

# FATE: Full-head Gaussian Avatar with Textural Editing from Monocular Video

## Supplementary Material

This supplementary material provides additional implementation details and experimental results. In Sec. 1, we introduce the preliminaries related to 3DGS and PTI. Sec. 2 describes implementation details regarding datasets, methods, neural baking and head completion. In Sec. 3, we present additional experimental results, including monocular reconstruction, cross-reenactment, more results about full-head completion, and textural editing. Sec. 4 explains the trade-off between texture quality and rendering quality in neural baking. We discuss the failure cases and ethics considerations in Sec. 5 and Sec. 9, respectively. We integrate the performance under imperfect poses, computational efficiency, and additional ablation in Sec. 6, Sec. 7, and Sec. 8, respectively. We highly recommend watching our *supplementary video* for more visual results.

### 1. Preliminary

**3D Gaussian Splatting** 3D Gaussian Splatting [4] is a point-based volume rendering method that models each primitive as a Gaussian kernel, formalized as follows:

$$G(\mathbf{x}) = e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}, \quad (1)$$

where  $\boldsymbol{\mu}$  is Gaussian position and  $\boldsymbol{\Sigma}$  is 3D covariance matrix. To ensure that  $\boldsymbol{\Sigma}$  is positive semi-definite, the covariance matrix is further decomposed into a rotation matrix  $\mathbf{R}$  and a scaling matrix  $\mathbf{S}$ :

$$\boldsymbol{\Sigma} = \mathbf{R} \mathbf{S} \mathbf{S}^T \mathbf{R}^T. \quad (2)$$

In the rendering phase, 3D Gaussians are projected onto the image plane as 2D Gaussians. Zwicker *et al.* [14] derive the following formula to approximate the covariance of the projected 2D Gaussians:

$$\boldsymbol{\Sigma}' = \mathbf{J} \mathbf{W} \boldsymbol{\Sigma} \mathbf{J}^T \mathbf{W}^T, \quad (3)$$

where  $\mathbf{W}$  is viewing transformation and  $\mathbf{J}$  is the Jacobian of the affine approximation of the projective transformation. Volumetric rendering is then performed for each pixel to calculate the final color:

$$\mathbf{C} = \sum_{i \in N} \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \quad (4)$$

where  $\mathbf{c}_i$  is the color of each Gaussian and  $\alpha_i$  represents the density computed by the projected Gaussians with  $\boldsymbol{\Sigma}'$  multiplied by each Gaussian's opacity  $o_i$ .

**Pivotal Tuning Inversion** We introduce the overall PTI [9] optimization pipeline as follows. In the first stage, we search for the pivotal latent code  $w_p$  by minimizing:

$$\arg \min_w \sum_{i=0}^{M-1} \mathcal{L}_{\text{prec}} \left( \mathbf{I}_i^{\mathcal{M}_R}, \mathbf{I}_i^{\mathcal{G}} \right), \quad (5)$$

$$\mathbf{I}_i^{\mathcal{G}} = \mathcal{G}_P(w, c_i; \theta), \quad (6)$$

where  $M$  is the number of valid multi-view images,  $\mathcal{L}_{\text{prec}}$  denotes the perceptual loss [3],  $\mathbf{I}_i^{\mathcal{M}_R}$  is the face image restored by pretrained model  $\mathcal{M}_R$ ,  $\mathcal{G}_P$  is the freezed pretrained generator,  $c$  is the camera pose.

In the second stage, we finetune the generator parameters by minimizing the following loss term:

$$\mathcal{L}_{\text{pt}} = \sum_{i=0}^{M-1} \mathcal{L}_{\text{prec}} \left( \mathbf{I}_i^{\mathcal{M}_R}, \mathbf{I}_i^{\mathcal{G}} \right) + \lambda_{\text{L2}} \mathcal{L}_{\text{L2}} \left( \mathbf{I}_i^{\mathcal{M}_R}, \mathbf{I}_i^{\mathcal{G}} \right), \quad (7)$$

$$\mathbf{I}_i^{\mathcal{G}} = \mathcal{G}_P(w_p, c_i; \theta^*), \quad (8)$$

where  $\theta^*$  is the tuned weights initialized with the pretrained weights  $\theta$ .

### 2. Implementation Details

#### 2.1. Datasets

We used a total of 20 monocular portrait videos for our experiments. For 10 datasets with DECA-based preprocessing, we optimize the DECA-predicted FLAME coefficients during training and testing in line with IMAvatar [13]. For the test-time fine-tuning, we perform FLAME coefficients optimization for 50 epochs. We optimize the FLAME coefficients with a learning rate of  $5 \times 10^{-4}$ . For all datasets, a pre-trained segmentation model [12] is used to remove regions below the neck to facilitate comparison. All methods except MonoGaussianAvatar are trained for 10 epochs on the INSTA and Emotalk3D datasets and 50 epochs on the PointAvatar and NerFace datasets.

#### 2.2. Models

All methods are implemented by PyTorch [8] with differential Gaussian rasterization from 3DGS [4]. And all methods are optimized by Adam [6] optimizer. To model the mouth region, each method incorporates the FLAME template with additional faces to close the mouth cavity, similar to FlashAvatar [11].

**Ours** For our method, the learning rates for color, opacity, scale, rotation, and offset are  $2.5 \times 10^{-3}$ ,  $5.0 \times 10^{-2}$ ,  $5.0 \times 10^{-3}$ ,  $1.0 \times 10^{-3}$ , and  $1.6 \times 10^{-3}$ , respectively. The learning rate for the learnable blendshapes is  $1.0 \times 10^{-5}$ . The opacity of the Gaussians is reset every  $6k$  iterations, and sampling-based densification is performed every  $3k$  iterations by adding  $1k$  Gaussians. Pruning is conducted every  $2k$  iterations based on an opacity threshold of  $5.0 \times 10^{-3}$ .

**FlashAvatar** FlashAvatar maintains a fixed number of Gaussians in the canonical space and utilizes an MLP-based deformer to learn the offset of scale, rotation, and position. And We set the learning rate for the deformer to  $1.0 \times 10^{-4}$ , for color to  $2.5 \times 10^{-3}$ , for opacity to  $5.0 \times 10^{-2}$ , for scale to  $5.0 \times 10^{-3}$ , and for rotation to  $1.0 \times 10^{-3}$ . The deformer has a hidden dimension of 256 and an output dimension of 10. The output channel corresponding to rotation is activated using an exponential function to ensure non-negativity. The scale offset, after being activated by the exponential function, is applied multiplicatively to the original unactivated Gaussian scale. At initialization, we perform uniform UV sampling at a resolution of 128. In addition to the uniform sampling, we apply additional random sampling, resulting in a total of  $16k$  Gaussians.

