Learning Texture Generators for 3D Shape Collections from Internet Photo Sets

–Supplementary Material–

Rui Yu[1,2]
1996yurui@gmail.com
Yue Dong[2]
yuedong@microsoft.com
Pieter Peers[3]
ppeers@siggraph.org
Xin Tong[2]
xtong@microsoft.com

[1] University of Science and Technology of China
[2] Microsoft Research Asia
[3] College of William & Mary

1 Implementation Details

Network structure We follow the StyleGAN [5] generator and discriminator network structure exactly. We adopt SPADE-IN [8] for conditional input, and MD-GAN [7] for stable multiview training. The detailed network structure is summarized in Figure 1.

Proxy condition training We train the same generator and set of discriminators with the 3D shape collection condition (*i.e.*, shape silhouette) and proxy condition (*i.e.*, image silhouette). In practice, we train one batch with the 3D shape condition, followed by one batch with the proxy condition. For each batch, the generator and discriminators are trained once.

Training parameters We almost completely reuse the StyleGAN [5] training parameters. We initialize our network using He initialization [3] for both the generator and the discriminators. To improve inter-chart consistency, the constant initial vectors for each chart are trained separately, while being initialized to the same constant value: 1. Furthermore, both generator and discriminators are trained with the Adam [6] optimizer, with $\beta_1 = 0.0, \beta_2 = 0.99$. We also reuse the progressive training strategy of StyleGAN [5], by starting the progressive training from a resolution of 8×8, and scale the resolution by a factor of 2 for each step. We exactly follow the same logic as in [5] to determine the batch sizes and training iteration for each intermediate training resolution. Additionally, we also follow [5] and set the learning rate as 0.001 when the resolution is less than 128×128 and 0.0015 for the final resolution of 128×128. When generating low-resolution texture charts, the corresponding images are rendered at the same resolution.

Implementation We implemented our method in Tensorflow [1]. All experimental results in this paper are trained on a server with 4 NVidia V100 GPUs. The training performance varies with the number of charts for the generator and the number of discriminators. For the *Cars* and *Shoes* dataset with 6 charts and 5 discriminators, training takes approximately 90

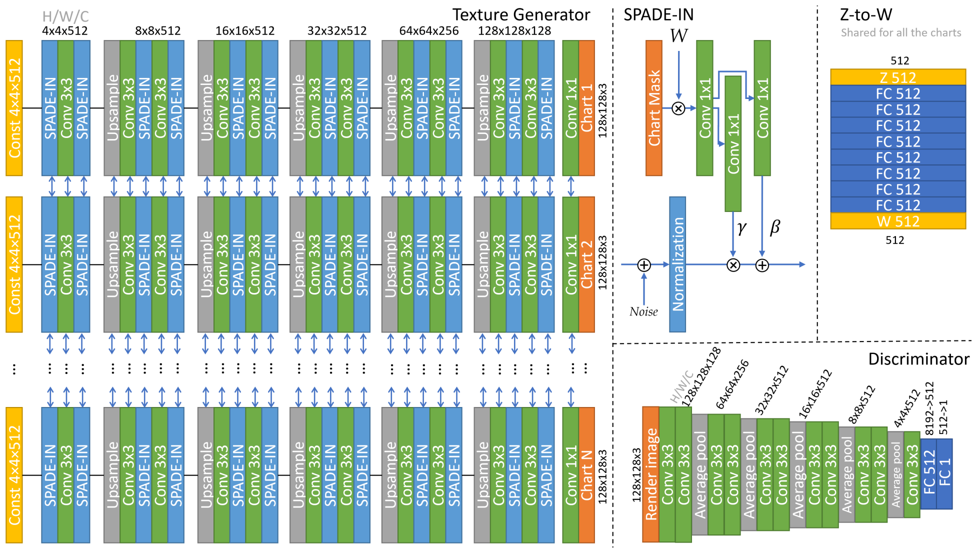

Figure 1: Network structure of the generator and discriminator.

Table 1: Ablation study results for a varying number of training shapes on the *Cars* dataset (GIQA is scaled by 10^2).

Number of training shapes	552	100	50	10
FID ↓	**32.59**	33.14	34.46	36.45
GIQA ↑	**9.910**	9.886	9.834	9.737

hours. For the *Faces* dataset with only 1 chart and 3 discriminators, training takes about 50 hours.

2 Additional Results

Textures for Varying Shapes We demonstrate the rich variations in generated textures in Figure 2 in a grid of rendered results on 3D meshes. This demonstrates that our method can generate a wide variety of textures on different object shapes.

