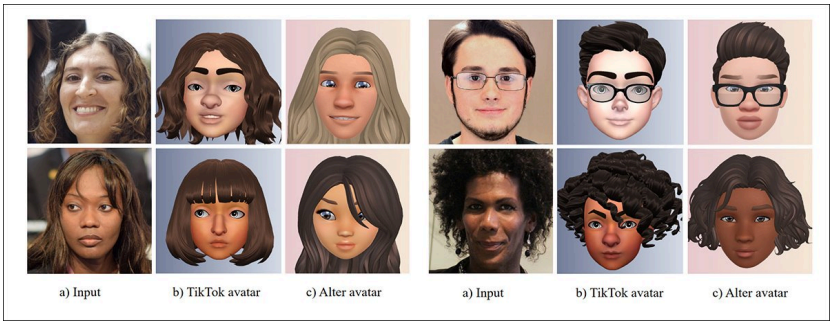
Creating Better Avatars with a Dual-Domain Approach

Originally published at https://arquivo.pt/wayback/20240203191302oe_/https://blog.metaphysic.ai/creating-better-avatars-with-a-dual-domain-approach/

January 20, 2023

A new Chinese academic/industry collaboration has devised a novel method of generating stylized user avatars for metaverse environments – by training on two distinct but parallel networks: one that's capable of reproducing deepfake-style likenesses of the user, and another which re-uses the same latent codes to generate an avatar.

The method, titled SwiftAvatar, uses StyleGAN and a host of other popular image synthesis technologies and libraries to create avatars that more closely resemble users, and which do not require excessive configuration at inference time.



Here, SwiftAvatar is creating representations for two other avatar creation engines – TikTok and Alter. Source: https://arxiv.org/pdf/2301.08153.pdf

In a quantitative evaluation and a human study, comprising 50 volunteers, the new approach was able to score higher than two comparable recent methods, and can also be repurposed for existing avatar creation pipelines.

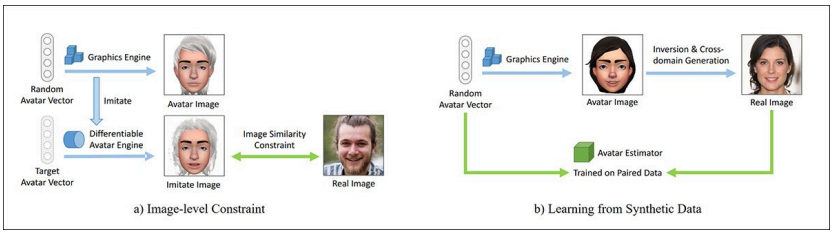
The new paper is titled *SwiftAvatar: Efficient Auto-Creation of Parameterized Stylized Character on Arbitrary Avatar Engines*, and comes from researchers at Beijing University of Posts and Telecommunications, and Douyin Vision (a subsidiary of ByteDance).

A Two-Faced Approach

Avatar creation is of increasing interest in the gaming industry and for the anticipated spread of metaverse environments, where individuals may wish to appear in a stylized or cartoon-like fashion. Though platforms such as ReadyPlayerMe, BitMoji and Zepeto offer avatar-creation pipelines to users, the authors of the new paper describe the experience of these services as *'tiresome and time-consuming'*, and opine that they are beset by excessive options – the combination of which may still not necessarily lead to more accurate avatars.

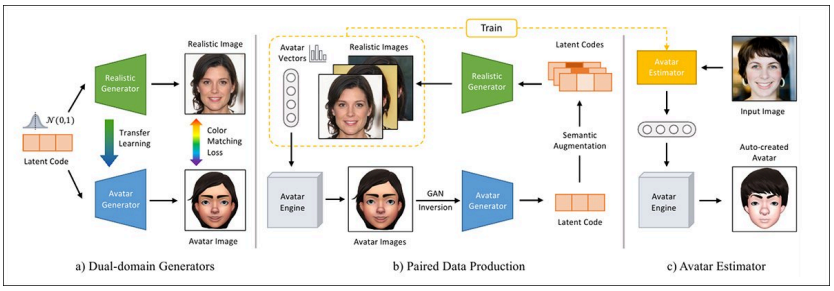
Part of the burden of traditional avatar methods of this type is that the facets are divided into *continuous* (face geometry) and *discrete* (hair, jewelry, etc.) elements, with no facility to simply input a current user image and generate a stylized avatar replete with both types of characteristic.

To boot, the previous methods are essentially *sequential*, in that a graphics engine decomposes the user's supplied face image, which is then passed through to a differentiable avatar engine, which recognizes similarity points and uses these to generate avatars, based on priors that accord with the avatar style of the platform.



Left, the broad schema of prior methods; right, the SwiftAvatar approach.

