HeadCraft: Modeling High-Detail Shape Variations for Animated 3DMMs Supplementary Document

1. Method: Technical Details

1.1. Displacement registration procedure

Here we explain the procedure in more detail. The vertices and displacements are modeled in the NPHM coordinate system, aligned with the scans, and the scaling of $30\times$ applied. The implementation of Butterfly subdivision of the FLAME template from MeshLab [3] was used. The parameters of the subdivision are constant across the scans and equal to 3 subdivision iterations with a threshold of 42.5. The subdivision produces around 100K vertices and 200K triangles for the original template consisting of 5023 vertices and 9976 triangles and smooths the surface. The description of the optimization problem features individual loss terms. The expanded expression for the terms are as follows. The Chamfer term $\mathcal{L}_{\text{Chamfer}}(P_1, P_2)$ quantifies the distance between the point clouds $P_1 \in \mathbb{R}^{|P_1|\times 3}$ and $P_2 \in \mathbb{R}^{|P_2|\times 3}$ is supposed to be differentiable w.r.t. the points of P_1. In our work, we apply the version pruned by the distance of the correspondences, i.e. when the Euclidean distance between point and its matched version exceeds the predefined threshold d, this correspondence is not accounted in the loss term.

$$
\begin{aligned}
\mathcal{L}_{\text{Chamfer}}(P_1, P_2) = \\
&\frac{1}{\sum_{p\in P_1}[d(p, nn(p, P_2)) \leq d]} \cdot \\
&\cdot \sum_{p\in P_1} (d(p, nn(p, P_2)) \cdot [d(p, nn(p, P_2)) \leq d]) \\
&+ \frac{1}{\sum_{p\in P_2}[d(p, nn(p, P_1)) \leq d]} \cdot \\
&\cdot \sum_{p\in P_2} (d(p, nn(p, P_1)) \cdot [d(p, nn(p, P_1)) \leq d]),
\end{aligned}
$$

where $d(\cdot, \cdot)$ stands for the Euclidean distance between two points in space and $nn(p, P) = \arg\min_{p'\in P} d(p, p')$ is the nearest neighbor of p in a point cloud P.

Edge length regularization is defined as follows.

$$
\mathcal{L}_{\text{edge}}(V, \mathcal{F}) = \frac{1}{|E|} \sum_{(e_a, e_b)\in E} d^2(V_{e_a}, V_{e_b}), \tag{1}
$$

where $E = E(\mathcal{F})$ is a set of graph edges derived from the faces $\mathcal{F}$. To construct it, we consider each face bringing three new edges and later leave only the unique edges in E.

Laplacian term is defined as the Euclidean distance between the vertex and its neighbors, which can be efficiently calculated via computing sparse Laplacian $L = L(V, \mathcal{F})$ of the graph:

$$
\mathcal{L}_{\text{lapl}}(V, \mathcal{F}) = \sqrt{\sum_{v\in V} \|Lv\|_2^2} \tag{2}
$$

The outer norm is used instead of e.g. L1 averaging to enforce the uniform smoothness of the mesh and avoid spikes that tend to appear otherwise (see, e.g., the documented example in PyTorch3D [8] repository).

During the vector displacement stage, only the scalp region (defined by the semantic mask shipped with FLAME [7]) is optimized. During the normal displacement stage, we also unfreeze the facial region but keep the neck, eyeballs, ears, and inner mouth region frozen (the latter is annotated manually in Blender [4] package and is frozen because of its absence in the ground truth scans, as it is placed fully interior). Each stage takes around 3 min for 1K steps on a single NVIDIA RTX 2080 Ti GPU. We used the PyTorch3D [8] functions for implementation of all the loss terms.

In Fig. 1, we show how the seam was annotated for the custom UV layout.

Blender [4] 4.0 package was used to annotate the seam for the custom UV layout. The seam is manually annotated to be symmetric w.r.t. the reflection symmetry of FLAME. Firstly, the layout was constructed via "Unwrap" Blender tool and adjusted manually via local translation and scaling tools to better fit the available space. After that, the UVs for vertices in the left part of the head were assigned with the mirrored UVs of the vertices in the right part to account for symmetry. This is repeated for the face and scalp separately.

