Learning High-Fidelity Face Texture Completion without Complete Face Texture (*Supplementry Material*)

Jongyoo Kim, Jiaolong Yang, Xin Tong
Microsoft Research Asia
{jongk,jiaoyan,xtong}@microsoft.com

In this supplementary material, we provide more detailed explanations and extended experimental results of our proposed face texture completion framework, Dual-Space-Discriminator Generative Adversarial Network (*DSD-GAN*).

1. Data Preparation - Details

To collect large-scale reliable texture training data, we propose a robust, automated training data preparation pipeline to obtain quality texture data without time-consuming manual work. As described in the main manuscript, the pipeline consists of three steps: (1) DNN-based 3D face reconstruction, (2) optimization-based refinement, and (3) data cleaning.

(1) 3D Face Reconstruction. In the first step, we employ an off-the-shelf 3D face reconstruction method of [5]. Given a single face image, this algorithm provides robust 3D reconstruction results including face shape, face pose, face texture, lighting parameter, and camera intrinsic, quickly. However, we observed that a small misalignment can lead to obvious artifacts in the texture space. Therefore, to further improve the alignment accuracy, we conduct an additional optimization-based 3D geometry refinement process.

(2) Optimization-based refinement. Following [2], we employ an analysis-by-synthesis approach for the optimization. Similar to [5], we utilize a photometric loss, regularization for 3DMM parameters, and a perceptual distance loss. For the detailed formulation of these three losses, please refer to the paper [5]. In addition, we employ a face parsing algorithm [9] for better accuracy especially around face silhouette. This is used to penalize the situation when the generated face model is larger than the face parsing boundary. We found this important to avoid background pixels around face boundary appearing in texture space. The combined loss is minimized via gradient descent in a Tensorflow framework [1]. Then we can obtain the refined 3DMM parameters, camera parameter, and an illumination coefficient. In the experiments, 120 iterations generally lead to convergence of the optimization.

Finally, to get a smooth visibility map in texture space, we combine two information: normal map and z-buffer. For the former one, we calculate a dot-product of the surface normal and viewing vector which represents a direction from a vertex to the camera center. The i-th texel is invisible if $\mathbf{n_i} \cdot \mathbf{v_i} \leq t$. We set $t = 9^\circ$ in our experiments. In addition, z-buffer data is used to check the self-occluded regions. The z-buffer-based visibility is not used for the other regions because it is rough and sparse in texture space.

(3) Data Cleaning. In general, the raw visibility maps are rough and include lots of isolated small masks. This high-frequency noisy information can affect the model to misunderstand the granularity of texture around mask boundaries. Therefore, we simplify the visibility mask boundary by using closing and opening morphological operations with the kernel size of 15. We additionally applied erosion with the kernel size of 9 to shrink the valid region slightly. Examples of mask simplification are shown in Fig. 1. Through this process, some small holes can be removed around the nose as shown in the figure, but it appears that most of our training data do not contain significant artifacts on small holes.

Then we remove background (non-face) pixels introduced in texture space by using the face parsing information that we generated during the optimization step. Specifically, the face parsing in image space is rendered into texture space using the same process as the face texture extraction. Then each texel can be determined whether it is background or not.

Finally, if the texture is unqualified due to too large hair, occlusion, or background pixels, we discard the data from the training set. This can be also determined by a face parsing map. Some of the discarded examples are shown in Fig. 2.

Figure 1: Examples of mask simplification.

Figure 2: Examples of detected bad textures.

