Texturify: Generating Textures on 3D Shape Surfaces

Yawar Siddiqui[1], Justus Thies[2], Fangchang Ma[3], Qi Shan[3], Matthias Nießner[1], and Angela Dai[1]

[1] Technical University of Munich
[2] Max Planck Institute for Intelligent Systems
[3] Apple

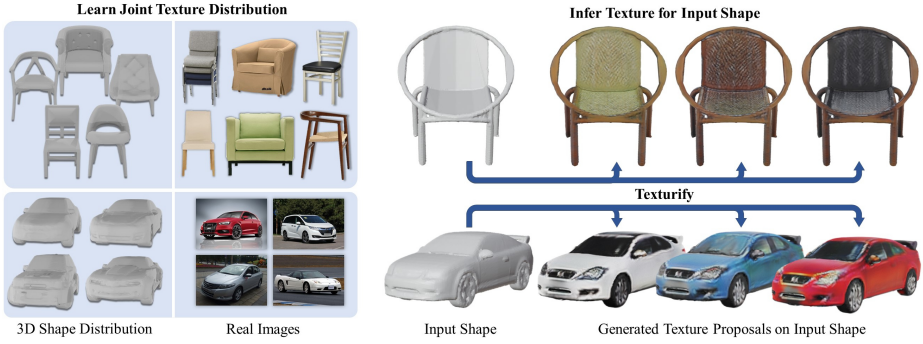


Fig. 1. *Texturify* learns to generate geometry-aware textures for untextured collections of 3D objects. Our method produces textures that when rendered to various 2D image views, match the distribution of real image observations. *Texturify* enables training from only a collection of images and a collection of untextured shapes, which are both often available, without requiring any explicit 3D color supervision or shape-image correspondence. Textures are created directly on the surface of a given 3D shape, enabling generation of high-quality, compelling textured 3D shapes.

Abstract. Texture cues on 3D objects are key to compelling visual representations, with the possibility to create high visual fidelity with inherent spatial consistency across different views. Since the availability of textured 3D shapes remains very limited, learning a 3D-supervised data-driven method that predicts a texture based on the 3D input is very challenging. We thus propose *Texturify*, a GAN-based method that leverages a 3D shape dataset of an object class and learns to reproduce the distribution of appearances observed in real images by generating high-quality textures. In particular, our method does not require any 3D color supervision or correspondence between shape geometry and images to learn the texturing of 3D objects. *Texturify* operates directly on the surface of the 3D objects by introducing face convolutional operators on a hierarchical 4-RoSy parameterization to generate plausible object-specific textures. Employing differentiable rendering and adversarial losses that critique individual views and consistency across views, we effectively learn the high-quality surface texturing distribution from real-world images. Experiments on car and chair shape collections show that our approach outperforms state of the art by an average of 22% in *FID* score.

1 Introduction

3D content is central to many application areas, including content creation for visual consumption in films, games, and mixed reality scenarios. Recent years have seen remarkable progress in modeling 3D geometry [7,10,31,40,43,46,1], achieving significant improvements on geometric fidelity, driven by new generative learning approaches on large-scale 3D shape datasets [6,12,47]. While strong promise has been shown in modeling geometry, generating fully textured 3D objects remains a challenge that is less explored. As such, textured 3D content generation still demands tedious manual efforts. A notable challenge in learning to automatically generate textured 3D content is the lack of high-quality textured 3D data. Large-scale shape datasets such as ShapeNet [6] have helped to drive the success of 3D geometric shape modeling, but tend to contain simplistic and often uniform textures associated with the objects. Furthermore, existing texture generation approaches primarily follow popular generative geometric representations that define surfaces implicitly over a volume in space [11,37,48], which results in inefficient learning and tends to produce blurry results, since textures are only well-defined on geometric surfaces.

We propose *Texturify* to address these challenges in the task of automatic texture generation for 3D shape collections. That is, for a given shape geometry, *Texturify* learns to automatically generate a variety of different textures on the shape when sampling from a latent texture space. Instead of relying on supervision from 3D textured objects, we utilize only a set of images along with a collection of 3D shape geometry from the same class category, without requiring any correspondence between image and geometry nor any semantic part information of the shapes. We employ differentiable rendering with an adversarial loss to ensure that generated textures on the 3D shapes produce realistic imagery from a variety of views during training. Rather than generating textures for 3D shapes defined over a volume in space as has been done with implicit representations [3,5,39,45] or volumetric representations [11,48], we instead propose to tie texture generation directly to the surface of the 3D shape. We formulate a generative adversarial network, conditioned on the 3D shape geometry and a latent texture code, to operate on the faces of a 4-way rotationally symmetric quad mesh by defining face convolutional operators for texture generation. In contrast to the common 2D texture parameterization with UV maps, our method enables generating possible shape textures with awareness of 3D structural neighborhood relations and minimal distortion. We show the effectiveness of Texturify in texturing ShapeNet chairs and cars, trained with real-world imagery; our approach outperforms the state of the art by an average of 22% FID scores.
In summary, our contributions are:

- A generative formulation for texture generation on a 3D shape that learns to create realistic, high-fidelity textures from 2D images and a collection of 3D shape geometry, without requiring any 3D texture supervision.
- A surface-based texture generation network that reasons on 3D surface neighborhoods to synthesize textures directly on a mesh surface following the input shape geometry and a latent texture code.

2 Related Works

Texturify aims at generating high-quality textures for 3D objects. Our generative method is trained on distinct sets of 3D shape and real 2D image data. Related approaches either require aligned 2D/3D datasets to optimize for textures, meshes with surface color, or learn a joint distribution of shape and appearance directly from 2D images. In contrast, our method predicts textures for existing 3D shapes using a parameterization that operates directly on the surface and convolutions on the 3D meshes.

Texturing via Optimization. Texture optimization methods address instance-specific texture generation by iteratively optimizing over certain objective functions requiring aligned 2D/3D data. Traditional methods [56] employ global optimization for mapping color images onto geometric reconstructions. To improve robustness against pose misalignments, Adversarial Texture Optimization [21] reconstructs high-quality textures from RGBD scans with a patch-based, misalignment-tolerant conditional discriminator, resulting in sharper textures. Note that these methods are not able to complete missing texture regions and heavily rely on the provided input images. Recently, Text2Mesh [32] proposes to optimize for both color and geometric details of an input mesh to conform to a target text using a CLIP-based [44] multi-view loss.

Texture Completion. IF-Net-Texture [9] focuses on texture completion from partial textured scans and completed 3D geometry using IF-Nets [8], with a convolutional encoder and an implicit decoder. This method learns locally-implicit representations and requires 3D supervision for training. SPSG [11] proposes a self-supervised approach for training a fully convolutional network to first predict the geometry and then the color, both of which are represented as 3D volumetric grids. In comparison to these completion methods, our proposed method only requires an uncoupled collection of 3D shapes and 2D images as supervision.

Retrieval-based Texturing. PhotoShape [41] proposes retrieval-based texturing using a dictionary of high-quality material assets. Specifically, it classifies the part material of an object in a 2D image and applies the corresponding material to the respective parts of a 3D object. While this approach is able to generate high-quality renderings, it requires detailed segmentations in the 2D image as well as for the 3D object, and is unable to produce material that is not present in the synthetic material dataset.

Generative Texturing. TextureFields [37] learns a continuous representation for representing texture information in 3D, parameterized by a fully connected residual network. While this approach can be trained in an uncoupled setting, it tends to produce blurry or uniform textures (see Sec. 4). The work closest to ours in terms of problem formulation is Learning Texture Generators From Internet Photos (LTG) [54], where the texture generation task is formulated as a shape-conditioned StyleGAN [25] generator. This approach requires several different training sets, each containing images and silhouettes from similar viewpoints, and multiple discriminators are employed to enforce correct rendering onto these corresponding viewing angles. In comparison, our method makes no

explicit assumptions on the partitioning of viewpoints. LTG also makes use of UV parameterization, where the texture atlases come from fixed views around the object, which inevitably results in seams and could struggle for non-convex shapes, whereas our method operates directly on mesh faces and is thus free from these two issues. Additionally, LTG utilizes only the silhouettes but not surface features for training, but our approach is geometry-aware.

