TM-NET: Deep Generative Networks for Textured Meshes

LIN GAO, Institute of Computing Technology, CAS and University of Chinese Academy of Sciences
TONG WU, Institute of Computing Technology, CAS and University of Chinese Academy of Sciences
YU-JIE YUAN, Institute of Computing Technology, CAS and University of Chinese Academy of Sciences
MING-XIAN LIN, Institute of Computing Technology, CAS and University of Chinese Academy of Sciences
YU-KUN LAI, School of Computer Science and Informatics, Cardiff University
HAO ZHANG, School of Computing Science, Simon Fraser University

We introduce TM-NET, a novel deep generative model for synthesizing *textured meshes* in a *part-aware* manner. Once trained, the network can generate novel textured meshes from scratch or predict textures for a given 3D mesh, without image guidance. Plausible and diverse textures can be generated for the same mesh part, while texture compatibility between parts in the same shape is achieved via conditional generation. Specifically, our method produces texture maps for individual shape parts, each as a deformable box, leading to a natural UV map with limited distortion. The network *separately* embeds part geometry (via a PartVAE) and part texture (via a TextureVAE) into their respective latent spaces, so as to facilitate learning texture probability distributions conditioned on geometry. We introduce a *conditional autoregressive model* for texture generation, which can be conditioned on both part geometry and textures already generated for other parts to achieve texture compatibility. To produce high-frequency texture details, our TextureVAE operates in a high-dimensional latent space via dictionary-based vector quantization. We also exploit transparencies in the texture as an effective means to model complex shape structures including topological details. Extensive experiments demonstrate the plausibility, quality, and diversity of the textures and geometries generated by our network, while avoiding inconsistency issues that are common to novel view synthesis methods.

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

Additional Key Words and Phrases: Mesh representation, Mesh texture, Shape generation

1 INTRODUCTION

With rapid advances in deep learning, many recent works propose deep neural networks to learn 3D shape representations [Chen et al. 2020; Chen and Zhang 2019; Gao et al. 2019; Groueix et al. 2018; Li et al. 2017; Mescheder et al. 2019; Mo et al. 2019a; Park et al. 2019; Wang et al. 2018a; Wu et al. 2019], 3D generative models [Chen and Zhang 2019; Gao et al. 2019; Li et al. 2017; Wu et al. 2016, 2019], and reconstructions from different sources such as images [Li and Zhang

Authors' addresses: Lin Gao, Tong Wu, Yu-Jie Yuan and Ming-Xian Lin are with the Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy of Sciences and also with the University of Chinese Academy of Sciences. Yu-Kun Lai is with the School of Computer Science and Informatics, Cardiff University. Hao Zhang is with the school of Computing Science, Simon Fraser University. Authors' e-mails: {gaolin, wutong19s, yuanyujie, linmingxian20g}@ict.ac.cn, LaiY4@cardiff.ac.uk, haoz@sfu.ca.

Textured meshes generated from random samples

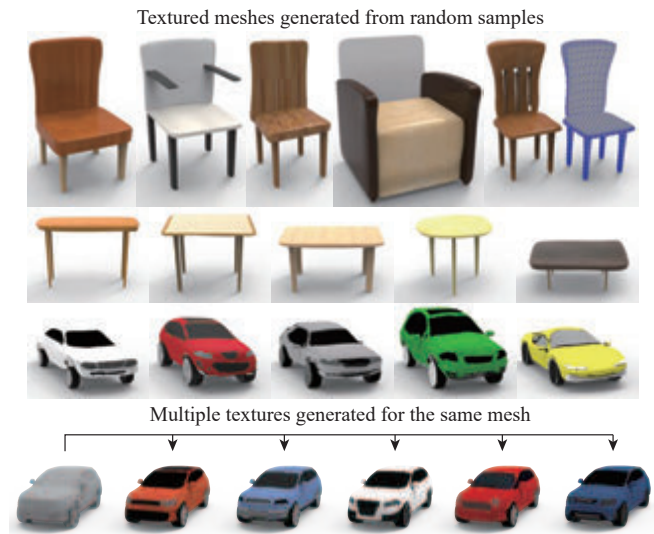Multiple textures generated for the same mesh

Fig. 1. Our deep generative network, TM-NET, can generate textured meshes from *random samples* in a latent space (top) and synthesize *multiple* textures for the same input shape (bottom). Our texture generation is part-aware, where different shape parts may be textured differently. Furthermore, even if the underlying mesh is relatively low-resolution (e.g., the top right chair has < 4K vertices), the generated textures can exhibit the *appearance* of topological details (e.g., holes between the slats on the chair back).

2021; Richter and Roth 2018; Wang et al. 2018b], point scans [Groueix et al. 2018; Park et al. 2019], and sketches [Han et al. 2017; Lun et al. 2017]. However, state-of-the-art methods still cannot produce 3D shapes with high visual quality, in terms of the realism and richness of the shapes' appearance. The generation and reconstruction of surface and topological details are especially difficult, due to lack of data and the technical and computational challenges in training deep models for high-resolution geometries.

In the absence of geometric details, a classical graphics "trick" is to create the *appearance* of surface richness by simulating details in the *image space*, e.g., via texture or bump mapping. As shown in Figure 1, textured meshes can exhibit the kind of realism and visual fidelity that geometry alone cannot convey, even when the underlying mesh resolution is fairly low. However, despite the flourishing of geometric deep learning, there have been few attempts at developing a deep generative model for *textured* meshes. To the
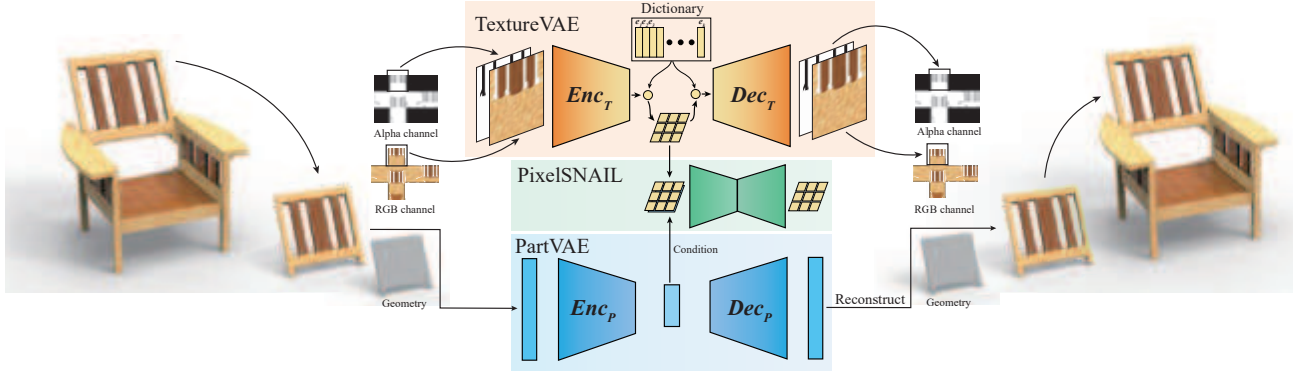
Fig. 2. An overview of the key components of TM-NET, for textured mesh generation. Each part is encoded using two Variational Autoencoders (VAEs): PartVAE for geometry with Enc_P as the encoder and Dec_P as the decoder, and TextureVAE for texture with Enc_T as the encoder and Dec_T as the decoder. For texture generation, TM-NET designs a conditional autoregressive generative model, which takes the latent vector of PartVAE as condition input and outputs discrete feature maps. These feature maps are decoded as texture images for the input mesh geometry.

best of our knowledge, existing works on textured shape generation have predominantly relied on single- or multi-view image guidance [Chen et al. 2019; Liu et al. 2019; Raj et al. 2019].

In this paper, we introduce TM-NET, a deep generative network for *textured meshes*, aiming to fill several gaps in learning-based realistic 3D shape modeling. First, we seek a *generic* decoder which can generate textured meshes with or without image guidance. Next, the network can generate novel textured meshes, e.g., from random samples in some latent space, and predict textures for a given 3D shape. Last but not least, the generative model should be *part-based* to allow different shape parts to be textured differently, which is naturally expected for many real-world objects.

With these goals in mind, our design for TM-NET draws inspiration from SDM-NET [Gao et al. 2019], a deep generative network for *structured deformable meshes*. Specifically, SDM-NET generates meshes formed by parts, where each part is homeomorphic to a cuboid box and finer-scale geometry of the part can be controlled by deforming the template box. Architecturally, the network consists of a part-level variational autoencoder (VAE) for learning genus-zero mesh parts and a structure-part VAE (SP-VAE), which jointly learns part structures and part geometries from a shape collection.

Figure 2 shows the key components of our network TM-NET, with the dataset and training processes detailed in Section 4. Similar to SDM-NET, our model learns a structured mesh representation, where a textured mesh is represented at both the part level and the overall shape level using VAEs. This allows both geometry and texture to be represented in a well-aligned canonical form: the rich geometry details are approximated and represented by deformations enabled by the template boxes, and these boxes work naturally with the UV-maps for texture mapping. Specifically, we can cut designated edges of a template box and flatten the box by unfolding it onto the UV space that contains texture information; see Figure 3, for example. The collection of deformable boxes, when mapped to the parameter space, forms a consistent texture atlas even for shapes with substantial structure and/or geometry variations, making effective learning of structured and textured shapes possible.
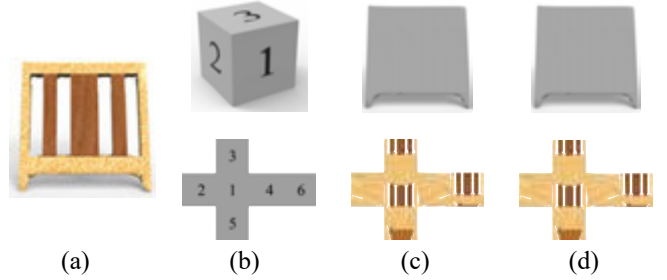

Fig. 3. Texture representation with deformed boxes: (a) input model; (b) template box and unfolded UV map where numbers on the box faces help illustrate the unfolding process; (c) geometry and texture representations with the deformable box; (d) decoded geometry and texture.

