has a large memory footprint. To make the matter worse, the 3D convolution needed to operate on a 3D grid produces intermediate feature maps that require even larger memory. The intensive memory requirement severely limits the grid resolution and in turn the generative model’s ability to learn geometric details.

We therefore seek for sidestepping 3D convolutions. Our generative model operates on the tri-plane feature map—which encodes a shape in three axis-aligned 2D feature maps—followed by a small multilayer perceptron (MLP) network that describes the generated shape as a neural implicit function [Mescheder et al. 2019; Peng et al. 2020]. The use of tri-plane feature map significantly reduces memory and computation cost. It allows the generative model to learn shape features across a wide range of scales, and the MLP network enables the model to output a shape at an arbitrary resolution.

To our knowledge, this is the first deep generative model that synthesizes novel 3D shapes from a single example while capturing shape features across multiple scales. We demonstrate generation results on various 3D shapes of different categories. We also perform quantitative evaluations to compare our method to several baselines and prior methods. In addition, we provide ablation studies to justify our network design, training strategy, and data construction.

2 RELATED WORK

3D model synthesis. Similar to texture synthesis [Wei et al. 2009], traditional 3D shape synthesis techniques can be largely classified into procedural and example-based methods. Procedural modeling techniques [Ebert et al. 2003] have been extensively studied over the years. They typically require the specification of many rules for generating shapes, and therefore they are often designed for specific classes of objects, such as terrains [Musgrave et al. 1989; Smelik et al. 2009], cities [Müller et al. 2006; Parish and Müller 2001; Talton et al. 2011], and trees [Longay et al. 2012; Mêch and Prusinkiewicz 1996; Prusinkiewicz et al. 2001].

Example-based methods are more general-purpose, aiming to synthesize new shapes by analyzing the given example(s). The pioneering work in this direction [Funkhouser et al. 2004] and others [Kalogerakis et al. 2012; Xu et al. 2012] synthesize novel shapes by assembling components retrieved from a database of segmented shapes. Another line of works [Bokeloh et al. 2010; Merrell 2007; Merrell and Manocha 2008; Zhou et al. 2007] analyze a single input shape and generate larger 3D models by exploiting repeated patterns within the input example. However, these techniques either rely on manual decomposition [Merrell 2007] or require exact symmetry [Bokeloh et al. 2010].

Our work also falls into the category of example-based 3D modeling. In contrast to prior methods that usually rely on hand-crafted rules, we leverage neural networks to automatically learn multi-scale shape features of the single input example. Our method is also related to multi-scale texture synthesis [Han et al. 2008], as we build our network in a hierarchical manner and train it on a voxel pyramid constructed from the input 3D shape.

Deep generative models for 3D shapes. Since the introduction of deep generative networks such as Generative Adversarial Network (GAN) [Goodfellow et al. 2014] and Variational Autoencoder (VAE) [Kingma and Welling 2014], developing deep generative models for 3D shapes has attracted immense research interest. Existing works learn to produce 3D shapes in different representations, including voxels [Chen et al. 2021; Wu et al. 2016], point clouds [Achlioptas et al. 2018; Cai et al. 2020; Li et al. 2021; Yang et al. 2019], meshes [Gao et al. 2021; Hertz et al. 2020; Liu et al. 2020; Nash et al. 2020; Pavlo et al. 2021], implicit functions [Chen and Zhang 2019; Kleineberg et al. 2020; Mescheder et al. 2019; Park et al. 2019], multi-charts [Ben-Hamu et al. 2018; Groueix et al. 2018], structural primitives [Jones et al. 2020; Li et al. 2017; Mo et al. 2019; Wu et al. 2020], and parametric models [Chen et al. 2020; Jayaraman et al. 2022; Wu et al. 2021]. Nearly all these methods are trained on a large dataset of category-specific 3D shapes, e.g., ShapeNet [Chang et al. 2015].

Although the use of a large dataset enables the generative model to learn rich information shared across various shape instances, it also limits the application scope of these methods. Unlike images, which can be easily captured and downloaded, collecting a large, high-quality dataset of 3D shapes is much more expensive. Moreover, many artistically designed shapes have unique structures, and thus it is hard, if not impossible, to learn them from a large data collection. In this work, instead of learning a shape distribution from a large dataset, we focus on learning shape features across multiple scales from a single 3D shape. Without the need of a large 3D dataset, even from a unique shape (e.g., designed by a 3D artist), our method is able to learn and generate similar but new shapes.

Learning a generative model from a single example. In several recent works, researchers start to explore generative models that are learned from a single example in various data domain, such as images [Chen et al. 222a; Granot et al. 2021; Hinz et al. 2021; Shaham et al. 2019; Shocher et al. 2019; Zhou et al. 2018], videos [Haim et al. 2021], audio [Greshler et al. 2021], 3D meshes [Hertz et al. 2020] and motion sequences [Li et al. 2022]. There are also data-driven methods on 3D mesh reconstruction [Hano et al. 2020] and stylization [Michel et al. 2022] under one-shot setting. A common theme
behind these works is to capture the internal patch distribution of a given example, and then apply the learned knowledge in the generation process. The seminal work, SinGAN [Shaham et al. 2019], proposes to learn a single image generative model by training a hierarchy of 2D convolutional patch-GANs. Their method is not restricted to stationary texture synthesis, and it can also synthesize natural images that present large-scale structures. But one cannot simply transfer their method for 3D shape synthesis, as it would require 3D CNNs on 3D voxel grids, which are inefficient and memory intensive, especially when a high resolution output is desired.

To the best of our knowledge, DGTS [Hertz et al. 2020] is the only prior work on learning a generative model from a single 3D shape. It learns a convolutional network on triangle meshes, predicting vertex displacement after subdivision. However, this method is mainly designed for geometric texture transfer, thus limited to the synthesis of stationary and isometric local geometric features. Moreover, its generated shapes are limited to the topology of the input. Our proposed method, on the contrary, learns from a voxel representation. It can capture geometric features across multiple scales and generate samples that have different topologies from the input. To avoid the memory cost for directly operating on 3D voxel grids, our generator leverages an efficient tri-plane hybrid representation, and thereby only 2D convolutions are needed.

3 METHOD

Overview: A natural way of capturing shape features across multiple scales is to use a hierarchical representation of the input shape. Provided an input shape \( x \), we voxelize it and construct a voxel pyramid \( \{ x_0, \ldots, x_N \} \), where \( x_i \) indicates a voxelized version of \( x \) at resolution \( s_N \cdot r^{-i} \), for a user-specified size \( s_N \) and a downsampling factor \( r < 1 \). All \( x_i \) are voxelized independently. Correspondingly, our generative model consists of a hierarchy of GANs (see our network overview in Fig. 2). At each level \( i \), the generator aims to synthesize a shape indistinguishable from \( x_i \). Its output is fed as an input to the generator at level \( i + 1 \) to propagate the synthesized large-scale structure to the finer level.

