# Supplementary Material: Generative Multiplane Images: Making a 2D GAN 3D-Aware

This supplementary is structured as follows:

- Sec. A: Details of the differentiable rendering in GMPI;
- Sec. B: Additional quantitative results;
- Sec. C: Implementation details;
- Sec. D: More qualitative results.

## A Differentiable Rendering in GMPI

In Sec. 3.3, we obtain the desired image $I_{v_{\text{tgt}}}$ which illustrates the generated MPI representation $\mathcal{M} = \{C, \{\alpha_1, \ldots, \alpha_L\}\}$ from the user-specified target view $v_{\text{tgt}}$ in two steps: 1) a warping step transforms the representation $\mathcal{M}$ from its canonical pose $v_{\text{cano}}$ to the target pose $v_{\text{tgt}}$; 2) a compositing step combines the planes into the desired image $I_{v_{\text{tgt}}}$. Importantly, both steps entail easy computations which are end-to-end differentiable such that they can be included into any generator. Here we provide details.

**Warping.** We warp the RGB image and the alpha map of the $i^{\text{th}}$ plane from the canonical view to the target view via

$$(C'_i, \alpha'_i) = \mathcal{H}_{i, v_{\text{cano}} \to v_{\text{tgt}}}(C, \alpha_i, d_i). \tag{S1}$$

Here, $\mathcal{H}_{i, v_{\text{cano}} \to v_{\text{tgt}}}$ represents the homography operation. Essentially, the homography $\mathcal{H}_{i, v_{\text{cano}} \to v_{\text{tgt}}}$ specifies a mapping: for each pixel coordinate $(p'_x, p'_y)$ in the image $C'_i$ and in the alpha map $\alpha'_i$ of the target view $v_{\text{tgt}}$, we obtain corresponding coordinates $(p_x, p_y)$ in the image $C$ and in the alpha map $\alpha_i$ of the canonical view $v_{\text{cano}}$. Bilinear sampling is applied on $(p_x, p_y)$ to obtain values for the pixel locations $(p'_x, p'_y)$. Concretely,

$$\begin{bmatrix} p_x & p_y & 1 \end{bmatrix}^\top = K_{v_{\text{cano}}} \left( R_{v_{\text{tgt}} \to v_{\text{cano}}} - \frac{\boldsymbol{t}_{v_{\text{tgt}} \to v_{\text{cano}}} \boldsymbol{n}^\top}{b_i} \right) K_{v_{\text{tgt}}}^{-1} \begin{bmatrix} p'_x & p'_y & 1 \end{bmatrix}^\top, \tag{S2}$$

where $\boldsymbol{n} \in \mathbb{R}^3$ is the normal of the plane defined in target camera coordinate system $v_{\text{tgt}}$, which is identical for all planes. $b_i$ is the depth of the plane from the target camera $v_{\text{tgt}}$. We let $K_{v_{\text{cano}}} \in \mathbb{R}^{3\times3}$ and $K_{v_{\text{tgt}}} \in \mathbb{R}^{3\times3}$ refer to the intrinsic matrices of the canonical view $v_{\text{cano}}$ and the target view $v_{\text{tgt}}$. Further, $R_{v_{\text{tgt}} \to v_{\text{cano}}} \in \mathbb{R}^{3\times3}$ and $\boldsymbol{t}_{v_{\text{tgt}} \to v_{\text{cano}}} \in \mathbb{R}^{3\times1}$ are the rotation and the translation from $v_{\text{tgt}}$ to $v_{\text{cano}}$.

**Alpha Compositing.** Given the warped image $C'_i$ and the warped alpha map $\alpha'_i$ for each plane $i$, we compute the final rendered 2D image $I_{v_{\text{tgt}}}$ via

$$I_{v_{\text{tgt}}} = \sum_{i=1}^{L} \left( C'_i \cdot \alpha'_i \cdot \prod_{j=1}^{i-1} (1 - \alpha'_j) \right). \tag{S3}$$

Table S1: **Comparing representations on FFHQ.** 1st and 3rd rows are copied from Tab. 2's 3rd and 9th rows in the main text. Row 2 and 3 infer with 96 planes. Row 1: depth to renderer; Row 2: depth to MPI to renderer; Row 3: MPI to renderer.

| | Row | | FID↓ | KID×100 ↓ | ID↑ | Depth↓ | Pose↓ |
|---|---|---|---|---|---|---|---|
| | 1 | LiftedGAN [58] | 29.8 | – | 0.58 | **0.40** | 0.023 |
| $256^2$ | 2-1 | D2A ($\epsilon = 1/64$) | 13.4 | 0.920 | 0.69 | 0.60 | **0.004** |
| | 2-2 | D2A ($\epsilon = 1/128$) | 13.5 | 0.867 | **0.70** | 0.60 | **0.004** |
| | 2-3 | D2A ($\epsilon = 1/256$) | 11.7 | **0.644** | **0.70** | 0.63 | 0.005 |
| | 2-4 | D2A ($\epsilon = 1/512$) | 12.6 | 0.684 | 0.69 | 0.62 | 0.005 |
| | 3 | **GMPI** | **11.4** | 0.738 | **0.70** | 0.53 | **0.004** |

Similarly, we approximate depth $D_{v_{\text{tgt}}}$ via

$$D_{v_{\text{tgt}}} = \sum_{i=1}^{L} \left( b_i \cdot \alpha'_i \cdot \prod_{j=1}^{i-1} (1 - \alpha'_j) \right), \tag{S4}$$

where $b_i$ is the distance mentioned in Eq. (S2). Notably, the combination of Eq. (4), Eq. (S1) and Eq. (S3) is end-to-end differentiable and hence straightforward to integrate into a generator. Importantly, the computations are also extremely efficient as only simple matrix multiplications are involved. It is hence easy to augment an existing generator like the one in StyleGANv2.

## B Additional Quantitative Results

### B.1 Comparisons of Representations

We provide additional ablations of the plane representation. Specifically, we consider the following three ablations:

**Depth map to renderer:** We let the generator predict a single channel depth map (instead of multiple-channel alpha maps) and use a differentiable renderer to supervise the transformed image. LiftedGAN [61] serves as this ablation.