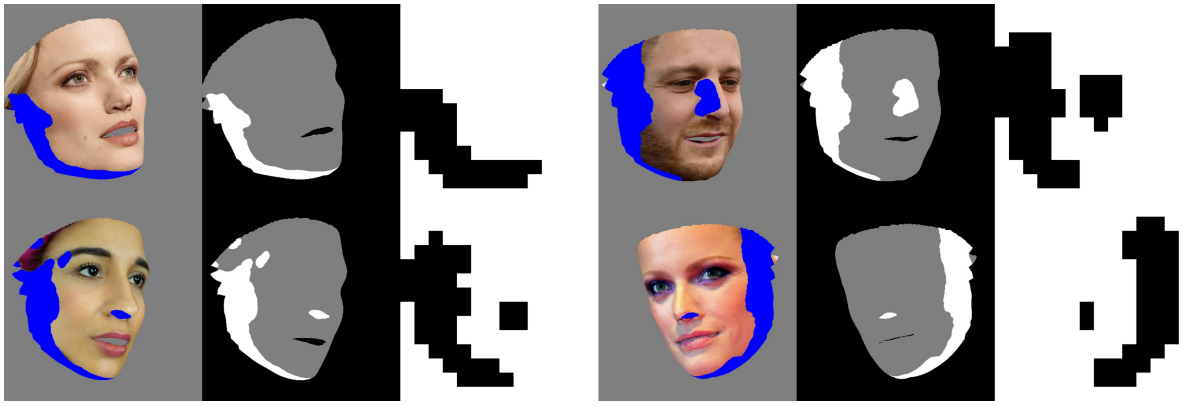

Figure 3: Examples of spatially-varying labels. Blue pixels indicate the hole region.

2. Generating Spatially-Varying Labels

As described in the main manuscript, we use spatially-varying labels for the image-space discriminator. To obtain the label, we first render an input visibility mask using a mesh renderer into image space. The rendered visibility image has 448×448 resolution. Then this map is resized into 14×14 which is the same size as the output of the image-space discriminator. For the resizing, we conducted low-pass filtering followed by stridden sampling. To reflect the actual receptive field size of the image-space discriminator, we used the same number and size of the kernels to the discriminator structure for low-pass filtering. As a result, we obtain a raw 14×14 label image. Finally, this map image is binarized with a threshold of 0.9. That is, larger than 0.9 means 'real' label, and smaller than 0.9 means 'fake' labels. Examples of the spatially-varying labels are shown in Fig. 3. Blue pixels indicate the hole region.

3. Model Architecture

The proposed framework consists of a generator, an image-space discriminator, and a texture-space discriminator. The detailed model architectures are described in Tables 1, 2, 3, and 4. A residual block in the generator is defined in Fig. 4. In the tables, 'Batch norm' indicates batch normalization [7], 'Spec norm' indicates spectral normalization [12], and 'Concat' denotes a concatenation layer.

Input images and visibility masks are rescaled into $[-1, 1]$ and $[0, 1]$, respectively. Since we apply per-channel mean subtraction to input images, the input images of the generator can range between $[-2, 2]$. Accordingly, we use $2 \cdot tanh$ for the output of the generator. Then the subtracted mean is added to the output to reconstruct the original per-channel brightness. We adopt spectral normalization [12] for all the discriminators, which helps to stabilize the GAN training. For the texture-space discriminator, we have a concatenation ('Concat') layer which merges the output of the 'conv7' layer and patch coordinate embeddings. The patch coordinate embedding module consists of a single 'Dense' layer (having 128 channels) where the inputs are normalized x and y coordinates ranging between $[0, 1]$. In the case of the coordinate regression module, it takes the output of the 'Conv7' layer in the texture-space discriminator and outputs a two-dimensional vector.

Layer	Kernel size	Number of channels	Stride / Up factor	Normalization	Nonlinearity
EncConv1	7	64	1	Batch norm	Leaky ReLu
EncConv2	3	128	2	Batch norm	Leaky ReLu
EncConv3	3	192	2	Batch norm	Leaky ReLu
ResBlock1	3	192	1	Batch norm	ReLu
ResBlock2	3	192	1	Batch norm	ReLu
ResBlock3	3	192	1	Batch norm	ReLu
ResBlock4	3	192	1	Batch norm	ReLu
ResBlock5	3	192	1	Batch norm	ReLu
DecConv3	3	128	2	Batch norm	Leaky ReLu
DecConv2	3	64	2	Batch norm	Leaky ReLu
DecConv1	7	3	1	-	2·tanh

Table 1: The architecture of the generator.