Crucial to this hierarchy of GANs is the structure of each generator. To operate on voxel grids, typically used is the 3D convolution. But the hierarchy of voxel grids and the 3D convolution results—which extend a 3D grid by another dimension, namely the number of channels—require a significant amount of memory (see Table 2 later). This intensive memory footprint prevents us from using a high resolution grid, thus limiting the finest features that the generative model can learn. To overcome this limitation, we design the generator to operate not on a voxel grid directly, but on its tri-plane feature map (see Sec. 3.1). The generator takes as input a tri-plane map and outputs another tri-plane map. The output map, equipped with an MLP network, serves as a neural implicit function—one that allows the output shape to be constructed at an arbitrary resolution.

The discriminators in our GAN hierarchy have simple structures, each composed of three convolutional layers. A discriminator is responsible for distinguishing the synthesized and input shape in terms of local voxel patches of the same size (\( 11 \times 11 \times 11 \) in our implementation). On a different level of the hierarchy, the same size of a local patch captures shape features in different scales. In this way, the low-level discriminators force our generative model to preserve

---

ACM Trans. Graph., Vol. 41, No. 6, Article 224. Publication date: December 2022.
large-scale global structures, while the high-level discriminators encourage small-scale local variations.

### 3.1 Tri-plane Hybrid Representation

Although a 3D shape has a volume, its shape features are on its surface; away from its surface, the grid data is featureless. We therefore seek to encode a 3D shape into 2D feature maps. In particular, we build our generative model upon the tri-plane hybrid representation emerged in recent neural implicit function literature [Chan et al. 2021; Chen et al. 2022b; Peng et al. 2020]. Instead of representing a 3D shape on a voxel grid of size $D \times H \times W$, the tri-plane hybrid representation expresses the 3D shape using a tri-plane feature map $T$ together with a small MLP network $M$. The tri-plane feature map $T$ contains three axis-aligned 2D feature maps, $T = (F_{xy}, F_{xz}, F_{yz})$, where $F_{xy} \in \mathbb{R}^{C \times D \times H}$, $F_{xz} \in \mathbb{R}^{C \times D \times W}$ and $F_{yz} \in \mathbb{R}^{C \times H \times W}$, with $C$ being the number of channels. Consider a 3D position $p \in \mathbb{R}^3$. Its occupancy by a 3D shape is related to its tri-plane feature map in the following way (see Fig. 3). First, $p$ is projected onto the three axis-aligned planes to obtain its 2D coordinates, $p_{xy}, p_{xz}$, and $p_{yz}$, respectively. Based on the 2D coordinates, we query the tri-plane feature map to obtain $p$’s feature vector $f(p)$:

$$
\begin{align*}
    f_{xy}(p) &= \text{interp}(F_{xy}, p_{xy}), \\
    f_{xz}(p) &= \text{interp}(F_{xz}, p_{xz}), \\
    f_{yz}(p) &= \text{interp}(F_{yz}, p_{yz}), \\
    f(p) &= \lceil f_{xy}, f_{xz}, f_{yz} \rceil,
\end{align*}
$$

where interp$(\cdot, q)$ performs bilinear interpolation of a 2D feature map at position $q$, akin to a 2D texture value lookup; and $f(p) \in \mathbb{R}^C$ is simply a concatenation of the three interpolation results. Lastly, an additional lightweight MLP decoder $M$ takes in $f(p)$ and outputs $p$’s occupancy probability,

$$
o(p) = M(f(p)),
$$

which is an implicit representation of the 3D shape. If $o(p)$ is determined, one can use the Marching Cubes algorithm [Lorensen and Cline 1987] to construct the shape’s triangle mesh.

Note that unlike previous works [Chan et al. 2021; Peng et al. 2020], here we intentionally exclude the 3D position of $p$ from the input of the MLP. This is to ensure the generator position-invariant—a desired property that will become clear in Sec. 3.2.

From now on, we denote $M(T) \in \mathbb{R}^{D \times H \times W}$ as decoding the entire 3D volume of grid size $D \times H \times W$ from the tri-plane feature map $T$, a shorthand for querying every grid point through $M$.

### 3.2 Multi-scale Generation

For the purpose of multi-scale shape generation, we need to organize tri-plane feature maps in a hierarchical fashion. This also sets our tri-plane maps apart from those previously used in neural implicit functions [Chan et al. 2021; Peng et al. 2020].

**Generator.** In our GAN hierarchy, the generator $G_i$ at level $i$ outputs a tri-plane feature map $T_i$ whose spatial resolution matches $x_i$ (see Fig. 2). The output $T_i$ on the one hand is decoded by an MLP network $M_i$ into a generated shape $\hat{x}_i = M_i(T_i)$, which is in turn examined by the discriminator $D_i$ at the same level. On the other hand, $T_i$, after upsampling, serves as an input to the generator $G_{i+1}$ at the next level to produce tri-plane map $T_{i+1}$, namely,

$$
T_{i+1} = G_{i+1}(\hat{T}_i \uparrow, z_{i+1}), \quad i \geq 0,
$$

where $(\hat{T}_i) \uparrow$ bilinearly upsamples each of the three 2D feature maps $(F_{xy}^i, F_{xz}^i, F_{yz}^i)$ by a factor of $\frac{1}{2}$, such that their spatial dimensions match $x_{i+1}$. In this way, the larger scale shape structures synthesized by $G_i$ is propagated into the next generator $G_{i+1}$ to add finer scale details. The second input $z_{i+1}$ indicates the noise to introduce randomness in the generation process, containing three independent spatial Gaussian noise maps, $z_{xy}^{i+1}, z_{xz}^{i+1}$ and $z_{yz}^{i+1}$. 

---

Rundi Wu and Changxi Zheng

ACM Trans. Graph., Vol. 41, No. 6, Article 224. Publication date: December 2022.
Concretely, the generator \( G_{i+1} \) performs the following operations,

\[
\begin{align*}
F_{ixy}^{i+1} &= (F_{ixy}^{i}) \uparrow + \psi_{ixy}^{i+1}(z_{ixy}^{i+1} + (F_{iy}^{i})) \uparrow, \\
F_{ixo}^{i+1} &= (F_{ixo}^{i}) \uparrow + \psi_{ixo}^{i+1}(z_{ixo}^{i+1} + (F_{izo}^{i})) \uparrow, \\
F_{ixo}^{i+1} &= (F_{ixo}^{i}) \uparrow + \psi_{ixo}^{i+1}(z_{ixo}^{i+1} + (F_{izo}^{i})) \uparrow,
\end{align*}
\]

where \( \psi_{xy}, \psi_{xz}, \psi_{yz} \) are three 4-layer 2D convolutional networks, wherein each layer has a kernel size 3 x 3. Figure 5 depicts the generator structure for \( i \geq 1 \).