**Depth maps to alpha maps to renderer:** We compute multiple alpha maps from a predicted single-channel depth map in two steps: 1) we predict a depth map $\texttt{Depth} \in \mathbb{R}^{H \times W}$ in normalized coordinates; 2) we generate $L$ alpha maps from Depth by computing the alpha value for a pixel $[x, y]$ on the $i$-th alpha map $\alpha_i$ via:

$$\alpha_i[x, y] = \min \left( 1, \max \left( 0, \frac{d_i - (\texttt{Depth}[x, y] - \epsilon)}{2\epsilon} \right) \right),$$

where $d_i$ is $\alpha_i$'s depth. Essentially, alpha values linearly increase from 0 to 1 within the range $[\texttt{Depth}[x, y] - \epsilon, \texttt{Depth}[x, y] + \epsilon]$. Any depth closer than $\texttt{Depth}[x, y] - \epsilon$ is set to 0. Any depth further than $\texttt{Depth}[x, y] + \epsilon$ is set to 1. We call this representation D2A.

Table S2: **Ablation studies on truncation level $\psi$ during inference on FFHQ.** All results use the same checkpoint trained with DPC, plane-specific features $\mathcal{F}^h_{\alpha_i}$, and shading-guided training. We evaluate with 96 planes. We show more digits for 'Depth' and 'Pose' than we use in Tab. 2 to emphasize differences. With lower $\psi$, GMPI produces geometries with fewer artifacts (Depth) at the cost of less variety (FID/KID).

| | $\psi$ | $256^2$ | | | | | $512^2$ | | | | | $1024^2$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | FID↓ | KID↓ | ID↑ | Depth↓ | Pose↓ | FID↓ | KID↓ | ID↑ | Depth↓ | Pose↓ | FID↓ | KID↓ | ID↑ | Depth↓ | Pose↓ |
| (a) | 1.0 | **11.4** | **0.738** | **0.70** | 0.533 | 0.0042 | **8.29** | **0.454** | **0.74** | 0.455 | 0.0056 | **7.50** | **0.407** | **0.75** | 0.535 | 0.0068 |
| (b) | 0.9 | 12.6 | 0.837 | 0.70 | 0.515 | 0.0037 | 10.1 | 0.588 | 0.74 | 0.421 | 0.0047 | 8.83 | 0.496 | 0.75 | 0.490 | 0.0058 |
| (c) | 0.8 | 15.4 | 1.079 | 0.70 | 0.497 | 0.0033 | 13.7 | 0.885 | 0.73 | 0.386 | 0.0039 | 11.5 | 0.691 | 0.74 | 0.449 | 0.0049 |
| (d) | 0.7 | 20.3 | 1.510 | 0.70 | 0.477 | 0.0029 | 19.8 | 1.391 | 0.73 | 0.354 | 0.0032 | 16.0 | 1.024 | 0.73 | 0.408 | 0.0040 |
| (e) | 0.6 | 27.8 | 2.172 | 0.70 | 0.459 | 0.0026 | 28.8 | 2.160 | 0.73 | 0.326 | 0.0025 | 22.8 | 1.510 | 0.73 | 0.368 | 0.0031 |
| (f) | 0.5 | 38.5 | 3.112 | 0.70 | **0.445** | **0.0023** | 41.1 | 3.249 | 0.72 | **0.304** | **0.0020** | 32.0 | 2.145 | 0.72 | **0.333** | **0.0024** |

**Alpha maps to renderer:** We directly predict multiple-channel alpha maps. Our GMPI serves as this ablation.

In Tab. S1 we report the results. For D2A, we ablate several values of $\epsilon$ and find that GMPI always outperforms on FID and depth metrics, verifying the fidelity of th texture and the high-quality of the geometry.

### B.2 Depth Score Analysis

We inspect GMPI-synthesized images and find two main reasons for our suboptimal 'Depth' scores in Tab. 2: 1) artifacts produced by StyleGANv2; 2) specular reflections on images deteriorate geometry generation. We discuss both in detail below.

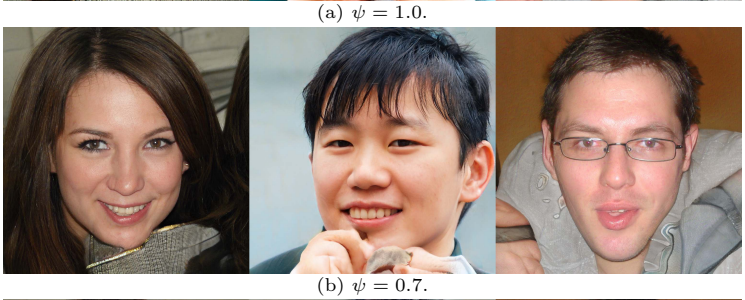
**1) Artifacts in StyleGANv2.** Truncation was introduced in [34, 7] to balance between variety and fidelity. Specifically, using a truncation level $\psi \in [0, 1]$, we replace the style embedding $\boldsymbol{\omega}$ in Eq. (3) with

$$\boldsymbol{\omega}' = \bar{\boldsymbol{\omega}} + \psi \cdot (\boldsymbol{\omega} - \bar{\boldsymbol{\omega}}), \tag{S5}$$