**GaussianAvatars** GaussianAvatars was originally designed for multi-view video datasets with accurate 3D mesh, whereas the preprocessing pipeline for monocular videos cannot obtain such precise geometry prior. Due to its specific binding mechanism, we set the learning rate for scale to  $1.7 \times 10^{-2}$ . Densification starts after  $10k$  iterations and is performed every  $2k$  iterations thereafter. The densification gradient threshold is  $1.0 \times 10^{-4}$ , and Gaussians are pruned with a minimum opacity threshold of  $5.0 \times 10^{-3}$ .

**MonoGaussianAvatar** MonoGaussianAvatar employs a series of MLPs to model geometry, deformation, and Gaussian attributes. The design of the MLPs follows the original implementation, with a learning rate of  $1.0 \times 10^{-4}$ . Densification of Gaussians is performed on an epoch-based scheduler, and the scheduler for the number of Gaussians added during densification remains consistent with the original paper. We perform densification every 5 epochs. Due to the slow convergence of MonoGaussianAvatar, we train each subject for 100 epochs.

**SplattingAvatar** SplattingAvatar constructs Gaussians that walk on triangles using UV coordinates. We set the learning rate for UV coordinates (and the normal offset  $d$ ) to  $1.6 \times 10^{-4}$ , while the learning rates for other attributes remain consistent with the original Gaussian configuration. Opacity is reset every  $3.5k$  iterations, and the walking triangle operation is performed every 100 iterations. The densification gradient threshold is set to  $2.0 \times 10^{-4}$ , and the minimum opacity for pruning is  $5.0 \times 10^{-3}$ . During initialization, we sampled  $10k$  Gaussian points.

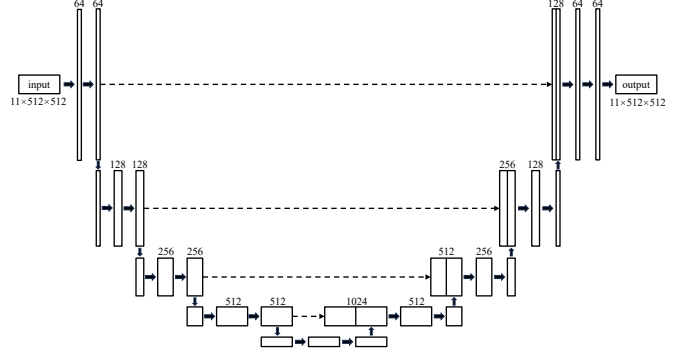


Figure 1. **BakeNet Architecture.** We adopt a U-Net architecture as the backbone of BakeNet, leveraging its ability to construct representations across various frequency bands from noise.

### 2.3. Neural Baking

We use a simple U-Net [10] as shown in Fig. 1 for BakeNet, with an input of an 11-channel noise map sampled from a Gaussian distribution, each channel having a size of 512. The first convolutional layer increases the number of channels to 64, and the encoder of the U-Net processes the channels up to 1024, doubling the number of channels at each layer. The decoder then reduces the number of channels back to 64, and the final convolutional layer adjusts the output channels to 11. Skip connections are used between the encoder and decoder.

The 11 channels represent the following: 3 channels for scale, 3 channels for rotation, 3 channels for color, 1 channel for opacity, and 1 channel for offset. Specifically, we use 3 channels to represent the rotation in axis-angle form.

Similar to GGHead [7], we apply a special normalization to the upsampled values from the output map corresponding to scale. We calculate the mean and maximum values of the unactivated scale for the avatar to be baked. Then, the sampled values  $v$  are processed as follows:

$$s = s_{max} - \log(1 + \exp(-(v + s_{mean}) + s_{max})), \quad (9)$$

where  $s_{mean}$  and  $s_{max}$  represent the mean and maximum values of the unactivated scale, respectively.

During training, we set the learning rate to  $1.0 \times 10^{-3}$  and use the Adam optimizer to optimize the U-Net.

### 2.4. Head Completion

We first render around the trained avatar for 30 frames. On average, DLib [5] deems 2 to 5 images valid. During PTI [9], we optimize the latent code for 200 iterations and fine-tune the generator parameters for 200 iterations.

We found that due to the FFHQ alignment used by SphereHead, the inversion often results in incomplete heads (see Fig. 2). This leads to the disappearance of some edge

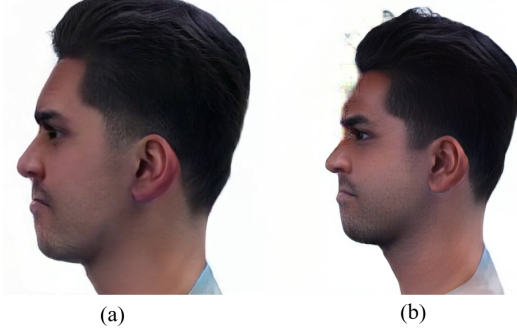


Figure 2. **Incomplete Inversion Issues.** In typical inversion optimization, the neck and top of the head of the portrait often fall outside the frame, as shown in (a). We obtained the result shown in (b) by adjusting the camera-to-object distance.

regions when used for completion. Since SphereHead assumes fixed camera intrinsics during training,

$$K = \begin{bmatrix} 4.2627 & 0 & 0.5 \\ 0 & 4.2627 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}, \quad (10)$$

directly modifying the camera intrinsics leads to poor out-of-domain results. We found a compromise by slightly increasing the camera radius from 2.7 to 3.2 while equivalently transforming the coordinates for the inverse transformation estimation to ensure the portrait appears within the viewing frustum.

After PTI is completed, we render 30 images in a full circle as pseudo-data. One potential issue is that the PTI results still differ from the real subject, and the coordinates of the monocular avatar and 3D-aware GAN are difficult to align. Therefore, we only used the latter half of the 30 images and incorporated random backgrounds during training to eliminate some artifacts.

### 3. Additional Results

#### 3.1. Monocular Results

We provide the quantitative results for each subject in Tab. 5 and Tab. 6, and more qualitative results are presented in Fig. 7. Our method demonstrates superior performance across multiple datasets.