The first generator \( G_0 \) is slightly different. Although it has the same network structure as others, it takes only one input, the tri-plane map, but not the noise. Thus, it has no skip connections (those shown in Fig. 5). Since there is no lower level generator to provide the input tri-plane map, the tri-plane map to \( G_0 \) comes from a 3D noise \( Z_0 \). We use a network module \( P \) (see Fig. 4 for its structure) to project \( Z_0 \) and produce the initial tri-plane feature map \( P(Z_0) \). In short, the first generator \( G_0 \) produces a tri-plane map \( T_0 \) through \( T_0 = G_0(P(Z_0)) \).

We note that a seemingly simpler option is to use three independent 2D noise maps to form a tri-plane map input to \( G_0 \), and thereby no projection network \( P \) is needed. This approach, however, produces suboptimal results, because according to the tri-plane map construction in (2), its three 2D feature maps must be correlated. See Sec. 4.2 for an ablation study on our choice.

Overall, the generators in our GAN hierarchy are fully convolutional, operating solely on 2D feature maps. Also the MLP decoder in (3) is position-invariant. Consequently, at inference time, we are able to generate shapes with the sizes and aspect ratios different from the input shape (see Fig. 6). This can be easily achieved by sampling an input 3D noise \( Z_0 \) with user specified spatial dimensions, not necessarily the input shape’s spatial dimensions.

**Discriminator.** Discriminators in our GAN hierarchy take as input either the generated shape \( X_i = M_i(T_i) \) or the example shape \( x_i \), both of which are provided in voxelized representation and have the same resolution. The discriminators all have the same simple structure: a 3-layer 3D convolutional network, wherein each layer has the kernel size 3 x 3 x 3, and the first layer (with the stride=2) halves the input grid’s resolution. Thus, every discriminator has the same receptive field \((11 \times 11 \times 11)\). The output of the discriminator is a 3D score map in which every element classifies the corresponding \(11 \times 11 \times 11\) patch of the input voxel grid to be fake or real.

Unlike the generators, the discriminators can afford to use 3D convolutions on 3D voxel grids, as they consume much less memory due to following reasons. 1) With the stride=2 in the first layer, the input grid’s resolution is halved. 2) More importantly, the discriminators are only needed at training time, and as we will describe next, they are trained sequentially from low level to high level. At any moment in time, only one discriminator needs to be kept in memory. In contrast, all the generators in the GAN hierarchy must be stacked together in order to generate a shape.

### 3.3 Training

We train our generators and discriminators in a progressive manner, from the lowest level (the coarsest scale) to the highest level (the finest scale). The loss function for training the \( i \)-th level includes an adversarial term and a reconstruction term, namely,

\[
\min_{G_i,M_i} \max_{D_i} \mathcal{L}_{\text{adv}}(G_i,M_i,D_i) + \alpha \mathcal{L}_{\text{rec}}(G_i,M_i),
\]

where the weighting factor \( \alpha \) is set to 10 in all our experiments. The projection module \( P \) is also jointly trained in the lowest level \( (i = 0) \) and kept fixed for the rest of the training. Once the training for the \( i \)-level is finished, the weights of \( G_i \) and \( M_i \) stay fixed, and \( D_i \) is discarded. In addition, the weights of \( M_i \) and \( D_i \) will be used to initialize the training of \( M_{i+1} \) and \( D_{i+1} \).

For the adversarial term \( \mathcal{L}_{\text{adv}} \), since the discriminator outputs a score map that classifies each of the \(11 \times 11 \times 11 \) patches of the voxel grid, we treat the average of the score map as the final discriminator score and use the WGAN-GP [Gulrajani et al. 2017] training objective, which we found through experiments produces the best results.

The reconstruction term \( \mathcal{L}_{\text{rec}} \) requires that a specific sequence of noise \( \{Z_{o}^{i}, z_{1}^{i}, ..., z_{N}^{i}\} \) is able to recover the input example at each scale. In our experiments, we set \( \{z_{1}^{i}, ..., z_{N}^{i}\} \) to zeros, and \( Z_{o}^{i} \) to be a fixed random 3D noise (i.e., drawn once and then kept fixed). The reconstruction loss is defined using the mean squared error,

\[
\mathcal{L}_{\text{rec}} = \frac{1}{2} \mathbb{E}_{x}[\|x - \hat{x}\|^2],
\]

where \( \hat{x} \) is the generated shape at scale \( i \) when using the specific noise sequence \( \{Z_{o}^{i}, 0, ..., 0\} \).

Following suggestion by Shaham et al. [2019], we also use the reconstructed shape to determine the standard deviation \( \sigma_i \) of the Gaussian noise \( z_i \) (one that is injected to the generator in each level). Specifically, we set

\[
\sigma_i = \hat{\sigma} \| (\hat{x}_{i-1}^{*}) \uparrow - x_i \|, \quad i > 0,
\]
where \( \sigma \) is a predefined value and we use \( \sigma = 0.1 \) in our experiments. For the initial 3D noise \( Z_0 \) at the coarsest scale, we simply set \( \sigma_0 = 1.0 \).

3.4 Implementation Details

When constructing the voxel pyramid \( \{ x_0, \ldots, x_N \} \) of the input shape, we set the downsampling factor \( r = 0.75 \) in all our experiments. We choose the number of scales \( N \) such that the largest dimension of \( x_0 \) is around \( 22 \sim 33 \) (i.e., \( 2 \sim 3 \) times larger than the receptive field of the discriminator). For example, we set \( N \) to be 6 or 7 for the resolution \( 3 \times \sigma_N = 128 \). If the smallest dimension of any \( x_i \) is less than 15 voxels, we resize that dimension to be 15. In addition, we apply a Gaussian filter with \( \sigma = 0.5 \) to each \( x_i \) to smooth its boundary.

Throughout our experiments, we set the number of tri-plane feature channels \( C = 32 \). Each MLP decoder \( M_i \) has one hidden layer of dimension 32, and we apply Sigmoid function to restrict its output within the range \([0, 1] \). Each convolution block in the generator \( G_5 \) and discriminator \( D_7 \) follows the form of Conv-InstanceNorm-LeakyReLU [Ulyanov et al. 2016], except the last one, which has no normalization layer and activation function. All convolutional layers use number of channels 32. The architecture details are included in the Table 4 of the appendix.

We train every GAN for 2000 iterations using the Adam optimizer [Kingma and Ba 2015] with a learning rate of \( 1 \times 10^{-4} \) and \( \beta_1 = 0.5 \). Each training iteration includes three discriminator updates followed by three generator updates. For the WGAN-GP objective, we set its gradient penalty weight to be 0.1. Our method is implemented using PyTorch [Paszke et al. 2019] and trained on an Nvidia GeForce RTX 3090 GPU. With these settings, it takes about 4 hours to train our model (at resolution of 256 along its largest dimension), and less than 0.1 second for generating a shape at inference time.

4 EVALUATION RESULTS

We now present our evaluation experiments and results. The generated 3D shape is represented as a neural implicit function, which we visualize in a standard process: we extract the surface mesh using the Marching Cubes algorithm [Lorensen and Cline 1987] followed by Laplacian Smoothing [Vollmer et al. 1999] to reduce Marching Cubes’ artifacts. In Table 5 of the appendix, we list the training resolution and number of scales for all shape examples in the paper.