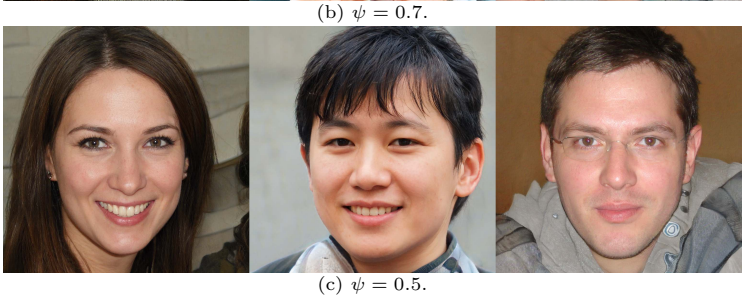
where $\bar{\boldsymbol{\omega}} = \mathbb{E}_{\boldsymbol{z}}[f_{\texttt{Mapping}}(\boldsymbol{z})]$ represents the style embedding space's center of mass. In practice, $\bar{\boldsymbol{\omega}}$ is approximated by computing the moving average of all $\boldsymbol{\omega}$ encountered during training. Without truncation, *i.e.*, for $\psi = 1.0$, we find StyleGANv2 can produce results with significant artifacts as shown in Fig. S1a. This demonstrates that the generator fails to convert the corresponding $\boldsymbol{\omega}$ properly. Although these artifacts are not being reflected in our 2D GAN metrics, *i.e.*, FID and KID, they do affect the alpha map generation adversely. Specifically, the alpha maps are produced based on feature $\mathcal{F}^h_{\alpha_i}$, which is largely determined by the style embedding $\boldsymbol{\omega}$ through Eq. (3) and Eq. (8). Consequently, artifacts cause an inferior 'Depth' score. To understand the effect of the truncation level $\psi$ during inference, we evaluate GMPI using various truncation levels $\psi$ in Tab. S2. As can be seen clearly, with smaller $\psi$, we consistently perform better on geometry metrics, *i.e.*, lower 'Depth' and 'Pose' error, while trading in variety, *i.e.*, higher FID/KID. We provide a qualitative example in Fig. S2. Note, we do not apply truncation for any other quantitative results reported in this paper. *I.e.*, we always use $\psi = 1.0$ for quantitative evaluation results except in Tab. S2.

**2) Specular reflections.** Our rendering model is not designed to handle specular reflections. Strong specular reflections in the training data tend to degrade geometry generation, producing artifacts such as concave foreheads. We provide a qualitative example in Fig. S3. Specular reflections are a common failure mode of geometry generation. For instance, see Sec. 5 and Fig. 8 in StyleSDF [55]. We leave addressing of this issue to future work.

(a) $\psi = 1.0$.

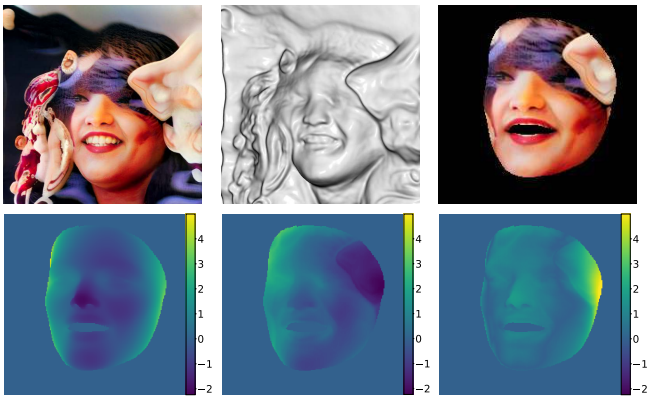


(b) $\psi = 0.7$.

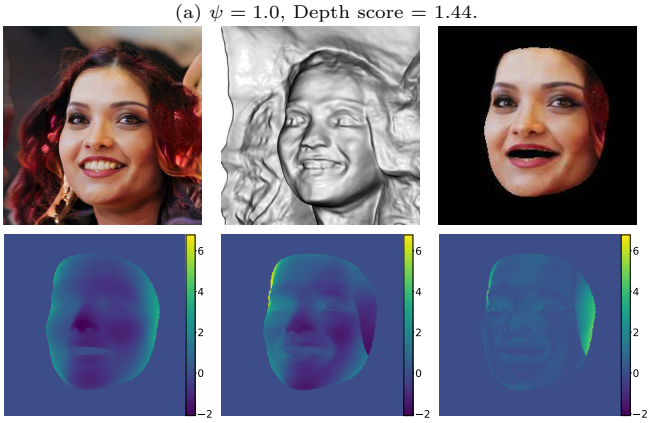


(c) $\psi = 0.5$.

Fig. S1: **Effects of truncation level $\psi$ for StyleGANv2.** Images are generated with the officially-released code and checkpoints.[6] Without truncation, *i.e.*, for $\psi = 1.0$, StyleGANv2 generates images with significant artifacts. This degrades alpha map generation. The images can be generated by running the command `python generate.py --outdir=out --trunc=1.0 --seeds=10,56,88 --network=https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pretrained/ffhq.pkl` while setting `--trunc` to 1.0, 0.7, and 0.5 respectively.



(a) $\psi = 1.0$, Depth score = 1.44.



(b) $\psi = 0.5$, Depth score = 0.83. The large difference in depth prediction appearing on the cheek is primarily due to hair. The parametric facial model does not reconstruct hair and accessories such as glasses (see Fig. 2 in [17]).

Fig. S2: **Truncation level $\psi$ affects geometry generation.** GMPI generated scenes with the same latent variable $\boldsymbol{z}$ using two different truncation levels. Each scene is rendered with the same pose where the 'Depth' score is computed. For each subplot, from top to bottom, left to right, we show 1) rendered image; 2) corresponding geometry; 3) predicted face model [17]; 4) normalized pseudo ground-truth depth map; 5) normalized GMPI generated depth map (the smaller value indicates closer distance to the camera); and 6) difference between normalized depth maps 4) and 5). Smaller truncation levels $\psi$ benefit 'Depth' scores at the cost of increased FID/KID values.

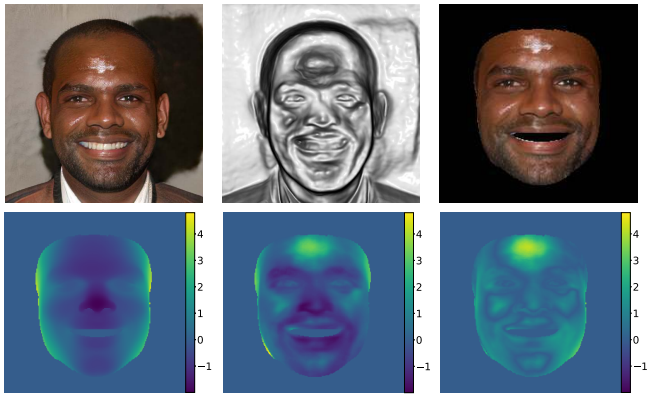


Fig. S3: **Specular reflection deteriorates geometry generation.** For this example, the depth score is 1.26. The strong lighting effect on the forehead breaks the Lambertian assumption and degrades alpha map generation.