Other methods, such as FlashAvatar, achieve excellent LPIPS scores on the INSTA dataset but perform poorly on the PointAvatar dataset, which contains complex poses and expressions. We attribute this to the deformation MLP in FlashAvatar overfitting the training set. In contrast, our method mitigates this tendency by employing a linear approach to implement personalized blendshapes, leading to better generalization.

MonoGaussianAvatar also utilizes personalized blendshapes. However, its Gaussian scales are computed through

the MLP, which prefers smoothness. This smooth nature produces blurred outputs, leading to relatively high PSNR and SSIM scores but poorer LPIPS performance.

#### 3.2. Full-head Completion Results

We provide additional results of full-head completion in Fig. 8. Since monocular videos lack supervision for side and back views, novel views at large angles tend to perform poorly before completion. After applying the completion framework, plausible rendering results are achieved across most angles. Furthermore, we extend the completion framework to other methods. As shown in Fig. 9, these methods also yield reasonable results after applying the completion framework.

We observed that for methods allowing free movement of Gaussians (*e.g.*, GaussianAvatars, SplattingAvatar), misalignment artifacts are more severe. This is because the overly flexible Gaussians overfit to misaligned views. However, these methods still achieve relatively satisfactory completion results.

#### 3.3. Cross-reenactment Results

We present the results of cross-reenactment in Fig. 10. We achieve face reenactment by transferring the expression and pose of the driving avatar to different subjects. Under monocular video settings, the shape parameters and expression are not well decoupled. To achieve effective transfer, we need to compute the delta of the expression between the driving avatar and the target avatar when both exhibit a neutral expression.

#### 3.4. Editing Results

More textural editing results are shown in Fig. 11. In sticker editing, we manually craft stickers with simple patterns and corresponding masks, applying them directly to the color texture map. In style transfer, we use off-shelf and classic style transfer models [2] to transfer the texture map. Since we do not employ non-zero-order spherical harmonic coefficients, the results are inherently multi-view consistent after editing. Compared to methods that require pre-trained models and optimize inconsistent editing results, direct editing on the texture map is a faster and easy-to-use approach.

### 4. Neural Baking Trade-off

As reported in the main content and Tab. 5, 6, neural baking causes certain metric degradation compared to the avatars optimized in a point-wise manner. We found that this is because convolutional neural networks (CNN) struggle to fit the complex distribution of Gaussian geometry (scale, rotation, and offset) in the UV space. Several experiments are designed to illustrate this observation.

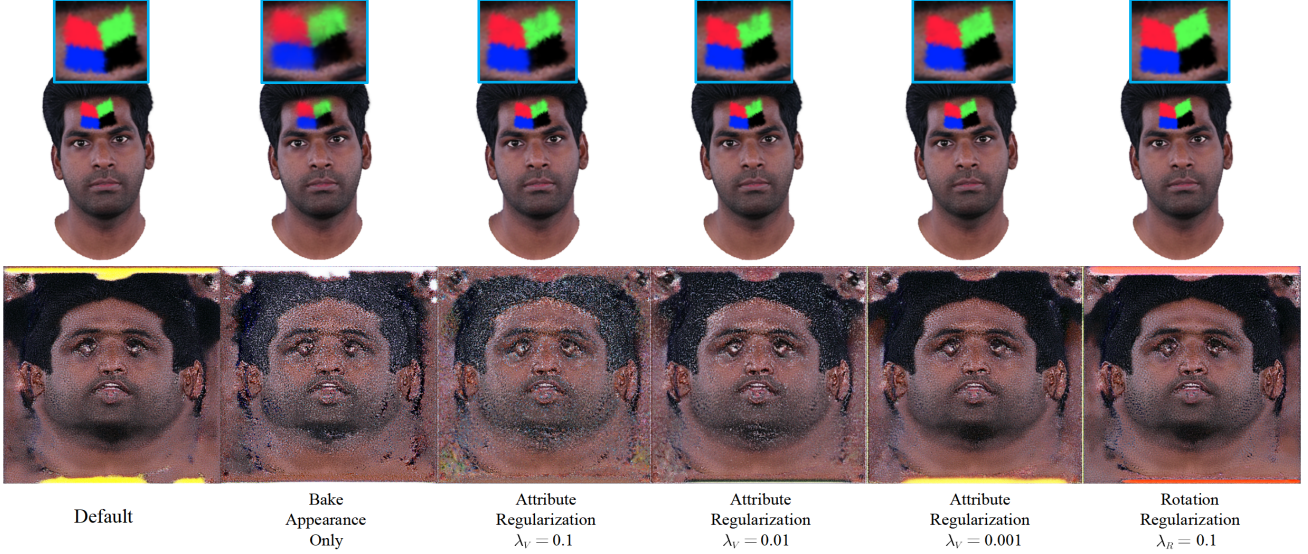


Figure 3. **Neural Baking Trade-off.** We visualize the color texture maps produced by neural baking under different settings and the results after editing with a checking sticker.

**Bake Appearance Only** We only use neural baking to obtain texture maps for color and opacity, while the scale, rotation, and offset are retained from the pre-trained avatar.

**Attribute Regularization** We minimize the difference between the attributes sampled from the BakeNet output and the corresponding attributes of the pre-trained avatar:

$$\mathcal{L}_V = \|v_* - \bar{v}_*\|_2, \quad (11)$$

where  $v$  denotes sampled values and  $*$  refers to Gaussian attributes. We add this regularization term to the baking training objective with a strength of  $\lambda_V$ .

**Rotation Regularization** We impose a regularization term on the sampled rotation. Since our rotation is relative to the local triangle, we enforce the rotation around its  $x$ -axis and  $y$ -axis to be close to 0. This encourages the Gaussian rotation around the face’s normal direction:

$$\mathcal{L}_R = \|r_x\|_2 + \|r_y\|_2, \quad (12)$$

where  $r_x$  and  $r_y$  are rotations in axis-angle representation.

Quantitative and qualitative results are shown in Tab. 1 and Fig. 3. When we bake only the color and opacity while retaining the pre-trained Gaussian geometric attributes, the LPIPS metric improves. However, it leads to noisy texture maps and blurry edited stickers. A straightforward idea is to make the baked attributes approximate the pre-trained ones. We conduct experiments under three levels of  $\lambda_V$ , but the results show that the metrics are still decreased even at the cost of degrading the texture maps. This suggests that CNN struggles to fit the complex geometric distribution of Gaussian attributes in UV space. We believe this is because the

Table 1. The quantitative results of the neural baking trade-off in *bala* case. **blue** indicate the best.