A gallery of generated shapes is shown in Fig. 6, and more results are provided in Fig. 13 and Fig. 14 of the appendix. We also provide an offline webpage in the supplementary material for interactive view of some example results. For each example, we show two generated samples that have the same bounding box size as the input shape, as well as samples that have different sizes and aspect ratios. As can be seen, all our generated shapes preserve the global structure of the input shape, while presenting rich local variations. Remarkably, the structural preservation and local variation adapt to input bounding box sizes and aspect ratios, allowing the user to quickly generate an ample assortment of shapes from a single input.

4.1 Comparison

**Compared methods.** We compare our method against two baselines. **SinGAN-3D** is an approach that we attempted by changing the 2D convolutions in SinGAN [Shaham et al. 2019] into 3D convolutions (to support 3D shape generation). For fair comparison, our implementation of SinGAN-3D uses the same discriminator structure and training strategy as used in our method. The only difference is that SinGAN-3D uses a 3D convolutional generator operating directly on voxel grids. **DGTS** [Hertz et al. 2020] is a prior 3D generative model working with a single example, although it aims to learn local geometric textures on mesh surfaces but not global structures. We use their published source code for comparison.

When comparing our method with SinGAN-3D, we set the largest dimension of the training shape to 128 voxels in both methods, and use 6 or 7 training scales depending on its aspect ratio. We note that SinGAN-3D has a much larger memory footprint than our method. Thus, in our experiments, we can not use it at a resolution as high as what we use in Fig. 6, because SinGAN-3D causes the GPU to run out of memory.

Also, unlike our method, DGTS takes as input a triangle mesh. Thus, for fair comparison, we use Marching Cubes to create a target mesh from a voxel grid that is also provided to our method. We then simplify it into a template mesh with 100 faces using an edge collapsing algorithm [Garland and Heckbert 1997]. With the target and template meshes, we use their optimization scheme to create the training meshes in 5 scales and finally train their model.

**Evaluation Metrics.** To measure the quality of generated shapes, we adopt the following metrics. **LP-IoU** and **LP-F-score** [Chen et al. 2021], according to their paper, measure the local plausibility of the generated shape. They are defined as the percentage of local patches (i.e., the \( 11 \times 11 \times 11 \) voxels in our setup) of a generated shape that are “similar” to at least one patch of the training shape. We consider two patches sufficiently “similar” if their IoU or F-score is above 0.95. To avoid sampling featureless patches—patches that are far away from the shape surface—we only consider voxel patches across the surface (i.e., having at least one occupied voxel and one unoccupied voxel in the central \( 3 \times 3 \times 3 \) area of the patch).

In addition, **SSFID** (Single Shape Fréchet Inception Distance) measures to what extent the generative model captures the patch statistics of the training shape. Similar to SIFTID (Single Image Fréchet Inception Distance) [Shaham et al. 2019], we use the deep features output by the second convolutional block in a pretrained 3D shape classifier (which we take from [Chen et al. 2021]). SSFID is defined as the Fréchet Inception Distance (FID) of those deep features between the generated and example shapes. Details of the metric evaluation are included in Appendix D.

We select 10 shapes in different categories as testing examples. These shapes have different topologies and patch variations across a range of scales. For each evaluated method and each testing example, we randomly generate 100 shapes and report the average scores of the metrics. For DGTS, we voxelize their generated meshes for the metric evaluation.

**Results.** The evaluation results are reported in Table 1, and the corresponding shapes are shown in Fig. 7 and Fig. 16 of the appendix. For nearly all testing examples, our method produces the best scores under all three metrics. Compared to the results of SinGAN-3D, our generated shapes better preserve the structure of the input shape (e.g., see the wall example in Fig. 7). In these tests, DGTS
Fig. 6. Randomly generated shapes. For each training shape, we show two randomly generated shapes that have the same spatial dimensions and additional shapes having different sizes and aspect ratios. Here, the largest spatial dimension of each training shape is 256, except for the vase and stalagmites example. For those two shapes, the largest spatial dimension is 192. From top to bottom, ©Acropolis by choly kurd under Standard License (turbosquid.com), ©Vase by davidmus under CC BY-SA, ©Terrain by BOXX3D under Editorial License (cgtrader.com), ©Table by akerStudio under RF, ©Castle by BlackMotion under RF, ©Desert cactus by exnihilo under RF, ©Stalagmites by wernie under RF, and ©Rock stairs by Misanthropiclion under RF.
Fig. 7. Visual comparison on our testing examples. Due to the limited space, here we only show comparison results on five testing examples and include the rest in Fig. 16 of the appendix. The quality score (Q) is LP-IoU defined in Sec. 4.1. The diversity score (D) is defined as the pairwise difference ($1 - \text{IoU}$) within a set of generated shapes (see Appendix D for detailed definition). For each method, we randomly generate 100 shapes to calculate both scores. From top to bottom, ©Acropolis by choly kurd under Standard License (turbo squid.com), ©Stone stairs by ivanzubak under Standard License (turbo squid.com), ©Wall by stray under RF, ©Cheese by thiennguyen2106 under RF, and ©Tree by Ada_King under Standard License (turbo squid.com).
We conduct ablation studies to validate the design choices of our (MISE) [Mescheder et al. 2019].

Table 1. Quantitative evaluation. ↑: a higher metric value is better; ↓: a lower metric value is better. We compare the generation quality of each method using three metrics. The last column is the average score over the ten testing examples.

| Metrics   | Methods          | Acropolis | Terrain | Stalagmites | Stairs | Examples | Rock | Wall | Vase | Cheese | Cactus | Tree | Avg. |
|-----------|------------------|-----------|---------|-------------|--------|----------|------|------|------|--------|--------|------|------|
| LP-IoU    | Ours             | 0.902     | 0.647   | 0.254       | 0.777  | 0.004    | 0.848| 0.175| 0.564| 0.583  | 0.015  | 0.477|
|           | SinGAN-3D        | 0.635     | 0.645   | 0.253       | 0.682  | 0.004    | 0.772| 0.173| 0.524| 0.343  | 0.001  | 0.403|
|           | DGTS             | 0.015     | 0.207   | 0.013       | 0.110  | 0.001    | 0.155| 0.004| 0.100| 0.028  | 0.001  | 0.063|
| LP-F-score| Ours             | 0.950     | 0.812   | 0.363       | 0.865  | 0.129    | 0.892| 0.214| 0.716| 0.847  | 0.149  | 0.594|
|           | SinGAN-3D        | 0.783     | 0.823   | 0.373       | 0.781  | 0.125    | 0.847| 0.207| 0.673| 0.642  | 0.048  | 0.530|
|           | DGTS             | 0.018     | 0.382   | 0.111       | 0.197  | 0.101    | 0.231| 0.007| 0.300| 0.243  | 0.059  | 0.165|
| SSFID     | Ours             | 0.037     | 0.050   | 0.078       | 0.102  | 0.020    | 0.272| 0.029| 0.065| 0.018  | 0.073  | 0.074|
|           | SinGAN-3D        | 0.065     | 0.058   | 0.089       | 0.093  | 0.035    | 0.475| 0.059| 0.145| 0.033  | 0.123  | 0.118|
|           | DGTS             | 3.62      | 2.08    | 1.81        | 2.00   | 1.51     | 3.67 | 1.95 | 1.36 | 0.403  | 0.201  | 1.724|