However, the generation of plausible, high-quality, *and diverse* textured meshes poses several new challenges:

- First, the mapping between shape parts and textures is generally far from one-to-one — the same shape part may possess drastically different textures and vice versa. Thus a joint learning of part geometry and part texture, which resembles the SP-VAE in SDM-NET, would be unsuitable. We need a different and more adaptive encoding approach.
- Second, a critical requirement to part-based texture representation is to ensure that *compatible* textures are generated for different parts. Hence, we must add a new layer of dependency, via conditional modeling, into our network design.
- Third, to produce textures with high-frequency details, we must address the tendency of VAEs to generate blurry outputs [Dosovitskiy and Brox 2016].

To address these challenges, TM-NET *separately* embeds part geometry (via a PartVAE) and part texture (via a TextureVAE) into their respective latent spaces (see Figure 4), as a means to facilitate learning the texture probability distribution conditioned on geometry. To resolve possible ambiguities due to identical or similar part
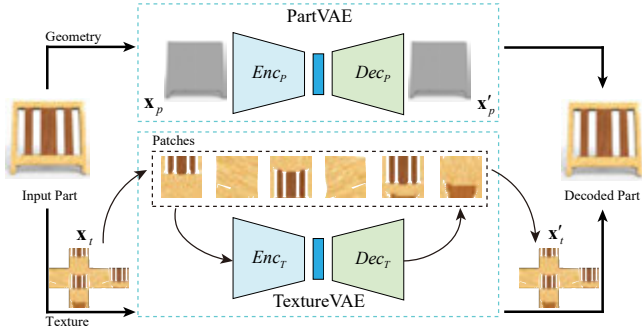
Fig. 4. Network architecture for representing a textured part: a PartVAE for encoding part geometry and a TextureVAE for texture.

geometries having different textures, we introduce a *conditional autoregressive* model based on PixelSNAIL [Chen et al. 2018] for texture generation. Within this framework, compatibility between textures generated for different shape parts is achieved via a conditional generative approach. Specifically, we designate a seed part for each shape category. The network is trained to texture the seed part conditioned on its geometry, while for other parts, the condition input is a concatenation of part geometry and the VGG [Simonyan and Zisserman 2014] image features of the texture of the seed part. In addition, we add three fully connected layers before feeding the condition into the autoregressive network. Overall, the learned mapping between geometry and texture is *one-to-many*.

Last but not least, since part geometry and texture possess different data characteristics, e.g., smooth geometry vs. high-frequency image details, the two part-level VAEs have different architectures. In particular, to address the blurry image issue with conventional VAEs, we adapt VQ-VAE-2 [Razavi et al. 2019a,b] for learning and generating textured mesh parts. With the aid of vector quantization (VQ), our approach allows the learning of a high-dimensional texture latent space to facilitate the generation of quality textures, with high-frequency details and less blurriness.

Our main contributions can be summarized as follows:

- To the best of our knowledge, TM-NET is the first deep generative model which learns to synthesize high-quality textured meshes in a *part-aware* manner.
- Once trained, our network can generate novel textured meshes from random samples in a latent space or predict textures for a given 3D mesh, with or without image guidance. Plausible and diverse textures can be generated for the same mesh part, with texture compatibility between parts in the same shape.
- By utilizing texture *transparency* via the alpha channel, TM-NET is able to generate and reproduce the *visual appearance* of topological details such as holes in the chair back; see Figure 1. Visually, the obtained results are superior than those attainable by state-of-the-art generative networks.

We study the generative capabilities of TM-NET under several application scenarios, including autoencoding and novel generation of textured meshes, shape-conditioned and image-guided shape texturing, as well as textured shape interpolation in the latent space.

The quality of the textured meshes generated by our network is demonstrated both qualitatively and quantitatively. Evaluation is conducted via ablation studies and comparisons to state-of-the-art generative models, including SDM-NET [Gao et al. 2019], BSP-Net [Chen et al. 2020], and Texture Fields [Oechsle et al. 2019].

2 RELATED WORK

We cover works most closely related to ours: deep generative models for 3D shapes, texture synthesis, and joint 2D-3D encoding.

Deep Generative Models for 3D Shapes. With the development of the deep neural networks such as Variational Autoencoders (VAEs) [Kingma and Welling 2013] and Generative Adversarial Networks (GANs) [Goodfellow et al. 2014], novel 3D shapes can be effectively generated by learning the distributions of existing data. For generating 3D shapes, existing deep generative models produce shapes in different representations, including voxels [Wu et al. 2016], point clouds [Achlioptas et al. 2018], implicit functions [Chen and Zhang 2019; Mescheder et al. 2019; Park et al. 2019], deformable meshes [Tan et al. 2018], multi-patch representations [Groueix et al. 2018; Smirnov et al. 2021] and and structured representations, e.g. GRASS [Li et al. 2017], StructureNet [Mo et al. 2019a] and SDM-NET [Gao et al. 2019]. Recent methods with improved shape representations allow 3D shapes with richer geometric details and more complex structures to be synthesized or reconstructed, e.g., [Li and Zhang 2021; Xu et al. 2019]. However, none of these methods generate 3D models with textures. Furthermore, textures are crucial for the visual realism and rich appearance of 3D shapes. In this paper, we build our texture generation method on SDM-NET, a recent shape generation method based on structured deformable meshes, where each part is represented by using deformations from a box. The method produces shapes with fine geometric details. For more discussions on 3D generative models, we refer readers to [Ahmed et al. 2018; Bronstein et al. 2017; Xiao et al. 2020; Yuan et al. 2021].

Texture synthesis. Texture synthesis for 2D images [Efros and Leung 1999] and 3D solids [Kopf et al. 2007] have been widely studied in computer graphics. In recent years, neural network based methods have been proposed [Gatys et al. 2015; Henzler et al. 2021; Snelgrove 2017; Zhou et al. 2018] for the synthesis problem, leading to improved results, although these methods are still restricted to image textures, rather than generating textures over surfaces of 3D shapes. Another popular texture synthesis problem is synthesizing geometric textures [Berkiten et al. 2017; Hertz et al. 2020; Lai et al. 2005], which mainly focus on transferring geometric details from an example model to the target surface.

For surface texture generation, the texture representation strongly relies on the underlying geometry representation. Although point clouds are flexible, and suitable for applications such as segmentation and classification [Qi et al. 2017a,b], they are too sparse to represent geometry details [Hu et al. 2019], and also suffer from this major issue when used for representing high-resolution textures. For the voxel-based representation, the texture representation based on voxels [Tulsiani et al. 2017] is also limited by the resolution due to the high memory overhead.

Recently, some works propose to generate textures using deep generative models. According to the statistical properties, textures

on 3D shapes can be divided into two categories, namely stochastic textures such as wood and marble which can often be compactly modeled using a procedural generation process or some samples, and non-stochastic textures which are generic images on surfaces. Several methods [Chen et al. 2015; Park et al. 2018] work on providing stochastic textures or material suggestions for 3D shapes and 3D scenes. These methods can search for suitable textures or materials in a database but are not designed to generate novel textures. Henzler et al. [2019] propose a method to encode stochastic textures from 2D exemplars for synthesizing 3D solid textures. While solid textures are useful for certain applications, they are not associated with shape surfaces. The method is also restricted to stochastic textures, which are insufficient for texturing general 3D objects.

Generating textured 3D Shapes. To obtain textured 3D shapes, some works use single-view color images as 2D supervision to recover 3D objects with colors. Hu et al. [2019] use a two-stage method that first infers the object coordinate map and then uses reprojection onto the input image to recover dense point clouds with colors. The method is able to handle input images of different (and even unseen) object categories; however, it only generates a partial point cloud with visible colored points for a given input image, and multi-view fusion is required to obtain complete shapes. Other methods, e.g., [Chen et al. 2019; Liu et al. 2019; Pavllo et al. 2020], rely on differentiable renderers to train networks to predict vertex positions and colors for meshes with a fixed topology. Recently, deep implicit surface representations [Chen et al. 2020; Chen and Zhang 2019; Mescheder et al. 2019; Park et al. 2019] have achieved a great success in geometry modeling. Several pioneering implicit representation methods take RGB or RGB-D scans as input to infer both implicit surfaces and associated textures for human body shapes [Saito et al. 2019, 2020], man-made objects [Huang et al. 2020] and indoor scenes [Dai et al. 2021]. These methods rely mainly on the input image or depth to provide constraints, whereas our approach aims to generate textures without image or depth guidance. Moreover, the use of a fixed-topology shape such as a sphere as the starting point also prevents these methods from handling shapes with complicated structures or rich geometric details.

A fundamental challenge for texturing 3D shapes is an effective texture mapping from the 3D surface to some texture (UV) space. UV mapping is usually difficult to be consistently defined across shape collections, and previous UV-mapping based methods mainly deal with models with fixed topology including planar topology (for faces) [Saito et al. 2017] and spherical topology (for birds) [Kanazawa et al. 2018]. These representations however cannot be generalized to cope with shapes with complex structures. Recent work Texture Fields [Oechsle et al. 2019] avoids this problem by learning textures using mapping from the 3D texture space. However, the use of 3D solid space is not only expensive, but also harder to learn, as textures of interest are usually only shape surfaces. Raj et al. [2019] propose a two-stage approach to generating textured meshes, where in the first stage, a network is trained to map 2.5D depth images to texture images, and in the second stage, multiview texture images are fused. However, the method does not give details how to ensure seamless synthesis of texture atlas, and how to ensure sufficient coverage of texture images, especially for shapes with complex structures

where self-occlusion is a problem, and shows examples of a single category (cars) which are of spherical topology. Unlike all these existing methods, our approach is able to generate higher-quality textured meshes with more complex structures. Our method can also generate geometry and texture simultaneously, which cannot be achieved with existing methods.