	PSNR↑	SSIM↑	LPIPS↓
Default	<b>29.27</b>	0.9278	0.0584
Bake App. Only	28.76	<b>0.9298</b>	<b>0.0522</b>
Attribute Regu. $\lambda_V = 0.1$	29.13	0.9239	0.0582
Attribute Regu. $\lambda_V = 0.01$	29.18	0.9264	0.0583
Attribute Regu. $\lambda_V = 0.001$	29.18	0.9268	0.0581
Rotation Regu. $\lambda_R = 0.1$	29.22	0.9272	0.0592

attributes describing the Gaussian geometry lack local similarity, making them ill-suited for learning with CNN. Additionally, we introduce a rotation regularization term during baking, which worsens LPIPS but improves the quality of the texture maps and editing effects.

These experiments demonstrate that we can flexibly balance rendering quality and texture map quality in practice. If better rendering quality is desired, we can opt not to bake the geometric attributes of Gaussians. Conversely, if smoother texture maps or better editing effects are desired, applying regularization terms, such as rotation regularization, can make the Gaussians more isotropic and closer to the surface, thereby resulting in smoother texture maps.

## 5. Failure Case and Limitation

Our neural baking and full-head completion still have limitations. As mentioned in Sec. 4, since CNN is tricky to construct Gaussian geometry, neural baking may fail for intricate geometry. For instance, in the case of the woman



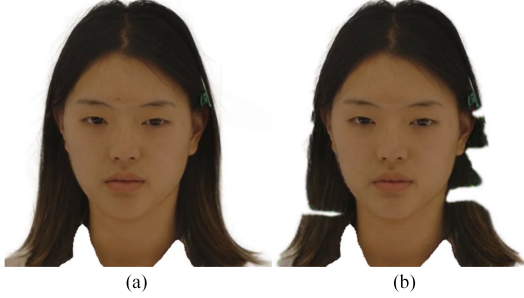


Figure 4. **Neural Baking Failure.** For long hair subjects, as in (a), direct neural baking will damage the fine geometry of the Gaussians composing the hair as in (b).



Figure 5. **Full-head Completion Failure.** Since the PTI results still differ from the real avatar, artifacts appear at the junction, as shown in the red box in (a). And for avatars with almost no side view in the training data, as shown in (b), it is difficult to estimate the exact geometry during PTI, leading to the identity change in the side view.

with long hair shown in Fig. 4, the hair requires Gaussians with delicate scale and rotation. However, neural baking makes it difficult to recover the desired geometry.

In full-head completion, we are training the unobserved view with pseudo images and the frontal view with real images. Artifacts, as shown in Fig. 5 (a), may appear in a certain side-view angle due to the transition between the two regions. Additionally, for subjects that have almost no side view in monocular videos (e.g., Internet video focusing on talking), PTI does not estimate the head with the correct geometry, resulting in identity change.

## 6. Noisy Pose Simulation

To train head avatars from monocular videos, we require frame-by-frame RGB images along with the corresponding tracked coefficients. We further evaluate the differences between our method and GaussianAvatars when the camera translation is imperfect. We add Gaussian noise with varying  $\sigma$  to camera translations to simulate real-captured data with inaccurate tracking. Fig. 6 shows our method is more robust than GA to such conditions. We attribute this to the regularization of the UV embedding, which constrains the Gaussians from freely moving to a blurred average solution.

Table 2. Running Time on Optional Parts.

	Mono. Recon.	Pseudo Gen.	Completion	Neural Baking
Time	$\sim 1.0h$	$\sim 10min$	$\sim 15min$	$\sim 0.5h/1.0h$

## 7. Computational Efficiency

In Tab. 3, we supplement the training time and rendering FPS under identical hardware conditions. Our method outperforms other UV space-based methods (FA, SA) regarding shorter training time and higher FPS. Compared to GA, our method achieves comparable efficiency with superior rendering quality.

We also measure the average running time of each part in our proposed method on the INSTA dataset. We just fine-tune for 1 epoch during completion and 5 epochs during baking, with training times ranging from 0.5 to 1 hour, depending on whether only the frontal face or the entire head is baked. The average running time is shown in Tab. 2

## 8. More Ablations

As shown in Tab. 4, we introduce more ablation settings on two representative datasets and further report the number of Gaussian in different settings. We additionally conduct experiments as *w/o densify\**, where \* indicates that Gaussians are removed based on opacity criteria. The suboptimal results further highlight the effectiveness of sampling-based densification. Moreover, we align our method with densification strategies based on SA and GA. GA-based densification tends to produce blurrier results, while SA-based densification introduces too many redundant Gaussians.

And we supplement more experiments comparing *Two-stage* and *One-stage*. Concretely, we find baking only the appearance (denoted as *Two-stage baking App.*) improves rendering quality compared to *Two-stage baking* but causes blurred editing effects, which is visualized and discussed in Sec. 4 of our supplementary material. Besides, we additionally report GS numbers in Tab. 4 to show that *Two-stage baking* leveraging the evolved distribution achieves comparable performance with much fewer Gaussians than *One-stage baking* that uses initialized uniform distribution.

## 9. Ethics

We used four subjects from EmoTalk3D [1], with all participants signing the consent for using their videos in this research and publication. Data from consenting subjects will be made publicly available. Our method generates realistic and animatable head avatars, enabling the creation of videos of real people performing synthetic poses and expressions. We strictly oppose any misuse of this work to create deceptive content intended to spread misinformation or damage reputations.