Instead of attempting to transform and pick apart a real-world image in a linear workflow in this way, SwiftAvatar uses *dual-domain* generators to process the image *twice* – once to obtain a deepfake-style 'real'-looking face, and once to generate a paired avatar image, informed by random priors of avatars in any selected style.



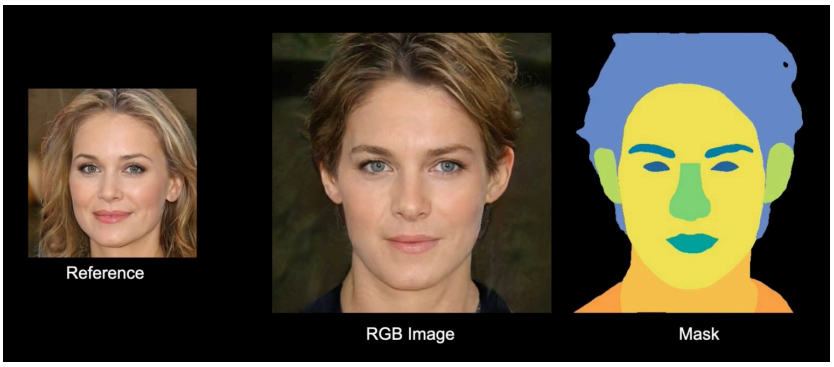
Via the output module, called the 'Avatar Estimator', the latent codes generated by the user images are re-used for both deepfake-style generation and for avatar generation. The codes are passed into a pre-trained avatar vector domain that already contains mappings for stylized facial features, which can in the process be informed by the 'real world' latent codes extracted from the user photo, and generate a user avatar in the 'house style'.

As we can see in the above image, the entire process breaks down into three stages: the dual-domain generator phase, where the input photo is processed through 'realistic' and 'stylized' domains (though the latent vectors are naturally solely obtained from the genuine photo); a training phase, where augmented (i.e., generic avatar) data is used to help the mapping; and the Avatar Estimator, the lightweight inference module that provides output.

The final latent code used for the SwiftAvatar output has, by this time, become a composite representation of the two trained domains, which adapts it to the Avatar Estimator (which is not designed to produce life-like representations).

Method

Besides the native latent code generated by the user image, SwiftAvatar makes use of a SemanticStyleGAN module pretrained on CelebAMask-HQ.

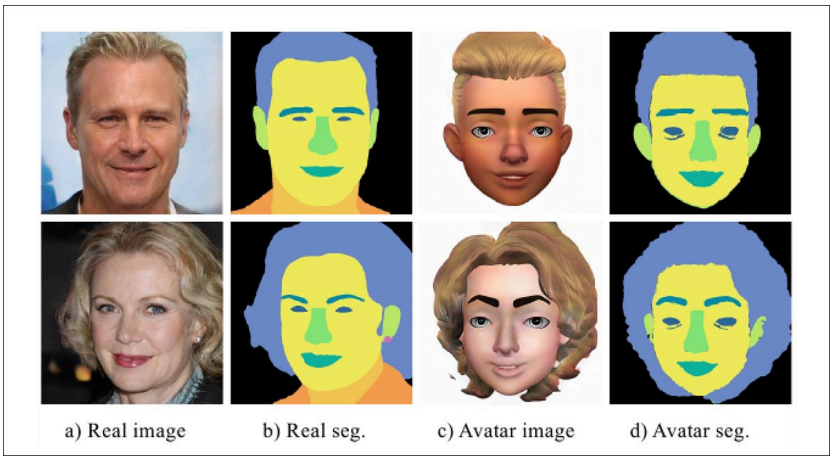


SemanticStyleGAN uses semantic segmentation to aide targeting of reconstruction or simulation of areas of the face, and is used in SwiftAvatar to help map recognized characteristics into the simpler bounds of an avatar-appropriate style. Source: https://semanticstylegan.github.io/

In the new system, SemanticStyleGAN is trained on randomly-sampled data from the avatar vector, and acts as the central architecture for both the realistic and stylized parallel processing workflows, aiding cross-domain consistency.

Further, SemanticStyleGAN is also used to separate out the component parts of the face, and attempt to create a good match between the colors for the realistic face (which is discarded, and only used as input data) and the ultimate avatar that will be created.

This is an important and sensitive aspect, because specious or anomalous lighting in source user images can inadvertently cause a *change in apparent race*, or other critical identity characteristics. Though this problem is not entirely solved by the new method, semantic segmentation (see above image) helps to analyze color differences across the facial region more intelligently, increasing the likelihood of an avatar that reflects the identity of the user.



Thanks to the semantic segmentation capabilities of SemanticStyleGAN, both real and stylized outputs can be quantified into areas of the face, aiding identity-matching as well as color fidelity.