Novel view synthesis (NVS). Given a single image, humans can easily extrapolate what the object/scene looks like from another view. NVS focuses on generating images from an unknown viewpoint with an input image. Sun et al. [2018] takes multiple images from different source viewpoints and predicts a flow field to move the pixels from the source image to the target image. Then it aggregates predictions from different viewpoints to produce the target image. SRN [Sitzmann et al. 2019] proposes a continuous 3D-structure-aware 3D scene representation that maps the world coordinates into a feature representation which helps preserve the geometry surface from novel viewpoints. GeLaTO [Martin-Brualla et al. 2020] moves a step forward to generate rendered images of transparent objects from a new viewpoint and interpolate rendered images from different viewpoints using U-NET [Ronneberger et al. 2015] and AtlasNet [Groueix et al. 2018]. However, their intermediate texture is an implicit representation, which cannot be directly mapped onto 3D shapes for typical graphics applications. Although textured meshes can be rendered from novel viewpoints, our method is fundamentally different from NVS methods, which have been an active research topic in recent years: not only does our method not require a single-view image as input (and can generate textured meshes directly), our method also ensures the rendered images from different views are consistent, which cannot be guaranteed with NVS.

3 METHOD

Given a collection of textured 3D meshes, TM-NET encodes the geometry associated with textures for generation and other applications such as conditional shape texturing and textured shape interpolation. As discussed before, we build on recent SDM-NET [Gao et al. 2019] which provides structural deformable boxes to represent 3D objects, each corresponding to a semantic part. Thanks to this representation, it intrinsically provides a way to define consistent texture atlases for objects, even if shapes in the collection may not have identical topology (e.g. a table may have different numbers of legs). Each box can be easily unfolded to the texture domain, with no extra distortion at this stage. In the following subsections, we will describe our texture representation and how to convert textured 3D models to this representation. We then describe the details of our novel network architecture, i.e. how the geometry detail determines its texture, first at the part level and finally the object level.

The mesh geometry and structure are learned jointly by one global VAE, to ensure that the generated textured meshes are consistent in both geometry and structure. Since a single part can be textured differently while still being plausible, instead of generating textures conditioned on the input geometry in a one-to-one manner, we learn a probability distribution based on PixelSNAIL [Chen et al. 2018]. Once the distribution is learned, we can sample several times on the distribution which will lead to multiple reasonable textures for the input shape. Directly applying this network at the part level could
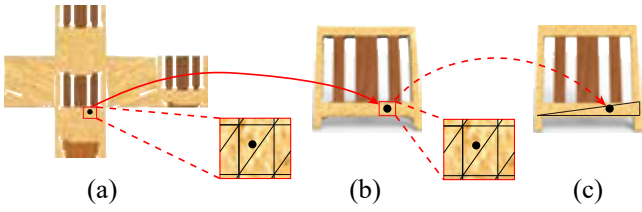
Fig. 5. Illustration of the procedure that maps the ground truth texture to the unfolded box to form a texture image. (a) the texture image to be filled, (b) the deformed box that approximates the input model part, (c) the input model containing ground truth texture.

Fig. 7. Network architecture of TextureVAE. The encoder maps the input image patch onto two continuous feature maps, top (t) and bottom (b). Then the dictionary-based vector quantization is performed. The decoder takes the discrete feature maps as input and reconstructs the image.
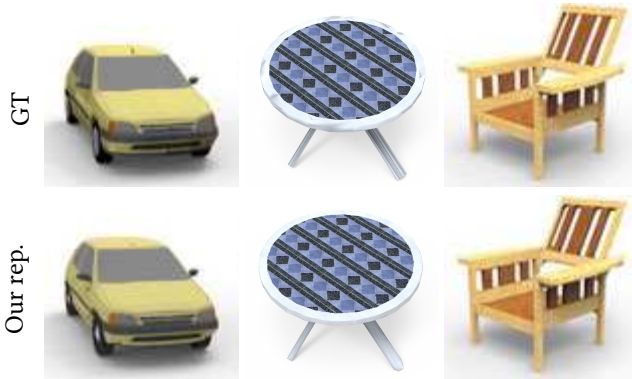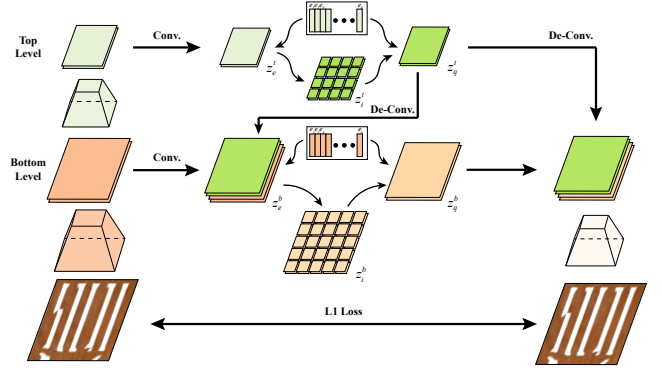

Fig. 6. Examples showing our part-based texture representation (bottom row), which is able to retain most fine-level details and visually reproduces the appearance of rich topological structures (e.g., holes in the chair's back) that cannot be represented using the underlying geometry.

not ensure the compatibility of textures between different parts on the same shape. Hence for each category we choose one part as the seed part (e.g. the table surface for tables). When modeling texture distributions of other parts, we append the VGG feature of the seed part's texture to the geometry condition. The texture generation of the seed part is conditional on its geometry, while texture generation of remaining parts depends on both their own geometry and the texture feature of the seed part. This strategy ensures that generated textures for different parts are coherent.

3.1 Part-based Texture Representation

To represent textures on a mesh, a key step is to build a texture atlas. Traditionally, such an atlas is generated individually for each mesh, which makes learning difficult.

Our method is based on SDM-NET [Gao et al. 2019], which provides a set of consistently segmented parts across each category. The geometry details of each part are represented by the deformations of a template box. In our setting, template boxes are also served as the domains for surface textures with predefined UV mapping. To minimize the mapping distortion from the template box to the 2D texture maps, the template box is unfolded in a straightforward manner as illustrated in Figure 3 (b). Let $I_{i,j}$ be the texture image of the jth part on the ith shape. By the unfolding operation, the template

box box_0 is unfolded and parameterized to the 2D mesh in the UV space to obtain $\widetilde{box_0}$ as shown in Figure 3, with texture image size $4l \times 3l$ where l is the length of the edge of the template box when mapped to the UV space. Throughout the paper, we fix l=256, while the texture resolution for a single part is set at $1,024 \times 768$. Although a larger l would lead to higher-resolution texture images that better represent details, the network training must require more data. Our current setting reflects a quality-cost trade-off.

With the UV mapping between the deformed box $b'_{i,j}$ and the corresponding texture image $I_{i,j}$ as defined above, the next step is to fill in each pixel $\tilde{u}$ on the texture image $I_{i,j}$. As shown in Figure 5, the 2D mesh $\widetilde{box_0}$ gives a partition of the texture image into triangles. Assuming that $\tilde{u}$ belongs to a 2D triangle $\tilde{f}$ on $\widetilde{box_0}$ with barycentric coordinates $bc(\tilde{u})$, we can find the corresponding 3D face f' located on the deformed box $b'_{i,j}$, and the corresponding 3D position p' based on barycentric coordinates $bc(\tilde{u})$ on f'. We then project p' onto the input shape to obtain the ground truth color for pixel $\tilde{u}$.

Note that a genus-zero box cannot geometrically model a high-genus part, e.g., the chair back in Figure 5 with holes, unless a topology-varying deformation is performed. We avoid the need for such a deformation by utilizing the *alpha channel* of the texture image to indicate *transparency*, so as to allow the appearance of a high-genus part to be produced through ray tracing. Specifically, if the projection of p' is not on the surface of the input shape, then the alpha channel for the pixel $\tilde{u}$ is set to 0, indicating total transparency, otherwise it is set to 1 for opaque. When rendering a shape whose parts are all of genus 0 using ray tracing, a ray that hits a point whose alpha value is 0 would continue to travel; see Figures 6 and 9 for some examples of such a rendering of decoded meshes by TM-NET. Without such transparency flags, which are encoded into the texture images, previous works such as SDM-NET [Gao et al. 2019] cannot produce the visual appearance of topological details.

3.2 Part-level Texture and Geometry Encoding

We first present how the geometry and texture are encoded for a given object part. For the geometry, we use the mesh-based variational autoencoder (PartVAE) in SDM-NET [Gao et al. 2019] to encode the geometric details. Please refer to [Gao et al. 2019] for its network structure. As for the texture, we introduce a TextureVAE for encoding texture images. Different networks are needed to address their fundamentally different characteristics.

Our TextureVAE is inspired by VQ (Vector Quantization)-VAE-2 [Razavi et al. 2019a,b] and its overall architecture is illustrated in Figure 7 and detailed network architecture is shown in the supplementary material. Here, VQ-VAE-2 is employed for the following reasons. It is difficult for the convolutional VAE [Kingma and Welling 2014] with a continuous representation to encode/decode high-resolution texture images (e.g. 256×256) well due to their high-frequency characteristics, which would lead to blurry results. In contrast, VQ-based VAE could represent them well. This is because there are often repetitive patterns on texture images, which could be effectively represented by a single codebook vector using vector quantization. In this way, modeling the whole image is converted into modeling smaller patterns, substantially reducing the learning difficulty. The input of TextureVAE is a four-channel texture patch in size 256×256, including three channels of RGB values and one channel of alpha values. Texture images in the UV space contain large areas of unused pixels (see Figure 5 for an example). To make texture encoding more efficient, we perform encoding based on 256×256 patches each corresponding to a face of the cube.