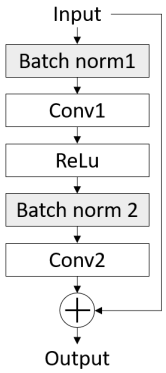

Figure 4: Residual block

Layer	Kernel size	Number of channels	Stride	Normalization	Nonlinearity
Conv1	4	64	2	Spec norm	Leaky ReLu
Conv2	4	128	2	Spec norm	Leaky ReLu
Conv3	4	192	2	Spec norm	Leaky ReLu
Conv4	4	192	2	Spec norm	Leaky ReLu
Conv5	4	192	2	Spec norm	Leaky ReLu
Conv6	4	192	1	Spec norm	Leaky ReLu
Conv7	4	1	1	Spec norm	-

Table 2: The architecture of the image-space discriminator.

4. Training Details

We used an Adam optimizer [8] ($\beta_1 = 0.5$, $\beta_2 = 0.999$) with the learning rate of 0.0001 for all the sub-models. The final losses of our model are defined as

$$\mathcal{L}_{\mathrm{G}} = w_{\mathrm{img}}\mathcal{L}^{G}_{\mathrm{img}} + w_{\mathrm{loc}}\mathcal{L}^{G}_{\mathrm{loc}} + w_{\mathrm{coord}}\mathcal{L}_{\mathrm{coord}} + w_{\mathrm{rec}}\mathcal{L}_{\mathrm{rec}} \tag{1}$$

$$\mathcal{L}_{\mathrm{D}} = w_{\mathrm{img}}\mathcal{L}^{D}_{\mathrm{img}} + w_{\mathrm{loc}}\mathcal{L}^{D}_{\mathrm{loc}} + w_{\mathrm{coord}}\mathcal{L}_{\mathrm{coord}} \tag{2}$$

where $\mathcal{L}_{\mathrm{G}}$ and $\mathcal{L}_{\mathrm{D}}$ indicate the total losses for the generator and the discriminators. $\mathcal{L}^{G}_{\mathrm{img}}$ and $\mathcal{L}^{D}_{\mathrm{img}}$ indicate the LSGAN [11] loss in image space for the generator and the discriminator, while $\mathcal{L}^{G}_{\mathrm{loc}}$ and $\mathcal{L}^{D}_{\mathrm{loc}}$ denote the LSGAN [11] loss in texture space for the generator and the discriminator, respectively. Each loss is defined in the main manuscript. And w indicates the loss weight of each term. In our experiments, we set $w_{\mathrm{img}} = 0.2$, $w_{\mathrm{loc}} = 1.0$, $w_{\mathrm{coord}} = 10$, and $w_{\mathrm{rec}} = 10$.

Layer	Kernel size	Number of channels	Stride	Normalization	Nonlinearity
Conv1	4	64	2	Spec norm	Leaky ReLu
Conv2	4	128	2	Spec norm	Leaky ReLu
Conv3	4	192	2	Spec norm	Leaky ReLu
Conv4	4	192	2	Spec norm	Leaky ReLu
Conv5	4	192	2	Spec norm	Leaky ReLu
Conv6	4	192	1	Spec norm	Leaky ReLu
Conv7	4	192	1	Spec norm	Leaky ReLu
Concat	-	192 + 128	-	-	-
Dense1	-	256	-	-	Leaky ReLu
Dense2	-	1	-	-	-

Table 3: The architecture of the texture-space discriminator.

Layer	Number of channels	Normalization	Nonlinearity
Input: the output of Conv7 in Tex-Space Discriminator			
Dense1	256	-	Leaky ReLu
Dense2	2	-	Linear

Table 4: The architecture of the coordinate regression module.

To improve the robustness of the model, for every n_{half} training step, we applied a random half mask to the input incomplete texture, and employed reconstruction loss on the masked region. The mask was randomly chosen from four candidates: (1) upper-half, (2) lower-half, (3) left-half, and (4) right-half masks. During this step, the following loss was minimized.

$$\mathcal{L}_{\text{G-half}} = w_{\text{loc}}\mathcal{L}_{\text{loc}}^{G} + w_{\text{rec}}\mathcal{L}_{\text{rec}} + w_{\text{half}}\mathcal{L}_{\text{half}} \tag{3}$$

$$\mathcal{L}_{\text{D-half}} = w_{\text{loc}}\mathcal{L}_{\text{loc}}^{D} \tag{4}$$

where $\mathcal{L}_{\text{half}}$ is the $L1$ reconstruction loss of the half-masked region. We set $n_{\text{half}} = 3$ and $w_{\text{half}} = 10$.

