Controllable Mesh Generation Through Sparse Latent Point Diffusion Models

Zhaoyang Lyu[1*] Jinyi Wang[1,2*] Yuwei An[1,4] Ya Zhang[1,2] Dahua Lin[1,3] Bo Dai[1]
[1]Shanghai AI Laboratory [2]Shanghai Jiao Tong University
[3]The Chinese University of Hong Kong [4]Tsinghua University
lyuzhaoyang@link.cuhk.edu.hk, jinyi.wang@sjtu.edu.cn
anyuwei@pjlab.org.cn, ya_zhang@sjtu.edu.cn
dhlin@ie.cuhk.edu.hk, daibo@pjlab.org.cn

Abstract

Mesh generation is of great value in various applications involving computer graphics and virtual content, yet designing generative models for meshes is challenging due to their irregular data structure and inconsistent topology of meshes in the same category. In this work, we design a novel sparse latent point diffusion model for mesh generation. Our key insight is to regard point clouds as an intermediate representation of meshes, and model the distribution of point clouds instead. While meshes can be generated from point clouds via techniques like Shape as Points (SAP), the challenges of directly generating meshes can be effectively avoided. To boost the efficiency and controllability of our mesh generation method, we propose to further encode point clouds to a set of sparse latent points with pointwise semantic meaningful features, where two DDPMs are trained in the space of sparse latent points to respectively model the distribution of the latent point positions and features at these latent points. We find that sampling in this latent space is faster than directly sampling dense point clouds. Moreover, the sparse latent points also enable us to explicitly control both the overall structures and local details of the generated meshes. Extensive experiments are conducted on the ShapeNet dataset, where our proposed sparse latent point diffusion model achieves superior performance in terms of generation quality and controllability when compared to existing methods. Project page, code and appendix: https://slide-3d.github.io.

1. Introduction

Being a fundamental representation of 3D objects in computer graphics, meshes are widely used in applications such as VR, AR, and games, and a generative model for meshes is thus of great value. By representing 3D objects with vertices, edges, and faces, meshes lead to more efficient modeling and computation of object geometries. However, such a specialized design also results in several intrinsic challenges for generative models. The first one is the irregular data structure of meshes, where the discrete vertex connections make it hard to define operations like convolution and up-sampling on meshes. Moreover, unlike point clouds and volumes, the topology of meshes is varying across different object instances in the same category. Therefore template-based methods [11,14,21,34,36,39] can only obtain meshes obeying the topology of used templates, causing defects like self-intersection when the deformation is significant.

Facing these challenges mentioned above, we propose to generate meshes indirectly via an intermediate representation that is easier to model. Inspired by recent successes of deep neural networks in modeling the distribution of point clouds [2, 22, 24, 38, 43] and reconstructing meshes from point clouds [11, 27], we propose to use point clouds as an intermediate representation of meshes. Consequently, the generation of meshes is effectively reformulated as the generation of point clouds, followed by transforming point clouds into meshes. Such a reformulation not only enables us to take advantage of the advances of point cloud generation methods, but also successfully bypasses the aforementioned challenges, as the distribution of point clouds is continuous and point clouds are unordered sets without explicit topology. In this paper, we adopt denoising diffusion probabilistic models (DDPMs) [13, 32], demonstrated promising results in modeling point clouds [22, 24, 43], to learn the distribution of the point clouds. And Shape as Points (SAP) [27] is employed to reconstruct meshes from the generated point clouds, which is a powerful surface reconstruction technique that can extract high-quality watertight meshes from point clouds at low inference times.

With the introduction of point clouds as the intermediate representation of meshes, we can use DDPMs to model the distribution of meshes. However, to ensure the quality of transformed meshes, the generated point clouds need

*Equal Contribution.

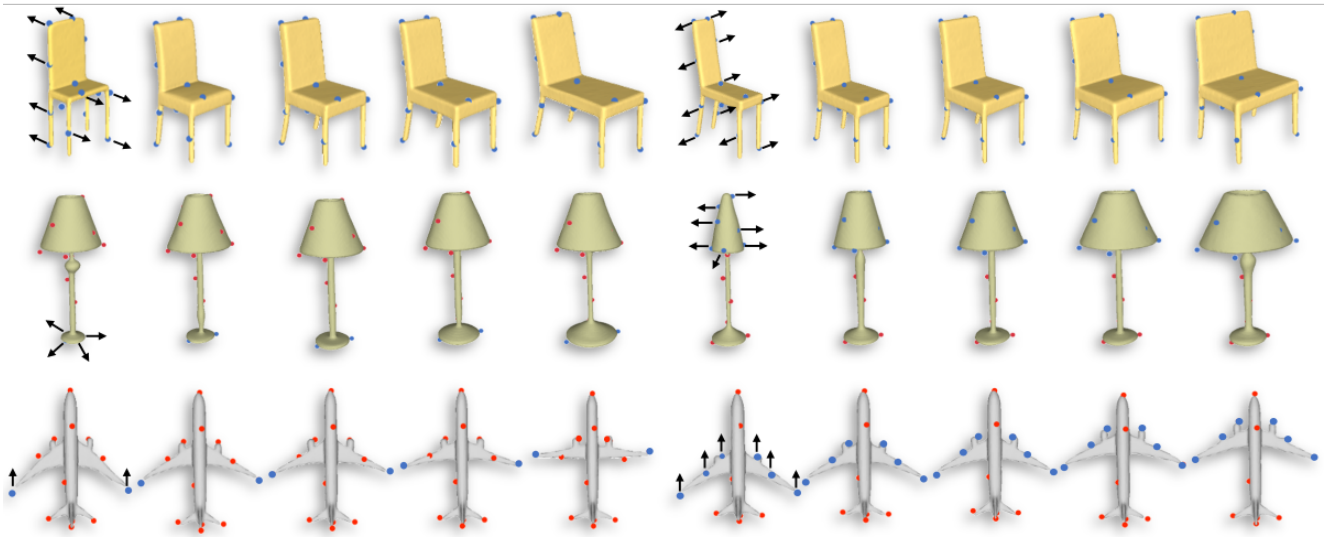


Figure 1. We can use the sparse latent points to control the shape of the generated meshes. Red points are stationary, and blue points are moving. Black arrows indicate the moving direction of the blue points. Some points are invisible because they are within the mesh. Note that the latent points are not always strictly lying on the surface of the generated meshes. This is because our point cloud decoder assumes that some noises exist in the positions of the sparse latent points. It will generate a mesh that best fits the latent points, but avoid generating defective meshes just to strictly fit the latent points.