Table 2. Comparison of GPU memory and time cost. OOM: out of memory. Since both methods are multi-scale generative models, in this experiment, we use 6, 7 and 8 training scales for output resolutions 128³, 192³ and 256³, respectively. Inference time is the average over 100 trials.

| Methods          | Output Resolution | GPU Memory | Inference Time |
|------------------|-------------------|------------|----------------|
| Ours             | 128³              | 4.06G      | 0.041s         |
|                  | 192³              | 7.27G      | 0.051s         |
|                  | 256³              | 19.34G     | 0.057s         |
| SinGAN-3D        | 128³              | 9.31G      | 0.082s         |
|                  | 192³              | 20.45G     | 0.435s         |
|                  | 256³              | OOM        | N/A            |

Table 3. Ablation study. Metric values shown here are averages over the 10 testing examples, under the same setting as Table 1. Metric values for each testing example are available in Table 6 of the appendix.

|                      | LP-IoU ↑ | LP-F-score ↑ | SSFID ↓ |
|----------------------|----------|--------------|---------|
| Proposed Method      | 0.479    | 0.594        | 0.072   |
| Initial 2D Noises    | 0.438    | 0.554        | 0.086   |
| No Gaussian Blur     | 0.456    | 0.564        | 0.084   |
| No Weights Reuse     | 0.451    | 0.566        | 0.078   |

performs the worst due to its inability to learn large structures and anisotropic geometric features, which is indeed one of the limitations acknowledged in their paper [Hertz et al. 2020]. Also, it can not learn well complex topologies (e.g., see the acropolis example in Fig. 7). In those cases, DGTS takes a long time (>15 hours) for training meshes preparation and the training process, possibly due to the large number of vertices and many self-intersections resulted from their mesh optimization process.

Besides the shape quality, the generation diversity is also of interest. However, measuring diversity solely without taking into account quality makes little sense. As an extreme example, a set of shapes with strong random noise is probably considered diverse (or different from each other), but those shapes are useless because of the poor quality. As an attempt, we measure diversity using the pairwise difference within a set of random generated shapes shown in Fig. 7. Under this metric, SinGAN-3D usually obtains a better score, although its output shapes have lower quality than ours.

Another advantage of our method over SinGAN-3D is its efficiency, because unlike SinGAN-3D, our generator requires no 3D convolution. Here, we use both SinGAN-3D and our model to generate shapes and examine their subsequent GPU memory and time costs. When executing our model, we query all grid points through the MLP network in a single forward pass. The results are presented in Table 2, which shows that our model requires much less GPU memory and runs faster than SinGAN-3D as the output resolution increases. We can further reduce the GPU memory footprint by using algorithms such as Multi-resolution IsoSurface Extraction (MISE) [Mescheder et al. 2019].

4.2 Ablation Study

We conduct ablation studies to validate the design choices of our model. In Table 3, we compare our proposed method with the following variants:

Initial 2D noise, in which we independently sample three 2D noise maps to construct the input tri-plane map to G0. In this variant, the projection network is not needed any more. In contrast, our method projects a sampled 3D noise onto the three feature planes.

No Gaussian Blur, in which we do not apply Gaussian filter to blur the voxel grid of the input shape (recall Sec. 3.4). Without the Gaussian blur, the grid values would be either 0 or 1, indicating whether or not each grid point is occupied by the shape. As a result, the discriminator can trivially distinguish a synthesized shape, because the grid data decoded by the MLP network will likely have values between 0 and 1—a clue for the discriminator to do its job without carefully examining the shape features. Such a discriminator, although effective for distinguishing synthesized shapes, offers little help for improving the generator.

No Weights Reuse, in which we randomly initialize the weights of the GAN in each level at the beginning of the training process. On the contrary, our proposed method initializes the network weights of a particular level using the weights of a lower level GAN (recall Sec. 3.3).

Our method outperforms all these variants (see Table 3), and thus our design choices are well justified.
Fig. 8. Choice of the coarsest scale. In this example, we fix the finest level resolution (256 × 54 × 256) and the downsampling factor \( r \) is 0.75. We build three voxel pyramids of the input shape using different numbers of scales, and thus their coarsest level resolution varies. From left to right, the numbers of scales are 10, 9, and 7, respectively, and the coarsest level resolutions are 20 × 5 × 20, 27 × 6 × 27 and 46 × 10 × 46. In each setting, we show three random generated samples from the trained generative models. We also show the shapes’ local variations in the inset figures. See discussion in Sec. 4.3.

4.3 Choice of Voxel Pyramid Resolution
Recall that we build a voxel pyramid of the input example to train our generative model. Here we conduct an experiment to understand how the range of scales covered by the pyramid affects the generation results.

The experiment is shown in Fig. 8, where the finest level resolution of the voxel pyramid stays fixed, but the number of scales varies. When the number of scales is too large, the coarsest level resolution is low, and every voxel in the coarsest level grid occupies a large spatial region. As a result, the discriminator at this level (whose receptive field is always 11 × 11 × 11) examines the shape features in a large region, too large to allow any structural variation (see the second column of Fig. 8). The generated shapes simply overfit the example shape. On the other extreme, when the number of scales is too small, the coarsest level resolution is still high, and thus a voxel in the coarsest level grid occupies only a small region. Consequently, the discriminator is unable to force the generator to learn large-scale structures, and the generated shape appear to be structurally incoherent (see the fourth column of Fig. 8).

As a rule of thumb, in practice, we typically choose the number of scales such that the largest dimension of the coarsest grid is about two or three times larger than the discriminator’s receptive field.

Fig. 9. Shape interpolation and extrapolation. We perform shape interpolation and extrapolation by linearly blending the input noises of the source and target shapes. From top to bottom, the blending weights are −0.5, −0.25, 0, 0.25, 0.5, 0.75, 1, 1.25, and 1.5, respectively. Note the smooth transition across the source and target shapes.

4.4 Shape Interpolation and Extrapolation
Our generative model naturally supports shape interpolation and to a certain extent extrapolation. Consider two shapes generated from two initial 3D noises \( Z_0^1 \) and \( Z_0^2 \), respectively. The shape interpolation is straightforward, i.e., by feeding into the generative model a linearly interpolated noise \( Z_0^α = (1 - α) \cdot Z_0^1 + α \cdot Z_0^2 \), where \( α \) is the interpolation parameter. In fact, \( α \) is not necessarily in the range of [0, 1]; we can extend it to be negative or greater than one for shape extrapolation. Here, noise at other scales \( z_1, ..., z_N \) stay fixed.