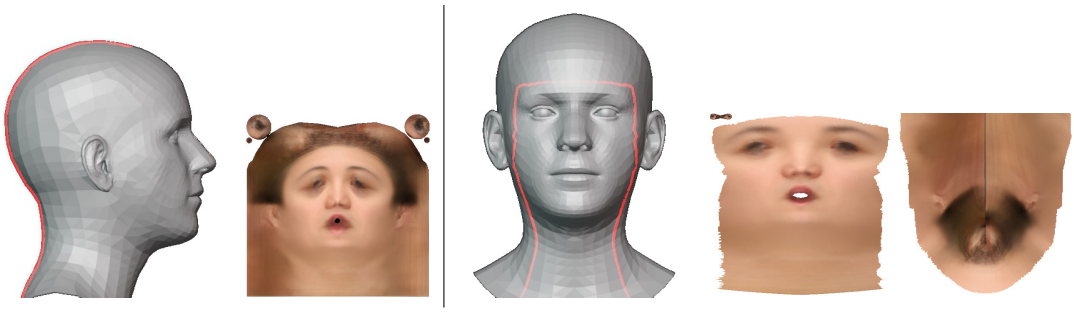

Figure 1. Comparison of the FLAME layouts. The standard, commonly used unwrapping for FLAME *(left)* features a seam corresponding to the vertical line in the back of the head and pays more attention to the facial region than to the scalp. In the hand-crafted custom layout that we employ *(right)*, a different seam around the face border is selected, thus making the regions of face and scalp separated and of similar size to avoid unnecessary breaks and expand the region of interest. Lower neck is not modeled by our method and hence made partially unseen in UV.

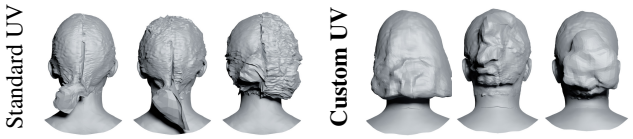

Figure 2. Ablation over the choice of the UV layout. Our method utilizes custom UV layout that allows us to model more consistent geometry by mitigating seam artifacts.

Finally, the face, scalp, and eyeballs parts were scaled and translated to fit the unit square. Regions, for which displacements are not modeled, are either given smaller space in the layout (eyeballs) or moved outside of the unit square (unmodeled parts of the neck).

The displacement vectors entries typically belong to the [-2, +2] range, while some large shape variations (e.g. a ponytail) can introduce the offsets into a large range up to [-20, +20]. We clip any displacements, obtained after full registration, to the [-20, +20] range. As the last step, the displacements are rendered in UV space, and each UV map is saved as uint16 image files linearly renormalized from [-20, +20] to $[0, 2^{16} - 1]$. Saving in uint16 (double-byte intensity value) instead of the widely used uint8 (single-byte intensity value) is important, since most of the displacements vector entries are concentrated around the small neighborhood of zero and the precision can be lost when renormalizing from [-20, +20] to $[0, 2^8 - 1]$ instead and discretizing. Saving UV maps as raw files would otherwise facilitate much slower training of the generative model due to the time-consuming loading and memory usage overhead.

1.2. Generative model

For training StyleGAN, we used the stylegan2-ada-lightning implementation with ADA and augmentations turned off and the following hyperparameters:

latent dim	# layers (z $\rightarrow$ w)	G lr	D lr
512	8	0.002	0.001

λ_{gp}	λ_{plp}	img size	batch size
4.0	2.0	256×256	8

The model was trained on four NVIDIA RTX 2080 Ti GPUs. For training of the StyleGAN, the ground truth UV displacement maps were transformed from $256 \times 256 \times 3$ into $256 \times 256 \times 6$ by first bilinearly upscaling the displacement maps over the width dimension (resulting in the map of shape $256 \times 512 \times 3$) and then splitting it into two three-channel maps by the width dimension. This effectively means that StyleGAN predicts the face and scalp as two separate three-channel images, which increases the spatial area in the output dedicated to each region. Additionally, we replaced the facial part of the registered UV maps of subjects in the NPHM dataset with the corresponding facial part from the neutral expression scan of the same person. This has been done to better support various expressions at the inference time. We also found it beneficial to disable the StyleGAN noise at the inference time, typically injected into the generator, for the face part of the UV map to smooth out the generated facial displacements relative to the scalp displacements that normally require a higher level of detail. For the scalp region, the StyleGAN noise is constant, initialized separately for each generator layer before training. This noise schedule separation is performed via two passes through StyleGAN – one with zero noise and another with constant noise – and combining the corresponding results.

Post-processing. To ensure a smooth transition from the face to the scalp part around the seam, we first create a mesh by querying the UV map as is, then identify the seam vertices and apply Laplacian smoothing [9] to the mesh in the K-vertex vicinity of the seam (vicinity obtained via BFS-style expansion of the seed vertices defining the seam in

a graph defined by the mesh edges). In our experiments, $K = 10$ (corresponding to the subdivided template with roughly 100K vertices), and Laplacian smoothing is repeated 10 times, only affecting the seam vicinity area. Since the face displacements are modeled with lower smoothness due to the disabled StyleGAN noise in that area, the post-processing technique results in a smooth transition from the relatively smooth face displacements to sharper details in the scalp area while maintaining local geometric consistency. The visualization is shown in Fig. 3.