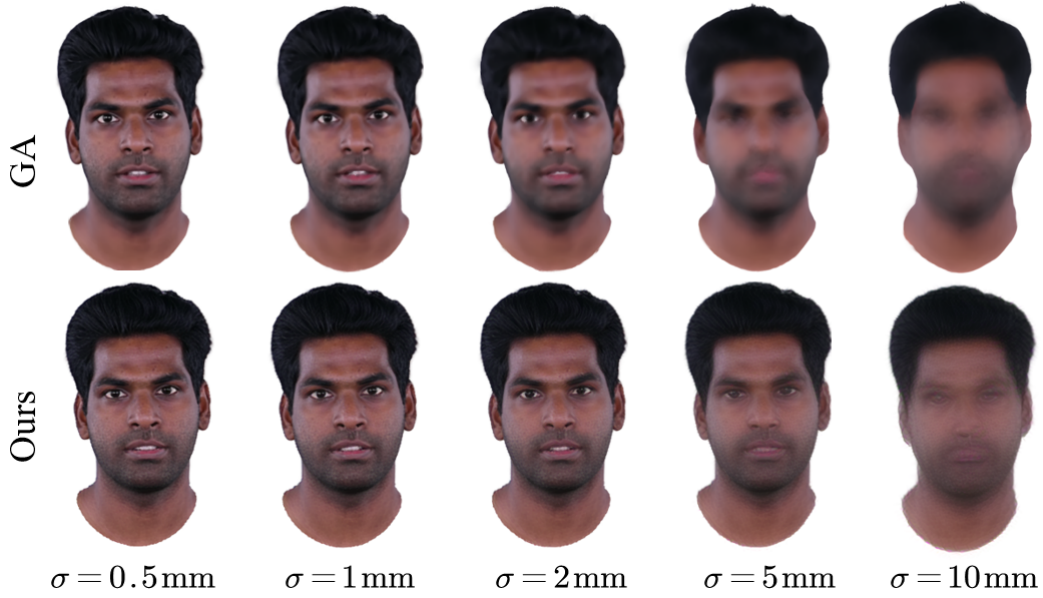


Figure 6. **Robustness to Imperfect Poses** We add noise to camera translation to simulate less well-processed datasets. Note that 1 mm in the figure approximately corresponds to 1 cm in the real world.

Table 3. Evaluation on Computational Efficiency.

Datasets	INSTA Dataset			IMAvatar Dataset			NerFace Dataset			EmoTalk3D Dataset		
	GS num.	Training time	FPS	GS num.	Training time	FPS	GS num.	Training time	FPS	GS num.	Training time	FPS
FA	16k	1.4h	190	16k	4.5h	208	16k	5.5h	206	16k	0.5h	214
SA	558k±188k	1.2h	106	617k±274k	8.0h	109	497k±142k	9.6h	112	489k±171k	1.0h	116
MGA	100k	7.2h	16	100k	13h	16	100k	14h	16	100k	7.5h	16
GA	72k±33k	0.4h	212	38±14k	3.1h	228	31k±7k	4.2h	223	55±12k	0.8h	229
Ours	49k±6k	1.0h	203	38k±6k	3.7h	216	42k±0.5k	4.5h	215	58k±2k	0.5h	216

Table 4. Ablation Study in *yufeng* and *bala*.

	yufeng				bala			
	PSNR↑	SSIM↑	LPIPS↓	GS num.	PSNR↑	SSIM↑	LPIPS↓	GS num.
Ours	29.36	0.9239	0.0694	30k	29.23	0.9329	0.0507	54k
w/o densify*	28.81	0.9195	0.0799	14k	28.83	0.9309	0.0526	45k
w/o densify	29.13	0.9217	0.0740	65k	28.91	0.9311	0.0528	65k
w/o $\Delta\mathcal{E}$ and $\Delta\mathcal{P}$	24.78	0.8820	0.1112	33k	24.46	0.9015	0.1081	54k
w/ GA densify	29.62	0.9327	0.0941	80k	26.89	0.9270	0.0966	118k
w/ SA densify	27.74	0.8699	0.1896	803k	26.37	0.8417	0.1827	917k
Two-stage baking	27.78	0.9104	0.0979	30k	29.27	0.9278	0.0584	54k
Two-stage baking App.	28.84	0.9190	0.0797	30k	29.53	0.9298	0.0522	54k
One-stage baking	27.42	0.9085	0.1088	65k	29.12	0.9208	0.0602	65k
Decode only	25.56	0.8878	0.1506	30k	28.25	0.9071	0.0827	54k