We implemented our framework using Tensorflow [1]. We trained our model on four GPUs (NVIDIA Tesla P100 16GB), where the batch size of each GPU was 4.

5. Extended Experimental Results

To further verify the robustness or our algorithm, we show extended inferred results of DSD-GAN in Figs 5, 6, and 7. We classified the results according to the input image's yaw angle into three: less than 10°, between 10° and 40°, and larger than 40°, and each is shown in each figure, respectively. The results are best viewed by zooming-in. For each face image, the first column is the input, the second column is the inferred texture, and the fourth and fifth columns show rendered images with two fixed poses. It shows that DSD-GAN can yield consistently good results for various input data, and the rendered images also look convincing.

Our DSD-GAN yields the compelling quality for both near-frontal and large poses with rich high-frequency details generated. For the former, DSD-GAN is able to look up the neighboring pixels on the same side and generate seamless contents. For the latter, DSD-GAN can refer to the opposite side and produce globally-consistent results. It is worth noting that DSD-GAN *does not naively replicate the textures from the other visible side*. Instead, we observed that it generates semantically-symmetric content (such as the positions of hair, beard, and skin properties) but with diverse and natural textural details, as can be seen in the figures.

6. Extended Pose Invariance Experiment

To check whether the model can generate consistent textures from different face poses, we generated 7 visibility maps having yaw angles from -75° to 75°. In Fig. 8, the images in the first column show the ground-truth full textures, and the others are inferred images by DSD-GAN. It can be observed that DSD-GAN can deal with various poses from frontal ones to large ones by combining the neighboring and opposite-side pixel information properly.

Figure 5: Extended inferred results (yaw angle $< 10^{\circ}$). **Best viewed with zoom.**

Figure 6: Extended inferred results ($10^\circ \leq$ yaw angle $< 40^\circ$). **Best viewed with zoom.**