The encoder in VQ-VAE-2 [Razavi et al. 2019a] that is capable of encoding 256×256 images is a two-level image pyramid which extracts two feature maps. We refer to them as the top- and bottom-level feature maps z_e^t and z_e^b, respectively. Each feature map z_e^*, where $* = b$ or t, goes through a vector quantization which maps each feature vector to its closest vector in the dictionary e^* to obtain the discretized feature map z_q^*. The dictionary is updated during the training process as in [Razavi et al. 2019a], to capture the data distribution. The decoder takes the discrete feature maps as input to produce the reconstructed image. For more information about vector quantization and discrete vectors for quantization, please refer to the appendix of [Razavi et al. 2019a]. This architecture captures multiscale texture characteristics and allows high-frequency details often present in textures to be faithfully represented using a low-dimensional latent vector, thanks to the dictionary-based vector quantization and low-dimensional embedding.

With PartVAE encoding the geometry and TextureVAE encoding the texture, our method can represent meshes with fine geometry and fine textures. Moreover, our texture contains transparency information so the texture is also leveraged for its geometric representation ability, enabling our method to represent a single part with possibly many holes in it, as shown in Figure 6.

Since the geometry and texture distributions lie in their own manifold spaces, PartVAE and TextureVAE are trained separately. Let $Enc_P(.)$ and $Dec_P(.)$ denote the encoder and decoder of part deformations for representing geometric details, $Enc_T(.)$ and $Dec_T(.)$ denote the encoder and decoder for the associated texture images, x_p and x_t be the respective input deformation representation and

texture images, and $z_p = Enc_P(x_p)$ and $z_t = Enc_T(x_t)$ be the encoded latent vectors of geometry and texture, respectively, where $z_t = (z_q^t, z_q^b)$ is the concatenation of discrete feature maps from the VAEs of the two layers. And finally, let $x_p' = Dec_P(z_p)$ and $x_t' = Dec_T(z_t)$ be the reconstructed deformation feature vector and texture image, respectively.

The loss function of PartVAE follows [Gao et al. 2019], so that the TextureVAE would minimize the following loss:

$$L_{TextureVAE} = L_{recon} + L_{seam} + \alpha_1 L_{embedding}, \qquad (1)$$

where $L_{recon} = \|x_t' - x_t\|_1$ is the reconstruction loss to ensure faithful reconstruction. The embedding loss, $L_{embedding}$, is defined as:

$$L_{embedding} = \sum_{* \in \{t,b\}} \|sg[z_e^*] - e^*\|_2^2 + \beta_1 \|sg[e^*] - z_e^*\|_2^2, \quad (2)$$

where sg stands for stop gradient operator which does not influence the forward pass but has zero partial derivatives in the backward pass. The first term aims at updating the embedding space, i.e., making the dictionary e^* closer to the feature maps z_e^* extracted by the encoder. The second term updates the encoder's parameters to make the feature map move towards the embedding space or the dictionary e^*. Please refer to [Razavi et al. 2019a,b] for more detail. And L_{seam} penalizes the differences between texture boundary pixels that are in the same template box edge before unfolding to promote synthesizing seamless textures. For more detail about L_{seam}, please refer to the supplementary file. α_1 and β_1 control the relative importance of each loss term.

3.3 Geometry Guided Texture Generation

Mesh geometry and texture could be encoded by PartVAE and TextureVAE respectively. In many cases, there are clear correlations between geometry and texture, e.g., beams always appear in the front of the car and they are usually white, while tires are black. In other cases however, the same geometry may be textured in different, and equally plausible ways, e.g., color of the car body.

To enable the texture generation network to output different textures for an input shape, we develop an autoregressive model to predict the conditional probability distribution of each pixel in the texture image, and sample different textures from the predicted distribution. Our model design is inspired by PixelSNAIL [Chen et al. 2018], which has shown success for discrete feature map generation by integrating temporal convolution with an attention mechanism.

PixelSNAIL models the distribution of every single pixel in the discrete representations z_i^*. We adapt the conditional PixelSNAIL with three fully connected layers for feeding the geometry latent vector, as shown in Figure 8. Two such different conditional autoregressive models learn the high-dimensional distributions of the two discrete index matrices z_i^t and z_i^b. It should be clarified that we learn the data distribution of the entire texture image instead of that of individual texture patches. That is, we concatenate the discrete index matrices of different patches in a texture image together and feed it into the autoregressive model.

The top level autoregressive model takes the top level index matrix z_i^t as input and the geometry latent vector z_p (for the seed part) or concatenation of geometry latent vector z_p and the seed part
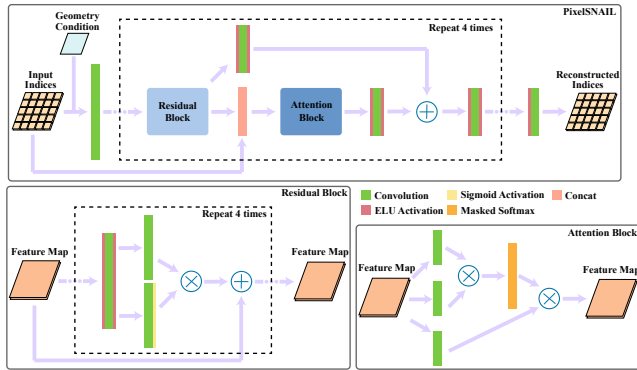
Fig. 8. The architecture of our autoregressive generative model. The network takes an index matrix as input and geometry latent vector as condition and outputs reconstructed index matrix.

texture's VGG feature [Simonyan and Zisserman 2014] (for other parts) as condition, while the bottom level autoregressive model takes the bottom level discrete index matrix z_i^b as input and the top level discrete index matrix z_i^t as condition.

To aggregate geometry feature and the seed part texture's VGG feature, three fully connected layers are used to fuse them. For simplicity, let x and c denote the input and the condition of the conditional autoregressive model f. In the training process, the conditional autoregressive model minimizes the following loss:

$$L_{Conditional} = L_{CrossEntropy}(x, f(x, c)). \tag{3}$$

Note that we use the SP-VAE from SDM-NET [Gao et al. 2019] to jointly encode the geometry and structure of different parts to represent the whole 3D shape.

3.4 Applications

We now present how our method can be applied to various applications with textured meshes.

Shape-Conditioned Mesh Texturing. Our network can be used to generate texture for a given shape automatically. Given a new shape, we first convert it into our representation. Then we can obtain the geometry latent vector of each part in the latent space using PartVAE. The autoregressive generative model PixelSNAIL for the seed part models the distribution of the two-level index matrices. Then we sample on the distribution for several times to generate different index matrices z_i^t and z_i^b, which are later used as indexes to obtain corresponding code vectors in the codebook to form discrete feature maps z_q^t and z_q^b. Finally, the decoder of TextureVAE decodes z_q^t and z_q^b to generate textures. Once the texture of the seed part is generated, we extract its VGG feature and concatenate it with the geometry of other parts as the condition for their autoregressive models, which will lead to coherent texture for different parts.

Generation and Interpolation. Given our deep generative model TM-NET, we can randomly generate textured mesh models and interpolate mesh models in the latent space. For textured mesh generation, we first generate the structured meshes with fine geometry by randomly sampling from the SP-VAE latent space. For each part

we encode the geometry into the latent space of PartVAE. Then the problem is converted to conditional texturing the generated shape as mentioned above. For interpolation, the geometry and structure of two input shapes are interpolated in the latent space of SP-VAE, and the textures are interpolated in the continuous feature maps z_e^* of TextureVAE for generating in-between textured meshes.

Image-Guided Mesh Generation. Our method can be adapted for textured shape generation as guided by a single-view image, where the untextured shape geometry is produced by existing methods with a pre-trained geometry prediction network. Specifically, we first predict the 3D shape and camera viewpoint from the input image using DISN [Xu et al. 2019]. The predicted shape is then segmented by PointNet++ [Qi et al. 2017b] pretrained on our dataset and encoded using our PartVAE. With the segmentation and camera viewpoint, we project all parts onto the image plane to obtain an image mask. Finally, a pretrained PixelSNAIL network takes the VGG features of the masked input image and the part geometry features as condition to produce the part textures.

4 DATASETS AND NETWORK IMPLEMENTATION

In this section, we describe our network architecture and training process in detail. The experiments are performed on a cluster with three deep learning servers, each of which is equipped with an I7 6850K CPU, 64GB RAM , and four Nvidia RTX 2080Ti GPUs. Twelve Nvidia RTX 2080Ti GPUs are used for training TM-NET. All the networks are implemented with PyTorch [Paszke et al. 2019]. Jittor [Hu et al. 2020] implementation is also supported. Codes are available at https://github.com/IGLICT/TM-NET.

4.1 Dataset

The dataset used in our experiments consists of segmented meshes with textures from the dataset employed by SDM-Net [Gao et al. 2019], which is a subset of ShapeNet Core V2 [Chang et al. 2015] with part segmentations provided by SDM-Net. See Table 1 for the numbers of textured meshes in different categories in our dataset.

4.2 Network Architecture

Our network is composed of four parts including PartVAE for encoding part geometry, TextureVAE for encoding part texture, conditional PixelSNAIL for generating textures, and SP-VAE for jointly encoding both global structure and part geometries. The architectures of PartVAE and SP-VAE follow [Gao et al. 2019].

As shown in Figure 7, the encoder of TextureVAE has three convolutional blocks, each followed by one convolution layer to obtain two different levels of continuous feature representations, which then go through vector quantization. The decoder concatenates the discrete latent variables from different levels and outputs the reconstructed texture images through deconvolution blocks.

Table 1. Numbers of segmented, textured meshes in our experiments.

Object category	Car	Chair	Plane	Table
# Segmented, textured meshes	1,824	3,746	2,690	5,266
# Texture patches	73,482	162,984	164,766	167,628

The detailed architecture of encoding and decoding networks can be found in the supplementary material. For the hyperparameters in the loss function of TextureVAE, they are the same as those in [Razavi et al. 2019a]. Further, the conditional texture generation network architecture is the same as the original implementation, except that before putting conditional input into PixelSNAIL, we add three fully connected layers to convert the conditional input into the same size as the input discrete index matrix.