Generative Models for Geometry and Appearance. 3D-aware image synthesis and automatic generation of 3D models have gained attention recently. Early approaches use discretized spatial representations such as a 3D voxel grid [13,19,33,34,51,52,57], with the downside of high memory footprint that grows cubically with resolution. Scaling up to high-resolution image generation with 3D voxel grids can be challenging, even with sparse volumetric representations [18]. To further alleviate memory consumption, upsampling of synthesized images with 2D convolutional networks has been proposed [35]. An alternative approach is neural implicit representations [5,39,45,14] which encode both the geometry and texture into a single multi-layer perceptron (MLP) network. Seminal work along this line of research includes HoloGAN [33], GIRAFFE [36], GRAF [45], and PiGAN [3]. Several concurrent works focus on the generation of high-resolution images with implicit representations, for example, EG3D [4], StyleNeRF [15], CIPS-3D [55], and StyleSDF [38]. These approaches generate high-quality 2D views rather than the textures for given input meshes, which is the focus of this paper. Additionally, these works generate both the pseudo-geometry and colors in a coupled fashion, resulting in tangled geometry and texture that are difficult to be separated for downstream applications. More recent work [42] generates textured triangle meshes from either random noise latent or 3D semantic layouts, which is also different from our problem formulation which conditions on untextured geometry.

Convolutions on 3D Meshes. Several approaches have been proposed for applying convolutions on meshes [50,30,49,17,22]. For instance, MeshCNN [17] proposes a trimesh-based edge convolution operator, demonstrating part segmentation on relatively small or decimated 3D shape meshes, while our approach aims to generate high-fidelity textures on the faces of high-resolution shape meshes. TangentConv [49] proposes tangent convolution, which projects local surface geometry on a tangent plane around every point and applies 2D planar convolutions within these tangent images. TextureConv [22] defines a smooth, consistently oriented domain for surface convolutions based on four-way rotationally symmetric (4-RoSy) fields and demonstrate superior performance compared to TangentConvs for semantic segmentation. Our approach builds on this TextureConv 4-RoSy parameterization to generate textures on mesh faces.

3 Method

Given a collection of untextured meshes of a class of objects, our method aims to learn a texture generator using only 2D image collections as supervision. We do not assume any correspondence between the shapes and the image collection, except that they should represent the same class category of objects.

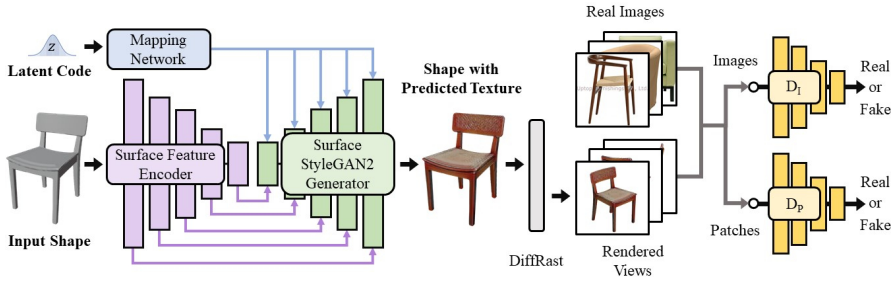


Fig. 2. *Texturify* Overview. Surface features from an input 3D mesh are encoded through a face convolution-based encoder and decoded through a StyleGAN2-inspired decoder to generate textures directly on the surface of the mesh. To ensure that generated textures are realistic, the textured mesh is differentiably rendered from different view points and is critiqued by two discriminators. An image discriminator D_I operates on full image views from the real or rendered views, while a patch-consistency discriminator D_P encourages consistency between views by operating on patches coming from a single real view or patches from different views of rendered images.

We propose a generative adversarial framework to tackle this problem (see Fig. 2). Given a 3D mesh of an object, and latent texture code, our generator produces textures directly on the mesh surface. To this end, we parameterize the mesh with a hierarchy of 4-way rotationally symmetric (4-RoSy) fields from QuadriFlow [23] of different resolutions. This parameterization enables minimal distortion without seams and preserving geodesic neighborhoods. Thus we define convolution, pooling and unpooling operations that enable processing features on the mesh surface and aggregate features across resolution hierarchy levels. Using these operators, we design the generator as a U-Net encoder-decoder network, with the encoder as a ResNet-style feature extractor and the decoder inspired by StyleGAN2 [26], both modified to work on object surfaces. We use differentiable rendering to render the 3D meshes with the generated textures and enforce losses to match the appearance distribution of real image observations. Specifically, we apply two discriminators against the real image distribution for supervision: the first discriminator is inspired by StyleGAN2 on individual rendered images to match the real distribution, while the other encourages global texture consistency on an object through patch discrimination across multiple views. The pipeline is trained end-to-end using a non-saturating GAN loss with gradient penalty and path length regularization.

3.1 Parameterization

Since textures are naturally a surface attribute, parameterizing them in 3D space is inefficient and can result in blurry textures. Therefore, we aim to generate textures directly on the surface using a surface parameterization. One popular way of representing textures on the surface of a mesh is through UV mapping. However, generating a consistent UV parameterization for a set of shapes of

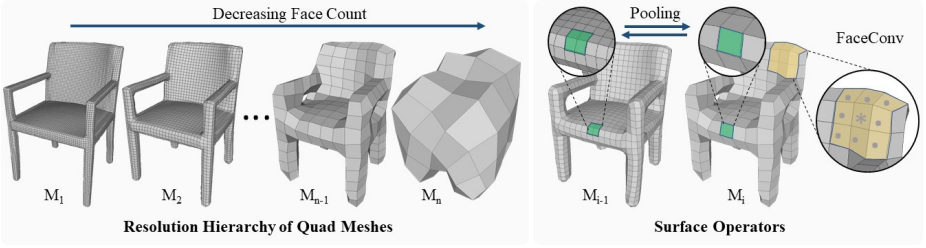


Fig. 3. Texturify generates textures on 4-RoSy parameterized quad meshes, where we define face convolutions with pooling and unpooling operations to operate on a hierarchy of the quad meshes. This enables reasoning about local surface neighborhoods across multiple resolutions to obtain effective global structures as well as fine details.

varying topology is very challenging. Furthermore, it can introduce distortions due to flattening as well as seams at surface cuts. Seams in particular make learning using neighborhood-dependent operators (e.g., convolutions) difficult, since neighboring features in UV space might not necessarily be neighbors in the geodesic space. To avoid these issues, we instead generate surface texture on the four-fold rotationally symmetric (4-RoSy) field parameterization from Quadriflow [23], a method to remesh triangle meshes as quad meshes. A 4-RoSy field is a set of tangent directions associated with a vertex where the neighboring directions are parallel to each other by rotating one of them around their surface normals by a multiple of 90°. This can be realized as a quad-mesh, without seams and with minimal distortions, near-regular faces, and preservation of geodesic neighborhoods, making it very suitable for convolutional feature extraction (as shown in Fig. 3). To facilitate a hierarchical processing of features on the surface of the mesh, we precompute this 4-RoSy field representation of the mesh $\mathcal{M}$ at multiple resolution levels to obtain quad meshes $\mathrm{M}_1, \mathrm{M}_2, .., \mathrm{M}_n$ with face count $\frac{|M_1|}{4^{l-1}}$, with $|M_1|$ being the face count at the finest level (leftmost in Fig. 3).