to be sufficiently dense. This inevitably leads to two issues. At first, the overall computational complexity is high, since sampling thousands of points from DDPMs is quite time-consuming. It is also difficult to explicitly control the structure of meshes via dense point clouds, as the semantics of their points are not sufficiently compact. We therefore further encode a point cloud to a sparse set of semantic latent points with features attached to every point, and learn a **S**parse **L**atent po**I**nt **D**iffusion mod**E**l (**SLIDE**) for mesh generation following the framework of latent diffusion models [28, 33]. Specifically, we train two DDPMs to learn the distribution of this latent space. The first DDPM learns the distribution of positions of the sparse latent points, and the second one learns the distribution of the features conditioned on the positions of the points. By cascading these two DDPMs together, we can perform unconditional generation of sparse latent points and their features. We adopt farthest point sampling (FPS) to obtain the positions of sparse latent points from a dense point cloud, and a neural encoder is deployed to attach each latent point a semantically meaningful feature. Accordingly, a neural decoder is used to recover a dense point cloud from the positions and features of the sparse latent points and then reconstruct the mesh. In this way, we maintain the quality of generated meshes while being able to control their overall structures and local details respectively via controlling configurations and semantic features of the sparse latent points, as shown in Figure 1. Moreover, we find that sampling in this sparse latent point space is significantly faster than directly sampling dense point clouds.

We conduct experiments on the ShapeNet [4] dataset to compare both point cloud and mesh generation performance of SLIDE with other methods. SLIDE achieves superior performances in terms of both visual quality and quantitative metrics. It also demonstrates great flexibility in controlling the overall structures and local part shapes of the generated objects without using any part-annotated 3D data. In summary, the main contributions of our work are: **1)** We propose to use point clouds as the intermediate representation of meshes. By generating point clouds first and then reconstructing surface from them, we can generate meshes with diverse topology and high quality. **2)** We design a novel point cloud autoencoder to further encode point clouds to a sparse set of latent points with features attached to them. Sampling in this latent space is more efficient than directly sampling dense point clouds. **3)** By decomposing the learning of the positions of the sparse latent points and features of them, we can perform both unconditional point cloud generation and controllable point cloud generation based on the positions of the sparse latent points as shown in Figure 1. We can also perform both global and local interpolations in this latent space.

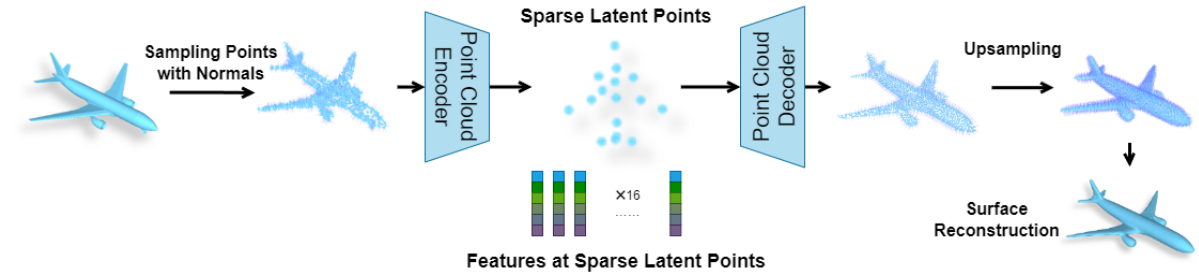
2. Background

2.1. Denoising Diffusion Probabilistic Model

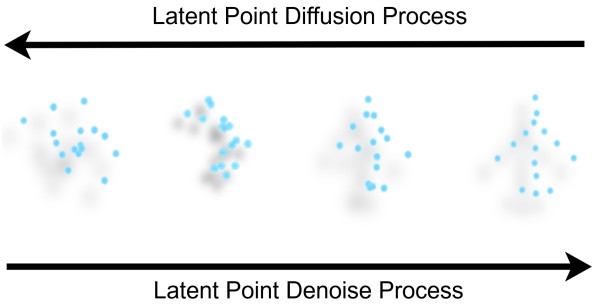
Denoising Diffusion Probabilistic Models (DDPMs) are a kind of generative models that learn the distribution of samples in a given dataset. A DDPM consists of two processes: A diffusion process and a reverse process. The diffusion process is defined as

$$q(\boldsymbol{x}^1, \cdots, \boldsymbol{x}^T|\boldsymbol{x}^0) = \prod_{t=1}^{T} q(\boldsymbol{x}^t|\boldsymbol{x}^{t-1}), \quad (1)$$

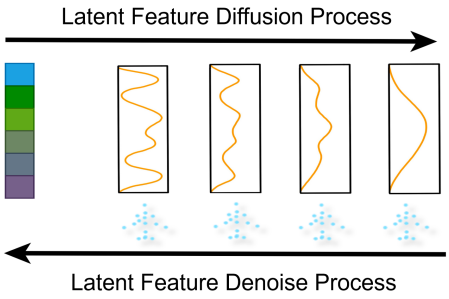
$$\text{where } q(\boldsymbol{x}^t|\boldsymbol{x}^{t-1}) = \mathcal{N}(\boldsymbol{x}^t; \sqrt{1-\beta_t}\boldsymbol{x}^{t-1}, \beta_t \boldsymbol{I}), \quad (2)$$



(a) The autoencoder encodes a mesh to features at the sparse latent points and decodes it back to a mesh.



(b) The DDPM learns the distribution of the sparse latent points.



(c) The DDPM learns the distribution of features at latent points.

Figure 2. Overview of our sparse latent point diffusion model and the two latent diffusion models.

