Supplementary Material for 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

1 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 [6] 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.

2 Network Architecture

A detailed description of the network for our method can be found in Fig. 1 and 2. Fig. 1 details the encoder, while Fig. 2 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%).

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

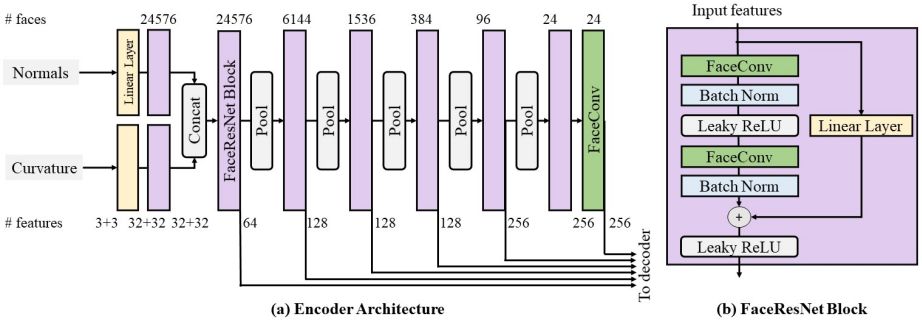

Fig. 1. Encoder architecture. (a) The encoder takes in face normals and curvature at the finest resolution of the hierarchy and extracts features using FaceResNet blocks (b). Features are extracted at all levels of the hierarchy using inter hierarchy pooling and are passed on to the decoder in a U-Net style with skip connections (see Fig. 2). (b) A FaceResNet block is a ResNet block that uses FaceConv instead of Conv2D, and therefore can operate on surface of the mesh.

in TextureNet [5] 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. 1, 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. 4). **GraphConvs.** While GCNs have been shown to work for part segmentation, they can also be adapted to work with such a vertex color representation by treating triangle meshes as graphs. We train a baseline using a graph convolution network for texture generation. In particular, we replace the FaceConvs in Texturify with SAGEConvs [4] which we found to be most effective amongst the graph convolution variants we tested. Since weight modulation is not trivial in a message passing framework, we condition the generated features on style using concatenation. This GCN variant achieves an FID and KID of 75.93 and 5.21×10^{-2} compared to our method's FID and KID of 26.17 and 1.54×10^{-2} on the chair dataset. Qualitative results are shown in Fig. 3.

Table 1. Comparison against a modified version of our network that uses TextureConv [5] 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. 4).

Method	Chairs		Cars	
	KID$\times 10^{-2}\downarrow$	FID$\downarrow$	KID$\times 10^{-2}\downarrow$	FID$\downarrow$
TexureConv [5]	1.88	31.67	6.13	80.10
Ours	**1.54**	**26.17**	**4.97**	**59.55**

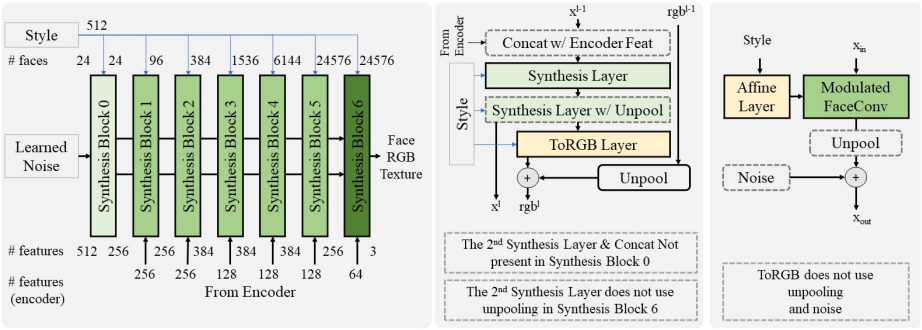

Fig. 2. Decoder architecture. (a) The decoder is inspired by the StyleGAN2 generator, but rather than operating on a 2D image hierarchy, it operates on quad mesh hierarchy. A learned noise over a cube, which is always the coarsest resolution with 24 faces in the hierarchy, is upsampled to a specific 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.

3 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 [3] in Tab. 2 and additional qualitative comparisons in Fig. 6 and 7. The methods differ mainly in their parametrization as discussed below.