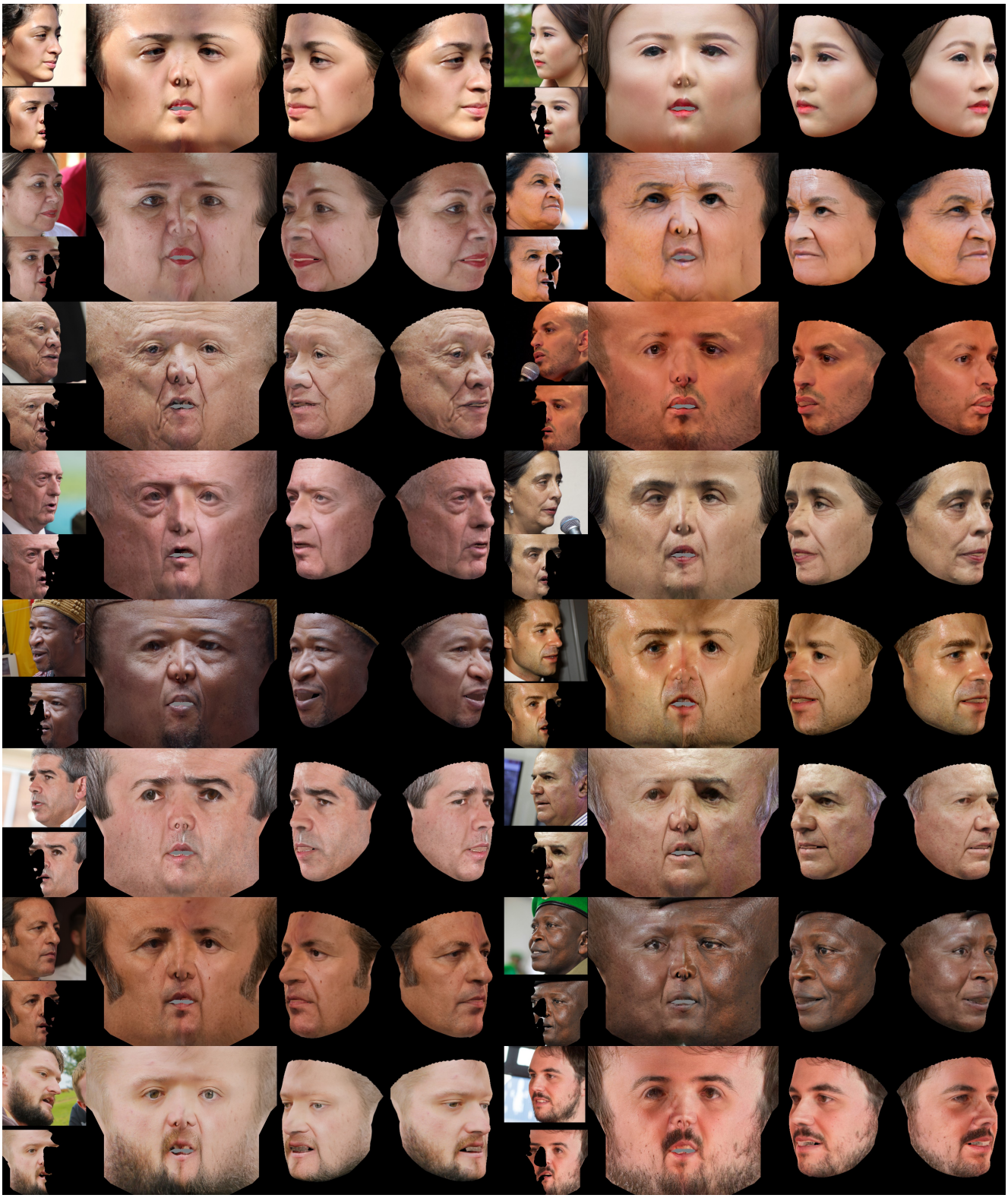

Figure 7: Extended inferred results (yaw angle $> 40°$). **Best viewed with zoom.**

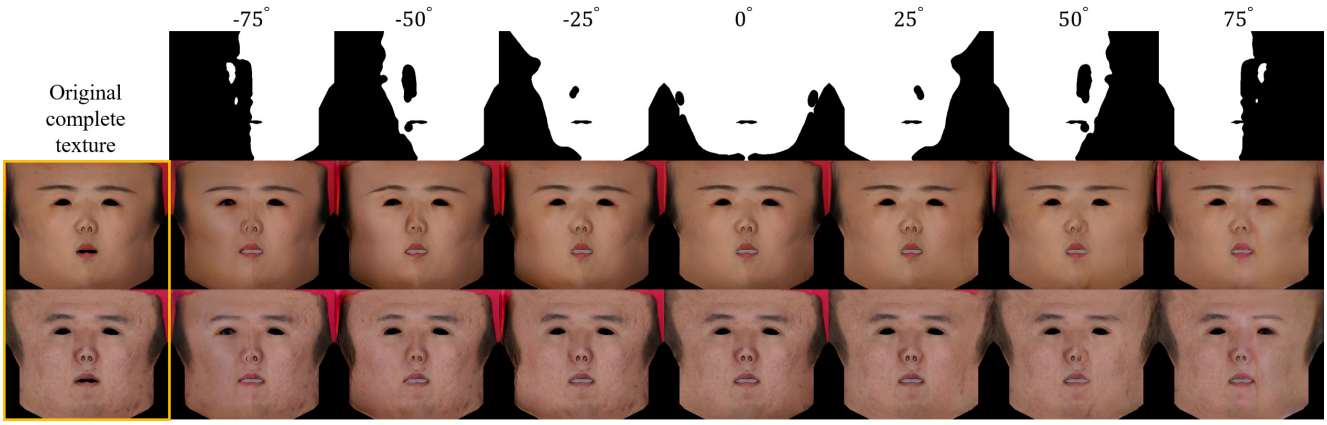

Figure 8: Pose invariance test.