The network architecture of SP-VAE follows the SP-VAE in [Gao et al. 2019]. Leaky ReLU is set as the activation function. We first train PartVAE and SP-VAE. After that, we train TextureVAE. The conditional autoregressive network is trained at last.

4.3 Metrics for Evaluation

To evaluate and compare quality of the generated shapes and textures, we employ the Learned Perceptual Image Patch Similarity (LPIPS) [Zhang et al. 2018] and Structure Similarity Image Metric (SSIM) [Zhou et al. 2004], which are image similarity measures between input textured shapes and decoded textured meshes. Since the texture and geometry are interconnected, we can evaluate their quality together by rendering the textured shapes. Specifically, we render the decoded textured meshes and the input textured meshes in 12 viewpoints and use SSIM to measure the difference of rendered images from the same viewpoint and under the same lighting.

Also, to evaluate the realism of generated textures for a given shape, we use fooling rate which is a human judgment approach proposed in [Zhang et al. 2016]. This test shows a real image and a synthesized image to a human participant for one second each and asks them to tell which is "real". The fooling rate is the possibility of synthesized images being mistaken for real images.

4.4 Parameters

Hyper-parameters are fixed for datasets of different object categories. We demonstrate how the results change with the hyper-parameters. Each dataset is randomly divided into training and test sets, with a 9:1 split, using the data released from SDM-NET. We use Structural Similarity Index (SSIM) (see more in Sec. 5) to measure image perceptual similarity between rendered images of input textured meshes and those of decoded textured meshes. α_1 and β_1 are set to 1 and 0.25.

We performed ablation studies on these parameters. For the hyper-parameter β_1, we fixed α_1 to 1 and performed an experiment on how its value influences the SSIM score between the input texture image and the output. As shown in Table 2, TextureVAE reaches its best performance when β_1 is set to 0.25. Then we fixed β_1 to 0.25 and performed an ablation study on α_1. The results in Table 3 show that $\alpha_1 = 1$ gives the best reconstruction quality.

Table 2. Comparison of average texture reconstruction quality (measured by SSIM) between the input texture images and decoded textured images in the test dataset of table w.r.t changing hyper-parameter β_1.

β_1	0.1	0.25	0.5	1.0	2.0
SSIM on textures	0.886	**0.915**	0.741	0.739	0.729

Table 3. Comparison of average texture reconstruction quality (measured by SSIM) between the input texture images and decoded textured images in the test dataset of table w.r.t changing hyper-parameter α_1.

α_1	0.1	0.25	0.5	1.0	2.0
SSIM on textures	0.837	0.856	0.882	**0.915**	0.874

ALGORITHM 1: Training steps

1. Train PartVAE for each part;
2. Train SP-VAE for each shape;
3. Train TextureVAE for all parts;
4. Load checkpoint and extract PartVAE and TextureVAE latents z_p, z_i^t and z_i^b for the seed part;
5. Train conditional PixelSNAIL for the seed part in the top and bottom level discrete feature spaces;
6. Load checkpoint and extract PartVAE and TextureVAE latents z_p, z_i^t, z_i^b, and VGG feature of the corresponding seed part's texture for other parts;
7. Train conditional PixelSNAIL for other parts in the top and bottom level discrete feature spaces.

4.5 Training Details

Our network is trained stage by stage. On the geometry side, similar to SDM-NET, every semantic part has a corresponding PartVAE. All of these PartVAEs are trained separately. After training the PartVAEs, we then concatenate the latent vector of PartVAE and the structure code as the input to train SP-VAE. The PartVAE is trained with 20,000 iterations and SP-VAE with 120,000 iterations with batch size 128. On the texture side, a single TextureVAE is trained for textures of all object parts. The TextureVAE is trained with 400,000 iterations with batch size set as 120.

After the training process of SP-VAE and TextureVAE are finished, we train the top and bottom level PixelSNAILs. The top level PixelSNAIL is trained with 200,000 iterations and bottom level PixelSNAIL is trained with 300,000 iterations with batch size 6. For all the networks, every training batch is randomly selected from the training dataset. The detailed training process is summarized in Algorithm 1. In all the neural networks, we use the ADAM optimizer with an initial learning rate 0.0003, which decays every 100 steps with a decay rate of 0.999. Take the car dataset with seven parts as an example, the whole training process takes about 5 days. Once the networks are trained, generating a textured shape takes about 10 minutes.

5 EXPERIMENTAL RESULTS AND EVALUATIONS

In this section, we present results of textured mesh autoencoding, novel generation, shape-conditioned and image-guided texturing, as well as latent-space interpolation. We also perform ablation studies to show the benefits of each component in our design.

5.1 Reconstruction and Generation of Textured Meshes

Textured Mesh Autoencoding. First, we qualitatively and quantitatively compare TM-NET with state-of-the-art geometry deep learning methods for the *shape* autoencoding task. To this end, only

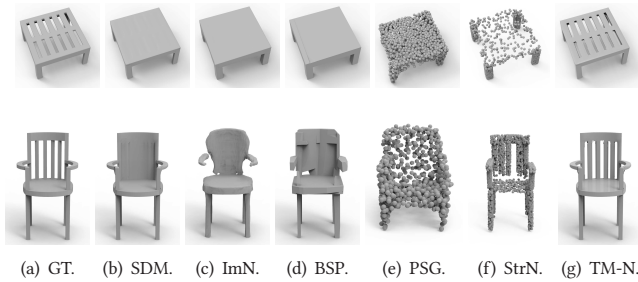(a) GT. (b) SDM. (c) ImN. (d) BSP. (e) PSG. (f) StrN. (g) TM-N.

Fig. 9. Comparing shape autoencoding between TM-NET (TM-N) and several state-of-the-art generative models: SDM=SDM-NET, ImN=IM-NET, BSP = BSP-NET, StrN = StructureNET, with GT = ground truth. Outside of TM-NET, the part-based SDM-NET tends to obtain the best overall shape and structure, but is unable to reproduce topological details such as holes on the chair back and tabletop. With texturing and transparency, TM-NET is the only method that can produce the appearance of such details.

the alpha channel is used, while color textures are ignored. In Figure 9, we show two representative autoencoding results on the test set, where the input shapes possess fine-grained topological details. A comparison with other recent deep models shows that TM-NET and SDM-NET tend to produce the cleanest shape *structures* due their their structure-aware autoencoding. With the utilization of texture transparencies, TM-NET is the only network that is able to reproduce the appearance of topological details. Table 4 provides quantitative comparison results, using the multi-view SSIM metric, demonstrating that TM-NET outperforms the other methods over all four major shape categories of ShapeNet.

For autoencoding *textured* meshes, we are not aware of existing alternatives from the literature. Hence, we elect to show some qualitative results in Figure 10, which provides the ground truth meshes with texture, the input textured meshes with our representation, and the decoded textured meshes by TM-NET. The example results shown were *randomly* selected from each shape category to provide a general assessment of how well the TM-NET autoencoding is able to preserve the geometry and texture details and structures. More random results can be found in the supplementary material.

Textured Mesh Generation. As a generative model, and compared to SDM-Net [Gao et al. 2019], TM-NET can not only output plausible shape structures and geometries but also generate a suitable texture for each part. On the geometry side, our SP-VAE encodes the structural information of different parts together with their geometry. On the texture side, the TextureVAE encodes texture information to ensure texture quality, while the conditional autoregressive model learns the relationship between geometry and texture to ensure compatibility between them. To obtain a textured mesh from scratch, we first sample on the latent space of SP-VAE to generate a structured deformable mesh without texture. Then similar to the shape-conditioned texturing procedure, we feed the geometry latent vectors into the conditional autoregressive network to predict TextureVAE index matrices. Finally, the decoder of TextureVAE takes these predicted index matrices as input to decode plausible textures

Table 4. Quantitative comparisons, using the SSIM metric, between TM-NET and other recent deep representations for 3D geometry autoencoding, without color textures. TM-NET outperforms all these other methods over all four shape categories. Note that StructureNET uses PartNet [Mo et al. 2019b] as input which does not contain the car or plane category.

Categories	SSIM metric ↑			
	Car	Chair	Plane	Table
SDM-NET	0.901	0.915	0.919	0.925
IM-NET	0.893	0.907	0.874	0.896
BSP-NET	0.899	0.908	0.869	0.919
PSG	0.724	0.619	0.713	0.693
StructureNet	-	0.803	-	0.798
Ours	**0.914**	**0.923**	**0.920**	**0.930**

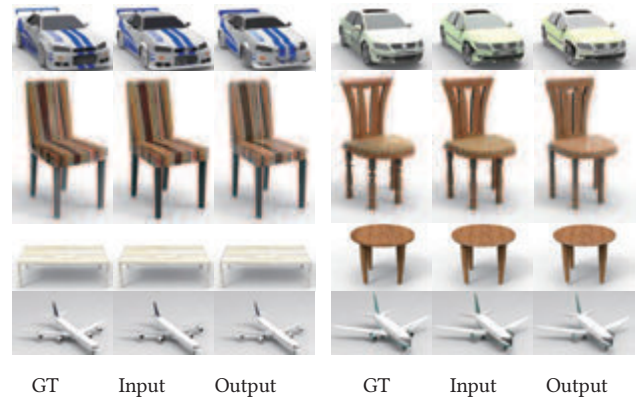GT   Input   Output   GT   Input   Output

Fig. 10. Randomly sampled results of textured mesh autoencoding by TM-Net, on meshes from the test set. These results are better viewed by zooming in. In each triplet, the left shows the ground-truth textured shape, the middle is our input representation relaying both geometry and texture features, which are generally well preserved by the decoded mesh (right), demonstrating that TM-NET learns an effective embedding.

for the generated shape. Figure 11 shows a sampler of output textured meshes using the above procedure for each shape cateogry. Note that these results were *randomly* selected, except for the first chair in the first row, which was picked to show an example of using texture transparency to "fake" topological details.