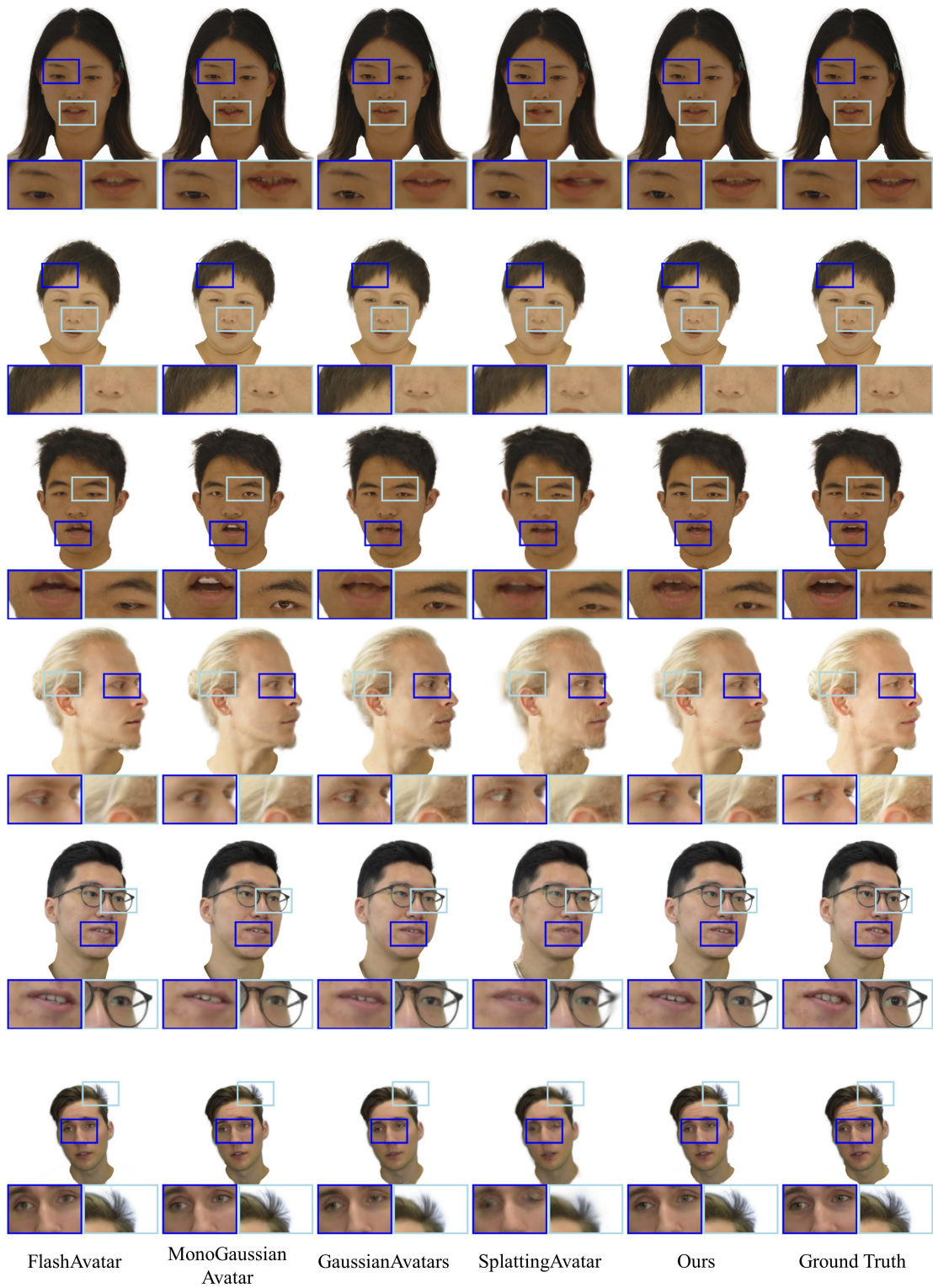


Figure 7. **More Reconstructed Results.** Our method excels at capturing fine structures and preserving high-frequency details (*e.g.*, eyebrows, hair strands, eyeglass frames, and pupil colors.).





Figure 8. **More Full-head Completion Results.** *Odd* rows display the results under novel views without applying the Full-head completion framework, while *even* rows show the results after completion. Our completion framework significantly enhances rendering quality under large viewing angles.



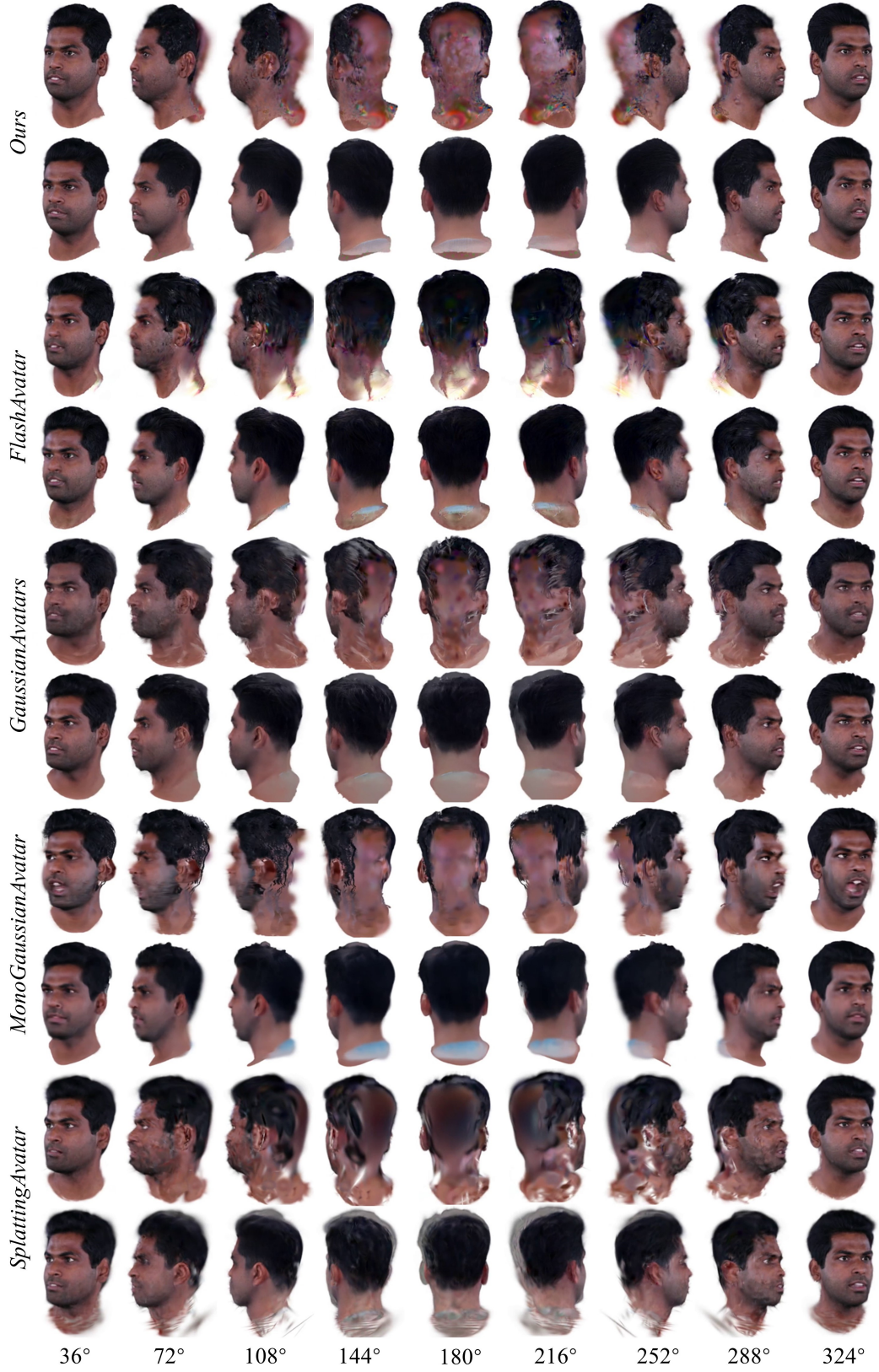


Figure 9. **Universal Completion Results.** *Odd* rows display the results under novel views without applying the Full-head completion framework, while *even* rows show the results after completion. Our completion framework applies to various monocular reconstruction methods.