2. Results

Unconditional sampling. In Fig. 7, we provide more unconditional samples from our model from different viewpoints. All the samples have been produced by sampling $z \in \mathcal{N}(0, \mathbb{I})$ with a truncation trick [1] with the power $\psi = 0.7$. For the evaluation in the Table 1 in the main text, the implementation of the metrics MMD, JSD, COV from PointFlow [10] was used. Since FaceVerse contains samples grouped by subjects, the nearest neighbor of a ground truth scan is typically a scan of the same subject with a different expression. Because of that, we only select one ground truth sample per subject (with the same neutral expression for all subjects) to calculate COV. All FaceVerse scans are used to calculate MMD and JSD. As a distance measure between individual point clouds, aggregated over multiple observations in MMD and COV, we use Chamfer Distance.

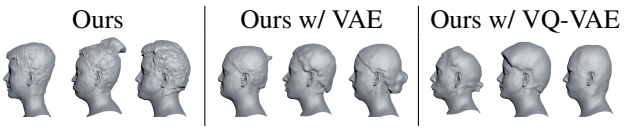

Figure 4. Ablation over the model design. VAE and VQ-VAE both follow the ResNet-18 encoder and decoder architecture, while our method is based on StyleGAN2. [VQ-]VAE mostly match the diversity of the training data but not the level of detail.

Ablating over the choice of the UV layout. We assess the effect of a manually hand-crafted UV space for FLAME on the quality of generations in Fig. 2. As observed, moving the seam from the vertical middle line, as in the standard UV layout for FLAME, to the face border, allows us to model more consistent and complex geometry without large distinction between a left and a right part.

Ablating over the choice of the generative model architecture. The VAE used in our experiments is based on the Lightning Bolts library. The encoder follows the ResNet-18 architecture consisting of blocks of 2 convolutions each, with every second convolution with a stride of two (starting from the third) to downsample the activations spatially the increasing number of channels (64 in the first two blocks, then 128, 256, 256, 256, 512, 512 in the next blocks, respectively). The Lightning Bolts implementation adds two fully-connected layers on top of the encoder (one for the μ and one for the σ prediction). The dimension of the latent space equals 512. The decoder follows the architecture symmetric to the encoder, where the stride two for some convolutions is replaced with a nearest-neighbor 2x upscaling.

For VQ-VAE implementation, we used the implementation of the VQ layer from vector-quantize-pytorch. Pixelcnn-pytorch served as a basis for the PixelCNN sampler implementation. Similarly to VAE, ResNet-18 encoder and decoder were used, with the exception that fewer downsampling operations have been used: they were introduced at each second layer (starting from the third), not each first layer. This is introduced to maintain a trade-off between the autoencoder quality and sampling ability, i.e. not to make PixelCNN operate in a too small latent space. The spatial resolution of the bottleneck is 32×32, which we found to be optimal, as the sampling performance of PixelCNN degrades from the top-left corner to the bottom-right corner and it is very noticeable already at the 64×64 bottleneck spatial resolution. The number of channels is 64, 128, 128, 32 for each two consecutive blocks, respectively. VQ-VAE is trained for 10K steps with batch size of 8, which we found to be enough to reach the sufficient visual quality of autoencoding. To facilitate the sampling, we obtain a dataset of VQ indices and learn PixelCNN to autoregressively sample from those for 200 epochs with a batch size of 32.

Visual comparison of the generative model choices is demonstrated in Fig. 4.

Behavior of the registration procedure. In Fig. 8, 9, 10, 11, we show how the mesh deforms as a result of the vector displacements optimization and normal displacements optimization.

Registration quality. In Table 1, we report exactly the same metrics as in NPHM [6] for both stages of our registration, averaged over identities. N.C. refers to Normal Consistency score; see [6] for clarification on all metrics. We excluded the region below the threshold by vertical axis (-0.30 in standard NPHM coordinate system) in order to not account for non-modeled region (clothes and neck). The registration precision is naturally better by all metrics than FLAME that does not provide a hair fit and better face fit.

Registration	L_1-Chamfer ↓	N.C. ↑	F-Score @ 1.5 ↑
FLAME	3.33e-1	0.763	0.266
Stage 1 only	9.20e-2	0.817	0.861
Stage 1+2 (Ours)	5.68e-2	0.841	0.949

Table 1. Metrics reflecting the registration quality for our method w.r.t. only using Stage 1 of the registration procedure (vector offsets) and FLAME fits.