3.2 Surface Operators

Given a 4-RoSy parameterized quad-mesh, we process features directly on its surface by defining convolutions on the faces. A face convolution operates on a face with feature $\mathbf{x}_i$ and its neighboring faces' features $\mathcal{N}_i = [\mathbf{y}_1, \mathbf{y}_2, ...]$ by:

$$\text{FaceConv}(\mathbf{x}_i, \mathcal{N}_i) = \mathbf{w}_0^T \mathbf{x}_i + \sum_{j=1}^{|\mathcal{N}_i|} \mathbf{w}_j^T \mathbf{y}_j + \mathbf{b} \tag{1}$$

with $\mathbf{x}_i \in \mathbb{R}^{C_0}$, $\mathbf{y}_j \in \mathbb{R}^{C_0}$, learnable parameters $\mathbf{w} \in \mathbb{R}^{C_0 \times C_1}$, $\mathbf{b} \in \mathbb{R}^{C_1}$, where C_0 and C_1 are input and output feature channels respectively. We use a fixed face neighborhood size $|\mathcal{N}_i| = 8$, since the vast majority of the faces in the quad mesh have a neighborhood of 8 faces, with very few singularities from Quadriflow remeshing. In the rare case of singularities (see suppl. doc.), we zero-pad the

faces so that the number of neighbors is always 8. Additionally, neighbors are ordered anticlockwise around the face normal, with the face having the smallest Cartesian coordinates (based on x, then y, then z) as the first face.

For aggregating features across mesh resolution hierarchy levels, we define inter-hierarchy pooling and unpooling operators. The pooled features $\mathbf{x}_{l+1}^j$ are given as $\mathbf{x}_{l+1}^j = \text{agg}\left(\left\{\mathbf{x}_l^i : i \in \text{F}_{l+1}^j\right\}\right)$, where F_{l+1}^j defines the set of face indices of the finer layer l which are nearest to the j-th face of the coarser layer $l+1$ in terms of a chamfer distance and with 'agg' as an aggregation operator. The unpooled features are computed as: $\mathbf{x}_l^j = \mathbf{x}_{l+1}^{\hat{\text{F}}_l^j}$, where $\hat{\text{F}}_l^j$ defines correspondence of the j-th face of the fine layer to the coarse layer (in terms of minimal chamfer distance).

3.3 Surface Texture GAN Framework

With our hierarchical surface parameterization and surface features operators, we design a GAN framework that generates colors on the mesh surface that can be trained using only a collection of images and untextured shapes without any explicit 3D texture supervision.

Our generator takes a U-shaped encoder-decoder architecture. Face normals and mean curvature are used as input features to the network. The encoder is then designed to extract features from the mesh surface at different resolution levels. These features are processed through a series of FaceResNet blocks (ResNet blocks with FaceConv instead of Conv2D) and inter-hierarchy pooling layers as defined above. Features extracted at each level of the hierarchy are then passed to the appropriate level of the decoder through U-Net skip connections. This multi-resolution understanding is essential towards generating compelling textures on a 3D shape, as the deeper features at coarse hierarchy levels enable reasoning about global shape and textural structure, with finer hierarchy levels allowing for generation of coherent local detail.

The decoder is inspired by the StyleGAN2 [26] generator, which has proven to be a stable and efficient generative model in 2D domain. We thus use a mapping network to map a latent code to a style code. Additionally, we upsample (via inter-hierarchy unpooling) and sum up RGB contributions at different mesh hierarchy levels, analogous to StyleGAN2 upsampling and summation at different image resolutions. Instead of the style-modulated Conv2D operators of StyleGAN2 style blocks, we use style-modulated FaceConvs. In contrast to the StyleGAN2 setting, where the generated image has a fixed structure, we aim to generate textures on varying input 3D shape geometries. Therefore, we further condition the style blocks on surface features extracted by the mesh encoder. This is achieved by concatenating the features generated by the decoder at each hierarchy level with the encoder features from the same level. The decoder outputs face colors for the highest resolution mesh ($l = 1$) representing the texture.

To enable training using only 2D image collections for texture supervision, the resulting textured mesh is rendered as an image from multiple viewpoints using a rasterization-based differentiable renderer, Nvdiffrast [29]. Note that we do not

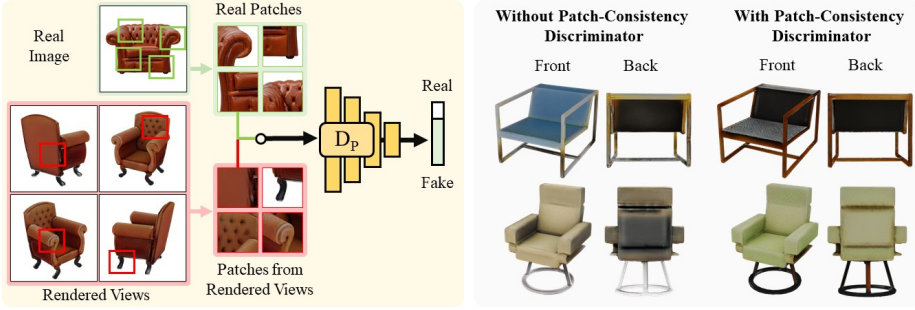


Fig. 4. The patch-consistency discriminator encourages global consistency in generated shape textures across multiple views. (a) While for real image data we are only considering patches from the same view (since the 2D image dataset does not contain multi-view data), we use patches from multiple views in the scenario of generated images. (b) Without patch-consistency discriminator rendered texture can end up having inconsistent styles across viewpoints. Using the patch consistency discriminator prevents this issue.

assume a known pose for the individual images in the real distribution; however, we assume a distribution on the poses from which viewpoints are sampled. Images of the generated textured mesh are then rendered from views sampled from the distribution, which are then critiqued by 2D convolutional discriminators. We use two discriminators: the first, like conventional discriminators, considers whether a single image comes from the real or generated distributions; the second considers whether a set of rendered views from a generated textured shape is consistent across the shape. Since we do not have access to multiple views of the same sample from the real distribution, we consider multiple patches from a single real image sample. For generated images, we then consider multiple patches from different views as input to the patch-consistency discriminator. As patches coming from the same view have a consistent style, the generated patches across views are also encouraged to have a matching style. Operating at patch level is important since for small patches it is harder to distinguish if the patches are coming from the same or from different viewpoints. Fig. 4(b) shows the effect of this patch-consistency discriminator, as considering only single views independently can lead to artifacts where the front and back of a shape are textured inconsistently, while the patch consistency across views leads to a globally consistent textured output. Both discriminators use the architecture of the original StyleGAN2 discriminators and use adaptive discriminator augmentation [24].

3.4 Implementation Details

We use a hierarchy of $n = 6$ quad-meshes with number of faces as (24576, 6144, 1536, 384, 96, 24) from finest to coarsest resolution respectively. Pooling layers use a *mean* operation for aggregating features. During training, each mesh is rendered at a resolution of 512×512 across 4 random viewpoints. The patch

consistency discriminator uses patches cropped to a resolution of 64×64, with 4 patches extracted from each generated viewpoint, yielding a total of 16 patches as input. The generator uses an empirically determined weighting of 10:1 for losses coming from the image discriminator and the patch discriminator.
Our Texturify model is implemented using Pytorch and trained using Adam [27] with learning rates of 1×10^{-4}, 2×10^{-3} and 1×10^{-3} for the encoder, decoder and both discriminators respectively. We train on 2 NVIDIA A6000s for 70k iterations ($\sim$ 80 hours) until convergence. We plan to open source our model, data and data-processing scripts.

4 Experiments

Data. We evaluate our method on 3D shape geometry from the 'chair' and 'car' categories of the ShapeNet dataset [6]. For chairs, we use 5,097 object meshes split into 4,097 train and 1,000 test shapes, and 15,586 images from the Photoshape dataset [41] which were collected from image search engines. For

Fig. 5. Qualitative results on ShapeNet chairs dataset trained with real images from the Photoshape dataset. While methods producing textures in 3D space like SPSG [11], EG3D [4] and TextureFields [37] produce blurry textures, UV based methods like LTG [54] show artifacts at UV seams, specially for non-convex shapes like chairs. By operating on the surface, our method can generate realistic and detailed textures.