Texture Interpolation Like other GAN-based generation methods, ours also supports interpolation of the generated texture via the latent z-vector. Figure 3 illustrates several interpolation examples among different generated textures.

2.1 Additional Ablation Results

Impact of Number of Training Shapes Our goal is train a texture generator for a collection of 3D shapes. Here we evaluate how the number of 3D shapes impacts the training quality. We train texture generators with $100, 50$, and 10 randomly selected 3D models from the 552 shapes of the *Cars* dataset. Table 1 lists the corresponding FID [4] and GIQA [2] scores measured on textures generated over all the shapes. We found that the generated texture quality gracefully decreases in concert with the number of training shapes.

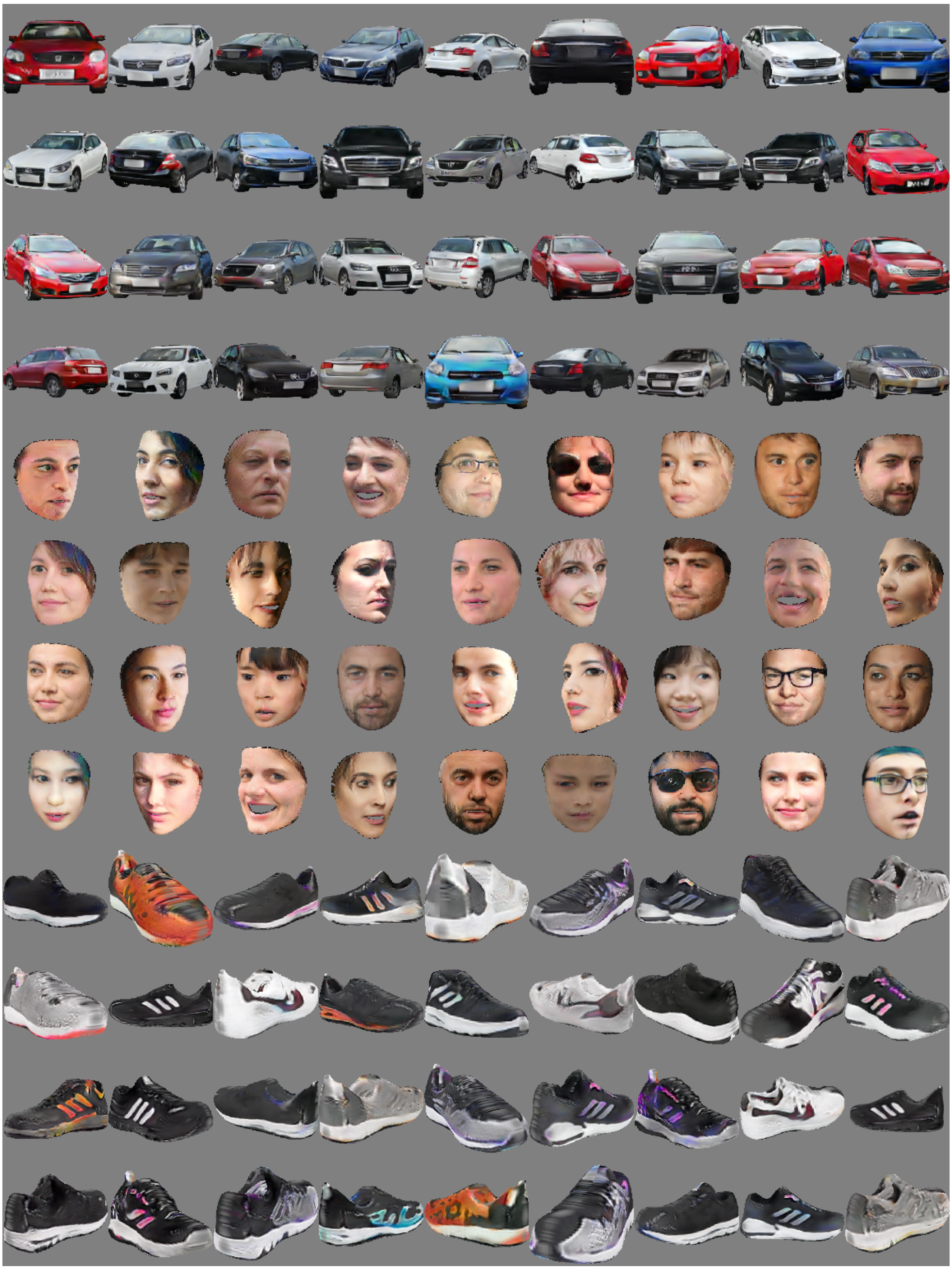

Figure 2: Additional rendering results of various shapes textured with our generator.