Table 2. 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 [8]	Global Implicit	6.29	5.14
SPSG [2]	Sparse 3D Grid	6.38	7.19
UV Baseline	UV	7.29	7.84
LTG [10]	UV	7.39	7.90
EG3D [1]	Tri-plane Implicit	7.58	7.85
Ours	4-RoSy Field	**7.73**	**7.99**

TextureFields. For TextureFields [8], we use the official code and configuration of its GAN variant. We found that training with purely real-world images made

Fig. 3. Results on chairs with graph convolutional generator instead of FaceConvs.

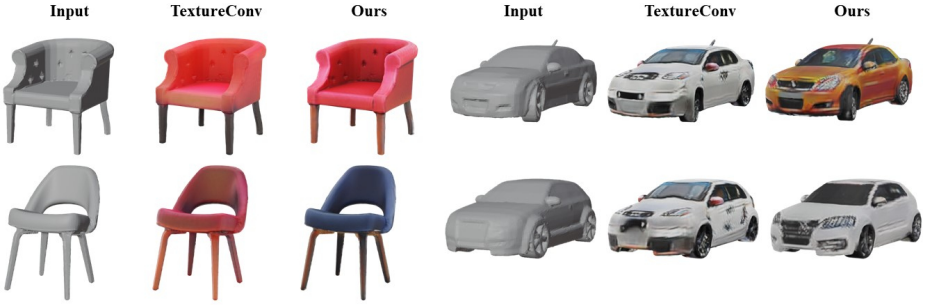

Fig. 4. Comparison with a modified version of our network using texture convolutions from TextureNet [5].

the network unstable, so we used a mix (with a probability $p = 0.5$) of real images and synthetic renders with ShapeNet textures.

SPSG. For the SPSG [2] inspired baseline, we use the exact same architecture as ours (Fig. 1 and 2), 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 [10], 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 [1] inspired baseline architecture is shown in Fig. 5. 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.

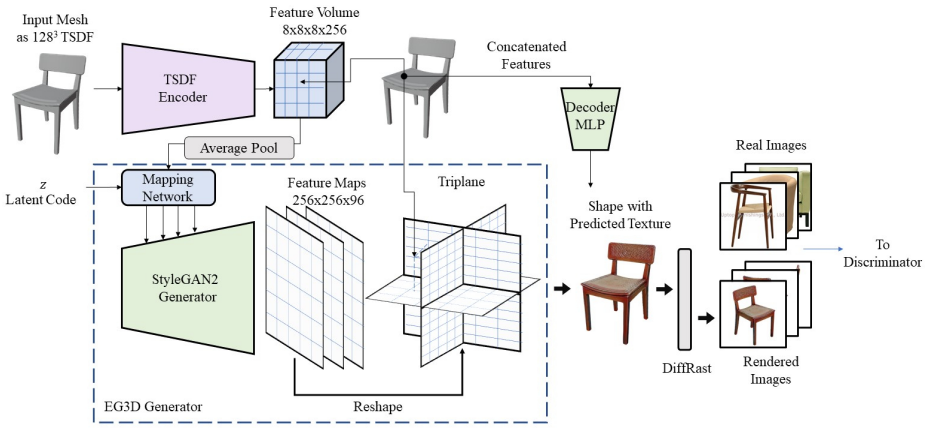

Fig. 5. EG3D [1] inspired baseline architecture. A StyleGAN2 generator outputs a triplane representation with style conditioned on a mesh code. The input mesh represented as a TSDF grid is additionally encoded into an 8^3 feature volume. For points on the mesh surface, features are sampled from the Triplane 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.

4 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 [7] 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 [9] scores between query and the renders are maximized. In Fig. 9, 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

Fig. 6. Qualitative results on ShapeNet chairs dataset trained with real images from the Photoshape dataset