Table S3: **Ablation studies on #planes during training.** No truncation is applied. We evaluate at a resolution of $512^2$. `DPC` refers to discriminator pose conditioning (Sec. 3.4), $\mathcal{F}^h_{\alpha_i}$ refers to the plane-specific feature introduced in Eq. (8), and `Shading` indicates the shading-guided training discussed in Sec. 3.5. `#planes` denotes the number of planes we used during training. During inference, we use 32 planes. We observe, the more planes we provide during training, the better the results of GMPI. Please see Fig. S5 for qualitative examples.

| | `#planes` | `DPC` | $\mathcal{F}^h_{\alpha_i}$ | `Shading` | FFHQ | | | | | AFHQv2-Cat | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | FID↓ | KID↓ | ID↑ | Depth↓ | Pose↓ | FID↓ | KID↓ |
| (a) | 32 | ✓ | ✓ | ✓ | **7.40** | **0.337** | **0.74** | **0.457** | **0.006** | 7.93 | 0.489 |
| (b) | 16 | ✓ | ✓ | ✓ | 9.83 | 0.575 | 0.74 | 0.574 | 0.007 | **6.82** | **0.358** |
| (c) | 8 | ✓ | ✓ | ✓ | 11.4 | 0.657 | 0.72 | 0.778 | 0.008 | 7.26 | 0.384 |
| (d) | 4 | ✓ | ✓ | ✓ | 16.4 | 1.043 | 0.65 | 0.992 | 0.007 | 8.53 | 0.467 |

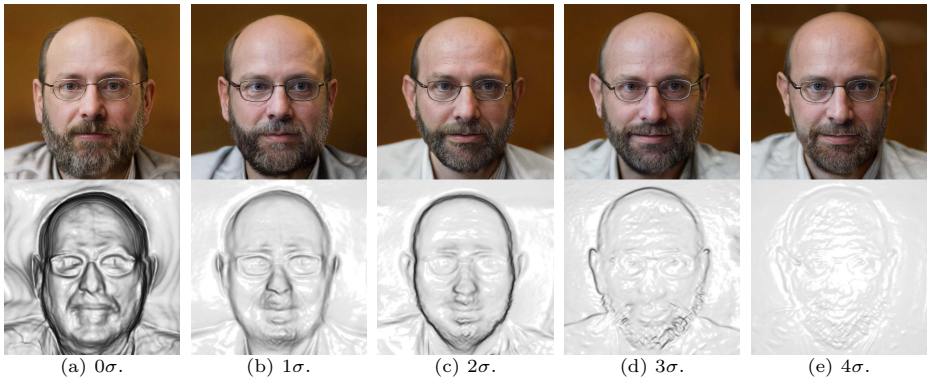
### B.3 More Ablations

**#Planes During Training.** We ablate the `#planes` during training in Tab. S3 and Fig. S5. We observe: the more planes we can provide during training, the better the results of GMPI.

**Robustness to inaccurate camera poses.** To understand the robustness to inaccurate camera poses, we add noise to the observer's camera pose. Specifically, we follow [9] to first compute per-element standard deviation $\sigma$ of the estimated camera pose matrices for real images. During training, we add noise of $\{1\sigma, 2\sigma, 3\sigma, 4\sigma\}$ to each element of the camera pose matrix. We report results in Tab. S4 and Fig. S4. We want to emphasize, the estimated camera poses for real images are not perfect. Therefore, $0\sigma$ does not indicate fully-accurate camera pose information. Fig. S4 verifies that the more noise we add, the less photo-realistic the geometry we obtain. This is aligned with the trend of the Depth

Table S4: **Ablation study on camera pose accuracy.** No truncation is applied. We evaluate at a resolution of $512^2$. `Noise` indicates the level of noise we added to the camera poses during training. Specifically, $1\sigma$ denotes camera poses are corrupted with one standard deviation of the estimated camera pose matrices for images from the corresponding dataset (see Sec. B.3). All results are trained with `DPC`, plane-specific features $\mathcal{F}^h_{\alpha_i}$, and shading-guided training with 32 planes during training.

| | Noise | FFHQ | | | | | AFHQv2-Cat | |
|---|---|---|---|---|---|---|---|---|
| | | FID↓ | KID↓ | ID↑ | Depth↓ | Pose↓ | FID↓ | KID↓ |
| (a) | $0\sigma$ | 8.29 | 0.454 | 0.74 | 0.46 | 0.006 | 7.79 | 0.474 |
| (b) | $1\sigma$ | 6.02 | 0.269 | 0.81 | 0.81 | 0.017 | 5.95 | 0.296 |
| (c) | $2\sigma$ | 5.38 | 0.221 | 0.86 | 1.00 | 0.027 | 5.51 | 0.266 |
| (d) | $3\sigma$ | 5.26 | 0.191 | 0.89 | 1.37 | 0.038 | 7.05 | 0.397 |
| (e) | $4\sigma$ | 4.72 | 0.176 | 0.90 | 1.68 | 0.046 | 6.92 | 0.393 |



(a) $0\sigma$. (b) $1\sigma$. (c) $2\sigma$. (d) $3\sigma$. (e) $4\sigma$.