We qualitatively evaluate the novelty of the generated meshes by showing the closest neighbor shapes from the training set, both in terms of geometry and appearance. These results demonstrate that TM-NET is able to generate diverse and novel textured meshes with quality. However, the generated car and plane textures tend to be blurrier than those from the other categories. This is because these textures typically contain a mixture of definitive sharp features (e.g., plane windows, car headlights, etc.) and smooth or constant textures (e.g., the color of the remaining parts). Such mixtures are more difficult to encode using the VQ- or compression-based TextureVAE in our framework, as compared to the chair/table textures that have

Fig. 11. Randomly sampled results of textured mesh generation by TM-NET (best viewed by zooming in), except for the first chair, which was picked to show texture transparency. We show four results per shape category, with more in the supplementary material. In each triplet, the left one is the generated textured mesh; the middle shows its closest neighbor from the training set, in terms of shape geometry as measured using Chamfer Distance (CD) on 1,024 uniformly sampled points on each mesh surface; the right shows the closest neighbor from the training set, in terms of appearance as measured by SSIM on multi-view images. For each mesh, we normalize it to a unit cube and render it from six fixed viewpoints.

a relatively simpler frequency-domain characterization. More mesh generation results can be found in the supplementary material.

As for a comparison to existing methods, we are not aware of other networks designed for textured *3D* shape generation. The work by Zhu et al. [2018] called Visual Object Networks (VON) could be compared in terms of generating different and novel *views* of object textures. VON does not generate a textured 3D shape, but only 2D images of an object after texture generation from a sampled texture code. As such, it is difficult to maintain texture consistency between multiple views, as shown in Figure 12, in contrast to results from textured mesh generation by TM-NET.

Table 5 shows a quantitative comparison in Fréchet Inception Distance (FID) [Heusel et al. 2017] between VON and TM-NET, where we evaluate the rendered images of the generated textured shapes against those from the training set.

Shape-Conditioned Mesh Texturing. Given an untextured 3D shape, TM-NET can predict plausible and diverse textures for the shape

Fig. 12. A comparison to VON [Zhu et al. 2018] for novel view synthesis of object textures (top row). The two VON results were randomly selected, then for each, we randomly selected a TM-NET result with similar overall color as the VON result. We observe texture inconsistency from different views since each viewpoint was generated separately with no global regularization. In contrast, TM-NET generates an entire textured 3D shape which, when rendered in multiple views, ensures texture consistency.

Table 5. Quantitative comparison between VON and TM-NET in terms of their generative capabilities, measured in **FID** ↓ [Heusel et al. 2017]. Note that VON did not contain the plane or table categories.

Method	Car	Chair	Plane	Table
VON	83.3	51.8	-	-
TM-NET	**56.9**	**46.4**	34.1	42.3

that are conditioned on the shape's geometry. Figure 13 shows such shape texturing results, where the same input shape can lead to different textures, by varying the sampling. The part-by-part texture generation is clearly revealed as different parts of the same shape can also be textured differently (e.g., chair seat vs. back and tabletop vs. leg). One could argue that some part textures generated by TM-NET are not quite compatible, e.g., see the second chair result in the first row. In regards to this, we shall point out that even more drastic texture "incompatibilities" can be observed from the training set, as shown in Figure 14. In the supplementary material, we show more generated samples for shape texturing, as well as the closest training textures to the generated samples.

We compare our method to Texture Fields (TF) [Oechsle et al. 2019], a state-of-the-art method for texturing an input 3D shape with a texture image sampled from a latent space. The comparisons were done on the same input shapes, from the same viewpoints, and using the same rendering parameters, but the output textures are generally different since both were sampled. We show some visual results in Figure 15, while for a quantitative study, we compare the LPIPS values between different results generated by the same method to demonstrate the diversity, and report fooling rate through a user study to evaluate realism. As demonstrated by results in Table 6, texturing results by TM-NET were deemed to be both more realistic and more diverse than those from Texture Fields.

Image-Guided Mesh Generation. As described in Section 3.4, TM-NET can be adapted for image-guided generation of textured meshes. In Figure 16, we show several randomly sampled results and compare to DVR [Niemeyer et al. 2020]. Note that due to the sampling strategy of PixelSNAIL, our method can generate multiple textured shapes

Fig. 13. Randomly selected shape-conditioned texturing results by four samplings of the learned probability distribution conditioned only on input geometry (left). The seed parts are chair seats and tabletops, respectively. Note the plausible and diverse textures generated by TM-NET, as well as different textures generated for different parts in the same shape.

Fig. 14. Several chair meshes from the training set whose part textures may be perceived as "incompatible", yet "stylish".
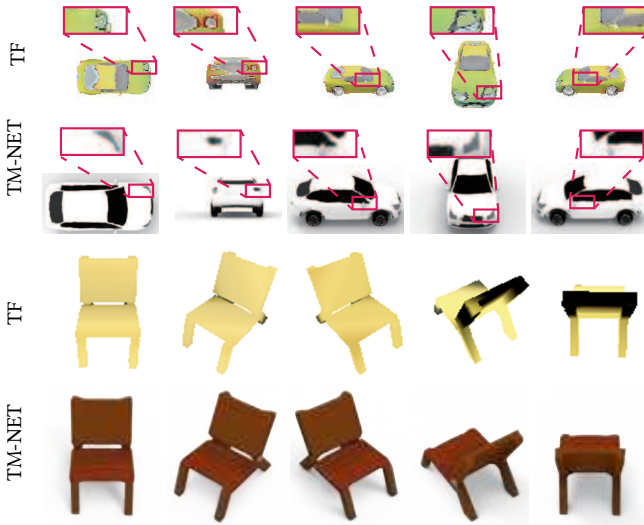

Fig. 15. Comparison of our mesh texturing with Texture Fields (TF) [Oechsle et al. 2019], on several randomly picked examples.

from which it would be quite easy to find one that is the closest in appearance to the input image. In general, this result is more faithful to the input than the DVR outputs.

Table 6. Comparison to Texture Fields [Oechsle et al. 2019] in terms of diversity (measured by LPIPS) and realism (measured by fooling rate). As a diversity measure, larger values of LPIPS is better, i.e., more diverse.

	Realism	Diversity
Method	Fooling rate ↑	LPIPS ↑
Texture Fields	23.8%	0.148
Ours	**41.5%**	**0.263**

(a) Image guidance. (b) DVR result. (c) TM-NET result 1. (d) TM-NET result 2.

Fig. 16. Image-guided generation of textured shapes and a comparison to DVR [Niemeyer et al. 2020], on a few randomly sampled results. Our network can generate multiple textured shapes, containing results that are generally closer to and more faithful to the guidance image (c-d).

Textured Shape Interpolation. In our method, shape geometry and texture are separately encoded into the latent spaces of SP-VAE and TextureVAE, respectively. Hence, we can perform linear interpolation in both latent spaces and then recover the textured meshes by the decoders of SP-VAE and TextureVAE. In Figure 17, we show a sampler of such interpolation results on chairs, and compare to VON [Zhu et al. 2018], TF [Oechsle et al. 2019], and simple alpha blending between textures on a per-part basis. Note that the original TF implementation only supported texture interpolation. Since their model outputs rendered images using a depth map, a shape code, and a texture code, we obtained the TF interpolation results by interpolating on all these three components. On the other hand, VON only generates in-between views of textured objects and it is not part-aware. Since it is a GAN-based model, it is difficult to compare VON to other methods using exactly the same source and target shapes. As a remedy, for each set of comparisons, we first run VON interpolation in both shape and texture latent spaces to obtain a *random* pair of source and target shapes. We then run TM-NET, TF, and part-based alpha blending using source and target shapes from our *test set* that are respectively the *closest*, as measured by SSIM, to the source and target shapes generated by VON.

We can see that compared to TF and VON, the interpolation results by TM-NET are more plausible and faithful in both geometry and texture, demonstrating the advantage of our structured approach and the learned latent spaces. Alpha blending also works well, with results quite close to those from TM-NET. We believe that this can be attributed to the part-by-part texture blending, where each part is simply a box, which facilitates part correspondence.

Fig. 17. Randomly sampled results of latent-space linear interpolation between chairs (first and last columns), comparing TM-NET to VON [Zhu et al. 2018], TF [Oechsle et al. 2019], and part-by-part alpha blending.

5.2 Self evaluations

Dividing texture image into patches. As mentioned in Sections 3.2 and 3.3, we divide the input texture image into six patches, each corresponding to a face of the template box. Compared with our divide and conquer approach, a straight-forward strategy is to feed the whole texture image of size 1024×768 into the TextureVAE. If we feed the whole texture image of size 1024×768 into the TextureVAE, we have to adopt a three-level hierarchy TextureVAE and augoregressive model as VQ-VAE-2 [Razavi et al. 2019a] does and use three index matrices of size 32×32, 64×64 and 128×128, to cope with the higher input resolution. Here, we compare shape texturing results using these two strategies in Figure 18. The

Fig. 18. Comparison of mesh texturing results. Top row: dividing a texture image into patches. Bottom row: regarding the same texture image as a whole. Results show that with the patch-based, divide-and-conquer approach, TM-NET can generate textures with higher quality.
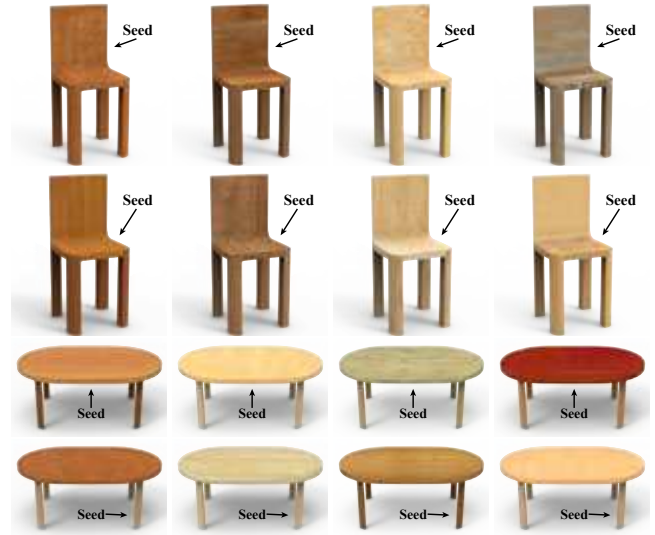

Fig. 19. Mesh texturing results on the same shape, randomly selected, by specifying different seed parts. Row 1: chair back as seed; row 2: chair seat as seed; row 3: tabletop as seed; row 4: left front leg as seed.

result shows that by dividing the whole image into patches, the autoregressive model can generate textures with higher quality.