$\boldsymbol{x}^0$ is a clean sample from the dataset, $\boldsymbol{x}^1, \cdots, \boldsymbol{x}^T$ are latent variables, T is the number of diffusion steps, $\mathcal{N}$ denotes the Gaussian distribution and β_t's are predefined small positive constants. See Appendix A.4 for details of these hyperparameters. The diffusion process gradually adds noise to the clean sample $\boldsymbol{x}^0$ and eventually turns it into a Gaussian noise $\boldsymbol{x}^T$ given that T is large enough. The reverse process is defined as

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}^0, \cdots, \boldsymbol{x}^{T-1}|\boldsymbol{x}^T) = \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}^{t-1}|\boldsymbol{x}^t), \quad \text{where } p_{\boldsymbol{\theta}}(\boldsymbol{x}^{t-1}|\boldsymbol{x}^t) = \mathcal{N}(\boldsymbol{x}^{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}^t, t), \sigma_t^2 \boldsymbol{I}), \tag{3}$$

and the mean $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}^t, t)$ is parameterized by a neural network. We use the proposed method in [13] to reparameterize $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}^t, t)$ as

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}^t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{x}^t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}^t, t) \right), \tag{4}$$

where $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. The reverse process simulates the reverse process of the diffusion process: It iteratively uses the network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}^t, t)$ to denoise a Gaussian noise and turn it into a clean sample. To generate a sample from the DDPM, we first sample $\boldsymbol{x}^T$ from the Gaussian distribution, then use Equation 3 to iteratively sample $\boldsymbol{x}^{T-1}, \boldsymbol{x}^{T-2}, \ldots, \boldsymbol{x}^0$ and finally obtain the sample $\boldsymbol{x}^0$.

We use the simplified loss proposed in [13] to train the DDPM:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}^0 \sim p_{\text{data}}} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x}^0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\|^2, \tag{5}$$

where p_{data} is the distribution of the dataset, t is sampled uniformly from $1, 2, \ldots, T$, and $\boldsymbol{\epsilon}$ is sampled from a Gaussian noise. We can see that the network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ actually learns to predict the noise $\boldsymbol{\epsilon}$ added to the clean sample $\boldsymbol{x}^0$. In other words, the network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ learns to denoise noisy samples. The architecture of the denoising network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ depends on the data format of $\boldsymbol{x}^0$. If $\boldsymbol{x}^0$ are images, a common choice of $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ is an Unet that predicts a per-pixel adjustment of the input noisy image $\boldsymbol{x}^t$. If $\boldsymbol{x}^0$ are point clouds, we can choose Point-Voxel CNN [43] or PointNet++ [24] for $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ that can predict a per-point displacement of the noisy input point cloud $\boldsymbol{x}^t$.

2.2. Latent Diffusion Model

Latent diffusion models [28, 33] are proposed for high-resolution image synthesis. Directly training DDPMs on high-resolution images and sampling from them are quite time-consuming. Latent diffusion models circumvent this problem by first encoding a high-resolution image to a low-dimensional latent space, and then training DDPMs in this latent space. Samples generated in this latent space are then decoded back to images. We follow the same procedures as [28] to train latent diffusion models. First, train an autoencoder in the data space. Then, train a DDPM using encoded samples from the dataset, namely, we can regard the variable $\boldsymbol{x}^0$ in Section 2.1 as the encoded variable of the original data sample by the pre-trained autoencoder.

3. Sparse Latent Point Diffusion Models

It is difficult to directly train a generative model on meshes, because meshes have irregular data structures. In general, a mesh is composed of vertices and faces. Vertices are points in the 3D space, while faces characterize the connections among vertices. It is easy for a generative model to model the positions of vertices, but it is dif-

ficult to model the connections among vertices. To tackle this problem, we propose to use point clouds with normals as an intermediate representation of meshes for their simple structure and efficient representation. Point clouds with normals can be sampled from the surface of meshes (2048 points in our experiments). Then we can use existing generative models [3, 23, 43] to model the distribution of point clouds. Finally, we use SAP [27] to reconstruct meshes from the generated point clouds. SAP is composed of an upsampling network and a Differentiable Poisson Surface Reconstruction (DPSR) algorithm. We refer readers to the original work or Appendix A.1 for details of SAP.

As mentioned above, we can use point clouds with normals as the intermediate representation of meshes. However, we think that point clouds are still a redundant representation of 3D shapes. In addition, point clouds are difficult to manipulate and control. To this end, we propose to further encode point clouds with normals to some sparse latent points with features as shown in Figure 2a. The intuition behind this representation is that a 3D shape can be decomposed to its skeleton that encodes the overall structure of the shape, and features located on the skeleton that encodes the local geometric details of the shape. To make the sparse latent points stretch over a given point cloud, we use farthest point sampling (FPS) to sample a given number (16 in our experiments) of points as the sparse latent points. We use two strategies to choose the first point in FPS: The first is choosing the centroid (mean coordinates of all the points) as the first point in FPS. The second is randomly choosing a point as the first point. We then design a point cloud encoder to encode a point cloud with normals to features attached to the sampled sparse latent points. We also design a point cloud decoder to decode the sparse latent points with features back to the input point cloud with normals. The details of the point cloud encoder and decoder are explained in the next section.

3.1. Architecture of the autoencoder

As mentioned above, we need a point cloud encoder to encode a point cloud to a sparse set of points with features, and a decoder to decode the sparse latent points back to the input point cloud. In this section, we explain the detailed architectures of the point cloud encoder and decoder.

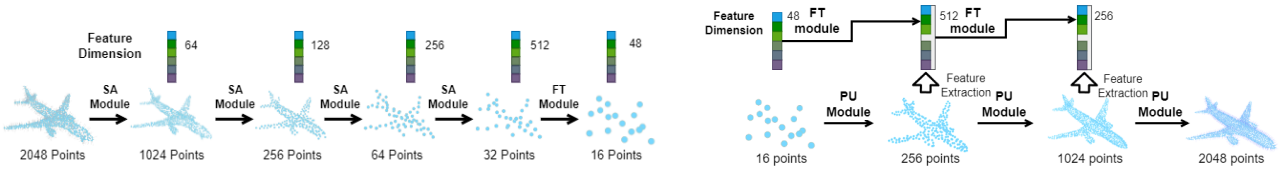
Point cloud encoder. The encoder needs to encode a point cloud to features at the FPS sampled sparse set of points. The overview of the encoder is shown in Figure 3a. It mainly consists of the improved Set Abstraction (SA) modules with attention mechanism proposed in PDR [24]. We briefly repeat the design of the SA module. The input of the SA module is a set of points with a feature attached to each point. The SA module subsamples the input points by farthest point sampling (FPS) and then propagates features to the subsampled points. Specifically, for every point in the subsampled points, it finds K nearest neighbors in the input points. Then it transforms the features of the neighbors by a shared Multi-layer perceptron (MLP) and aggregates the transformed features to this point through the attention mechanism. We refer readers to the original work [24] for details of the SA module. In our encoder, there are 4 cascaded SA modules. They iteratively downsample the input point cloud (2048 points) to $1024, 256, 64, 32$ points and propagate features to the downsampled points. The features of the input point cloud are simply the 3D coordinates and normals of every point.