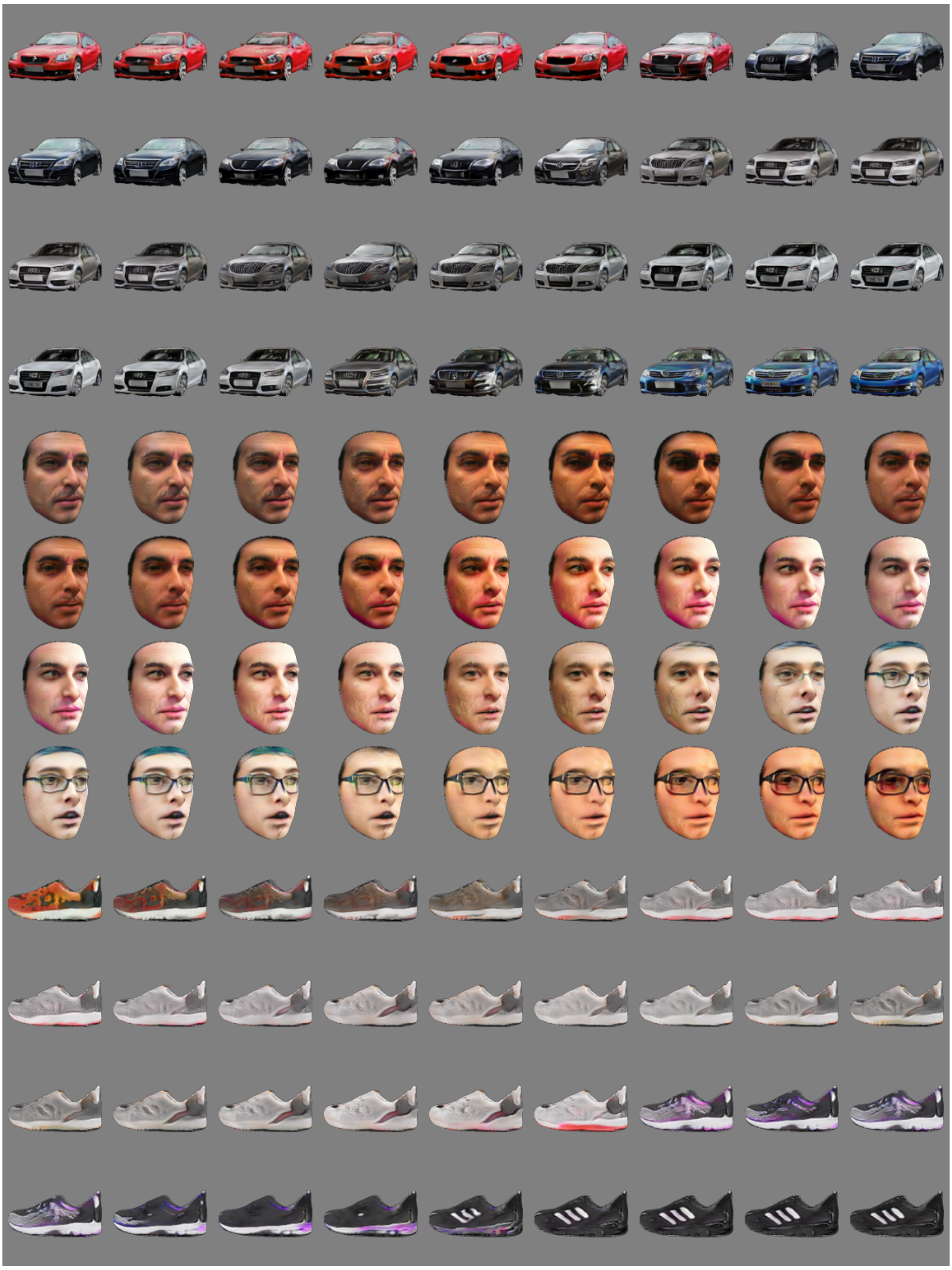

Figure 3: Interpolating textures in the latent z-vector space.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Shuyang Gu, Jianmin Bao, Dong Chen, and Fang Wen. Giqa: Generated image quality assessment. In *ECCV*, pages 369–385. Springer, 2020.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[4] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. In *NIPS*, 2017.

[5] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, pages 4401–4410, 2019.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[7] Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. Synthesizing 3d shapes from silhouette image collections using multi-projection generative adversarial networks. In *CVPR*, pages 5535–5544, 2019.

[8] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, pages 2337–2346, 2019.