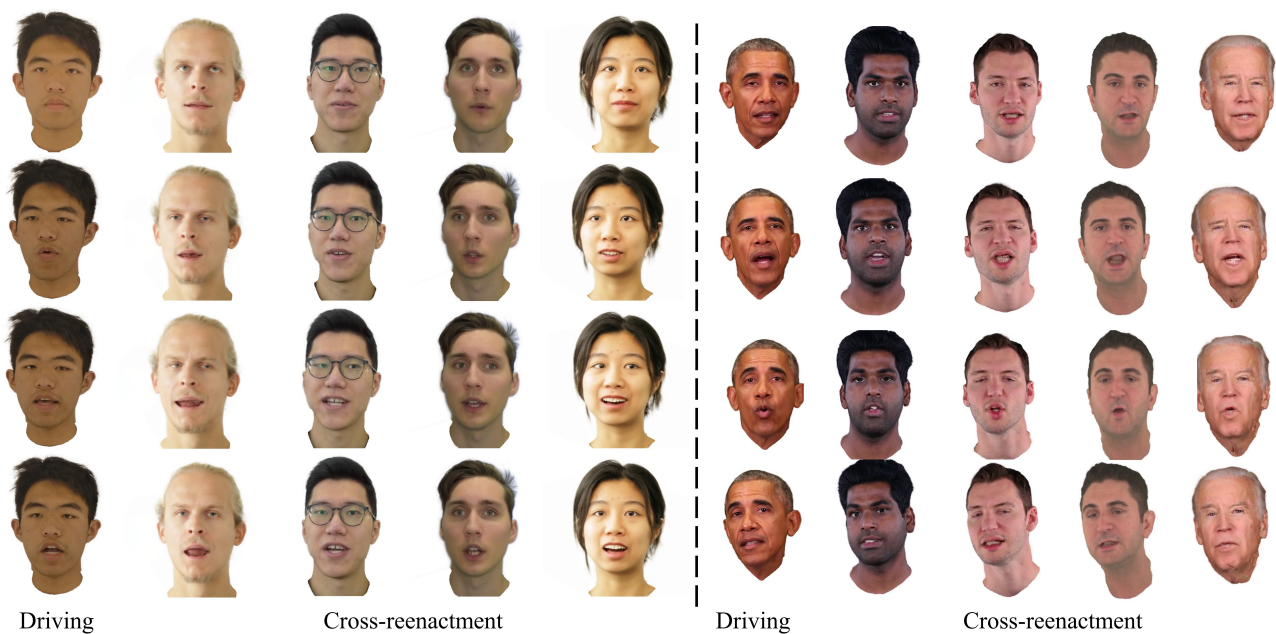


Figure 10. **Cross-reenactment Results.** We use the expression and pose sequences from the driving source to animate different subjects, enabling the transfer of dynamic facial expressions and poses across various avatars.

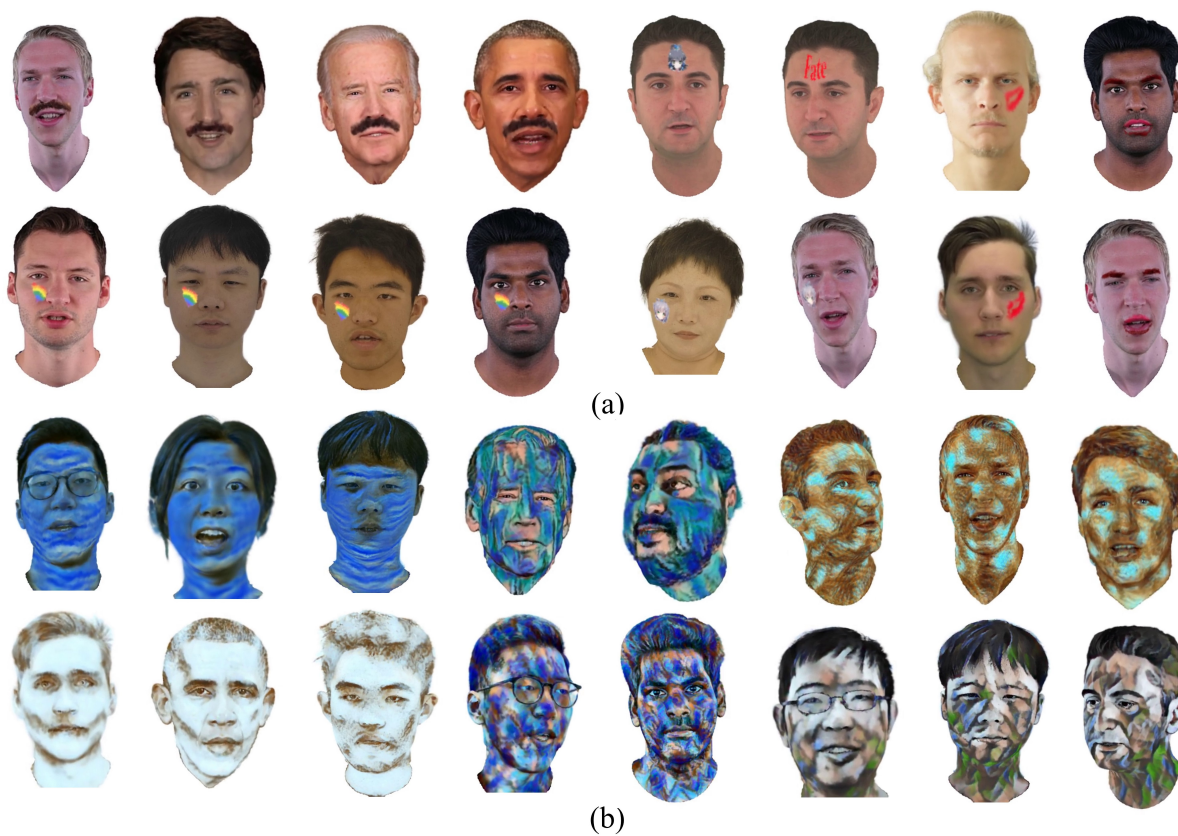


Figure 11. **Editing Results.** In (a), we show several results of directly editing the texture map by adding stickers, such as anime portraits, rainbows, kisses, mustaches, and logos. In (b), we present the results of applying style transfer to the texture map.



Table 5. Full comparison of quantitative results with state-of-the-art methods on INSTA dataset. **blue** and **lightblue** indicate the 1st and 2nd best.

