AgileAvatar: Stylized 3D Avatar Creation via Cascaded Domain Bridging - Supplementary Materials

S. Sang, T. Zhi, G. Song, M. Liu, C. Lai, J. Liu, X. Wen, J. Davis, and L. Luo

1 PORTRAIT STYLIZATION DETAILS

Segmentation Models: The avatar segmentation model is trained using 20k randomly sampled avatar vectors with neural pose, expression and illumination. For real image segmentation, we used an open-source pre-trained BiSeNet module[1] [Yu et al. 2018].

Distribution Prior $\mathcal{W}$: To sample W+ distribution prior, we inverse CelebA dataset [Liu et al. 2015] into W+ space using a pre-trained e4e encoder [Tov et al. 2021].

Normalized Style Exemplar Set $\mathcal{Y}$: For training stylized generator $\mathcal{G}_{\phi_t}$, we synthetically rendered a diverse set of 150 avatar imageries with normalized facial expressions.

2 AVATAR PARAMETERIZATION DETAILS

2.1 Imitator

To train our module in a self-supervised way, we plug-in a differentiable neural renderer (i.e. imitator) in our learning framework. As we mentioned in the main paper, the imitator can take a relaxed avatar vector as input, although the imitator itself is trained with strict avatar vector. No matter the input is a relaxed or strict avatar vector, it can produce a valid rendering. In this way, we can supervise the training in image space without any ground-truth for the parameters. Due to the differentiability of the imitator, the parameterization stage can be trained with gradient descent. To achieve high fidelity rendering quality, we leverage the StyleGAN2 generator [Karras et al. 2019] as our backbone, which is capable of generating high quality renderings matching the graphics engine. The imitator consists of an encoder $\mathcal{E}_i$ implemented using MLP and a generator $\mathcal{G}_i$ adopted from StyleGAN2. The encoder translates an input avatar vector to a latent code $w+$. The generator then produces a high-quality image given the latent code.

Training: In order to fully utilize the image generation capability of StyleGAN2, we propose to train the imitator in two steps: 1) we first train a StyleGAN2 from scratch with random rendering samples generated by our graphics engine to obtain a high-quality image generator, without any label or conditions; then 2) we train the encoder and the generator together with images and corresponding labels, result in a conditional generator. Given an avatar vector v, a target image $\mathcal{I}_{gt}$ and the generated image $\mathcal{I}_{gen} = \mathcal{G}_i(\mathcal{E}_i(v))$, we use the following loss function combination to perform the second step training:

$$\mathcal{L}_{imitator} = \lambda_1 \|\mathcal{I}_{gen} - \mathcal{I}_{gt}\|_1 + \lambda_2 \mathcal{L}_{lpips} + \lambda_3 \mathcal{L}_{id} \tag{1}$$

[1]https://github.com/zllrunning/face-parsing.PyTorch

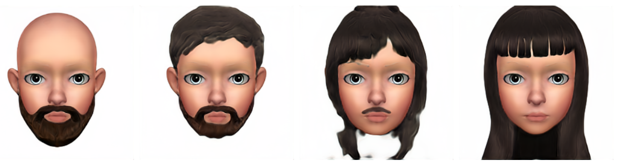
Figure 1: Interpolation of avatar vectors. The neural rendering imitator which temporarily replaces the traditional graphics engine is differentiable, allowing the relaxation of the strict constraint on discrete types. Linear interpolation between two avatar vectors results in the gradual disappearance of the beard and the gradual growth of the hair.

where the first term is an L1 loss, which encourages less blurring than L2. In addition, $\mathcal{L}_{lpips}$ is the LPIPS loss adopted from [Zhang et al. 2018],

$$\mathcal{L}_{lpips} = \|\mathcal{F}(I_1) - \mathcal{F}(I_2)\|_2 \tag{2}$$

where $\mathcal{F}$ denotes the perceptual feature extractor. $\mathcal{L}_{id}$ is the identity loss which measures the cosine similarity between two faces built upon a pretrained ArcFace [Deng et al. 2019] face recognition network $\mathcal{R}$,

$$\mathcal{L}_{id} = 1 - cos(\mathcal{R}(I_1), \mathcal{R}(I_2)) \tag{3}$$

We set $\lambda_1 = 1.0$, $\lambda_2 = 0.8$, $\lambda_3 = 1.0$, empirically.

Interpolation property: Fig. 1 provides an example of the interpolation property of the imitator which enables relaxed optimization over the discrete parameters.

Implementation: To train the imitator, we randomly generate 100,000 images and corresponding parameters. Note that although random sampling leads to strange avatars, our imitator can generate images matching the graphics engine well by seeing plenty of samples in the parameter space. Please refer to our supplementary video for a side-by-side comparison.

We train StyleGAN2 using the official source code[2] with images of size $256 \times 256 \times 3$, thus the latent code $w+$ has a shape of 14×512. We build the encoder $\mathcal{E}_i$ with 14 individual small MLPs, each is responsible for mapping from the input vector to one latent style. Given the pretrained generator, we train the encoder and simultaneously finetune the generator with Adam [Kingma and Ba 2015]. We set the initial learning as 0.01 and decay it by 0.5 each two epochs. In our experiments, it takes around 20 epochs to converge.

2.2 Mapper

We use CelebA-HQ [Lee et al. 2020] and FFHQ [Karras et al. 2019] as our training data. To collect a high quality dataset for training,

[2]https://github.com/NVlabs/stylegan2-ada-pytorch

we use the Azure Face API [3] to analyze the facial attributes and keep only facial images that meet our requirements:

1) within a limited pose range ($yaw < 8°, pitch < 8°, roll < 5°$)
2) without headwears
3) without extreme expressions
4) without any occlusions

Finally, we collect 21,522 images in total for mapper training.

The input is an 18×512 latent code taken from the Stylization module. Each one of the 18 layers latent code is passed to an individual MLP. The output features are then concatenated together. After that, we apply two MLP heads to generate continuous and discrete parameters separately.

We apply a scaling before the softmax function for discrete parameters:

$$\mathcal{S}(x) = \frac{e^{\beta x_k}}{\Sigma_{i=1}^{N} e^{\beta x_i}}, k = 1, ...N \tag{4}$$

where $\beta > 1$ is a coefficient that performs non-maximum suppression over some types that contribute less than the dominant ones, and N is the number of discrete types. During training, we gradually increase the coefficient β to perform an easy-to-hard training by decreasing the smoothness. Empirically, we increase β by 1 for each epoch. We train the mapper for 20 epochs.

REFERENCES

Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. 2019. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

Tero Karras, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 4401–4410.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR).*

Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. 2020. MaskGAN: Towards Diverse and Interactive Facial Image Manipulation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision.* 3730–3738.

Omer Tov, Yuval Alaluf, Yotam Nitzan, Or Patashnik, and Daniel Cohen-Or. 2021. Designing an encoder for stylegan image manipulation. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–14.

Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. 2018. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV).* 325–341.

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

[3]https://azure.microsoft.com/en-us/services/cognitive-services/face