Fig. 6. Qualitative results on ShapeNet cars trained with real images from the CompCars dataset. Our method can generate realistic and diverse cars textures on varying shape geometries like sedans, sportscars, SUVs and hatchbacks.

cars, we use 1,256 cars split into 956 train and 300 test shapes. We use 18,991 real images from the CompCars dataset [53] and use an off-the-shelf segmentation model [28] to obtain foreground-background segmentations.

Evaluation Metrics. Our evaluation is based on common GAN image quality and diversity metrics. Specifically, we use the Frechet Inception Distance (FID) [20] and Kernel Inception Distance (KID) [2] for evaluating the generative models. For each mesh, we render images of the textured shapes produced by each method at a resolution of 256×256 from 4 random view points using 4 random latent codes, and evaluate these metrics against all available real images segmented from their background. Note that we do not have ground truth textures available for the 3D shapes and specific style codes, thus, a classical reconstruction metric (e.g., an ℓ_1 distance) is not applicable.

Comparison against state of the art. Tab. 1 shows a comparison to state-of-the-art methods for texture generation on 3D shape meshes. We compare with

Table 1. Comparison against state-of-the-art texture generation approaches on ShapeNet chairs and cars learned on real-world 2D images.

Method	Parameterization	Chairs		Cars	
		KID$\times 10^{-2}\downarrow$	FID$\downarrow$	KID$\times 10^{-2}\downarrow$	FID$\downarrow$
Texture Fields [37]	Global Implicit	6.06	85.01	17.14	177.15
SPSG [11]	Sparse 3D Grid	5.13	65.36	9.59	110.65
UV Baseline	UV	2.46	38.98	5.77	73.63
LTG [54]	UV	2.39	37.50	5.72	70.06
EG3D [4]	Tri-plane Implicit	2.15	36.45	5.95	83.11
Ours	4-RoSy Field	**1.54**	**26.17**	**4.97**	**59.55**

Texture Fields [37], which generates textures as implicit fields around the object, Yu et al. [54] which learns texture generation in the UV parameterization space, and a modified version of EG3D [4] such that it uses a hybrid explicit-implicit tri-plane 3D representation to predict textures for a given mesh conditioned on its geometry. Additionally, we compare with the voxel-based 3D color generation of SPSG [11]; since this was originally formulated for scan completion, we adopt its differentiable rendering to our encoder-decoder architecture using 3D convolutions instead of FaceConvs, with sparse 3D convolutions in the final decoder layer. Finally, we also compare to a UV-based baseline which takes our network architecture with 2D convolutions to learn directly in UV space rather than on a 4-RoSy field. In contrast to these alternatives, our approach generates textures directly on the mesh surface, maintaining local geometric reasoning which leads to more compelling appearance generation for both chair and car meshes (see Fig. 5 and 6). The network architectures for our method and baselines are detailed in the supplementary.

Which surface features are the most informative for texture generation? We evaluate a variety of different local surface features used as input to the encoder network, see Tab. 2. In particular, we consider a case *'None'*, where we do not use a surface feature encoder, thus, the surface StyleGAN generator is reasoning only via the mesh neighborhood structure, a case where we input the 3D *position* of the face centroid, and the cases with local geometric neighborhood characterizations using *Laplacian*, *curvature*, *discrete fundamental forms*, and *normals* as input features. We find that using surface features help significantly over using no features (*'None'*). Further, features dependent on surface geometry like curvature, fundamental forms perform better than positional features like absolute 3D position and Laplacian, with a combination of normals and curvature providing the most informative surface descriptor for our texture generation (see Fig. 7).

What is the impact of the patch-consistency discriminator? When a patch consistency discriminator is not used (see Fig. 4), our method can end up generating textures that might look valid from distinct view points, but as a whole incorporate different styles. Using a discriminator that considers patch consistency across multiple different views enables more globally coherent texture generation (Fig. 4, right), also reflected in improved KID and FID scores (Tab. 2, last two lines).

What is the effect of the mesh resolution on the quality? We compare our base method with 6 hierarchy levels with number of faces as (24576, 6144, 1536, 384, 96, 24) against our method with 5 hierarchy levels with number of faces as (6144, 1536, 384, 96, 24). As seen in Tab. 2 and Fig. 8(a), the increased mesh resolution helps to produce higher-quality fine-scale details, resulting in notably improved performance. Even higher resolutions on our setup were prohibitive in memory consumption.

How does the rendered view resolution affect results? Tab. 2 and Fig. 8(b) show the effect of several different rendering resolutions during training: 64, 128,

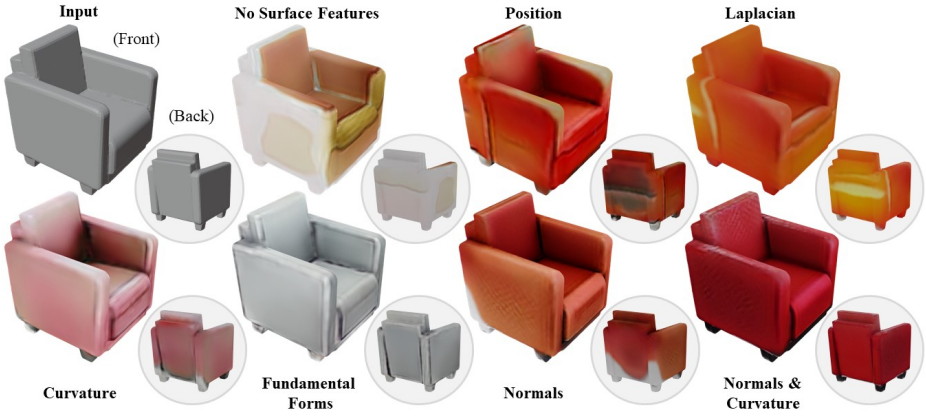


Fig. 7. Effect of different input surface features. Using no surface features (i.e. no surface encoder, only the decoder) produces poor textures since the decoder has limited understanding of the shape. 3D location based features such as position and Laplacian suffer from the inability to effectively align texture patterns with geometric ones. Curvature and fundamental forms introduce spurious line effects due to strong correlation with curvature. Surface normal are quite effective, and are further stabilised when used along with curvature.

256, and 512. Increasing the rendering resolution results in improved quality enabling more effective generation of details.

Does rendering from multiple viewpoints during training help? We consider a varying number of viewpoints rendered per object during each optimization step while training in Tab. 2, using 1, 2, and 4 views. We see that using more views continues to help when increasing from 1 to 2 to 4. Specifically, we see that using multiple views helps avoid artifacts that appear in the mesh across vastly different viewpoints as shown in Fig. 8(c). Note that the multi-view setting also allows patch consistency discriminator to generate consistent textures. We use 4 views during training since increasing the number of views has a decreasing marginal benefit as views will become more redundant.

Learned texture latent space behavior. For a fixed shape, our learned latent space is well behaved with a smooth interpolation yielding valid textures, as shown in Fig. 10. Furthermore, the learned latent space is consistent in style across different shapes, i.e. the same code represents a similar style across shapes (Fig. 9), and can be used, for example, in style transfer applications.