We show three examples in Fig. 9, and include more results in Fig. 15 of the appendix. All the results present smooth transition
Learning to Generate 3D Shapes from a Single Example

4.5 Higher Resolution Output
The neural implicit function, enabled by the MLP network $M_N$ in our model, allows the model to discretize the output shape at an arbitrary resolution. We demonstrate this by showing a $\times 8$ upsampling result in Fig. 10. Here, the output shape is discretized by a grid whose resolution is 8 times higher than the input grid’s resolution. We compare the result with a simple upsampling strategy based on trilinear interpolation. The experiment shows that our result is much smoother while retaining sharp features. However, the ability to query at an arbitrary resolution does not imply arbitrary geometric details. The level of details that our model can synthesize is still limited by the training resolution.

4.6 Failure Cases
Although our generative model can learn from many different types of shapes, certain shapes still pose challenges to our model. 1) If the example shape does not have rich local features—such as a simple chair, which presents predominantly an overall structure (Fig. 11 top)—then our model is unable to learn and generate local variations, and it simply overfits the large structure. 2) Voxel-based deep generative models typically struggle to learn and generate thin structures, and ours is no different (Fig. 11 bottom). Primitive guidance [Tang et al. 2019] may help the generative model to learn thin structures. We leave it for future work.

5 LIMITATIONS AND FUTURE WORK
In this work, we propose a generative model that learns from a single example. Our method contrasts starkly to existing 3D shape generative models that require a large set of class-specific 3D shapes for training. Trained on a voxel pyramid of the input shape and operating on a hierarchy of tri-plane feature maps, our model is able to produce diverse shape variations across different bounding box sizes and aspect ratios. Meanwhile, all the generated shapes share similar large-scale structures as the input shape.

Our method also has some limitations. As discussed in recent works [Choi et al. 2021; Xu et al. 2021], zero boundary padding of the convolutional layers serves as an implicit spatial bias. Leveraging this bias, a convolutional generator can learn to produce globally-structured samples from spatially i.i.d. noises. Meanwhile, this bias leads to unbalanced output variation. For example, the output samples have less variation near the corners, because the zero-padding pattern is unique in corner regions. We reveal this limitation in our method through a purposely constructed example (see Fig. 12). We also tested the sinusoidal positional encoding strategy proposed
in [Xu et al. 2021], but find it no better, possibly because the periodic prior does not apply for 3D shapes. The tri-plane representation helps reduce the memory and time cost. But the highest resolution of the voxel pyramid is still limited (typically ≤ 256), because the discriminators still perform 3D convolutions on voxel grids. In future, we may further reduce our model’s memory cost by either using a more sophisticated training strategy or leveraging a more efficient voxel representation (e.g., octree [Wang et al. 2022]), deformable tetrahedron [Gao et al. 2020]).

Another interesting direction is to enable user control in the generation process, possibly via a conditional input to the network. For example, it would be useful to enable the user to use sketch or brush as a guidance to delete, add, or modify certain parts of the original shape [Li et al. 2020], while the generative model automatically makes coherent changes. Lastly, we would like to combine the internal (i.e., within the example) and external (i.e., from other shapes) patch information [Park et al. 2020] for more expressive shape generation.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback. This work was partially supported by the National Science Foundation (1910839).

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitlidakis, and Leonidas Guibas. 2018. Learning representations and generative models for 3D point clouds. In International conference on machine learning. PMLR, 40–49.

Heli Ben-Hamu, Haggai Maron, Itay Kerner, Gal Avineri, and Yaron Lipman. 2018. Multi-chart generative surface modeling. ACM Transactions on Graphics (TOG) 37, 6 (2018), 1–15.

Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. 2010. A connection between partial symmetry and inverse procedural modeling. In ACM SIGGRAPH 2010 papers. 1–10.

Ruojin Cai, Guandao Yang, Hedar Averbuch-Elor, Zekun Hao, Sergio Belongie, Noah Heli Ben-Hamu, Haggai Maron, Itay Kezurer, Gal Avineri, and Yaron Lipman. 2021. Efficient geometry-aware 3D generative adversarial networks. arXiv preprint arXiv:2112.07945 (2021).

Aiyang X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiang Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, et al. 2021. Efficient geometry-aware 3D generative adversarial networks. arXiv preprint arXiv:2112.07945 (2021).

Huawei Chen, Jiayi Liu, Weikai Chen, Shichen Liu, and Yajie Zhao. 2022a. Exemplar-based Pattern Synthesis with Implicit Periodic Field Network. arXiv preprint arXiv:2204.01671 (2022).

Zhiqin Chen, Vladimir G Kim, Matthew Fisher, Noam Aigerman, Hao Zhang, and Siddhartha Chaudhuri. 2021. Decor-gan: 3D shape detailization by conditional refinement. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 15740–15749.

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. Bsp-net: Generating compact meshes via binary space partitioning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 45–54.

Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 5039–5048.

Jooyoung Choi, Jungbeom Lee, Yonghyun Jeong, and Sungroh Yoon. 2021. Toward spatially unbiased generative models. arXiv preprint arXiv:2108.01285 (2021).

David S Ebert, F Kersten Muangsri, Darwyn Peachey, Ken Perlin, and Steven Worley. 2003. Texturing & modeling: a procedural approach. Morgan Kaufmann.

Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefel, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. 2004. Modeling by example. ACM transactions on graphics (TOG) 23, 3 (2004), 652–663.

Jun Gao, Wenzheng Chen, Tommy Xiang, Alex Jacobson, Morgan McGuire, and Sanja Fidler. 2020. Learning deformable tetrahedral meshes for 3D reconstruction. Advances in Neural Information Processing Systems 33 (2020), 9936–9947.

Lin Gao, Tong Wu, Yu-Jie Yuan, Ming-Xian Lin, Yu-Kun Lai, and Hao Zhan. 2021. Tim-net: Deep generative networks for textured meshes. ACM Transactions on Graphics (TOG) 40, 6 (2021), 1–15.

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In Proceedings of the 24th annual conference on Computer graphics and interactive techniques. 209–216.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 27—Proceedings. 2672–2680.

Karim Goodfellow, Soumya Parikh, and Daniel Cohen-Or. 2017. Interactive texture synthesis and transfer using convolutional neural features. ACM Transactions on Graphics (TOG) 36, 6 (2017), 1–18.

Ishana Gudrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved training of wasserstein gans. Advances in neural information processing systems 30 (2017).

Niv Haim, Ben Feinstein, Niv Granot, Assaf Shocher, Shai Bagon, Tal Dekel, and Michal Irani. 2021. Diverse Generation from a Single Video Made Possible. arXiv preprint arXiv:2109.08591 (2021).