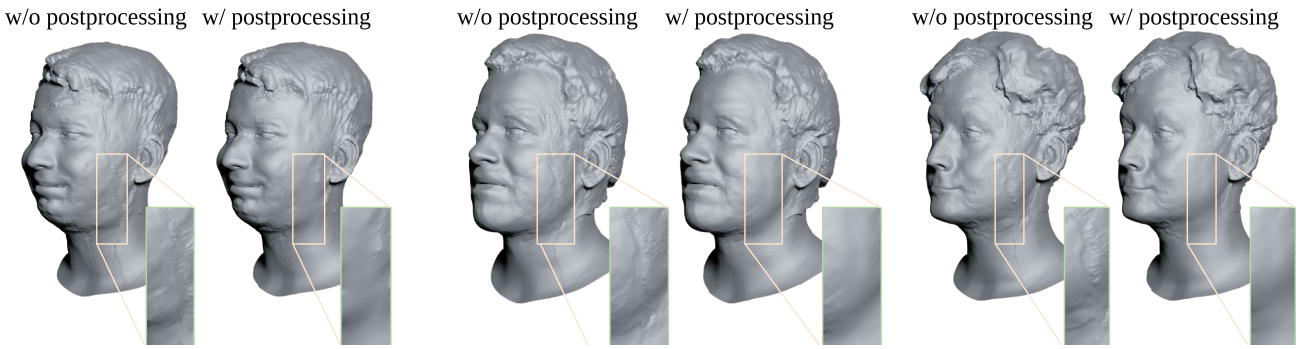

Figure 3. Comparison w/ and w/o the post-processing procedure that enables smoother transition between the face and scalp parts modeled in unconnected regions of the UV space.

The necessity of the second stage is also motivated accordingly. Note that even though the numbers are not directly comparable to the ones in the NPHM paper [6], since there the evaluation over the face region only was conducted, and for us it is for the whole FLAME surface, they are still in the same scale. This indicates that the quality attained by non-rigid face refinement procedure in NPHM has been mostly achieved by our registration procedure for both face and hair.

Consistency of registrations. In Fig. 13, we demonstrate the analysis as to which template vertices are selected by the registration procedure to cover various regions of different meshes. Since we know the UV coordinates of all template vertices, this can be done by rendering the meshes with a UV checker texture image. Note that the long hair parts, such as pony tails, are mostly explained by the same regions of the layout as the vertices they originate from.

Comparison of the hair length. In Fig. 5, we show the comparison of the offset length in the scalp region between the ground truth data (*NPHM scans*) and the model predictions (*HeadCraft*). Since this offset length approximates the haircut size, we use it as a measure of hair length (not accounting for any hair curvature). This is demonstrated for comparison of the occurrence frequency of long hair samples in the model's predictions w.r.t. the ground truth data distribution.

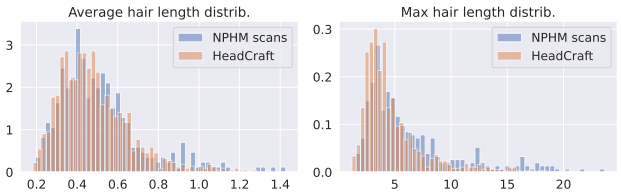

Figure 5. Comparison of the average offset length distribution in the scalp region for our method's predictions and 3D scans from the NPHM dataset. We consider the offset length in the scalp region the approximation of the haircut size (or, simplified, a measure of hair length, without accounting for the hair curvature). The diversity of the average/maximum of this parameter per scan for our generated samples is on par with the training data. The horizontal axis quantifies the length in the NPHM coordinate system and the vertical axis stands for the bin height of the histogram.

2.1. Applications

Fitting the latent code to a full scan. To fit the latent to the complete head scan, we have to apply preliminary steps, similar to the ones used to construct the training set. Firstly fit the FLAME to the scan, then apply our registration procedure to get a UV map U_{gt}. After that, we fit a $w \in \mathcal{W}+ \subset \mathbb{R}^{16\times512}$ latent code for the StyleGAN generator $g(w) : \mathcal{W}+ \rightarrow \mathbb{R}^{H\times W\times3}$ to satisfy the following loss terms:

$$\begin{aligned}\mathcal{L}_{opt}^{full}(w|U_{gt}, \boldsymbol{\lambda}) \\ = \lambda^{\text{LPIPS}} \cdot \text{LPIPS}(g(w), U_{gt}) \\ + \lambda^{L_1} \cdot L_1(g(w), U_{gt}),\end{aligned}$$

where $L_1(\cdot,\cdot)$ is an average pixel-wise L1 distance between two images and $LPIPS(\cdot,\cdot)$ corresponds to the LPIPS score [11]. To calculate LPIPS, we cut the 256×256 UV maps (both predicted $U = g(w)$ and ground truth U_{gt} into sixteen 64×64 patches, evaluate LPIPS between the respective patches of U and U_{gt}, and average the obtained sixteen scores. The parameters of the loss equal to $\lambda^{\text{LPIPS}} = 0.1$ and $\lambda^{L_1} = 3$. The loss is being optimized via Adam algorithm with the learning rate of 10^{-2} for 1K steps. The w is initialized as the average latent predicted by the trained StyleGAN mapping network, evaluated over 10^5 codes $z \in \mathcal{N}(0, \mathbb{I})$.