Seed Part. To ensure the coherence between textures generated for different shape parts, we introduced the use of seed parts, where the generation of textures for the remaining parts is conditioned on the seed part's generated texture features and the remaining parts' geometry features. We evaluate how the choice of seed parts influences the generated textures in Figure 19, which demonstrates experimentally that while the final texturing results do differ as the seed parts vary, the generated textures remain plausible and exhibit compatability across the whole shapes.

Texture Image Resolution. Finally, we test how TM-NET performs, specifically on the textured mesh autoencoding task, as we vary the resolution of the texture images. As shown in Table 7, when the image resolution is low at 64×64, the SSIM value is low which means that the reconstruction error on the test dataset is high. However, the reconstruction quality does not always improve as the resolution increases, since when the texture resolution is too high, the generalization capability of the network can be compromised. Experimentally, we find that the best quality is attained by setting

Table 7. SSIM measured on rendered images from autoencoding the chair test set, as we vary the texture resolution. Recall that l is the length of the edge of the template box when mapping to UV space; see Section 3.1.

l	64	128	256	512
SSIM on autoencoding textures	0.837	0.859	**0.928**	0.905

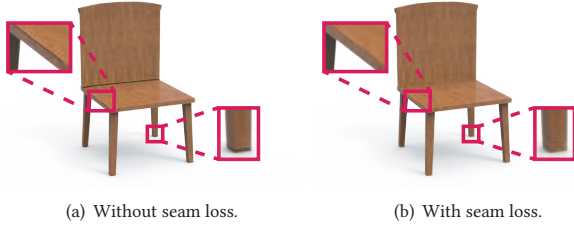(a) Without seam loss.   (b) With seam loss.

Fig. 20. Comparison of decoded results without and with seam loss L_{seam}. Due to unfolded boundaries of the texture image in our texture representation, the generated texture images may show some artifacts (inconsistency in the unfolded boundaries, as highlighted with red rectangles in (a)). The loss can prevent the artifacts (b). The results demonstrate the seam loss is effective for removing artifacts related to unfolded boundaries.

$l = 256$ and the texture resolution for a single part to $1,024 \times 768$; this is the setting we adopt throughout our experiments.

5.3 Ablation Studies

We perform several ablation studies to demonstrate the necessity of key components of our network architecture.

With vs. without seam loss. Since our texture representation relies on unfolding boxes (see Figure 3), pixels on the boundary during unfolding are mapped to two or more places in the texture space. This may lead to artifacts when pixels on the boundaries are not consistent. So we introduced a seam loss to prevent this, which ensures the unfolded boundaries in the texture images are as consistent as possible. Figure 20 shows the decoded results of a chair without and with the seam loss L_{seam}. The results show black line artifacts (on legs and seat and highlighted with red rectangles) exist near the unfolded boundaries without the seam loss, and our result with the seam loss does not have such artifacts.

With vs. without texture part compatibility. As demonstrated in the sections above, the autoregressive model is able to generate reasonable textures for each part. However, generating textures for each part individually can lead to incompatible between different parts. To solve this issue, a greedy strategy is applied to ensure the coherence between different parts. Firstly, one selected part's texture (usually the selected part's center is closest to the shape center) is generated, and then the VGG feature of its texture is extracted and concatenated with other parts' geometry latent code to be the conditional input of other parts' conditional autoregressive network. The comparison of mesh texturing results without and with seed part's VGG feature conditional input is shown Figure 21. We can find that the generated textures are incompatible among parts without the seed part's VGG feature condition input.

(a) Untextured mesh.   (b) No part compatibility.   (c) With part compatibility.
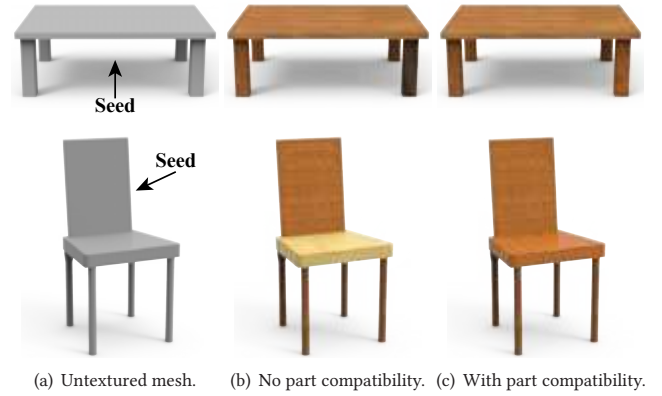
Fig. 21. Comparison of mesh texturing without vs. with part compatibility. Row 1: without using VGG features of the tabletop (the seed part) as conditional input, the texture generated for one of the table legs is inconsistent with the rest. Row 2: with chair back as the seed, the chair seat has an incoherent texture generated if no part compatibility is enforced.

6 CONCLUSION, LIMITATION, AND FUTURE WORK

We present a part-aware deep generative model, TM-NET, for textured meshes embedded in 3D. Our network utilizes a simple and effective consistent texture atlas obtained from unfolding deformable boxes. It separately encodes box deformation and textures using VAEs at the part level and learns one-to-many relationships between geometry and texture. Textures are encoded using dictionary-based vector quantization to allow high-frequency details to be faithfully synthesized. Our generative model is quite generic and supports several typical applications in the realm of realistic 3D shape modeling, e.g., synthesizing textures for a given 3D shape, generation of novel textured meshes, latent-space interpolation, and image-guided generation of textured shapes, all under a unified framework.

As a first attempt at the challenging problem of neural and generative modeling of both the geometry and texture of 3D shapes, our work is still quite preliminary and limited on several fronts. To start, we have only explored the dependency of part texture on shape geometry and structure, without taking into account the correlation between texture and *material* properties, which are often tied to *object functionality*. For example, to be sat on comfortably, sofa textures are often those of fabrics and leather. A joint learning of texture, material, and functionality is left for future work.

In general, textures are highly complex signals that can exhibit a wide variety of frequency-domain characteristics. For example, the wood or leather grain textures on furniture objects are quite different from image patterns on an airplane. As shown in Figure 11, our texture encoding using VQ-VAEs is not as effective in learning the car and plane textures due to their more complex and varied frequency-domain characteristics, compared to the furniture textures. The resulting artifacts include blurrier decoded textures and cross-fading during latent-space interpolation (see Figure 22). In addtition, the VQ-VAE in our TextureVAE does not impose constraints on the intermediate feature space as the original VAE or normalizing

Fig. 22. Cross-fading results from latent space interpolation described in Sec. 5.1 by TM-NET on the car and plane examples (best viewed by zooming in). Row 2 is further enlarged in row 3 for clearer detail visualization.

(a) Ground truth.　(b) Reconstruction.　(c) Image guidance.　(d) Generation.

Fig. 23. Some failure cases. (a-b): TM-NET cannot accurately reconstruct the small slits on top of the ground truth table. (c-d): Image-guided generation of textured shapes by TM-NET may produce unexpected texture.

flows [Berthelot et al. 2018], thus it cannot ensure that all the entries in the codebook are mapped to reasonable textures images.

In addition, while TM-NET can represent 3D shapes with high genus and rely on the texturing trick to create the visual illusion of topological details, using the alpha channel in texture space, it is not always able to decode the fine-grained shape details. As shown in Figure 23(a-b), the tabletop contains many small slits. Our current method cannot reconstruct such geometric details accurately, e.g., some of the gaps are closed in the decoded shape. The details around the slits appear over small regions on the texture image, which are difficult to encode and decode precisely. Still, we believe that the generated textures can provide sufficient constraints to help refine the geometry in a postprocess. To train a network to accomplish this in an end-to-end manner left for future work.

Another possible failure case occurs for our image-guided generation of textured 3D shapes, as shown in Figure 23(c-d). The shape generation uses the guidance image as a condition. However, due to the sampling strategy employed by the conditional generative network, the input texture is not guaranteed to be reproduced or even closely approximated. It is worth considering in future work how to reconstruct both shapes and textures more deterministically by making better use of the input image via neural rendering.

As for other possible directions to explore, we would like to extend our method to other applications, e.g. single-view 3D reconstruction. Expansions to other types of input, such as line drawings for both shape and texture, and to other encoding schemes are also interesting to explore. We also plan to extend our current representation with procedural texture generation, e.g., based on Perlin noise style texture [Henzler et al. 2019] to improve texture generation. Finally, our current shape-conditioned texturing is unable to generate stylish outputs such as the ones shown in Figure 14 which exhibit significant deviations between textures in different parts. Learning *texturing styles* would be an interesting direction for future work.

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2018. Learning Representations and Generative Models for 3D Point Clouds. In *International Conference on Machine Learning (ICML)*, Vol. 80. 40–49.

Eman Ahmed, Alexandre Saint, Abd El Rahman Shabayek, Kseniya Cherenkova, Rig Das, Gleb Gusev, Djamila Aouada, and Björn E. Ottersten. 2018. Deep Learning Advances on Different 3D Data Representations: A Survey. *CoRR* abs/1808.01462 (2018). arXiv:1808.01462

Sema Berkiten, Maciej Halber, Justin Solomon, Chongyang Ma, Hao Li, and Szymon Rusinkiewicz. 2017. Learning Detail Transfer based on Geometric Features. , 361–373 pages.