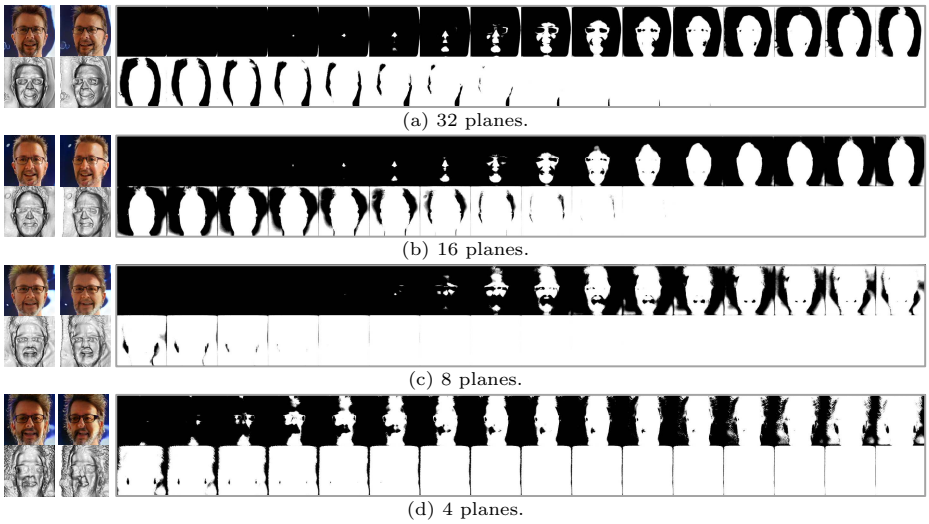
Fig. S4: **Ablate robustness to inaccurate camera poses.** Panels (a)-(e) correspond to Tab. S4's rows (a)-(e). All results are generated from the same latent code $\boldsymbol{z}$. Within expectation, the more inaccurate camera poses the model is trained with, the less 3D geometry we can obtain.

and Pose metrics in Tab. S4. Note, FID, KID, and ID metrics are misleading as we do not observe much difference. This verifies again that 2D metrics lack the ability to capture 3D errors.

## C Implementation Details

**Hyperparameters.** Tab. S5 summarizes the hyperparameters we used for training GMPI. We perform all training on 8 Tesla V100 GPUs using 5000 iterations. For all experiments except FFHQ256, we use a batch size of 32. This exposes the discriminator to 0.16 million real images. FFHQ256 is trained on 0.32 million real images since it uses a larger batch size of 64. We apply the minibatch

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



(a) 32 planes.

(b) 16 planes.

(c) 8 planes.

(d) 4 planes.

Fig. S5: **Ablate #planes during training.** For each subplot, from left to right, we show two views and the 32 alpha maps that GMPI produces during inference. Each subplot's caption indicates the number of planes used during training. We observe: the more planes we can provide during training, the better the results of GMPI.

standard deviation layer [31] independently on each image. We apply $x$-flip data augmentation on AFHQv2 and MetFaces. To determine the channel number $\texttt{dim}_h$ of $\mathcal{F}^h$ (Eq. (2)) and $\mathcal{F}^h_{\alpha_i}$ (Eq. (8)), we follow [32]'s official implementation[7] using:

$$\texttt{dim}_h = \min(\frac{2^{15}}{h}, 512), \tag{S6}$$

where $2^{15}$ is the channel base number. The only exception is FFHQ256, where we use a channel base number of $2^{14}$ following the official implementation. All experiments utilize R1 penalties [46] with weight 10.0 and learning rate $2 \times 10^{-3}$. We use Adam [36] as the optimizer. In all experiments, we use 32 planes during training.

**Mixed precision.** We follow StyleGANv2's official code and use half precision for both generator and discriminator layers corresponding to the four highest resolutions.

**Near/far depths.** We set the near and far depths of MPI for each dataset as 0.95/1.12 (FFHQ and MetFaces), 2.55/2.8 (AFHQv2-Cat).

**Depth normalization.** The depth $d_i$ $\forall i$ is normalized to the range $[0, 1]$ before being used in the embedding function $f_{\texttt{Embed}}$ in Eq. (8). Specifically, we use $d'_i = \frac{d_i - d_1}{d_L - d_1}$.

**Shading-guided training.** As mentioned in Sec. 3.5, we utilize Eq. (10) to apply shading. For the first 1000 iterations, we set $k_a = 1.0$ and $k_d = 0.0$. We

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

Table S5: **Hyperparameters used for training GMPI.**

| | FFHQ256 | FFHQ512 | FFHQ1024 | AFHQv2 | MetFaces |
|---|---|---|---|---|---|
| Resolution | $256^2$ | $512^2$ | $1024^2$ | $512^2$ | $1024^2$ |
| #GPUs | 8 | 8 | 8 | 8 | 8 |
| Training length (iters) | 5k | 5k | 5k | 5k | 5k |
| Training length (#imgs) | 0.32M | 0.16M | 0.16M | 0.16M | 0.16M |
| Batch size | 64 | 32 | 32 | 32 | 32 |
| Minibatch stddev [31] | 1 | 1 | 1 | 1 | 1 |
| Dataset $x$-flips | ✗ | ✗ | ✗ | ✓ | ✓ |
| Channel base | $\frac{1}{2}\times$ | $1\times$ | $1\times$ | $1\times$ | $1\times$ |
| Learning rate ($\times 10^{-3}$) | 2 | 2 | 2 | 2 | 2 |
| R1 penalty weight [46] | 10 | 10 | 10 | 10 | 10 |
| Mixed-precision | ✓ | ✓ | ✓ | ✓ | ✓ |

linearly reduce $k_a$ to 0.9 and linearly increase $k_d$ to 0.1 during iteration 1001 to 2000. Starting from the $2001^{\text{st}}$ iteration, we fix $k_a = 0.9$ and $k_d = 0.1$. Meanwhile, lighting direction $\boldsymbol{l}$ is randomly sampled. We follow [56] to use $l_h \sim \mathcal{N}(0.0, 0.2)$ and $l_v \sim \mathcal{N}(0.2, 0.05)$, where $l_h$ and $l_v$ represent horizontal and vertical angles for lighting directions respectively.

**Structure of $f_{\texttt{Embed}}$ (Eq. (8)).** For each resolution $h \times h$, we utilize three modulated-convolutional layers [35] to construct $f_{\texttt{Embed}}$, whose channel numbers are $\texttt{dim}_{h/4}$, $\texttt{dim}_{h/2}$, $\texttt{dim}_h$ respectively.

**Alternative representation of $f_{\texttt{Embed}}$ (Eq. (8)).** Besides conditioning $f_{\texttt{Embed}}$ on specific depth $d_i$, we also studied use of a learnable token. Concretely, $f_{\texttt{Embed}}$ was represented using a tensor of shape $h \times h \times \texttt{dim}_h$. This resulted in reasonable geometry but we find conditioning of alpha maps on depth to provide better results. We leave further exploration of learnable tokens to future work.