Limitations. While our approach takes a promising step towards texture generation on shape collections, some limitations remain. Since our method uses a real image distribution that comes with lighting effects, and our method does not factor out lighting, the textures learned by our method can have lighting effects baked in. Further, the resolution of the texture generated by our method is limited by the number of faces used at the highest level of the 4-RoSy param-

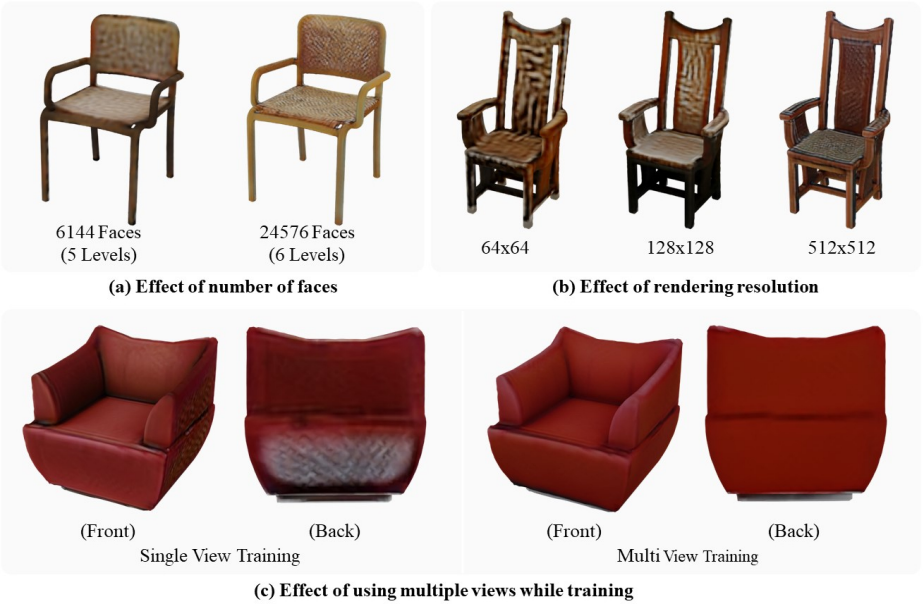


Fig. 8. (a) Increased mesh resolution enables synthesis of higher quality texture. (b) Higher rendering resolution during training helps synthesize more details. (c) Using a single viewpoint per mesh for discrimination can introduce artifacts in the texture across different regions of the mesh. In contrast, using multiple views instead encourages more coherent textures across views.

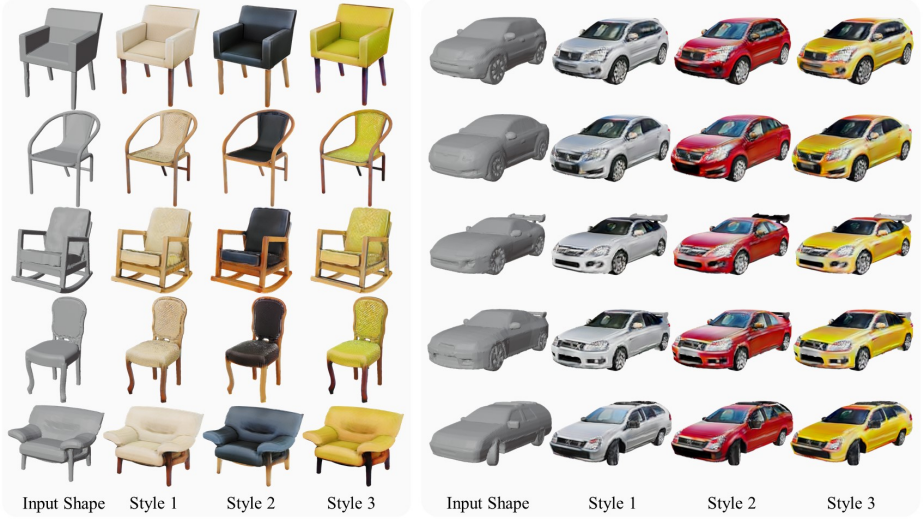


Fig. 9. The latent texture space is consistent across different shape geometry, such that the same latent code gives a similar texture style for different geometries.

Table 2. Ablations on geometric input features, mesh and image resolution, number of views, and the patch-consistency discriminator for our method on ShapeNet chairs.

Input Feature	Mesh-Res	Render-Res	# views	Patch D	KID×10^{-2}↓	FID↓
None	24K	512	4	✓	2.53	37.95
Position	24K	512	4	✓	2.10	34.45
Laplacian	24K	512	4	✓	2.05	34.18
Curvatures	24K	512	4	✓	1.79	29.86
Fundamental Forms	24K	512	4	✓	1.80	30.91
Normals	24K	512	4	✓	1.68	27.73
Normals + Curvature	6K	512	4	✓	2.01	33.95
Normals + Curvature	24K	64	4	✓	2.35	39.32
Normals + Curvature	24K	128	4	✓	1.93	32.22
Normals + Curvature	24K	256	4	✓	1.54	26.99
Normals + Curvature	24K	512	1	✓	1.67	27.83
Normals + Curvature	24K	512	2	✓	1.56	26.95
Normals + Curvature	24K	512	4	X	1.64	27.22
Normals + Curvature	24K	512	4	✓	**1.54**	**26.17**

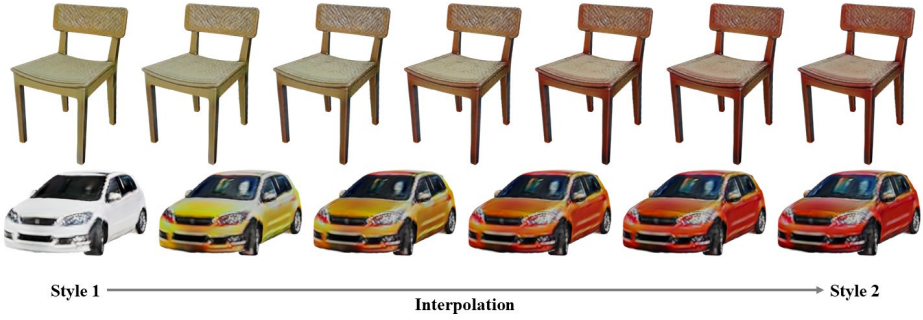


Fig. 10. The texture latent space learned by our method produces smoothly-varying valid textures when traversing across the latent space for a fixed shape.

eterization, whereas learned implicit functions or explicit subdivison at these highest-level faces could potentially capture even higher texture resolutions.

5 Conclusion

We have introduced *Texturify*, a new approach to generate textures on mesh surfaces from distinct collections of 3D shape geometry and 2D image collections, i.e., without requiring any correspondences between 2D and 3D or any explicit 3D color supervision. Our texture generation approach operates directly on a given mesh surface and synthesizes high-quality, coherent textures. In our experiments we show that the 4-RoSy parameterization in combination with face convolutions using geometric features as input outperforms the state-of-the-art methods both quantitatively, as well qualitatively. We believe that *Texturify* is an important step in 3D content creation through automatic texture generation of 3D objects which can be used in standard computer graphics pipelines.

Acknowledgements. This work was supported by the Bavarian State Ministry of Science and the Arts coordinated by the Bavarian Research Institute for Digital Transformation (bidt), a TUM-IAS Rudolf Mößbauer Fellowship, an NVidia Professorship Award, the ERC Starting Grant Scan2CAD (804724), and the German Research Foundation (DFG) Grant Making Machine Learning on Static and Dynamic 3D Data Practical. Apple was not involved in the evaluations and implementation of the code. Further, we thank the authors of LTG [54] for assistance with their code and data.