Charles Han, Eric Risser, Ravi Ramamoorthi, and Eitan Grinspun. 2008. Multiscalar texture synthesis. In ACM SIGGRAPH 2008 papers. 1–8.

Rana Hanocka, Gal Metzer, Raja Gürses, and Daniel Cohen-Or. 2020. Point2Mesh: A Self-Prior for Deformable Meshes. ACM Trans. Graph. 39, 4 (2020). https://doi.org/10.1145/3385659.3392415

Amir Hertz, Rana Hanocka, Raja Gürses, and Daniel Cohen-Or. 2020. Deep Geometric Texture Synthesis. ACM Trans. Graph. 39, 4 (2020). https://doi.org/10.1145/3385659.3392471

Tobias Hinz, Matthew Fisher, Oliver Wang, and Stefan Wermter. 2021. Improved techniques for training single-image gans. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 1300–1309.

Pradeep Kumar Jayaraman, Joseph G Lambourne, Nishkrit Desai, Karl DD Willis, Aditya Sanghi, and Nile JN Morris. 2022. SolidGen: An Autoregressive Model for Direct B-rep Synthesis. arXiv preprint arXiv:2203.13944 (2022).

R. Kenny Benson, Theresa Barton, Xiangyan Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2020. ShapeAssembly: Learning to Generate Programs for 3D Shape Structure Synthesis. ACM Transactions on Graphics (TOG), 39, 6 (2020), Article 234.

Evangelos Kalogerakis, Siddharta Chaudhuri, Daphne Koller, and Vladlen Koltun. 2012. A probabilistic model for component-based shape synthesis. ACM Transactions on Graphics (TOG) 31, 4 (2012), 1–11.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings. http://arxiv.org/abs/1412.6980

Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In Proceedings of the 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1312.6114

Marian Kleinberg, Matthias Fey, and Frank Weichert. 2020. Adversarial Generation of Continuous Implicit Shape Representations. In Eurographics 2020 - Short Papers. 5ca3e9b122f61f8f06494c97b1afccf3.

Changjian Li, Hao Pan, Adrien Boussard, and Niloy J Mitra. 2020. Sketch2cad: Sequential modeling by sketching in context. ACM Transactions on Graphics (TOG) 39, 6 (2020), 1–14.

Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. 2017. Grass: Generative recursive autoencoders for shape structures. ACM Transactions on Graphics (TOG) 36, 4 (2017), 1–14.
A  NETWORK ARCHITECTURES

Table 4 describes the architecture for each module of our model for a single scale, and all scales share the same architecture. We will release all the code when the paper is published.

Table 4. Network architectures. AdaptAvgPool3D: adaptive average pooling, whose out channels denote the output size of x, y or z-axis. IN: instance normalization layer. LReLU: leaky ReLU with negative slope 0.2.

| Module (for each plane) | Layers | Out channels | Kernel size | Stride |
|-------------------------|--------|--------------|-------------|--------|
| Projection              | AdaptAvgPool3D | 8           | -           | -      |
| Conv2D                  | 32      | (3, 3)       | (1, 1)      | -      |
| Generator               | Linear+ReLU | 32          | (3, 3)      | (1, 1) |
| Conv2D                  | 32      | (3, 3)       | (1, 1)      | -      |
| MLP Decoder             | Linear+Relu | 32          | (3, 3)      | (1, 1) |
| Conv2D                  | 32      | (3, 3)       | (1, 1)      | -      |
| Discriminator           | Conv3D+IN+LReLU | 32          | (3, 3, 3)   | (2, 2, 2) |
| Conv2D                  | 32      | (3, 3, 3)    | (1, 1, 1)   | -      |
| Conv3D                  | 32      | (3, 3, 3)    | (1, 1, 1)   | -      |

B  DATA CONFIGURATION

In Table 5, we list the configuration (i.e. training resolution, number of scales) for all training shapes used in the paper.

C  MORE RESULTS

We show more of our random generation results in Fig. 13 and Fig. 14, more shape interpolation results in Fig. 15, and the remaining comparison results in Fig. 16. We also show more examples of querying higher resolution in Fig. 17. In addition, we present detailed metric values for our ablation study in Table 6. Finally, we recommend the reader to open the offline webpage in the supplementary material, for interactive view of some generation and comparison results.

D  EVALUATION METRICS

For LP-IoU and LP-F-score, we sample voxel patches of size $11 \times 11 \times 11$ with a stride of 3. A patch is considered valid only if it has at least one occupied voxel and one unoccupied voxel in its center $3 \times 3 \times 3$ area. We use all sampled valid patches from real shape as reference, denoted as $P_r$, and randomly select 1000 sampled valid patches from the generated shape, denoted as $P_g$. LP-IoU and LP-F-score are then defined as

$$
\text{LP-Score}(P_r, P_g) = \frac{1}{|P_g|} \sum_{x \in P_g} \mathbb{1}[\max_{y \in P_r} \text{Score}(x, y) > \delta],
$$

where Score is either IoU or F-score. F-score is calculated in a voxel-wise binary classification manner. The threshold $\delta$ is set to 0.95.

For SSFID, we take the 3D CNN classification network pretrained on ShapeNet [Chen et al. 2021] and use the deep feature map output by the second convolutional block to calculate the Fréchet Inception Distance (FID). The deep feature map is spatially 4 times smaller than the input voxel grid, and has 64 channels.

We compute the diversity score in Fig. 7 as follows. For each shape in the a set of $k = 100$ generated shapes $\{S_i\}$, we calculate its average distance to the other $k - 1$ shapes. Here we use $1 - \text{IoU}(\cdot, \cdot)$ as the distance measure. Then the score is the average value over all shapes in the set. Concretely, the diversity score is defined as

$$
\text{Div}(\{S_i\}) = \frac{1}{k} \sum_{i \neq j} \frac{1}{k - 1} \sum_{l \leq k \neq j} 1 - \text{IoU}(S_i, S_j).
$$

Table 5. Data configuration.