**Background plane.** We treat the last plane of the MPI-like representation, *i.e.*, the $L^{\text{th}}$ plane, as the background plane. To color this plane, we treat the left-most and right-most 5% pixels of the synthesized image $C$ as the left and right boundary. RGB values of all remaining pixels on this plane are linearly interpolated between the left and the right boundary. We find this simple procedure to work well in our experiments.

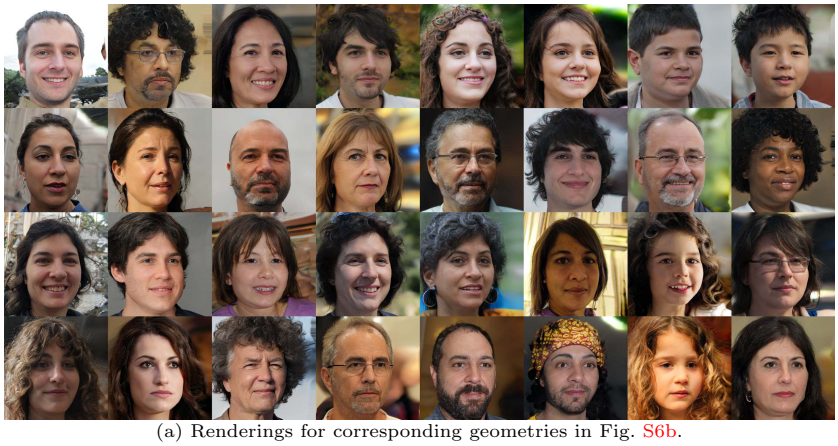
**Mesh generation.** We utilize the marching cube algorithm [45] implemented in PyMCubes for generating meshes.[8] We also utilize its smoothing function for better visualization.

# D Additional Qualitative Results

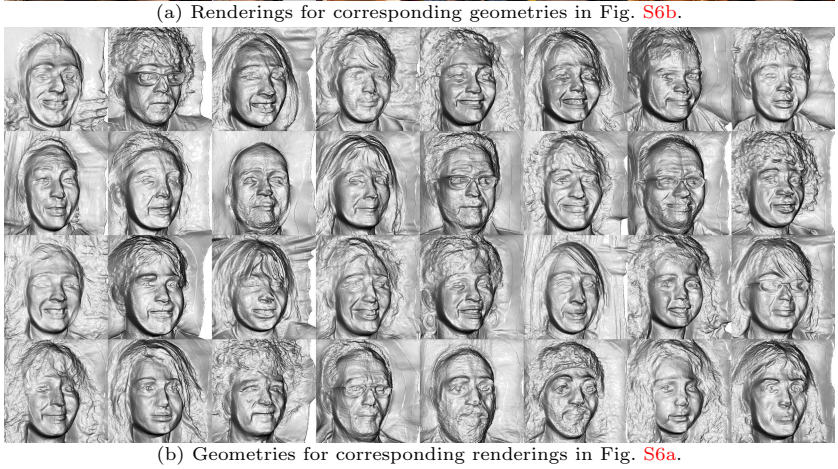
## D.1 Uncurated Results

We provide uncurated results on FFHQ (Fig. S6), AFHQv2 (Fig. S7), and MetFaces (Fig. S8). We observe GMPI to generate high-quality geometry.

[8] https://github.com/pmneila/PyMCubes



(a) Renderings for corresponding geometries in Fig. S6b.



(b) Geometries for corresponding renderings in Fig. S6a.

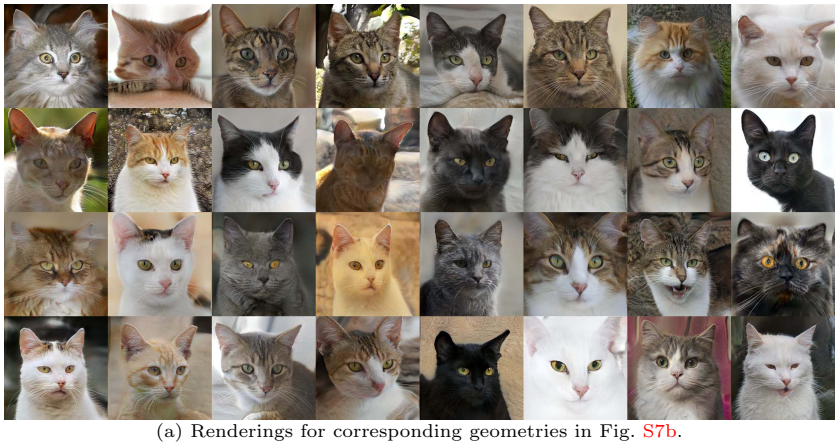
Fig. S6: **Uncurated results on FFHQ.** From top to bottom, left to right, we show generations with seed 1-32. Results are generated with truncation $\psi = 0.5$ [34].

### D.2 Style Mixing

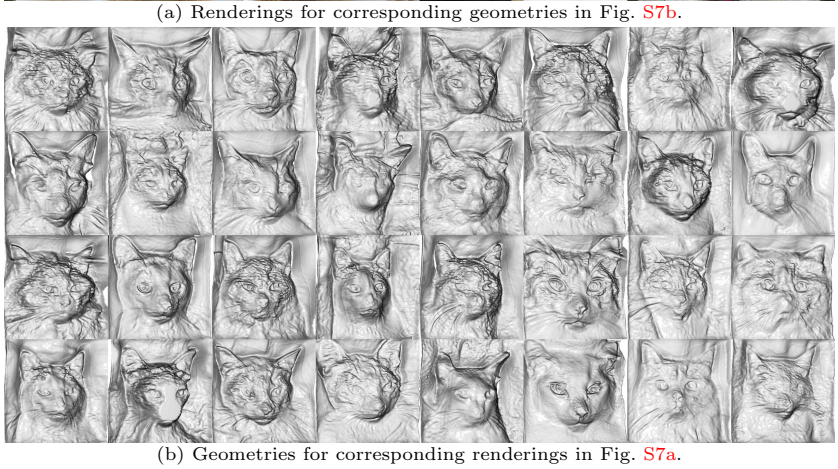
We illustrate style mixing [34] results on FFHQ (Fig. S9), AFHQv2 (Fig. S10), and MetFaces (Fig. S11). GMPI successfully disentangles coarse and fine levels of generations.