References

1. Azinović, D., Martin-Brualla, R., Goldman, D.B., Nießner, M., Thies, J.: Neural rgb-d surface reconstruction. arXiv preprint arXiv:2104.04532 (2021) 2
2. Bińkowski, M., Sutherland, D.J., Arbel, M., Gretton, A.: Demystifying mmd gans. arXiv preprint arXiv:1801.01401 (2018) 10
3. Chan, E., Monteiro, M., Kellnhofer, P., Wu, J., Wetzstein, G.: pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In: arXiv (2020) 2, 4
4. Chan, E.R., Lin, C.Z., Chan, M.A., Nagano, K., Pan, B., Mello, S.D., Gallo, O., Guibas, L., Tremblay, J., Khamis, S., Karras, T., Wetzstein, G.: Efficient Geometry-aware 3D Generative Adversarial Networks. ArXiv (2021) 4, 9, 10, 11, 21, 22
5. Chan, E.R., Monteiro, M., Kellnhofer, P., Wu, J., Wetzstein, G.: pi-GAN: Periodic implicit generative adversarial networks for 3D-aware image synthesis. In: CVPR (2021) 2, 4
6. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An Information-Rich 3D Model Repository. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015) 2, 9
7. Chibane, J., Alldieck, T., Pons-Moll, G.: Implicit functions in feature space for 3d shape reconstruction and completion. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6970–6981 (2020) 2
8. Chibane, J., Alldieck, T., Pons-Moll, G.: Implicit functions in feature space for 3d shape reconstruction and completion. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6970–6981 (2020) 3
9. Chibane, J., Pons-Moll, G.: Implicit feature networks for texture completion from partial 3d data. In: European Conference on Computer Vision. pp. 717–725. Springer (2020) 3
10. Dai, A., Diller, C., Nießner, M.: Sg-nn: Sparse generative neural networks for self-supervised scene completion of rgb-d scans. In: CVPR. pp. 849–858 (2020) 2
11. Dai, A., Siddiqui, Y., Thies, J., Valentin, J., Nießner, M.: Spsg: Self-supervised photometric scene generation from rgb-d scans. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1747–1756 (2021) 2, 3, 9, 10, 11, 21
12. Fu, H., Jia, R., Gao, L., Gong, M., Zhao, B., Maybank, S., Tao, D.: 3d-future: 3d furniture shape with texture. International Journal of Computer Vision pp. 1–25 (2021) 2

13. Gadelha, M., Maji, S., Wang, R.: 3D shape induction from 2D views of multiple objects. In: 3DV (2017) 4
14. Gafni, G., Thies, J., Zollhofer, M., Nießner, M.: Dynamic neural radiance fields for monocular 4d facial avatar reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8649–8658 (2021) 4
15. Gu, J., Liu, L., Wang, P., Theobalt, C.: StyleNeRF: A style-based 3D-aware generator for high-resolution image synthesis. ArXiv (2021) 4
16. Gu, S., Bao, J., Chen, D., Wen, F.: Giqa: Generated image quality assessment. In: European Conference on Computer Vision. pp. 369–385. Springer (2020) 20
17. Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., Cohen-Or, D.: Meshcnn: a network with an edge. ACM Transactions on Graphics (TOG) **38**(4), 1–12 (2019) 4
18. Hao, Z., Mallya, A., Belongie, S., Liu, M.Y.: GANcraft: Unsupervised 3D neural rendering of minecraft worlds. In: ICCV (2021) 4
19. Henzler, P., Mitra, N.J., Ritschel, T.: Escaping Plato's cave: 3D shape from adversarial rendering. In: ICCV (2019) 4
20. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017) 10
21. Huang, J., Thies, J., Dai, A., Kundu, A., Jiang, C., Guibas, L.J., Nießner, M., Funkhouser, T., et al.: Adversarial texture optimization from rgb-d scans. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1559–1568 (2020) 3
22. Huang, J., Zhang, H., Yi, L., Funkhouser, T., Nießner, M., Guibas, L.J.: Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4440–4449 (2019) 4, 19, 20, 21
23. Huang, J., Zhou, Y., Niessner, M., Shewchuk, J.R., Guibas, L.J.: QuadriFlow: A Scalable and Robust Method for Quadrangulation. Computer Graphics Forum (2018). https://doi.org/10.1111/cgf.13498 5, 6
24. Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., Aila, T.: Training generative adversarial networks with limited data. Advances in Neural Information Processing Systems **33**, 12104–12114 (2020) 8
25. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. CVPR (2019) 3
26. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of stylegan. CVPR (2020) 5, 7
27. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2015) 9
28. Kirillov, A., Wu, Y., He, K., Girshick, R.: Pointrend: Image segmentation as rendering. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 9799–9808 (2020) 10
29. Laine, S., Hellsten, J., Karras, T., Seol, Y., Lehtinen, J., Aila, T.: Modular primitives for high-performance differentiable rendering. TOG (2020) 7
30. Masci, J., Boscaini, D., Bronstein, M., Vandergheynst, P.: Geodesic convolutional neural networks on riemannian manifolds. In: Proceedings of the IEEE international conference on computer vision workshops. pp. 37–45 (2015) 4
31. Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3d reconstruction in function space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4460–4470 (2019) 2

32. Michel, O., Bar-On, R., Liu, R., Benaim, S., Hanocka, R.: Text2mesh: Text-driven neural stylization for meshes. arXiv preprint arXiv:2112.03221 (2021) 3, 22, 24
33. Nguyen-Phuoc, T., Li, C., Theis, L., Richardt, C., Yang, Y.L.: HoloGAN: Unsupervised learning of 3D representations from natural images. In: ICCV (2019) 4
34. Nguyen-Phuoc, T., Richardt, C., Mai, L., Yang, Y.L., Mitra, N.J.: BlockGAN: Learning 3D object-aware scene representations from unlabelled images. In: NeurIPS (2020) 4
35. Niemeyer, M., Geiger, A.: GIRAFFE: Representing scenes as compositional generative neural feature fields. In: CVPR (2021) 4
36. Niemeyer, M., Geiger, A.: Giraffe: Representing scenes as compositional generative neural feature fields. CVPR (2021) 4
37. Oechsle, M., Mescheder, L., Niemeyer, M., Strauss, T., Geiger, A.: Texture fields: Learning texture representations in function space. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4531–4540 (2019) 2, 3, 9, 10, 11, 20, 21
38. Or-El, R., Luo, X., Shan, M., Shechtman, E., Park, J.J., Kemelmacher-Shlizerman, I.: Stylesdf: High-resolution 3d-consistent image and geometry generation. ArXiv (2021) 4
39. Pan, X., Xu, X., Loy, C.C., Theobalt, C., Dai, B.: A shading-guided generative implicit model for shape-accurate 3d-aware image synthesis. NeurIPS (2021) 2, 4
40. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. In: CVPR (2019) 2
41. Park, K., Rematas, K., Farhadi, A., Seitz, S.M.: Photoshape: Photorealistic materials for large-scale shape collections. arXiv preprint arXiv:1809.09761 (2018) 3, 9
42. Pavllo, D., Kohler, J., Hofmann, T., Lucchi, A.: Learning generative models of textured 3d meshes from real-world images. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 13879–13889 (2021) 4
43. Peng, S., Niemeyer, M., Mescheder, L., Pollefeys, M., Geiger, A.: Convolutional occupancy networks. arXiv preprint arXiv:2003.04618 **2** (2020) 2
44. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International Conference on Machine Learning. pp. 8748–8763. PMLR (2021) 3, 22
45. Schwarz, K., Liao, Y., Niemeyer, M., Geiger, A.: GRAF: Generative radiance fields for 3D-aware image synthesis. In: NeurIPS (2020) 2, 4
46. Siddiqui, Y., Thies, J., Ma, F., Shan, Q., Nießner, M., Dai, A.: Retrievalfuse: Neural 3d scene reconstruction with a database. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 12568–12577 (2021) 2
47. Sun, X., Wu, J., Zhang, X., Zhang, Z., Zhang, C., Xue, T., Tenenbaum, J.B., Freeman, W.T.: Pix3d: Dataset and methods for single-image 3d shape modeling. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018) 2
48. Sun, Y., Liu, Z., Wang, Y., Sarma, S.E.: Im2avatar: Colorful 3d reconstruction from a single image. arXiv preprint arXiv:1804.06375 (2018) 2
49. Tatarchenko, M., Park, J., Koltun, V., Zhou, Q.Y.: Tangent convolutions for dense prediction in 3d. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3887–3896 (2018) 4