David Berthelot, Colin Raffel, Aurko Roy, and Ian Goodfellow. 2018. Understanding and improving interpolation in autoencoders via an adversarial regularizer. *arXiv preprint arXiv:1807.07543* (2018).

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint arXiv:1512.03012* (2015).

Kang Chen, Kun Xu, Yizhou Yu, Tian-Yi Wang, and Shi-Min Hu. 2015. Magic decorator: automatic material suggestion for indoor digital scenes. *ACM Trans. Graph.* 34, 6 (2015), 1–11.

Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. 2019. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer. In *NeurIPS*. 9605–9616.

Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. 2018. Pixelsnail: An improved autoregressive generative model. In *International Conference on Machine Learning*. 864–872.

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. BSP-NET: Generating Compact Meshes via Binary Space Partitioning. In *CVPR*. 42–51.

Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *CVPR*. 5939–5948.

Angela Dai, Yawar Siddiqui, Justus Thies, Julien Valentin, and Matthias Nießner. 2021. Spsg: Self-supervised photometric scene generation from rgb-d scans. In *CVPR*. 1747–1756.

Alexey Dosovitskiy and Thomas Brox. 2016. Generating Images with Perceptual Similarity Metrics based on Deep Networks. *CoRR* abs/1602.02644 (2016).

Alexei A. Efros and Thomas K. Leung. 1999. Texture Synthesis by Non-parametric Sampling. In *ICCV*. 1033–1038.

Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao(Richard) Zhang. 2019. SDM-NET: Deep Generative Network for Structured Deformable Mesh. *ACM Trans. Graph.* 38, 6 (2019), 243:1–243:15.

Leon Gatys, Alexander S Ecker, and Matthias Bethge. 2015. Texture synthesis using convolutional neural networks. In *NeurIPS*. 262–270.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NeurIPS*. 2672–2680.

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. 2018. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *CVPR*. 216–224.

Xiaoguang Han, Chang Gao, and Yizhou Yu. 2017. DeepSketch2Face: a deep learning based sketching system for 3D face and caricature modeling. *ACM Trans. Graph.* 36, 4 (2017), 126:1–126:12.

Philipp Henzler, Valentin Deschaintre, Niloy J. Mitra, and Tobias Ritschel. 2021. Generative Modelling of BRDF Textures from Flash Images. *CoRR* abs/2102.11861 (2021). arXiv:2102.11861

Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. 2019. Learning a Neural 3D Texture Space from 2D Exemplars. *arXiv:1912.04158* (2019).

Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. 2020. Deep geometric texture synthesis. *arXiv preprint arXiv:2007.00074* (2020).

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *NeurIPS*. 6626–6637.

Shi-Min Hu, Dun Liang, Guo-Ye Yang, Guo-Wei Yang, and Wen-Yang Zhou. 2020. Jittor: a novel deep learning framework with meta-operators and unified graph execution.

Science China Information Sciences 63, 222103 (2020), 1–21.

Tao Hu, Geng Lin, Zhizhong Han, and Matthias Zwicker. 2019. Learning to Generate Dense Point Clouds with Textures on Multiple Categories. *arXiv:1912.10545* (2019).

Jingwei Huang, Justus Thies, Angela Dai, Abhijit Kundu, Chiyu Jiang, Leonidas J Guibas, Matthias Nießner, Thomas Funkhouser, et al. 2020. Adversarial texture optimization from rgb-d scans. In *CVPR*. 1559–1568.

Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. 2018. Learning Category-Specific Mesh Reconstruction from Image Collections. In *ECCV*. 371–386.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv:1312.6114* (2013).

Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

Johannes Kopf, Daniel Cohen-Or Chi-Wing Fu, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. 2007. Solid texture synthesis from 2D exemplars. *ACM Trans. Graph.* 26, 3 (2007), 2.

Yu-Kun Lai, Shi-Min Hu, Xianfeng Gu, and Ralph R. Martin. 2005. Geometric texture synthesis and transfer via geometry images. In *Proceedings of the Tenth ACM Symposium on Solid and Physical Modeling 2005, Cambridge, Massachusetts, USA, June 13-15, 2005*, Leif Kobbelt and Vadim Shapiro (Eds.). ACM, 15–26.

Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas J. Guibas. 2017. GRASS: Generative Recursive Autoencoders for Shape Structures. *ACM Trans. Graph.* 36, 4 (2017), 52:1–52:14.

Manyi Li and Hao Zhang. 2021. D2IM-Net: Learning Detail Disentangled Implicit Fields from Single Images. In *CVPR*. 10246–10255.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. (2019), 7708–7717.

Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 2017. 3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks. In *Proc. of the International Conference on 3D Vision (3DV)*. 67–77.

Ricardo Martin-Brualla, Rohit Pandey, Sofien Bouaziz, Matthew Brown, and Dan B Goldman. 2020. GeLaTO: Generative Latent Textured Objects. In *ECCV*, Vol. 12351. 242–258.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*. 4460–4470.

Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas Guibas. 2019a. StructureNet: Hierarchical Graph Networks for 3D Shape Generation. *ACM Trans. Graph.* 38, 6 (2019), 242:1–242:29.

Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. 2019b. PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding. In *CVPR*. 909–918.

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision. In *CVPR*. 3501–3512.

Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. 2019. Texture Fields: Learning Texture Representations in Function Space. In *ICCV*. 4531–4540.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *CVPR*. 165–174.

Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven M. Seitz. 2018. PhotoShape: Photorealistic Materials for Large-Scale Shape Collections. *ACM Trans. Graph.* 37, 6, Article 192 (2018).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. 8024–8035.

Dario Pavllo, Graham Spinks, Thomas Hofmann, Marie-Francine Moens, and Aurelien Lucchi. 2020. Convolutional Generation of Textured 3D Meshes. *arXiv preprint arXiv:2006.07660* (2020).

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017a. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*. 77–85.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017b. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NeurIPS*. 5105–5114.

Amit Raj, Cusuh Ham, Connelly Barnes, Vladimir Kim, Jingwan Lu, and James Hays. 2019. Learning to Generate Textures on 3D Meshes. In *CVPR Workshops*. 32–38.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. 2019a. Generating diverse high-fidelity images with vq-vae-2. In *NeurIPS*. 14866–14876.

Ali Razavi, Aäron van den Oord, and Oriol Vinyals. 2019b. Generating Diverse High-Resolution Images with VQ-VAE. In *Deep Generative Models for Highly Structured Data, ICLR 2019 Workshop, New Orleans, Louisiana, United States, May 6, 2019*.

Stephan R. Richter and Stefan Roth. 2018. Matryoshka networks: Predicting 3D geometry via nested shape layers. In *CVPR*. 1936–1944.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. 2019. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *ICCV*. 2304–2314.

Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. 2020. Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In *CVPR*. 84–93.

Shunsuke Saito, Lingyu Wei, Liwen Hu, Koki Nagano, and Hao Li. 2017. Photorealistic facial texture inference using deep neural networks. In *CVPR*, Vol. 3. 5144–5153.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556* (2014).

Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In *NeurIPS*. 1119–1130.

Dmitriy Smirnov, Mikhail Bessmeltsev, and Justin Solomon. 2021. Learning Manifold Patch-Based Representations of Man-Made Shapes. In *International Conference on Learning Representations (ICLR)*.

Xavier Snelgrove. 2017. High-Resolution Multi-Scale Neural Texture Synthesis. In *SIGGRAPH ASIA 2017 Technical Briefs (SA '17)*. ACM, 13:1–13:4.

Shao-Hua Sun, Minyoung Huh, Yuan-Hong Liao, Ning Zhang, and Joseph J Lim. 2018. Multi-view to Novel View: Synthesizing Novel Views with Self-Learned Confidence. In *ECCV*, Vol. 11207. 162–178.

Qingyang Tan, Lin Gao, Yu-Kun Lai, and Shihong Xia. 2018. Variational Autoencoders for Deforming 3D Mesh Models. In *CVPR*. 5841–5850.

Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. 2017. Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency. In *CVPR*. 2626–2634.

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018b. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*. Springer, 52–67.

Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. 2018a. Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes. *ACM Trans. Graph.* 37, 6 (2018), 217:1–217:11.

Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. 2016. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NeurIPS*. 82–90.

Zhijie Wu, Xiang Wang, Di Lin, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. 2019. SAGNet: Structure-aware Generative Network for 3D-Shape Modeling. *ACM Trans. Graph.* 38, 4 (2019), 91:1–91:14.

Yun-Peng Xiao, Yu-Kun Lai, Fang-Lue Zhang, Chunpeng Li, and Lin Gao. 2020. A survey on deep geometry learning: From a representation perspective. *Comput. Vis. Media* 6, 2 (2020), 113–133.

Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 2019. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. *arXiv preprint arXiv:1905.10711* (2019).

Yu-Jie Yuan, Yu-Kun Lai, Tong Wu, Lin Gao, and Ligang Liu. 2021. A Revisit of Shape Editing Techniques: From the Geometric to the Neural Viewpoint. *J. Comput. Sci. Technol.* 36, 3 (2021), 520–554.

Richard Zhang, Phillip Isola, and Alexei A Efros. 2016. Colorful Image Colorization. In *ECCV*, Vol. 9907. 649–666.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*. 586–595.

Kun Zhou, John Synder, Baining Guo, and Heung-Yeung Shum. 2004. Iso-charts: stretch-driven mesh parameterization using spectral analysis. In *Symposium on Geometry processing*. 45–54.

Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. 2018. Non-stationary Texture Synthesis by Adversarial Expansion. *ACM Trans. Graph.* 37, 4 (2018), 49:1–49:13.

Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Joshua B. Tenenbaum, and William T. Freeman. 2018. Visual Object Networks: Image Generation with Disentangled 3D Representations. In *NeurIPS*. 118–129.