Recall that the encoder needs to encode the input point cloud to features at the sampled sparse latent points (16 points). We achieve this by mapping features at the output of the last level SA module (consisting of 32 points) to the sparse latent points. We use the feature transfer (FT) module proposed in PDR [24] to map features from the last level SA module to the sparse latent points. The FT module can map features from one set of points to the second set of points, and the mapping is parameterized by a neural network. It is worth noting that the FT module requires points in the second set to have features, and these features are used as queries to aggregate features from the first set of points to the second set. We refer readers to Appendix A.2 or the original work [24] for details of the FT module. To utilize the FT module, we first use a lightweight PointNet++ to extract features for the sparse latent points themselves. Then we use it to map features from the last level SA module to the sparse latent points. The mapped features are concatenated to the original features at the sparse latent points to form the final features that represent the input point cloud.

Overall, as shown in Figure 3a, our point cloud encoder contains 4 SA modules to hierarchically extract features from the input point cloud, a light-weight PointNet++ to extract features for the sparse latent points, and an FT module to map features from the last level SA module to the sparse latent points.

Point cloud decoder. As mentioned above, the point cloud encoder can encode an input point cloud to features at the sampled sparse latent points. Next, we explain the point cloud decoder that can decode the sparse latent points with features back to the input point cloud with normals. Its overall structure is shown in Figure 3b. It contains 3 point upsampling (PU) modules that gradually upsample the sparse latent points (16 points) to 2048 points.

The input to the l-th PU module is a set of points $\boldsymbol{X}^l = \{x_j^l \in \mathbb{R}^3 | 1 \leq j \leq N^l\}$, with features attached to each point $\boldsymbol{F}^l = \{f_j^l \in \mathbb{R}^{d^l} | 1 \leq j \leq N^l\}$, where N^l is the number of input points to the l-th PU module, f_j^l is the feature at point x_j^l, and d^l is the dimension of the feature. The input to the first PU module is the sparse latent points $\boldsymbol{X}^1$ and their features $\boldsymbol{F}^1$. The PU module first uses a shared



(a) The point cloud encoder. (b) The point cloud decoder.

Figure 3. Architecture of the point cloud autoencoder.

Multi-layer Percpetron (MLP) to transform the feature f_j^l at every point x_j^l to γ displacements. Then the displacements are added to the original point x_j^l to obtain γ new points. In this way, the input points are upsampled by a factor of γ. To enforce the uniformness of the upsampled points, after upsampling, we use FPS to downsample the upsampled points by half. Overall, the input points are upsampled by a factor of $\gamma/2$ and we obtain the upsampled points $\boldsymbol{X}^{l+1} = \{x_j^{l+1} \in \mathbb{R}^3 | 1 \leq j \leq N_{l+1}\}$, where $N_{l+1} = \gamma N_l/2$. Note that the upsampled points $\boldsymbol{X}^{l+1}$ are controlled by the features $\boldsymbol{F}^l$ at the input points $\boldsymbol{X}^l$ and the learned MLP.

Next, we use the $(l+1)$-th PU module to further upsample the output of the l-th PU module, $\boldsymbol{X}^{l+1}$, but first we need to compute features for points in $\boldsymbol{X}^{l+1}$, because the PU module needs features at the input points to perform upsampling. We think that the features should consist of two parts. The first part is from the upsampled points $\boldsymbol{X}^{l+1}$ themselves. This part of the feature characterizes the shape of the current point cloud $\boldsymbol{X}^{l+1}$, and instructs how we can further upsample and refine it to make it more plausible. We use a improved PointNet++ [24] to extract this part of the feature from the set $\boldsymbol{X}^{l+1}$, and denote it as $\boldsymbol{F}_1^{l+1} = \{f_{1,j}^{l+1} \in \mathbb{R}^{d_1^{l+1}} | 1 \leq j \leq N_{l+1}\}$, where $f_{1,j}^{l+1}$ is the feature at point x_j^{l+1}, and d_1^{l+1} is the dimension of the feature.

The second part of the feature should come from the previous level PU module, namely, $\boldsymbol{X}^l$. This is because we want the information in the features at the sparse latent points can propagate along the PU modules layer by layer to control the shape of the final decoded point cloud. Afterall, all information of the input point cloud is encoded to the features at the sparse latent points, and we want the decoded shape to be consistent with the information stored in the features at the sparse latent points. Therefore, to obtain the second part of the feature for every point in $\boldsymbol{X}^{l+1}$, we use the FT module mentioned in the point cloud encoder to map features from $\boldsymbol{X}^l$ to $\boldsymbol{X}^{l+1}$, and the first part feature $\boldsymbol{F}_1^{l+1}$ at $\boldsymbol{X}^{l+1}$ are used as queries. After obtaining the second part of the features, $\boldsymbol{F}_2^{l+1} = \{f_{2,j}^{l+1} \in \mathbb{R}^{d_2^{l+1}} | 1 \leq j \leq N_{l+1}\}$, it is concatenated with the first part feature $\boldsymbol{F}_1^{l+1}$ to obtain the final feature for $\boldsymbol{X}^{l+1}$: $\boldsymbol{F}^{l+1} = \{(f_{1,j}^{l+1}, f_{2,j}^{l+1}) \in \mathbb{R}^{d^{l+1}} | 1 \leq j \leq N_{l+1}\}$, where $d^{l+1} = d_1^{l+1} + d_2^{l+1}$.

After upsampling the point cloud $\boldsymbol{X}^l$ to $\boldsymbol{X}^{l+1}$, and obtaining the features $\boldsymbol{F}^{l+1}$ at $\boldsymbol{X}^{l+1}$, we can use the PU module to further upsample $\boldsymbol{X}^{l+1}$. By applying the PU module and FT module iteratively, we can gradually upsample the sparse latent points to a point cloud of 2048 points. For the last PU module, we let it predict both γ displacements and γ normals, so that the final output point cloud has 2048 points with normals. Overall, the input to the point cloud decoder is the sparse latent points $\boldsymbol{X}^1$ (16 points) and their features $\boldsymbol{F}^1$. The decoder outputs the intermediate results $\boldsymbol{X}^2$ (256 points), $\boldsymbol{X}^3$ (1024 points), the final reconstructed point cloud $\boldsymbol{X}^4$ (2048 points) and normals $\boldsymbol{F}^4$.