### D.3 More Results

This supplementary material also includes an interactive viewer for the generated MPI representations and an HTML page with videos to illustrate generations from GMPI.



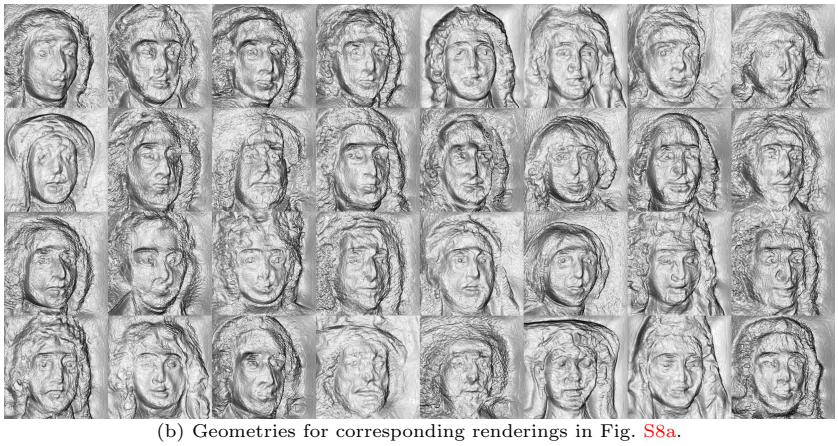
(a) Renderings for corresponding geometries in Fig. S7b.



(b) Geometries for corresponding renderings in Fig. S7a.

Fig. S7: **Uncurated results on AFHQv2.** From top to bottom, left to right, we show generations with seed 1-32. Results are generated with truncation $\psi = 0.7$ [34].

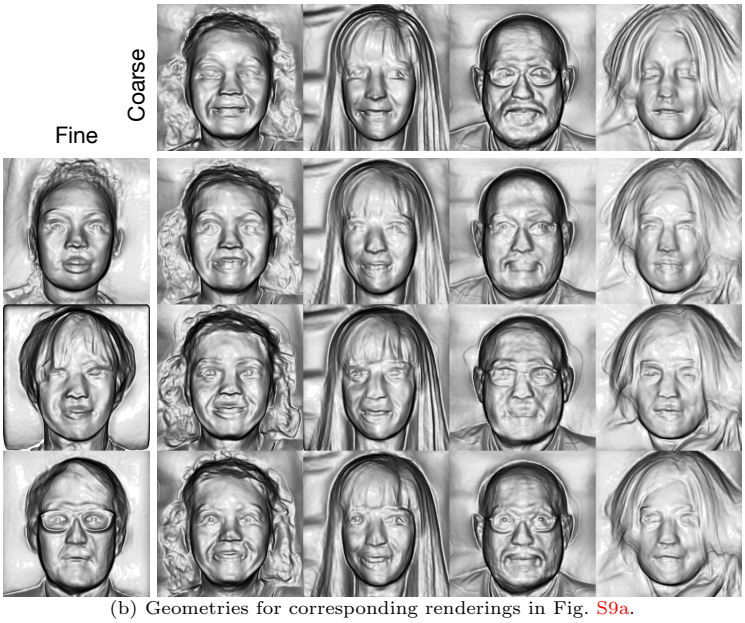
(a) Renderings for corresponding geometries in Fig. S8b.



(b) Geometries for corresponding renderings in Fig. S8a.

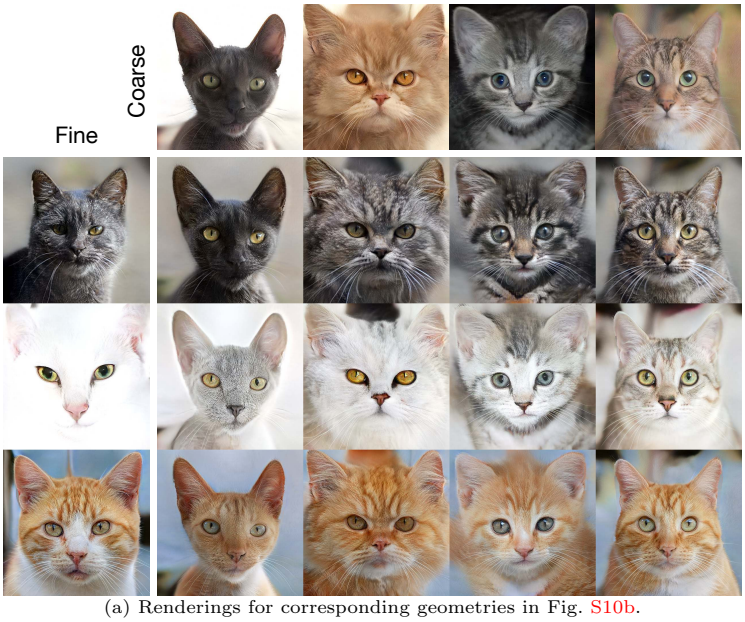
Fig. S8: **Uncurated results on MetFaces.** From top to bottom, left to right, we show generations with seed 1-32. Results are generated with truncation $\psi = 0.7$ [34].


(a) Renderings for corresponding geometries in Fig. S9b.