Datasets		INSTA Dataset									
		bala	biden	justin	malte_1	marcel	nf_01	nf_03	obama	person4	wojtek_1
PSNR↑	FlashAvatar	28.04	29.63	27.42	27.54	29.32	24.47	27.24	26.92	21.65	30.66
	SplattingAvatar	27.58	28.64	26.43	28.01	28.05	24.11	26.02	25.46	20.84	31.23
	MonoGaussianAvatar	28.95	29.97	27.28	27.93	28.84	24.58	27.22	27.39	21.98	29.90
	GaussianAvatars	27.99	28.52	26.70	27.53	29.10	23.80	26.00	25.12	20.58	31.27
	Ours	28.17	30.06	27.11	28.40	29.07	24.69	27.46	27.04	22.01	31.25
	Ours (Baked)	29.42	30.37	27.75	28.34	28.38	24.91	27.43	28.01	21.92	31.49
SSIM↑	FlashAvatar	0.9170	0.9595	0.9580	0.9310	0.9328	0.9303	0.9227	0.9413	0.9009	0.9530
	SplattingAvatar	0.9168	0.9501	0.9553	0.9352	0.9265	0.9201	0.9137	0.9341	0.8959	0.9563
	MonoGaussianAvatar	0.9297	0.9599	0.9601	0.9340	0.9316	0.9320	0.9200	0.9482	0.9090	0.9480
	GaussianAvatars	0.9397	0.9548	0.9609	0.9401	0.9394	0.9286	0.9257	0.9398	0.9032	0.9635
	Ours	0.9267	0.9659	0.9606	0.9434	0.9371	0.9344	0.9253	0.9524	0.9097	0.9600
	Ours (Baked)	0.9285	0.9672	0.9638	0.9424	0.9356	0.9347	0.9231	0.9547	0.9086	0.9602
LPIPS↓	FlashAvatar	0.0484	0.0267	0.0410	0.0418	0.0757	0.0768	0.0601	0.0375	0.1403	0.0296
	SplattingAvatar	0.1284	0.0694	0.0902	0.0838	0.1330	0.1385	0.1286	0.0820	0.1992	0.0659
	MonoGaussianAvatar	0.0969	0.0547	0.0585	0.0758	0.1352	0.1146	0.0980	0.0532	0.1471	0.0534
	GaussianAvatars	0.0618	0.0458	0.0616	0.0607	0.0943	0.1025	0.0827	0.0606	0.1626	0.0445
	Ours	0.0534	0.0341	0.0445	0.0414	0.0760	0.0811	0.0674	0.0426	0.1298	0.0324
	Ours (Baked)	0.0583	0.0371	0.0448	0.0451	0.0841	0.0859	0.0755	0.0422	0.1312	0.0346

Table 6. Full comparison of quantitative results with state-of-the-art methods on the PointAvatar dataset, NerFace dataset, and Emotalk3D dataset. **blue** and **lightblue** indicate the 1st and 2nd best.

Datasets		PointAvatar Dataset			NerFace Dataset			Emotalk3D Dataset			
		yufeng	marcel	soubhik	person1	person2	person3	subject1	subject2	subject3	subject4
PSNR↑	FlashAvatar	25.85	26.67	26.84	30.44	31.47	32.24	25.28	29.96	21.41	25.28
	SplattingAvatar	25.09	25.57	23.61	27.35	28.58	32.10	25.16	28.23	20.98	23.81
	MonoGaussianAvatar	28.50	27.01	28.98	32.09	33.89	35.35	25.31	30.22	21.46	24.74
	GaussianAvatars	25.12	25.15	23.27	27.03	28.62	31.54	25.12	27.71	20.77	23.10
	Ours	29.36	27.58	29.28	32.91	33.65	34.54	26.13	30.96	21.56	26.36
	Ours (Baked)	27.78	26.37	28.21	31.99	33.34	32.45	-	29.70	21.73	26.96
SSIM↑	FlashAvatar	0.8863	0.9224	0.9221	0.9620	0.9734	0.9569	0.9218	0.9411	0.9087	0.9037
	SplattingAvatar	0.8761	0.9056	0.8903	0.9416	0.9516	0.9509	0.9250	0.9350	0.9164	0.9020
	MonoGaussianAvatar	0.9259	0.9374	0.9448	0.9719	0.9812	0.9763	0.9434	0.9453	0.9126	0.8827
	GaussianAvatars	0.8938	0.9200	0.9095	0.9504	0.9613	0.9560	0.9394	0.9381	0.9200	0.9028
	Ours	0.9239	0.9341	0.9418	0.9716	0.9802	0.9691	0.9429	0.9530	0.9208	0.9265
	Ours (Baked)	0.9104	0.9282	0.9330	0.9655	0.9761	0.9579	-	0.9505	0.9199	0.9274
LPIPS↓	FlashAvatar	0.1043	0.1021	0.0607	0.0377	0.0217	0.0317	0.0668	0.0435	0.0827	0.0787
	SplattingAvatar	0.1502	0.1510	0.1490	0.0835	0.0660	0.0640	0.1403	0.0877	0.1109	0.1484
	MonoGaussianAvatar	0.0993	0.1280	0.0656	0.0473	0.0224	0.0249	0.0758	0.0546	0.0770	0.0921
	GaussianAvatars	0.1287	0.1321	0.1163	0.0681	0.0415	0.0432	0.0806	0.0590	0.0912	0.1035
	Ours	0.0694	0.0876	0.0586	0.0329	0.0186	0.0256	0.0705	0.0414	0.0843	0.0842
	Ours (Baked)	0.0979	0.1142	0.0740	0.0460	0.0240	0.0418	-	0.0593	0.0923	0.0952

## References

- [1] Qianyun He, Xinya Ji, Yicheng Gong, Yuanxun Lu, Zhengyu Diao, Linjia Huang, Yao Yao, Siyu Zhu, Zhan Ma, Songchen Xu, Xiaofei Wu, Zixiao Zhang, Xun Cao, and Hao Zhu. Emotalk3d: High-fidelity free-view synthesis of emotional 3d talking head. In *ECCV*, 2024. [5](#)
- [2] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. [3](#)
- [3] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, pages 694–711, 2016. [1](#)
- [4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM TOG*, 42(4), 2023. [1](#)
- [5] Davis E. King. Dlib - a toolkit for machine learning and computer vision, 2009. [2](#)
- [6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. [1](#)
- [7] Tobias Kirschstein, Simon Giebenhain, Jiapeng Tang, Markos Georgopoulos, and Matthias Nießner. Gghead: Fast and generalizable 3d gaussian heads. *ACM SIGGRAPH Asia*, 2024. [2](#)
- [8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*. Curran Associates, Inc., 2019. [1](#)
- [9] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. *ACM TOG*, 2021. [1](#), [2](#)
- [10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. [2](#)
- [11] Jun Xiang, Xuan Gao, Yudong Guo, and Juyong Zhang. Flashavatar: High-fidelity head avatar with efficient gaussian embedding. In *CVPR*, 2024. [1](#)
- [12] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *ECCV*, page 334–349. Springer-Verlag, 2018. [1](#)
- [13] Yufeng Zheng, Victoria Fernández Abrevaya, Marcel C. Bühler, Xu Chen, Michael J. Black, and Otmar Hilliges. I M Avatar: Implicit morphable head avatars from videos. In *CVPR*, 2022. [1](#)
- [14] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Ewa splatting. *IEEE TVCG*, 8(3):223–238, 2002. [1](#)