50. Verma, N., Boyer, E., Verbeek, J.: Feastnet: Feature-steered graph convolutions for 3d shape analysis. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2598–2606 (2018) 4
51. Wu, J., Zhang, C., Xue, T., Freeman, W.T., Tenenbaum, J.B.: Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In: NIPS (2016) 4
52. Xu, Q., Xu, Z., Philip, J., Bi, S., Shu, Z., Sunkavalli, K., Neumann, U.: Point-nerf: Point-based neural radiance fields. arXiv preprint arXiv:2201.08845 (2022) 4
53. Yang, L., Luo, P., Change Loy, C., Tang, X.: A large-scale car dataset for fine-grained categorization and verification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3973–3981 (2015) 10
54. Yu, R., Dong, Y., Peers, P., Tong, X.: Learning texture generators for 3d shape collections from internet photo sets (2021) 3, 9, 10, 11, 15, 21
55. Zhou, P., Xie, L., Ni, B., Tian, Q.: CIPS-3D: A 3D-Aware Generator of GANs Based on Conditionally-Independent Pixel Synthesis. ArXiv (2021) 4
56. Zhou, Q.Y., Koltun, V.: Color map optimization for 3d reconstruction with consumer depth cameras. ACM Transactions on Graphics (ToG) **33**(4), 1–10 (2014) 3
57. Zhu, J.Y., Zhang, Z., Zhang, C., Wu, J., Torralba, A., Tenenbaum, J.B., Freeman, W.T.: Visual object networks: Image generation with disentangled 3D representations. In: NeurIPS (2018) 4

A Network Architecture

A detailed description of the network for our method can be found in Fig. 11 and 12. Fig. 11 details the encoder, while Fig. 12 displays the StyleGAN2 inspired decoder. Both are based on our 4-RoSy parametrization and the FaceConv and pooling operations presented in the main paper.

Singularities. Singularities are vertices on a quad mesh that do not have a valancy of 4. Our method uses zero-padding on faces with singularity vertices to enforce a fixed size neighborhood. Quad meshes parameterized by quadriflow have few singularities. Specifically, we get 0.89% vertices with singularities for Chair category, and 1.95% for Car category. This is comparable to the proportion of pixel locations that need padding in a 256×256 image (1.56%).

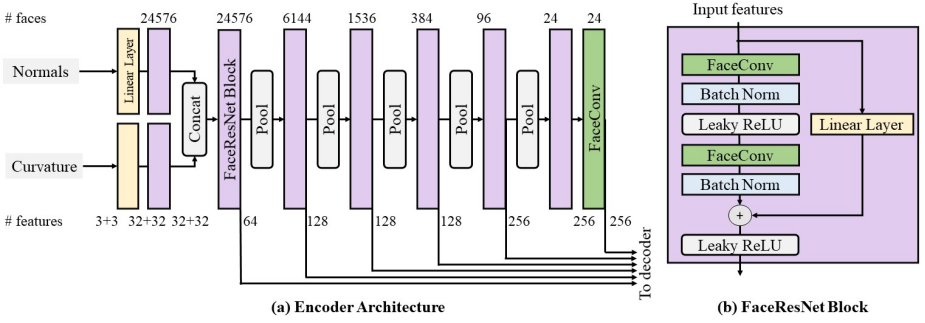
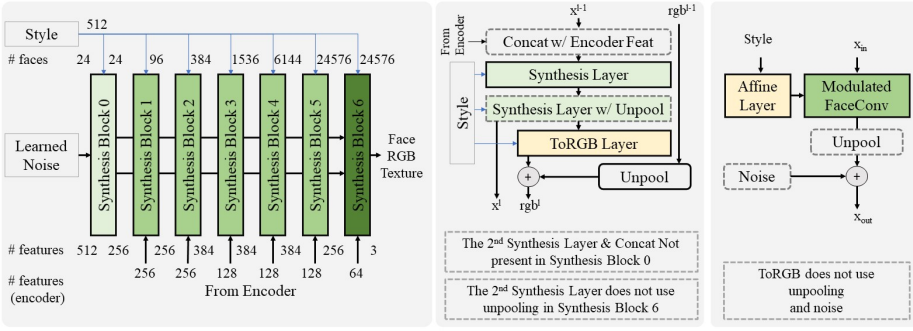


Fig. 11. Encoder architecture. (a) The encoder takes in face normals and curvature at the finest resolution of the hierarchy and extracts features using FaceResNet blocks (b). Features are extracted at all levels of the hierarchy using inter hierarchy pooling and are passed on to the decoder in a U-Net style with skip connections (see Fig. 12). (b) A FaceResNet block is a ResNet block that uses FaceConv instead of Conv2D, and therefore can operate on surface of the mesh.

FaceConvs. As described in the main paper, our approach uses FaceConvs as operator on a 4-RoSy surface. A similar 4-RoSy parameterization has been used in TextureNet [22] for the segmentation of point clouds. Instead of our FaceConvs, which use Cartesian ordering to resolve the 4 way ambiguity, TextureNet introduced TextureConvs a 4-RoSy surface convolutional operator. In Table. 3, we modify our proposed method and replace the FaceConvs with these TextureConvs. We observe that while TextureConvs work reasonably well for the chair category, it struggles with the placement of headlights and the front grill for the car category (Fig. 13).



(a) Decoder Architecture (b) Synthesis Block (c) Synthesis & ToRGB Layers

Fig. 12. Decoder architecture. (a) The decoder is inspired by the StyleGAN2 generator, but rather than operating on a 2D image hierarchy, it operates on quad mesh hierarchy. A learned noise over a cube, which is always the coarsest resolution with 24 faces in the hierarchy, is upsampled to a specific shape through a series of synthesis blocks that take in surface features coming from the encode and the style codes. (b) A synthesis block concatenates features coming from the encoder with the previous synthesis blocks generated features, and passes them through synthesis layers, one of which unpools them to a finer level in the hierarchy. Features generated at the current level are also decoded to an RGB texture and added to the unpooled RGB texture coming from previous layer. (c) Synthesis layer applies a FaceConv with weights modulated by a style code to the input features, and optionally unpools and adds noise to it.

Table 3. Comparison against a modified version of our network that uses TextureConv [22] instead of FaceConvs on ShapeNet chairs and cars learned on real-world 2D images. Our proposed FaceConvs lead to significantly better geometry-aware texture synthesis, especially, on the car dataset (see Fig. 13).

Method	Chairs		Cars	
	KID×10^{-2}↓	FID↓	KID×10^{-2}↓	FID↓
TexureConv [22]	1.88	31.67	6.13	80.10
Ours	**1.54**	**26.17**	**4.97**	**59.55**

B Baseline Methods

We describe the experimental setup for the various baseline comparisons with state-of-the-art texture generation, along with an additional quantitative comparison on the Generated Image Quality Assessment (GIQA) metric [16] in Tab. 4 and additional qualitative comparisons in Fig. 15 and 16. The methods differ mainly in their parametrization as discussed below.

TextureFields. For TextureFields [37], we use the official code and configuration of its GAN variant. We found that training with purely real-world images made the network unstable, so we used a mix (with a probability $p = 0.5$) of real images and synthetic renders with ShapeNet textures.

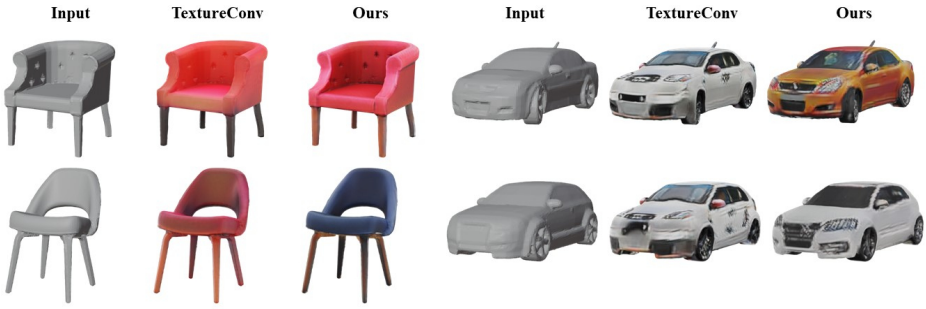


Fig. 13. Comparison with a modified version of our network using texture convolutions from TextureNet [22].