Finally, we optimize for the StyleGAN noise (only for the scalp region of the UV space) to better fit the tiny details of the map U_{gt}. This step can be omitted in practice if fitting very high-frequency details is not required. Exactly the same loss terms are being optimized, this time not with respect to w but with respect to the StyleGAN noise

tensors of all generator layers, while w remains fixed. The optimization is again carried out by Adam with the same learning rate and number of steps.

Fitting the latent code to a depth map. Fitting the latent representation to represent a partial observation poses a more challenging problem than trying to represent a full scan, since the resulting displacements must both resemble the original point cloud and complete it in a realistic way. This requires several changes to the fitting pipeline, described next.

Firstly, prior to applying the registration procedure to register part of the cloud P in the UV space, we identify the mask of points $m \in \{0,1\}^{|V|}$ that are allowed to be offset by selecting only the points within the convex hull of the point cloud, expanded by 1.5x from its center to account for the possible important regions missing in the point cloud. The points below a certain horizontal plane are not accounted for when estimating the convex hull to disregard the shoulders and clothing, usually featured in NPHM raw scans. The level of the horizontal plane is selected as a 30% quantile of the coordinates of the points along the vertical axis. Masking out the points too far from the convex hull of the point cloud is especially important when the point cloud covers the minority of the geometry (e.g. if it is coming from a single depth map), since in this case, these points tend to pull in to cover the parts that the points inside the hull cannot explain (e.g. due to the regularizations), and this results in a non-plausible shape. For the registration procedure itself, stronger regularization parameters for the first stage have been selected, namely $\boldsymbol{\lambda}_{\text{Stage 1}} = (\lambda^{\text{Chamfer}}_{\text{Stage 1}}, \lambda^{\text{edge}}_{\text{Stage 1}}, \lambda^{\text{lapl}}_{\text{Stage 1}}) = (2{\cdot}10^3, 8{\cdot}10^5, 10^5)$. The correspondence pruning threshold, on the contrary, is raised to 10.0 for the first stage to allow the points to move farther while maintaining higher smoothness of the overall geometry due to stronger regularizations. For the second stage, the threshold is on the contrary reduced to 0.1 to penalize for large false movement of points along the template normals to explain the individual points of the cloud.

At the end of the registration, we refine the mask of the points by only selecting those of them that are sufficiently close to the fitted point cloud: $m_i^{\text{final}} = m_i \cdot [d(v_i + D_{\text{Stage 2},i}, nn(v_i + D_{\text{Stage 2},i}, P)) \leq t]$, where t defines the proximity threshold, and its optimal value depends on the sparsity of the cloud. For the point cloud formed from a dense depth map, we set $t = 0.1$, and for a sparse cloud with only 1% points of the original depth map left, we set $t = 0.3$. The regressed displacements and the mask are separately baked in the UV map as two independent images, 3-channel real-valued U and 1-channel binary M, respectively. In Fig. 12, we demonstrate the typical result of the partial registration stages.

Another important change lies in the latent fitting procedure. In our observations, the optimization of $w \in \mathcal{W}+$ latent code works great for the visible part but tends to produce displacements closer to the average shape for the non-visible part. We explain it by not strong enough supervision from the prior during fitting in $\mathcal{W}+$ space. To mitigate that effect, we first fit the $z \in \mathcal{Z} \subset \mathbb{R}^D$ latent code of the StyleGAN mapping network $map(z) : \mathcal{Z} \to \mathcal{W}+$, obtain the respective $w = map(z) \in \mathcal{W}+$ and regress the delta to the w code: Δw. We found that optimizing z code yields much better, yet rougher result of completion, and refining the map by regressing the Δw greatly improves fitting of the details.

In more detail, during the first z optimization step, we optimize the following loss:

$$\begin{aligned}&\mathcal{L}^{z}_{opt}(z|U_{gt}, \boldsymbol{\lambda}) \\ &\quad = \lambda^{\text{LPIPS}} \cdot \text{LPIPS}(g(map(z)) \cdot M, U_{gt} \cdot M) \\ &\quad + \lambda^{L_1} \cdot L_1(g(map(z)) \cdot M, U_{gt} \cdot M),\end{aligned}$$

Similarly to the $\mathcal{L}^{full}_{opt}$, we use $\lambda^{\text{LPIPS}} = 0.1$ and $\lambda^{L_1} = 3$. The z is initialized from $\mathcal{N}(0, \mathbb{I})$ and further optimized by Adam with the learning rate of 10^{-2} for 500 steps. Here and further, $\text{LPIPS}(\cdot,\cdot)$ and $L_1(\cdot,\cdot)$ follow the same expressions as for the full scan fitting.