7. Extended Comparison to UV-GAN [4]

We show an extended comparison of DSD-GAN and UV-GAN [4]. As mentioned in the main manuscript, since they do not provide the pre-trained models or full training datasets, we took visual results of UV-GAN from their original paper. The comparison results are shown in Fig. 9. Here we summarize the key differences between DSD-GAN and UV-GAN:

- **Training data.** While DSD-GAN is trained without any complete face texture, UV-GAN is trained with 77K complete face textures in a supervised fashion.
- **Output resolution.** DSD-GAN generates 512×512 textures, while UV-GAN infers up to 256×256.
- **Detail preservation.** DSD-GAN always preserves visible details. However, UV-GAN preserves the details for the frontal view, on the other hand, for large-pose faces, UV-GAN regenerates the whole texture details (eyebrows, gaze, shading, *etc*.), as can be observed from the last three samples in Fig. 9.

In general, both models generate compelling results. When it comes to details, DSD-GAN appears to yield better high-frequency textures such as plausible beard and skin details, which can be checked by zooming-in images in Fig. 9. Given low-quality input like the second and third images in Fig. 9, DSD-GAN could generate blurry outputs. Please note that DSD-GAN is able to generate high-fidelity textures given high-resolution images, as shown in various examples of Figs 5, 6, and 7.

For profile views ($\sim90°$), 3D reconstruction can be less accurate which affects texture alignment. This sometimes leads to weird facial components when viewed in texture space as shown in Fig. 9 (c). Since DSD-GAN preserves all the visible details from the input, the output of DSD-GAN can also suffer from inaccurate alignment as shown in Fig. 9 (d).

8. Extended Comparison to GANFIT [6] and Lin *et al.* [10]

We also show an extended comparison to GANFIT [6] Lin *et al.* [10] using the displayed images from their paper. Since we do not have the original testing data to GANFIT, we searched for the most similar images on the Internet and the public datasets. The full comparison is shown in Fig. 10, where we also showed both inputs to GANFIT and DSD-GAN. For the fifth and sixth inputs to DSD-GAN, they are slightly different from the inputs to GANFIT, and for the last one, we used a low-resolution image. Note that GANFIT generates albedo textures so that the shading is different from the inputs. Since the GANFIT is built based on high-quality large-scale captured data (10K complete UV data) [3], it yields high-quality textures. However, it may not be the best way to preserve the original textural details of the input images.

On the other hand, Lin *et al.* [10] aims to refine the face texture from the reconstructed BFM texture, resulting in reasonable details. But the resolution is lower than ours as [10] generates per-vertex colors $\in \mathbb{R}^{35.7K\times3}$ while DSD-GAN generates texture images $\in \mathbb{R}^{262K\times3}$ (512×512). In addition, while [10] focuses on visible parts, ours aims to infer high-fidelity self-occluded texture.