Training of the autoencoder. The point cloud autoencoder is trained to encode the input point cloud and then reconstruct the point cloud. The input to the autoencoder is point cloud $\boldsymbol{X}_{\text{in}}$ (2048 points) with normals $\boldsymbol{F}_{\text{in}}$ sampled from the meshes in the dataset. The supervision is added on all the intermediate upsampling results in the point cloud decoder: $\boldsymbol{X}^2, \boldsymbol{X}^3, \boldsymbol{X}^4$. The loss is the sum of the Chamfer distance (CD) between $\boldsymbol{X}_{\text{in}}$ and $\boldsymbol{X}^2, \boldsymbol{X}^3, \boldsymbol{X}^4$, respectively. Note that when computing the CD loss between $\boldsymbol{X}_{\text{in}}$ and $\boldsymbol{X}^2, \boldsymbol{X}^3$, we first downsample $\boldsymbol{X}_{\text{in}}$ using farthest point sampling to the same number of points as $\boldsymbol{X}^2$ and $\boldsymbol{X}^3$, respectively. We also add a normal consistency loss between the ground-truth normals $\boldsymbol{F}_{\text{in}}$ and the predicted normals $\boldsymbol{F}^4$ with a weight of 0.1. See Appendix B.3 for details of this loss. We further add a slight Kullback–Leibler divergence loss (weight 10^{-5}) between the encoded features $\boldsymbol{F}^1$ and a standard normal distribution. This regularization term is to encourage the latent feature space to be simple and smooth, so that we can perform manipulation and interpolation in this space. Before the encoder encodes the input point cloud $\boldsymbol{X}_{\text{in}}$ to the sampled sparse latent points $\boldsymbol{X}^1$, we add a Gaussian noise with a standard deviation of 0.04 to the point positions in $\boldsymbol{X}^1$. This is to make the autoencoder more robust to the positions of the sparse latent points, so that even if the positions of the sparse latent points are not perfect (*e.g.*, human-edited sparse latent points), the autoencoder can still well reconstruct the input point cloud.

3.2. Train DDPMs in the Sparse Latent Point Space

After training the autoencoder, we can train latent DDPMs in the latent space of the autoencoder, while freezing the parameters of the autoencoder. Specifically, for each point cloud, we can encode it to features $\boldsymbol{F}^1$ at sparse latent points $\boldsymbol{X}^1$. We train two DDPMs in this latent space. The first one learns the distribution of the sparse latent points $\boldsymbol{X}^1$ and is illustrated in Figure 2b. The sparse latent points can be seen as a point cloud with very few points. Therefore, its distribution can be effectively learned by a DDPM

designed for point clouds, except that we can use a light-weight PointNet++ as the denoising network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}^t, t)$ in Equation 5.

The second DDPM learns the distribution of the feature $\boldsymbol{F}^1$ given the sparse latent points $\boldsymbol{X}^1$, which is illustrated in Figure 2c. For algebraic simplicity, we use $\boldsymbol{f}$ to denote the vector form of $\boldsymbol{F}^1$, namely, concatenating all feature vectors in $\boldsymbol{F}^1$ to a single vector $\boldsymbol{f}$. Similarly, let $\boldsymbol{x}$ be the vector form of $\boldsymbol{X}^1$. The second DDPM can be seen as a conditional DDPM that generates features $\boldsymbol{f}$ conditioned on the positions of the sparse latent points $\boldsymbol{x}$. To achieve this, we can simply replace the diffusion variable $\boldsymbol{x}$ in Equation 1 with $\boldsymbol{f}$, and adapt the reverse process in Equation 3 to

$$p_{\boldsymbol{\phi}}(\boldsymbol{f}^0, \cdots, \boldsymbol{f}^{T-1} | \boldsymbol{f}^T, \boldsymbol{x}) = \prod_{t=1}^{T} p_{\boldsymbol{\phi}}(\boldsymbol{f}^{t-1} | \boldsymbol{f}^t, \boldsymbol{x}),$$
$$\text{where } p_{\boldsymbol{\phi}}(\boldsymbol{f}^{t-1} | \boldsymbol{f}^t, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{f}^{t-1}; \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{f}^t, \boldsymbol{x}, t), \sigma_t^2 \boldsymbol{I}), \tag{6}$$

and $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{f}^t, \boldsymbol{x}, t)$ is parameterized as

$$\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{f}^t, \boldsymbol{x}, t) = \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{f}^t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\boldsymbol{f}^t, \boldsymbol{x}, t) \right). \tag{7}$$

To feed the input $\boldsymbol{f}^t$ and $\boldsymbol{x}$ to the denoising network $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}$, we concatenate each feature in $\boldsymbol{f}^t$ to the corresponding point in $\boldsymbol{x}$. In other words, the input to $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}$ can be seen as a sparse point cloud with noisy features attached to each point, and the output of the network $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}$ is to predict the noise added to each feature in $\boldsymbol{f}$. Therefore, we can use the improved PointNet++ in PDR [24] as the denoising network $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}$. Correspondingly, the training loss of the network $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}$ is

$$L(\boldsymbol{\phi}) = \mathbb{E}_{(\boldsymbol{X}_{\text{in}}, \boldsymbol{F}_{\text{in}}) \sim p_{\text{data}}} \| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\sqrt{\bar{\alpha}_t}\boldsymbol{f} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \boldsymbol{x}, t) \|^2,$$

where $\boldsymbol{X}_{\text{in}}$ is the point cloud (2048 points) sampled from a mesh in the dataset, $\boldsymbol{F}_{\text{in}}$ are the corresponding normals, $\boldsymbol{x}$ are sampled sparse latent points from $\boldsymbol{X}_{\text{in}}$, $\boldsymbol{f}$ is the encoded feature at $\boldsymbol{x}$ obtained by the trained authoencoder, t is sampled uniformly from $1, 2, \ldots, T$, and $\boldsymbol{\epsilon}$ is sampled from a Gaussian noise.