During the second Δw optimization step, we optimize a similar expression with a few additional terms:

$$\begin{aligned}&\mathcal{L}^{\Delta w}_{opt}(\Delta w|z, U_{gt}, \boldsymbol{\lambda}) \\ &\quad = \lambda^{\text{LPIPS}} \cdot \text{LPIPS}(g(map(z) + \Delta w) \cdot M, U_{gt} \cdot M) \\ &\quad + \lambda^{L_1} \cdot L_1(g(map(z) + \Delta w) \cdot M, U_{gt} \cdot M) \\ &\quad + \lambda^{\text{LPIPS}}_{\text{preserve}} \cdot \text{LPIPS}(g(map(z) + \Delta w) \cdot (1 - M), \\ &\qquad\qquad g(map(z) \cdot (1 - M)) \\ &\quad + \lambda^{L_1}_{\text{preserve}} \cdot L_1(g(map(z) + \Delta w) \cdot (1 - M), \\ &\qquad\qquad g(map(z)) \cdot (1 - M)),\end{aligned}$$

where M^{face} is a predefined mask of the face region in the UV space, reduced to the circle including the eyes, nose and mouth.

The third and second "preserve" terms are introduced to not let the map guided by the Δw optimization deviate much from the output corresponding to the regressed z in non-visible regions, which is essential due to the tendency of convergence to the average shape there when optimizing in the $\mathcal{W}+$ space. $\lambda^{\text{LPIPS}} = 0.1$ and $\lambda^{L_1} = 3$ remain the same as before, and $\lambda^{\text{LPIPS}}_{\text{preserve}} = 0.01$ and $\lambda^{L_1}_{\text{preserve}} = 0.3$ are selected $10\times$ less. The optimization is carried out by Adam with the same learning rate of 10^{-2} for 500 steps. The Δw is initialized with zeros.

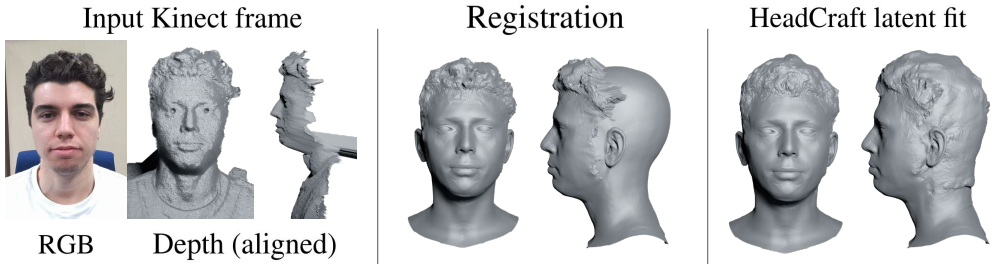

Figure 6. An example of the use case for the HeadCraft registration and fitting stages to Kinect depth data. An example of a frontal RGB frame is shown on the left (the color information is not used further), along with the observed Kinect depth, converted into a mesh and aligned with the standard FLAME coordinate system. We fit FLAME and apply the partial registration procedure, described in Sec. 2, to register the displacements in the observed region (middle) and further fit the HeadCraft latent representation to the corresponding part of the UV map.

Finally, we optimize the StyleGAN noise to improve the details in the visible part. Despite that we consider this step optional, we found that it helps reconstruct more detail even for a sparse cloud. We optimize the same expression as $\mathcal{L}^z_{opt}$, with the difference that it is only being optimized w.r.t. the StyleGAN noise tensors (only in the scalp region). The only modification is the introduced regularization that equals to the sum of the noise tensors L2 norms. The optimization is carried out by Adam with the same learning rate of 10^{-2} for 500 steps. The coefficient of this regularization is equal to 10^{-5}.

In the Supplementary Video, we demonstrate more results of fitting the latent to the point clouds with different sparsity.

Use case: Kinect data. We demonstrate that the HeadCraft pipeline can be applied to real-world depth scans in Fig. 6. For a sample RGB-D image captured by Kinect, we first convert it into a mesh by unprojecting the points with color and connecting the vertices by the triangles constructed from the image pixels. The resulting mesh with vertex colors is rendered onto three views (frontal, slight left, and slight right) to obtain the facial landmarks from each side via an image-based facial landmark detector [2] and aggregate them (jawline landmarks are obtained from slight left and slight right and the others from the frontal rendering). FLAME is fitted using these landmarks, and the displacements for the visible part of the mesh are obtained via the partial registration procedure described above and later baked into the UV displacement map. We also show the result of fitting the latent representation of HeadCraft to the visible part of the scan. Compared to the aforementioned procedure, we omit LPIPS loss to give L1 more relevance in predicting coarse shape and only supervise the latent in the scalp region.