Table 4. Comparison against state-of-the-art texture generation approaches on ShapeNet chairs and cars learned on real-world 2D images.

Method	Parameterization	GIQA$\times 10^{-2}\uparrow$	
		Chairs	Cars
Texture Fields [37]	Global Implicit	6.29	5.14
SPSG [11]	Sparse 3D Grid	6.38	7.19
UV Baseline	UV	7.29	7.84
LTG [54]	UV	7.39	7.90
EG3D [4]	Tri-plane Implicit	7.58	7.85
Ours	4-RoSy Field	**7.73**	**7.99**

SPSG. For the SPSG [11] inspired baseline, we use the exact same architecture as ours (Fig. 11 and 12), except that instead of surface, the networks now operate in a 3D grid. A TSDF grid at the finest resolution of 128^3 is input to the encoder, with features extracted and pooled using 3D ResNet blocks (ResNet block with Conv3D) and trilinear downsampling operators. The decoder uses modulated Conv3Ds instead of modulated FaceConvs and unpooling is performed using trilinear upsampling of features. The last synthesis block uses sparse convolutions because of memory constraints. The decoder outputs a 3D grid of RGB colors instead of per face RGB colors. This color grid is rendered to an image with the shape's TSDF using SPSG's TSDF differentiable rendering.

UV-Space. For the UV baseline, we again use broadly the same architecture as ours. Here, instead of operating on surface using 4-RoSy parameterization, we operate on the surface using UV parameterization. Specifically, we compute the UV maps for the shapes in a fashion similar to LTG [54], i.e. using 6 views (top, bottom, left, right, front, back) around the object. Input to the encoder are the normal and curvature atlas maps. Features are extracted using vanilla 2D ResNet blocks at multiple resolutions and passed to the decoder. The decoder is a regular 2D StyleGAN conditioned (through concatenation like ours) on features coming from encoder. It predicts texture atlases which are mapped to the shape during differentiable rendering. This pipeline differs from LTG as it does not use SPADE-IN blocks for conditioning on silhouettes but instead conditions on

surface features extracted via the encoder. Further, this baseline synthesises a single texture atlas unlike the multiple texture atlases in LTG.

EG-3D. Finally, the EG3D [4] inspired baseline architecture is shown in Fig. 14. Here, given an input mesh and its 3D TSDF representation, a StyleGAN2 network generates a triplane representation, while a 3D TSDF encoder encodes a 3D feature grid. An MLP decoder is then used to query face colors point in space, based on the features projected on the triplane and the feature on the grid at the query point. The mesh with it's face colors is then differentiably rendered and critiqued through a discriminator.

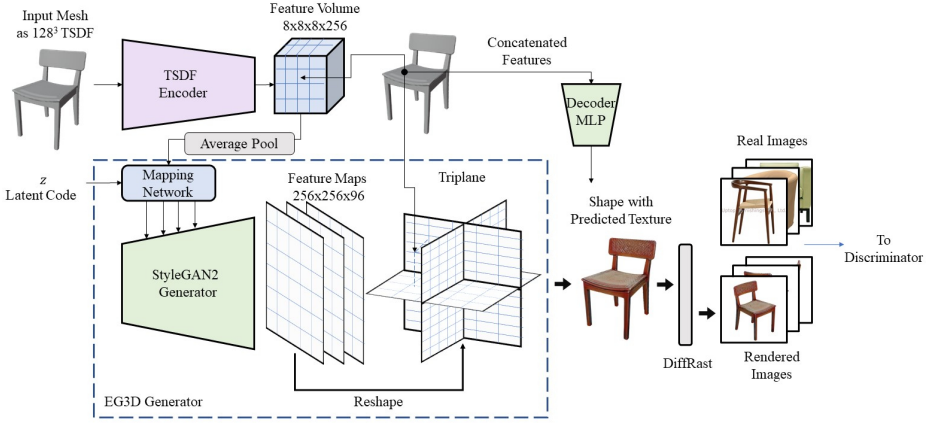


Fig. 14. EG3D [4] inspired baseline architecture. A StyleGAN2 generator outputs a triplane representation with style conditioned on a mesh code. The input mesh represented as a TSDF grid is additionally encoded into an 8^3 feature volume. For points on the mesh surface, features are sampled from the Triplane and 3D feature grid, concatenated, and decoded via an MLP to get face colors. The resulting mesh with face colors is differentiably rendered and critiqued by a discriminator.

C Discussion & Outlook

Our method learns to texture 3D objects from in-the-wild image datasets. It exhibits consistent global and local structural details and can also be used for text-based texture synthesis. To this end, we adapted the Text2Mesh [32] framework to take advantage of our texture model. Specifically, we optimize the latent code passed to our pretrained generator using an evolutionary algorithm such that the CLIP [44] scores between query and the renders are maximized. In Fig. 17, we show a comparison to the original Text2Mesh approach, where we only optimize for the colors on the surface of the mesh. Note that we disable the geometry optimization for this experiment. While Text2Mesh gives good textures when the queries specify a small scale texture description like "brick" or "cactus", it fails to synthesize textures in a semantically consistent way for

Fig. 15. Qualitative results on ShapeNet chairs dataset trained with real images from the Photoshape dataset

Fig. 16. Results on ShapeNet cars trained with real images from the CompCars dataset.

broader queries like “brown chair” or “blue sedan car”. For instance, in the case of the car mesh, Text2Mesh synthesizes smaller images of cars on the surface of the car mesh. In contrast, our method generates semantically consistent textures, also on a higher abstraction level.

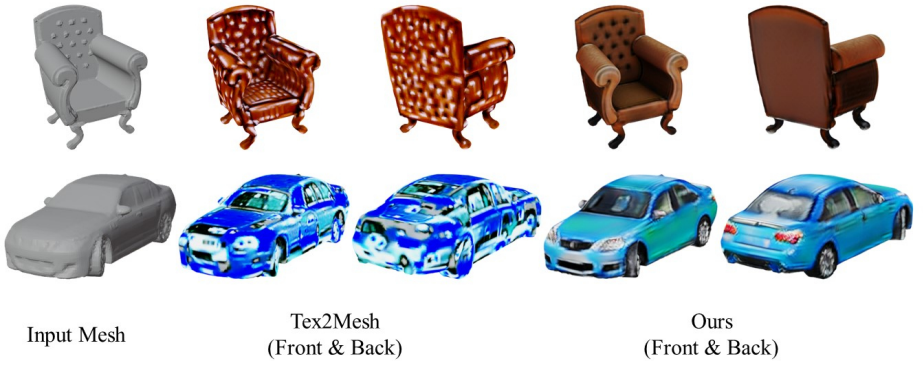


Fig. 17. Comparison with Text2Mesh [32] with queries “brown chair” (top) and “blue sedan car” (bottom).

While we already see a wide applicability of our method, there are limitations that we want to address in future work. As we learn from real world data, we also capture lighting effects, e.g., shadows or specular highlights in our texture. These ‘baked-in’ effects might look reasonable from one view-point, but view-dependent effects like specular highlights should not be synthesized in the texture since they are implausible from other view-points (see Fig. 18). Therefore, additional effort has to be invested to disentangle these effects from the actual diffuse texture. In addition, we think that combining our texture estimation approach that estimates per face colors, with local texture MLPs similar to IF-Nets or ConvOcc (which could predict a color for each point on a face) is an interesting avenue for future research.

Fig. 18. Since our method does not model illumination, the textures produced by our method can end up replicating the lighting effects found in the training images.