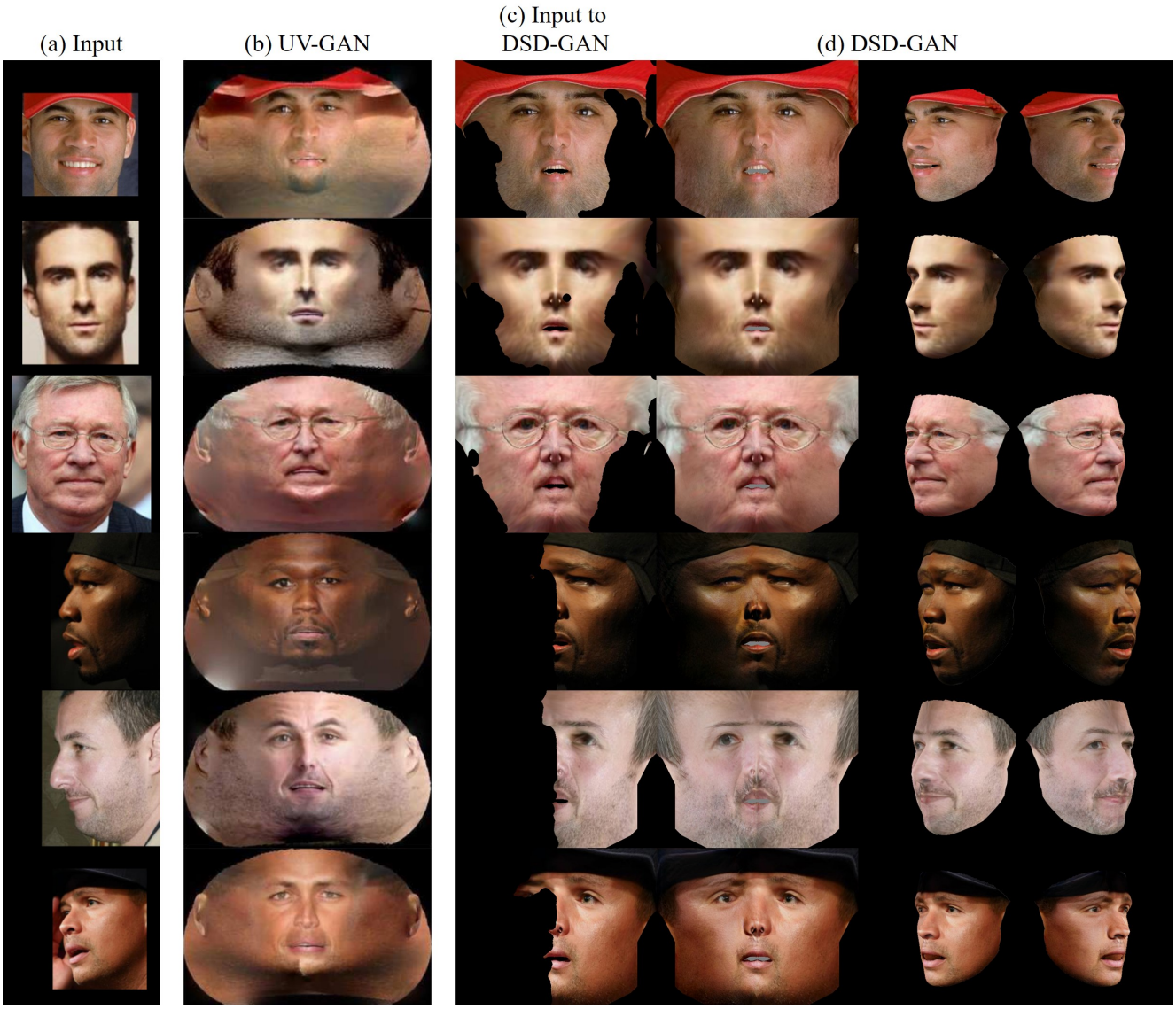

Figure 9: Extended comparison to UV-GAN [4]. Note that DSD-GAN always preserves raw visible texture details. In contrast, UV-GAN does not preserve the raw texture for large-pose cases (last three rows). **Best viewed with zoom.**

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2015. 1, 4

[2] V. Blanz and T. Vetter. A Morphable Model for the Synthesis of 3D Faces. In *ACM Siggraph*, SIGGRAPH '99, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 1

[3] J. Booth, A. Roussos, S. Zafeiriou, A. Ponniah, and D. Dunaway. A 3D Morphable Model Learnt From 10,000 Faces. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5543–5552, 2016. 8

[4] J. Deng, S. Cheng, N. Xue, Y. Zhou, and S. Zafeiriou. UV-GAN: Adversarial Facial UV Map Completion for Pose-Invariant Face Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7093–7102, 2018. 8, 9

[5] Y. Deng, J. Yang, S. Xu, D. Chen, Y. Jia, and X. Tong. Accurate 3D Face Reconstruction with Weakly-Supervised Learning: From