Animation. Here we expand on more details regarding applying displacements to a template, deforming over time. Compared to the simple unconditional scenario, where the displacements are also applied to a certain FLAME template, we have to introduce two key differences.

First, as mentioned in Subsec. 1.1, to apply the displacements to the template, we apply Butterfly subdivision, the MeshLab implementation of which also smooths the surface. However, the result of Butterfly is not consistent over various FLAME templates and yields a bit different number of vertices every time. To solve that, we come up with *consistent subdivision*, i.e. the way to construct the same topology for every FLAME. To do that, we first apply Butterfly subdivision to an arbitrary scan, and for each vertex after the subdivision, we find which triangle of the original template it belongs to and the barycentric coordinates w.r.t. that triangle. Later, for every new template, the locations of the subdivided vertices are evaluated based on these triangles and barycentric coordinates. To handle the seam accurately, we consider each vertex of every triangle after subdivision individually, thus accounting for the duplicate vertices.

An artifact of such procedure is that the smoothness of the surface, introduced in the MeshLab implementation of Butterfly subdivision, cannot be trivially transferred onto a new mesh this way. Because of this, the surface normals remain the same within the large triangles of the original template even after the subdivision, creating a non-appealing "tiling" effect. To mitigate that, we apply Laplacian smoothing [9] in its classical version to smooth the surface. In order to account for important subtle parts, we apply a different number of Laplacian smoothing iterations to different regions, namely, 3 times to the lips region, 5 times to the face skin (face except mouth, eyeballs and eye surroundings), and 10 times to the scalp and the neck. Since the realism of mouth, ears, and eyeballs is important for animation, they remain intact.

Second, as mentioned in the main text, we rotate the displacements according to the rotation of the surface normals of the template. To do that, we first estimate the local basis of the

TBN space [5] for each FLAME in a sequence. This basis defines the normalized tangent $\boldsymbol{t_i^k}$, bitangent $\boldsymbol{b_i^k}$, and normal $\boldsymbol{n_i^k}$, pre-estimated for the i-th vertex of the FLAME template $F^k = FLAME(\text{shape}, \text{exp}^k, \text{jaw}^k, \text{headpose}^k)$. In addition, we estimate the TBN basis $(\boldsymbol{t_i^{\text{neutral}}}, \boldsymbol{b_i^{\text{neutral}}}, \boldsymbol{n_i^{\text{neutral}}})$ for a FLAME, corresponding to the same person and a neutral expression and pose $F^{\text{neutral}} = FLAME(\text{shape}, \mathbf{0}, \mathbf{0}, \mathbf{0})$. The displacements D, queried from the generated UV map U, are first transferred from the object space into the neutral TBN space:

$$D^{\text{TBN}} = \left((\boldsymbol{t_i^{\text{neutral}}} \cdot \boldsymbol{d_i}), (\boldsymbol{b_i^{\text{neutral}}} \cdot \boldsymbol{d_i}), (\boldsymbol{n_i^{\text{neutral}}} \cdot \boldsymbol{d_i})\right)_{i=1}^{|D|}$$

Then, for each of the sequence frames, we transfer them into object space, this time w.r.t. the TBN basis of the given frame:

$$D_k^{\text{object}} = \left(\left[\boldsymbol{t_i^k} \;\; \boldsymbol{b_i^k} \;\; \boldsymbol{n_i^k}\right] \cdot \boldsymbol{d_i^{\text{TBN}}}\right)_{i=1}^{|D|}$$

(the $\boldsymbol{t_i^k}$, $\boldsymbol{b_i^k}$, $\boldsymbol{n_i^k}$, $\boldsymbol{d_i^{\text{TBN}}}$ vectors above treated as columns).

More examples of animations can be found in the Supplementary Video.