The detailed architecture of the two DDPMs is provided in Appendix A.3. After training the two DDPMs, we can use them to perform both unconditional 3D shape generation or controllable generation conditioned on the positions of the sparse latent points. To perform unconditional 3D point cloud generation, we can simply cascade the two DDPMs together: The first DDPM generates a set of sparse latent points, and the second DDPM generates features at the sparse latent points. Finally, the point cloud decoder decodes the sparse latent points with features to a point cloud. To achieve controllable generation, we can manipulate the positions of the sparse latent points, then feed the human-adjusted sparse latent points to the second DDPM to generate plausible features on them, and finally decode them to a point cloud.

4. Related Work

Mesh Generation. Most existing mesh generation methods rely on deforming a template mesh or another mesh [11, 14, 15, 21, 34, 36, 39], but meshes generated in this way are usually limited by the topology of the template or the initial mesh. And large deformations could cause defects. In contrast, our method is able to generate meshes from scratch with diverse topologies. Another line of works uses implicit representations of 3D shapes [6,7,10,17,25,26,31,41], but it usually requires dense neural network evaluations to extract meshes from the learned model.

Point cloud generation. Many learning-based methods are proposed to model the distribution of point clouds. Some works use generative adversarial networks (GANs) to generate point clouds [1, 19, 20, 30]. [1] also trains a latent GAN in the latent space of a point cloud autoencoder, but the autoencoder they use can only encode a point cloud to a global feature. Other works [16,18,38] use normalizing flows to model the distribution of point clouds. ShapeGF [3] learns gradient fields to move randomly sampled points to the surface of the objects. DDPMs have also been applied to point cloud generation [23, 43]. The generated point clouds of these methods can be transformed to meshes through surface reconstruction techniques [5, 8, 9, 12, 15, 29, 35, 37]. In this work, we choose SAP [27] for surface reconstruction for its efficiency and reconstruction quality.

Diffusion models. DDPMs are a kind of likelihood-based generative model that generate samples by gradually denoising a Gaussian noise [13,32]. They have shown promising results for 3D point cloud generation [23,43]. Our work is based on the recently proposed latent diffusion models [28, 33]. Latent diffusion models train diffusion models in the latent space of an autoencoder that encodes data samples to a more compact representation, and thus makes the training and sampling process of DDPMs faster.

Concurrent works. The concurrent work, LION [40], also proposes to use a latent diffusion model to learn the distribution of point clouds and then use SAP [27] to reconstruct meshes from point clouds, but the latent point cloud representation they use is a noisy point cloud with the same number of points (2048 points) as the original clean point cloud. In contrast, we encode the original clean point cloud to a sparse set of latent points (16 points) with features of dimension 48, which is a more compact representation and thus leads to faster training and sampling for DDPMs. This representation also enables us to perform controllable generation using the sparse latent points. Concurrently, NVMG [42] proposes to use voxels as the latent representations of meshes, but computational cost increases rapidly as the resolution of the 3D grid increases.

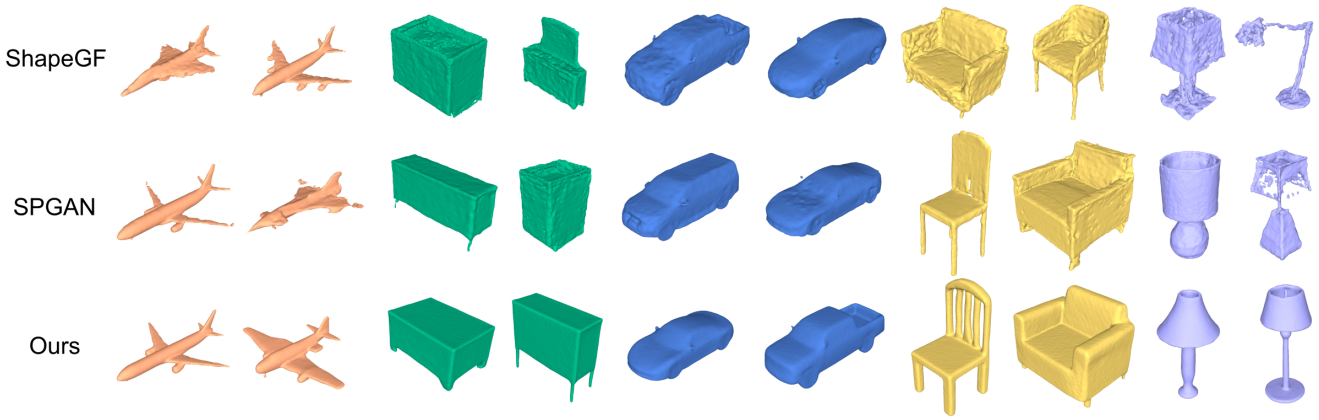


Figure 4. Meshes generated by our method and baselines. We can see that meshes generated by our method are more visually appealing. More examples of other baselines and our method are provided in Appendix B.5 and B.8.

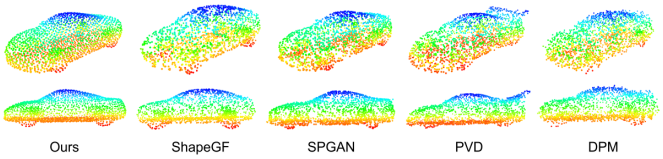


Figure 5. Point clouds generated by our method and baselines. More examples are provided in Appendix B.7 and B.8.

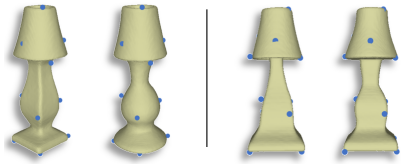


Figure 6. SLIDE is able to generate diverse meshes for the same set of sparse latent points due to the stochasticity in the feature generation process. Here are two pairs of generated lamps for the same set of latent points.

5. Experiment

We present our main experiment results in this section. We use ShapeNet [4] to train our mesh generative model, SLIDE, and compare it with other baselines. We use the pre-processed ShapeNet dataset provided by [27]. The detailed setups and complete experiment results are provided in Appendix B.