Additionally, the individuation of facial parts provided by SemanticStyleGAN allows SwiftAvatar to perform semantic augmentation on actual *sections* of faces, rather than just entire faces, since each section can be assigned its own latent code. In the training process, random noise is added to the latent codes for these discrete facial parts, enabling per-section transformations – a far cry from the entirely-entangled workflows of prior methods.

During the paired data production part of the pipeline, a number of random stylized avatar images are sampled as labels, and used to generate corresponding images for the user input. Each image is then passed through a Generative Adversarial Network (GAN, and the process of projecting an image into a GAN's latent space is called GAN Inversion), in order to obtain the latent code that will be passed further downstream.

Tests

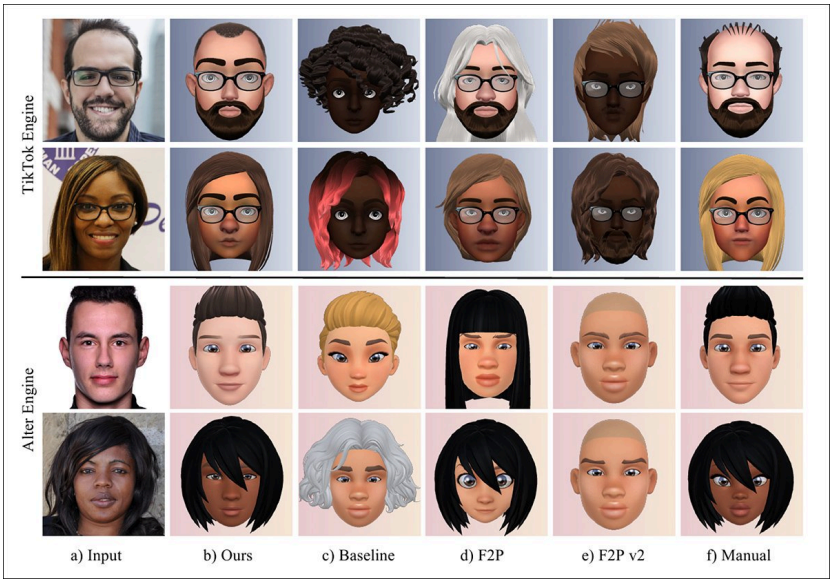
The researchers tested SwiftAvatar against prior methods – the TikTok engine and the Alter engine (acquired by Google in late 2022).

The TikTok engine makes use of the aforementioned discrete and continuous avatar parameters, so that central facial features and character styling are separated. By contrast, the Alter engine only contains discrete parameters. For parity, therefore, the researchers created 50,000 avatar vectors for the former, and 10,000 for the latter, each with corresponding image renders.

The tests were conducted against 116 images from FFHQ (there was no ground truth for these 116 images, since this would have involved commissioning 116 vector-style artworks, and the authors invite such contributions from artists, for future tests).

Tests were implemented over the PyTorch 1.10 library on NVIDIA V100 GPUs. Fine-tuning of the avatar generator followed the original settings of SemanticStyleGAN, at a batch size of 16, with lazy regularization applied every 16 mini-batches for the generator (see the paper and supplementary material for extensive further details of the training regimen).

SwiftAvatar was tested against methods including baseline, Face-to-Parameter (F2P), and F2P V2. The *baseline* method ignores the domain gap issue (the fact that a real face must be created in a non-real, i.e., stylized domain).



A comparison of styles tested. The final column shows avatars created by professional designers.

In a quantitative evaluation round, LPIPS was used as a perceptual metric to compare the results across the systems against avatars created by human designers, with SwiftAvatar notably leading the results table:

Method	TikTok engine ↓	Alter engine ↓	Speed ↑
Baseline	0.5033	0.3962	$\sim 10^2$ Hz
F2P	0.3562	0.3040	~ 1 Hz
F2P v2	0.4466	0.2825	$\sim 10^2$ Hz
Ours	**0.3110**	**0.2405**	$\sim \mathbf{10^2}$ **Hz**

Results from the quantitative evaluation round, with lower numbers better.

For the human evaluation test, fifty volunteers were asked to subjectively consider results from all the graphics engines, and match the most apposite avatar against the real-world source face. Here too, SwiftAvatar took the lead:

Method	Ours	baseline	F2P	F2P v2
TikTok	**59.11**%	5.66%	30.86%	4.37%
Alter	**63.68**%	18.94%	8.65%	8.73%

Human judgment on the various avatar iterations created across the tested systems.

The authors conclude:

'Compared with previous methods, our method is more concise in training stage and more efficient in inference stage. Results on quantitative evaluation and human rating demonstrate the superiority of our method. Also, the success of applying on two different avatar graphics engines demonstrates the generality of our method.'