| Figure    | Data          | Resolution | #Training scales |
|-----------|---------------|------------|-----------------|
| Fig. 1 &  | Acropolis     | 256 × 72 × 118 | 8               |
| Fig. 6 &  | Terrain       | 256 × 256 × 70 | 8               |
| Fig. 9    |                | 256 × 256 × 92 | 8               |
| Fig. 13 & | Canyon        | 256 × 256 × 50 | 8               |
| Fig. 9    | Plant pot     | 256 × 94 × 78  | 8               |
|           | Icelandic mountain | 256 × 62 × 256 | 9               |
|           | Floating wood | 256 × 70 × 84  | 8               |
|           | Cube hotel    | 146 × 256 × 126 | 8              |
| Fig. 14   | Terrain 2     | 256 × 60 × 256 | 9               |
|           | Curved vase   | 162 × 148 × 192 | 8             |
|           | Elm tree      | 164 × 256 × 152 | 9             |
|           | Stone wall    | 256 × 56 × 138 | 8               |
|           | Ruined building | 104 × 82 × 128 | 6             |
|           | Natural arch  | 132 × 142 × 256 | 9         |
|           | Industry house | 120 × 98 × 256 | 8             |
| Fig. 7 &  | Acropolis     | 128 × 36 × 60  | 6               |
| Fig. 10 & | Terrain       | 112 × 128 × 36  | 6             |
| Fig. 16   | Stalagmites   | 128 × 100 × 106 | 7            |
|           | Stone stairs  | 112 × 52 × 128 | 7               |
|           | Rock          | 128 × 128 × 128 | 7             |
|           | Wall          | 128 × 88 × 46  | 6               |
|           | Vase          | 128 × 128 × 128 | 7             |
|           | Cheese        | 48 × 78 × 128  | 6               |
|           | Cactus        | 74 × 128 × 72  | 6               |
|           | Tree          | 60 × 128 × 66  | 6               |
| Fig. 8    | Mountain with water | 256 × 54 × 256 | 9          |
| Fig. 11 & | Chair         | 104 × 128 × 100 | 7             |
| Fig. 12   | Tree branches | 110 × 128 × 96  | 7            |
|           | Strange terrain | 128 × 128 × 40 | 7             |
Fig. 13. More random generation results (1). For each training shape, we show two generated shapes of the same spatial dimensions, with the rest having different sizes and aspect ratios. From top to bottom, ©Canyon by splod67 under CC BY, plant pot from ShapeNet, ©Icelandic mountain by saz88 under RF, ©Floating wood under RF, ©Cube hotel by fredolegros91 under RF, ©Log by towercg under RF, ©Boulder stone by Iml46 under RF, and ©Small town by pedram-ashoori under RF.
Fig. 14. More random generation results (2). For each training shape, we show two generated shapes of the same spatial dimensions, with the rest having different sizes and aspect ratios. From top to bottom. ©Terrain 2 by 3dDigital under Standard License (turboquad.com), ©Curved vase by davidmus under CC BY-SA, ©Elm tree by darkqueencpn under RF, ©Stone wall by bumblebrush under Editorial License (cgtrader.com), ©Ruined building by bizkit001 under RF, ©Natural arch by smanor under RF, and ©Industry house by lukass12 under RF.
Fig. 15. More results for shape interpolation and extrapolation. We perform shape interpolation and extrapolation by linearly blending the input noises of the source and target shapes. From top to bottom, the blending weights are $-0.5, -0.25, 0, 0.25, 0.5, 0.75, 1, 1.25$ and 1.5, respectively. Note the smooth transition across the source and target shapes.
Fig. 16. **Visual comparison on our testing examples.** The quality score (Q) is LP-IoU defined in Sec. 4.1. The diversity score (D) is defined as the pairwise difference (1−IoU) within a set of generated shapes (see Appendix D for detailed definition). For each method, we generate 100 shapes to calculate both scores. From top to bottom, ©Terrain by BOXX3D under Editorial License (cgtrader.com), ©Stalagmites by wernie under RF, ©Rock by georgij-space under RF, ©Vase by davidmus under CC BY-SA, and ©Cactus by m4rios under Standard License (turbosquid.com).
Table 6. **Ablation study.** ↑: a higher metric value is better; ↓: a lower metric value is better. See Sec. 4.2 for the description of each variant.

| Metrics       | Methods               | Acropolis | Terrain | Stalagmites | Stairs | Rock  | Wall  | Vase  | Cheese | Cactus | Tree  | Avg.  |
|---------------|-----------------------|-----------|---------|-------------|--------|-------|-------|-------|--------|--------|-------|-------|
|               | **Proposed Method**   | 0.902     | 0.647   | 0.254       | 0.777  | 0.004 | 0.848 | 0.175 | 0.564  | **0.583** | **0.015** | 0.477 |
|               | Initial 2D Noises     | 0.816     | 0.638   | 0.247       | 0.711  | 0.003 | 0.818 | 0.174 | 0.579  | 0.412  | 0.003 | 0.440 |
|               | No Gaussian Blur      | 0.887     | 0.648   | 0.249       | 0.756  | 0.002 | **0.854** | 0.173 | 0.561  | 0.407  | 0.003 | 0.454 |
|               | No Weights Reuse      | 0.874     | 0.637   | 0.220       | **0.821** | 0.004 | 0.821 | 0.179 | 0.575  | 0.374  | 0.005 | 0.451 |

| Metrics       | Methods               | Acropolis | Terrain | Stalagmites | Stairs | Rock  | Wall  | Vase  | Cheese | Cactus | Tree  | Avg.  |
|---------------|-----------------------|-----------|---------|-------------|--------|-------|-------|-------|--------|--------|-------|-------|
|               | **Proposed Method**   | 0.950     | 0.812   | 0.363       | 0.865  | **0.129** | 0.892 | 0.214 | 0.716  | **0.847** | **0.149** | **0.594** |
|               | Initial 2D Noises     | 0.884     | 0.809   | 0.360       | 0.803  | 0.112 | 0.867 | 0.212 | 0.715  | 0.730  | 0.074 | 0.557 |
|               | No Gaussian Blur      | 0.933     | 0.811   | 0.356       | 0.854  | 0.103 | 0.889 | 0.215 | 0.687  | 0.716  | 0.065 | 0.563 |
|               | No Weights Reuse      | 0.931     | 0.793   | 0.316       | **0.929** | 0.112 | 0.864 | 0.223 | 0.705  | 0.720  | 0.085 | 0.568 |

| Metrics       | Methods               | Acropolis | Terrain | Stalagmites | Stairs | Rock  | Wall  | Vase  | Cheese | Cactus | Tree  | Avg.  |
|---------------|-----------------------|-----------|---------|-------------|--------|-------|-------|-------|--------|--------|-------|-------|
|               | **Proposed Method**   | 0.037     | 0.050   | 0.078       | 0.102  | **0.020** | 0.272 | **0.029** | 0.065 | **0.018** | **0.073** | **0.074** |
|               | Initial 2D Noises     | 0.058     | 0.051   | 0.081       | 0.070  | 0.022 | 0.376 | 0.033 | **0.064** | 0.028  | 0.089 | 0.087 |
|               | No Gaussian Blur      | 0.035     | 0.060   | 0.087       | 0.048  | 0.025 | 0.378 | 0.036 | 0.088 | 0.032  | 0.093 | 0.088 |
|               | No Weights Reuse      | 0.039     | 0.056   | 0.116       | **0.032** | 0.027 | 0.303 | 0.033 | 0.095 | 0.033  | 0.086 | 0.082 |

Fig. 17. **Querying at a higher resolution.** We show ×4/×8 upsampling results by querying \( o(p) \) (resulted from the generated tri-plane representation) on a voxel grid with a ×4/×8 training resolution. The training resolutions for the three shape example are 128 × 100 × 106, 128 × 128 × 128 and 128 × 128 × 128, respectively. In this figure, visualized meshes are the direct output from Marching Cubes [Lorensen and Cline 1987].