5.1. Evaluation Metrics

To evaluate the quality of generated meshes, we uniformly sample point clouds (2048 points) with normals from the generated meshes and reference meshes from the validation set. Then we use the commonly used point cloud evaluation metrics 1-NN [38], Minimum Matching Distance (MMD) and Coverage (COV) as our main evaluation tools. All of the metrics require a distance metric to compute the distance between two point clouds. We use the commonly used Chamfer distance (CD) and earth mover distance (EMD). We also use the normal consistency loss between two point clouds with normals. We find that it can better reflect the surface curvature differences between the two underlying meshes. Details of the normal consistency loss are described in Appendix B.3.

5.2. Point Cloud and Mesh Generation

We train SLIDE on 5 categories of the ShapeNet dataset: Airplane, cabinet, car, chair, and lamp. And compare with baselines TreeGan [30], SPGAN [20], ShapeGF [2], PVD [43], DPM [23]. All the baselines are trained by ourselves using their public codebase. We compare both the point clouds that they generate and meshes reconstructed from the point clouds using SAP. Meshes generated by SLIDE and baselines are shown in Figure 4. More examples and generated point clouds are shown in Appendix B.6, B.7, and B.8. We can see that SLIDE generates meshes of the highest visual quality, with smooth surfaces and sharp details. Since all the meshes are reconstructed from the generated point clouds using the same method, SAP. This means the quality of the generated point clouds greatly affects the quality of the reconstructed meshes. We provide an example of generated point clouds in Figure 5. More point cloud examples are provided in Appendix B.7. Indeed, we can see that point clouds generated by SLIDE spread more uniformly on the surface of the objects, and bear less noise compared with other methods. We attribute this to the design of our novel point cloud autoencoder. Quantitatively, we compute 1-NN, MMD, and COV on both generated point clouds and reconstructed meshes. Results are shown in Appendix B.4 and B.6. In terms of efficiency, the average generation time for a single point cloud of SLIDE is about 0.2s (See Appendix B.10 for more details of the generation time.) tested on a single NVIDIA A100 GPU, while the DDPM-based method that directly trains generative models on dense point clouds, PVD [43], need 2.93s to generate a point cloud tested on the same A100 GPU. LION [40] reports it needs 27.12s per shape. We also conduct an ablation study on the number of sparse latent points and the method to sample them. Results are shown in Appendix B.10.

Controllable Generation. As mentioned in Section 3.2, we can use the sparse latent points to control the gener-

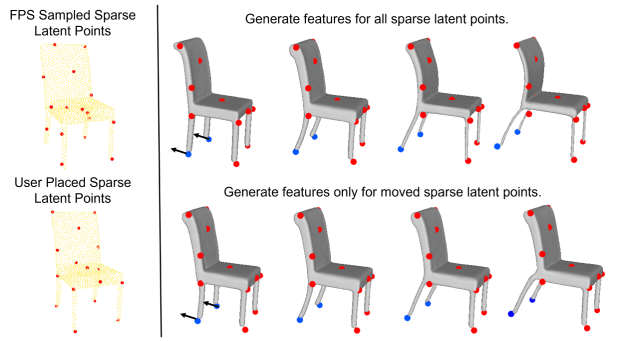


Figure 7. Use manually placed sparse latent points to control the rear legs of the generated chairs. The blue points are moving and the red points are fixed. The top row generates new features for all sparse latent points, and the bottom row generates new features only for moved points and fixes the features of the rest points.

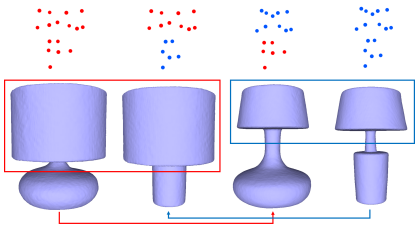


Figure 8. Perform shape combination. The first row are the sparse latent points of the original two lamps and the combined sparse latent points. The second row are the original two lamps (two sides) and the two lamps (middle two) obtained by combining the top part and bottom part of the original lamps.

ated mesh. Specifically, we can change the positions of the sparse latent points, then use the second DDPM to generate features at the latent points, and finally decode them to a point cloud and reconstruct the mesh. Several examples are shown in Figure 1. It shows that we can use the sparse latent points to control the overall scale of the generated mesh as well as change the position, scale, or shape of a part of the mesh. It is worth noting that we achieve this without any part annotations of the dataset. SLIDE is also able to generate diverse meshes even for the same set of sparse latent points due to the stochasticity in the feature generation process. Figure 6 gives two pairs of examples.

The sparse latent points in Figure 1 are obtained by FPS. At inference, we can also manually place the sparse latent points at regions of interest other than FPS sampled points and control the corresponding part. This is because we augment the FPS sampled sparse latent points with Gaussian noises during training and it makes our model robust to the positions of the sparse latent points. Figure 7 gives an example where we manually select the sparse latent points and control the rear legs of a chair. In addition, if we want to keep the rest part of a shape fixed while changing the part we want to edit, we can use the second DDPM to sample features only for moved sparse latent points and fix the features of rest points. See Figure 7 for an example. This is achieved by an algorithm similar to DDPM-based image inpainting and is described in Appendix A.5.

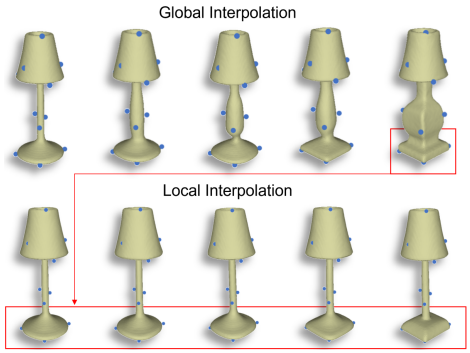


Figure 9. SLIDE is able to perform both global and local interpolations. The first row is an example of global interpolation. The second row interpolates between the bottom of the two lamps.

Shape Interpolation. To interpolate two shapes, we can interpolate both the positions and features between the corresponding latent points of the two shapes. See Appendix B.11 for how to establish correspondence between two sets of sparse latent points of two shapes. The top row of Figure 9 is an example of global interpolation. SLIDE is also able to perform local interpolation. We can interpolate only a part of the latent points, and keep the positions and features of the rest part of the latent points fixed. The bottom row of Figure 9 is an example of local interpolation.

Shape combination. We can also perform shape combinations using our sparse latent point-based representation of 3D shapes. We can simply combine the sparse latent points and their features from two or more source shapes to form new shapes. See Figure 8 for an example.