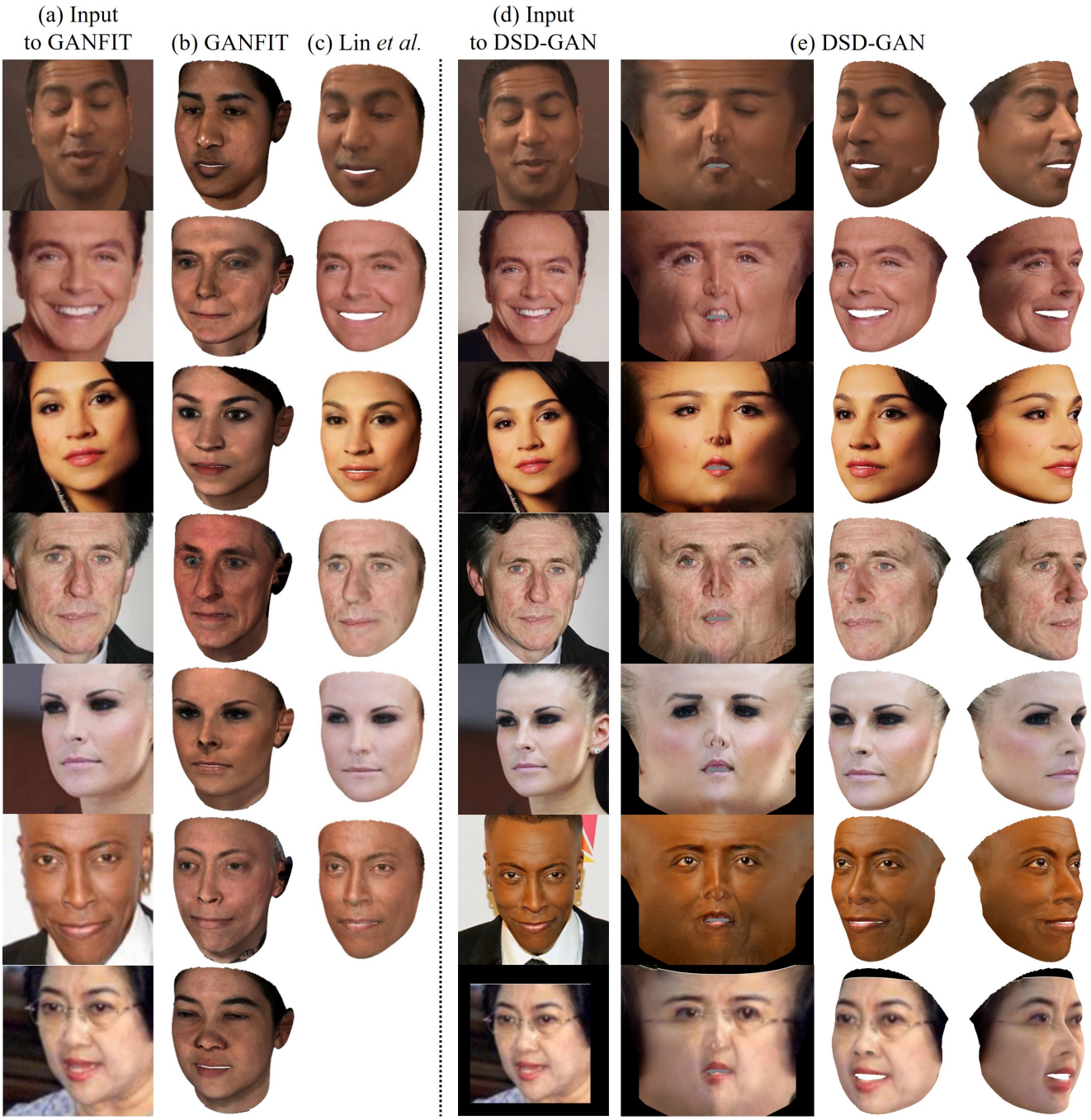

Figure 10: Extended comparison to GANFIT [6] and Lin *et al.* [10]. **Best viewed with zoom.**

Single Image to Image Set. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, Mar. 2019. 1

[6] B. Gecer, S. Ploumpis, I. Kotsia, and S. Zafeiriou. GANFIT: Generative Adversarial Network Fitting for High Fidelity 3D Face Reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, Feb. 2019. 8, 10

[7] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on International Conference on Machine Learning*, Mar. 2015. 3

[8] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015. 3

[9] J. Lin, H. Yang, D. Chen, M. Zeng, F. Wen, and L. Yuan. Face Parsing with RoI Tanh-Warping. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2019. 1

[10] J. Lin, Y. Yuan, T. Shao, and K. Zhou. Towards High-Fidelity 3D Face Reconstruction from In-the-Wild Images Using Graph Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, Mar. 2020. 8, 10

[11] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. Least Squares Generative Adversarial Networks. In *IEEE International Conference on Computer Vision*, Apr. 2017. 3

[12] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations*, Feb. 2018. 3