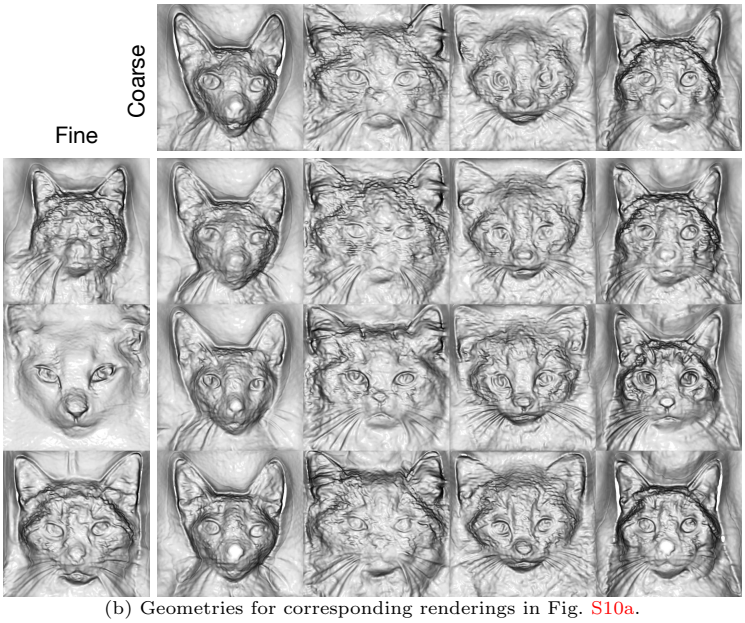



(b) Geometries for corresponding renderings in Fig. S9a.

Fig. S9: **Style mixing on FFHQ.** Results don't use truncation, *i.e.*, $\psi = 1.0$. To obtain each cell in the bottom right grid, we replace lower-level style embeddings $\boldsymbol{\omega}$ (Eq. (3)) in the *Fine* column with the corresponding $\boldsymbol{\omega}$ from the *Coarse* row. We observe, GMPI enables semantic editing: lower-level $\boldsymbol{\omega}$ (layers $0-6$) control the shape while upper-level $\boldsymbol{\omega}$ (layers 7 and higher) determine fine-grained styles.

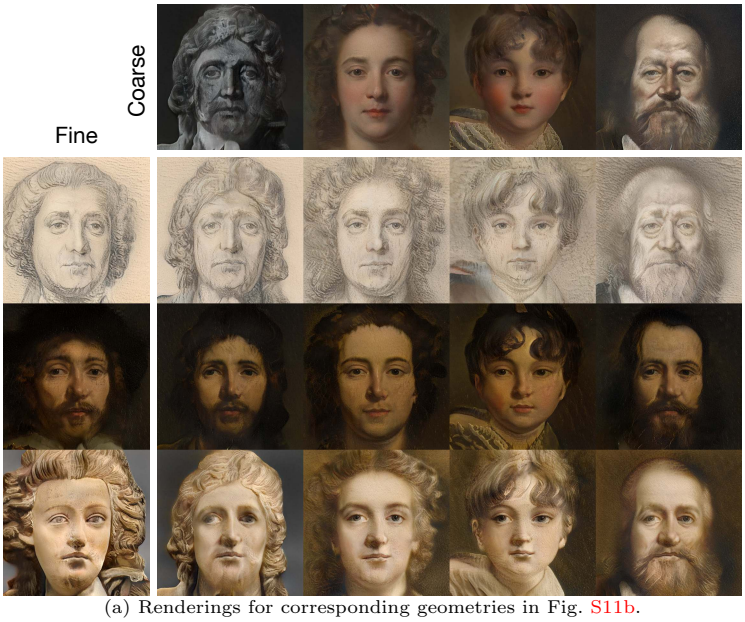



(a) Renderings for corresponding geometries in Fig. S10b.

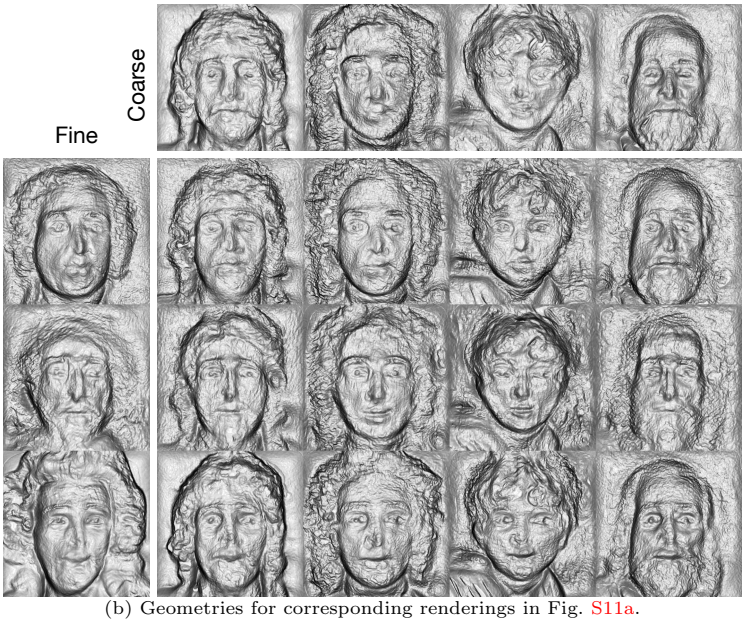



(b) Geometries for corresponding renderings in Fig. S10a.

Fig. S10: **Style mixing on AFHQv2.** Results don't use truncation, *i.e.*, $\psi = 1.0$. To obtain each cell in the bottom right grid, we replace lower-level style embeddings $\boldsymbol{\omega}$ (Eq. (3)) in the *Fine* column with the corresponding $\boldsymbol{\omega}$ from the *Coarse* row. We observe, GMPI enables semantic editing: lower-level $\boldsymbol{\omega}$ (layers $0-6$) control the shape while upper-level $\boldsymbol{\omega}$ (layers 7 and higher) determine fine-grained styles.




(a) Renderings for corresponding geometries in Fig. S11b.




(b) Geometries for corresponding renderings in Fig. S11a.

Fig. S11: **Style mixing on MetFaces.** Results don't use truncation, *i.e.*, $\psi = 1.0$. To obtain each cell in the bottom right grid, we replace lower-level style embeddings $\boldsymbol{\omega}$ (Eq. (3)) in the *Fine* column with the corresponding $\boldsymbol{\omega}$ from the *Coarse* row. We observe, GMPI enables semantic editing: lower-level $\boldsymbol{\omega}$ (layers $0-6$) control the shape while upper-level $\boldsymbol{\omega}$ (layers 7 and higher) determine fine-grained styles.