6. Conclusion

In this work, we propose to use point clouds as an intermediate representation of meshes. We train generative models on the point clouds sampled from the surface of the meshes, then we use SAP to reconstruct meshes from the generated point clouds. Meshes generated in this way demonstrate diverse topology. We propose to further encode dense point clouds to features at a sparse set of latent points, and train two DDPM in this latent space to learn the distribution of the positions and features of the latent points, respectively. Our sparse latent point diffusion model (SLIDE) outperforms DDPMs directly trained on point clouds in terms of both sample quality and generation speed. In addition, this sparse latent point representation allows us to explicitly control the shape of generated shapes, perform both global and local interpolations, and shape combination.

Acknowledgement. This project is funded in part by Shanghai AI Laboratory, CUHK Interdisciplinary AI Research Institute, and the Centre for Perceptual and Interactive Intelligence (CPII) Ltd under the Innovation and Technology Commission (ITC)'s InnoHK.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018. 6

[2] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *European Conference on Computer Vision*, pages 364–381. Springer, 2020. 1, 7

[3] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge J. Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part III*, volume 12348 of *Lecture Notes in Computer Science*, pages 364–381. Springer, 2020. 4, 6

[4] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 2, 7

[5] Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural dual contouring. *arXiv preprint arXiv:2202.01999*, 2022. 6

[6] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 5939–5948. Computer Vision Foundation / IEEE, 2019. 6

[7] Zhang Chen, Yinda Zhang, Kyle Genova, Sean Fanello, Sofien Bouaziz, Christian Häne, Ruofei Du, Cem Keskin, Thomas Funkhouser, and Danhang Tang. Multiresolution deep implicit functions for 3d shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 13087–13096, October 2021. 6

[8] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6970–6981, 2020. 6

[9] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. *Advances In Neural Information Processing Systems*, 33:9936–9947, 2020. 6

[10] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4857–4866, 2020. 6

[11] Kunal Gupta. *Neural mesh flow: 3d manifold mesh generation via diffeomorphic flows*. University of California, San Diego, 2020. 1, 6

[12] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2mesh: A self-prior for deformable meshes. *arXiv preprint arXiv:2005.11084*, 2020. 6

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020. 1, 3, 6

[14] Tomas Jakab, Richard Tucker, Ameesh Makadia, Jiajun Wu, Noah Snavely, and Angjoo Kanazawa. Keypointdeformer: Unsupervised 3d keypoint discovery for shape control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12783–12792, 2021. 1, 6

[15] Chiyu Jiang, Jingwei Huang, Andrea Tagliasacchi, and Leonidas J Guibas. Shapeflow: Learnable deformation flows among 3d shapes. *Advances in Neural Information Processing Systems*, 33:9745–9757, 2020. 6

[16] Hyeongju Kim, Hyeonseung Lee, Woo Hyun Kang, Joun Yeop Lee, and Nam Soo Kim. Softflow: Probabilistic framework for normalizing flow on manifolds. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 6

[17] Marian Kleineberg, Matthias Fey, and Frank Weichert. Adversarial generation of continuous implicit shape representations. *arXiv preprint arXiv:2002.00349*, 2020. 6

[18] Roman Klokov, Edmond Boyer, and Jakob Verbeek. Discrete point flow networks for efficient point cloud generation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXIII*, volume 12368 of *Lecture Notes in Computer Science*, pages 694–710. Springer, 2020. 6

[19] Chun-Liang Li, Manzil Zaheer, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov. Point cloud gan. *arXiv preprint arXiv:1810.05795*, 2018. 6

[20] Ruihui Li, Xianzhi Li, Ka-Hei Hui, and Chi-Wing Fu. Sp-gan: Sphere-guided 3d shape generation and manipulation. *ACM Transactions on Graphics (TOG)*, 40(4):1–12, 2021. 6, 7

[21] Minghua Liu, Minhyuk Sung, Radomir Mech, and Hao Su. Deepmetahandles: Learning deformation meta-handles of 3d meshes with biharmonic coordinates. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12–21, 2021. 1, 6

[22] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2837–2845, June 2021. 1

[23] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2837–2845, 2021. 4, 6, 7

[24] Zhaoyang Lyu, Zhifeng Kong, Xudong Xu, Liang Pan, and Dahua Lin. A conditional point diffusion-refinement

paradigm for 3d point cloud completion. *arXiv preprint arXiv:2112.03530*, 2021. 1, 3, 4, 5, 6

[25] Paritosh Mittal, Yen-Chi Cheng, Maneesh Singh, and Shubham Tulsiani. Autosdf: Shape priors for 3d completion, reconstruction and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 306–315, 2022. 6

[26] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019. 6

[27] Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. Shape as points: A differentiable poisson solver. *Advances in Neural Information Processing Systems*, 34:13032–13044, 2021. 1, 4, 6, 7

[28] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 2, 3, 6

[29] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021. 6

[30] Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 3858–3867. IEEE, 2019. 6, 7

[31] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020. 6

[32] Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015. 1, 6

[33] Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space. *Advances in Neural Information Processing Systems*, 34:11287–11302, 2021. 2, 3, 6

[34] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018. 1, 6

[35] Xingkui Wei, Zhengqing Chen, Yanwei Fu, Zhaopeng Cui, and Yinda Zhang. Deep hybrid self-prior for full 3d mesh generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5805–5814, 2021. 6

[36] Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1042–1051, 2019. 1, 6

[37] Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. Deep geometric prior for surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10130–10139, 2019. 6

[38] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4541–4550, 2019. 1, 6, 7

[39] Wang Yifan, Noam Aigerman, Vladimir G Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. Neural cages for detail-preserving 3d deformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 75–83, 2020. 1, 6

[40] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. Lion: Latent point diffusion models for 3d shape generation. *arXiv preprint arXiv:2210.06978*, 2022. 6, 7

[41] X Zheng, Yang Liu, P Wang, and Xin Tong. Sdf-stylegan: Implicit sdf-based stylegan for 3d shape generation. In *Computer Graphics Forum*, volume 41, pages 52–63. Wiley Online Library, 2022. 6

[42] Yan Zheng, Lemeng Wu, Xingchao Liu, Zhen Chen, Qiang Liu, and Qixing Huang. Neural volumetric mesh generator. *arXiv preprint arXiv:2210.03158*, 2022. 6

[43] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5826–5835, 2021. 1, 3, 4, 6, 7