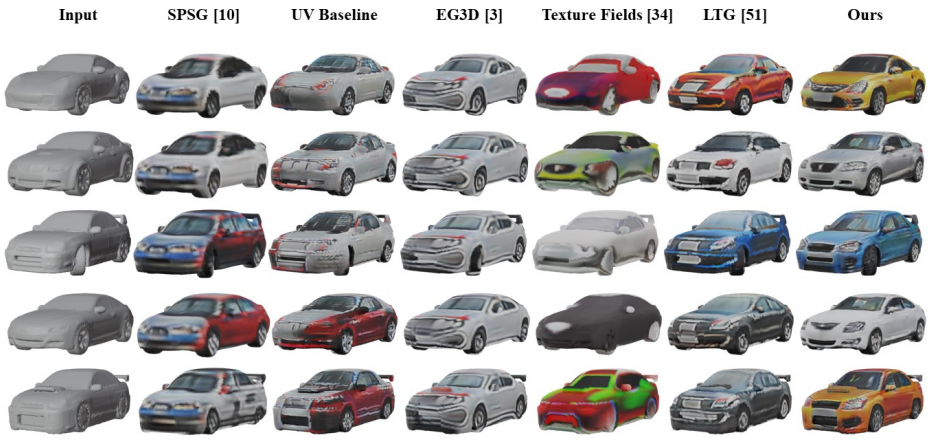

Fig. 7. Results on ShapeNet cars trained with real images from the CompCars dataset.

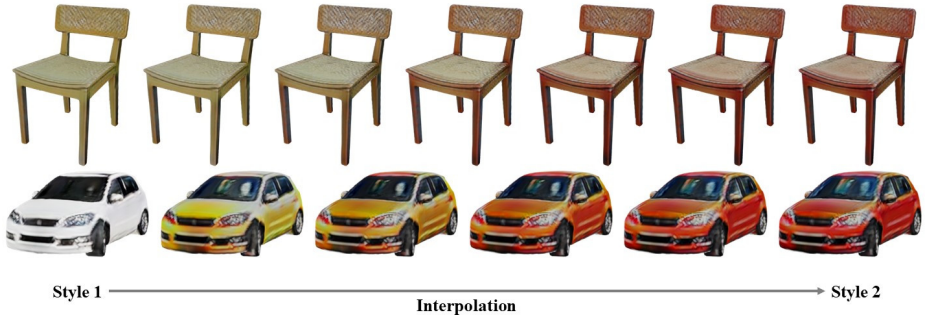

Fig. 8. The texture latent space learned by our method produces smoothly-varying valid textures when traversing across the latent space for a fixed shape.

to synthesize textures in a semantically consistent way for 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.

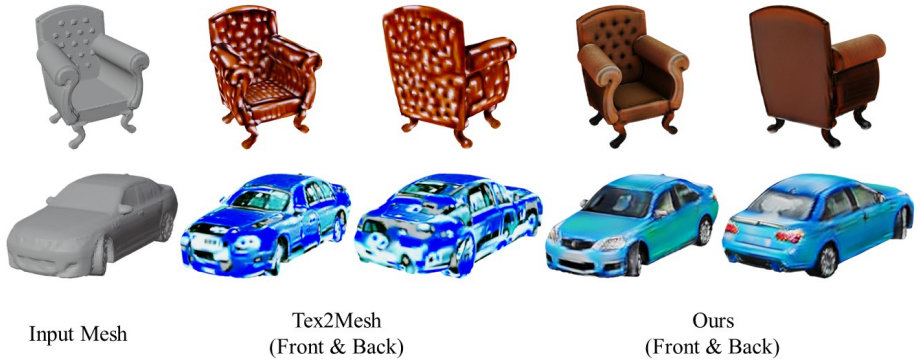

Fig. 9. Comparison with Text2Mesh [7] with queries "brown chair" (top) and "blue sedan car" (bottom).

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. 10). 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. 10. 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.

References

1. 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) 3, 5
2. 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) 3, 4
3. Gu, S., Bao, J., Chen, D., Wen, F.: Giqa: Generated image quality assessment. In: European Conference on Computer Vision. pp. 369–385. Springer (2020) 3
4. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017) 2
5. 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) 2, 4
6. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2015) 1
7. Michel, O., Bar-On, R., Liu, R., Benaim, S., Hanocka, R.: Text2mesh: Text-driven neural stylization for meshes. arXiv preprint arXiv:2112.03221 (2021) 5, 7
8. 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) 3
9. 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) 5

10. Yu, R., Dong, Y., Peers, P., Tong, X.: Learning texture generators for 3d shape collections from internet photo sets (2021) 3, 4