References

[1] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 3

[2] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks). In *International Conference on Computer Vision*, 2017. 6

[3] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. 1

[4] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 1

[5] Joey de Vries. Learn OpenGL: Normal Mapping. `https://learnopengl.com/Advanced-Lighting/Normal-Mapping`, 2014. 7

[6] Simon Giebenhain, Tobias Kirschstein, Markos Georgopoulos, Martin Rünz, Lourdes Agapito, and Matthias Nießner. Learning neural parametric head models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21003–21012, 2023. 3, 4

[7] Tianye Li, Timo Bolkart, Michael J Black, Hao Li, and Javier Romero. Learning a model of facial shape and expression from 4D scans. *ACM Trans. Graph.*, 36(6):194–1, 2017. 1

[8] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 1

[9] Jörg Vollmer, Robert Mencl, and Heinrich Mueller. Improved laplacian smoothing of noisy surface meshes. In *Computer graphics forum*, pages 131–138. Wiley Online Library, 1999. 2, 6

[10] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3D point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019. 3

[11] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 4

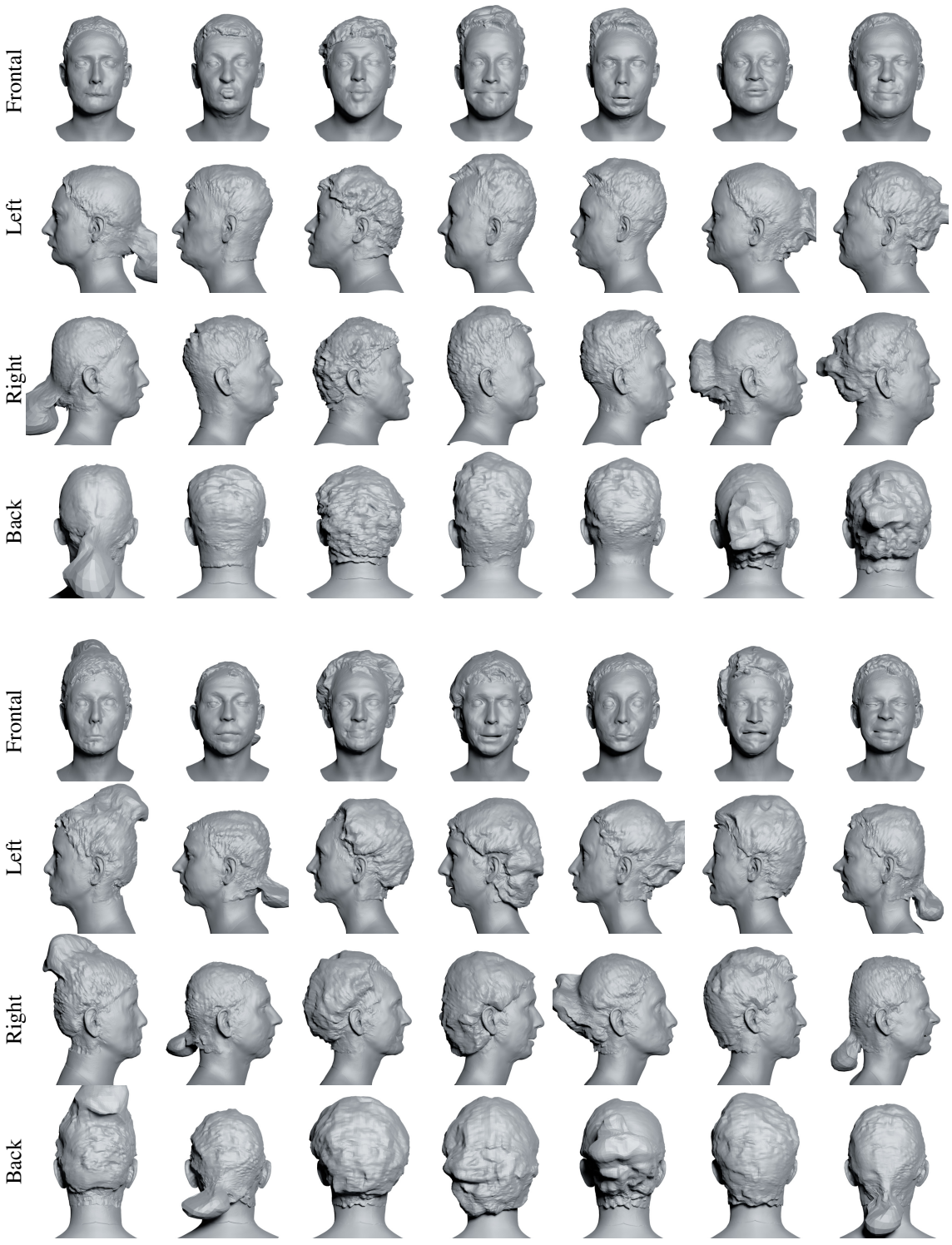

Figure 7. Additional demonstration of the diversity and level of detail of the unconditionally sampled generations from HeadCraft. The generations are obtained by randomly sampling $z \sim \mathcal{N}(0, \mathbb{I})$ latent code of the generative model. The displacements, returned by the model, are applied to the random FLAMEs sampled from Gaussian distribution with statistics calculated over the NPHM dataset.

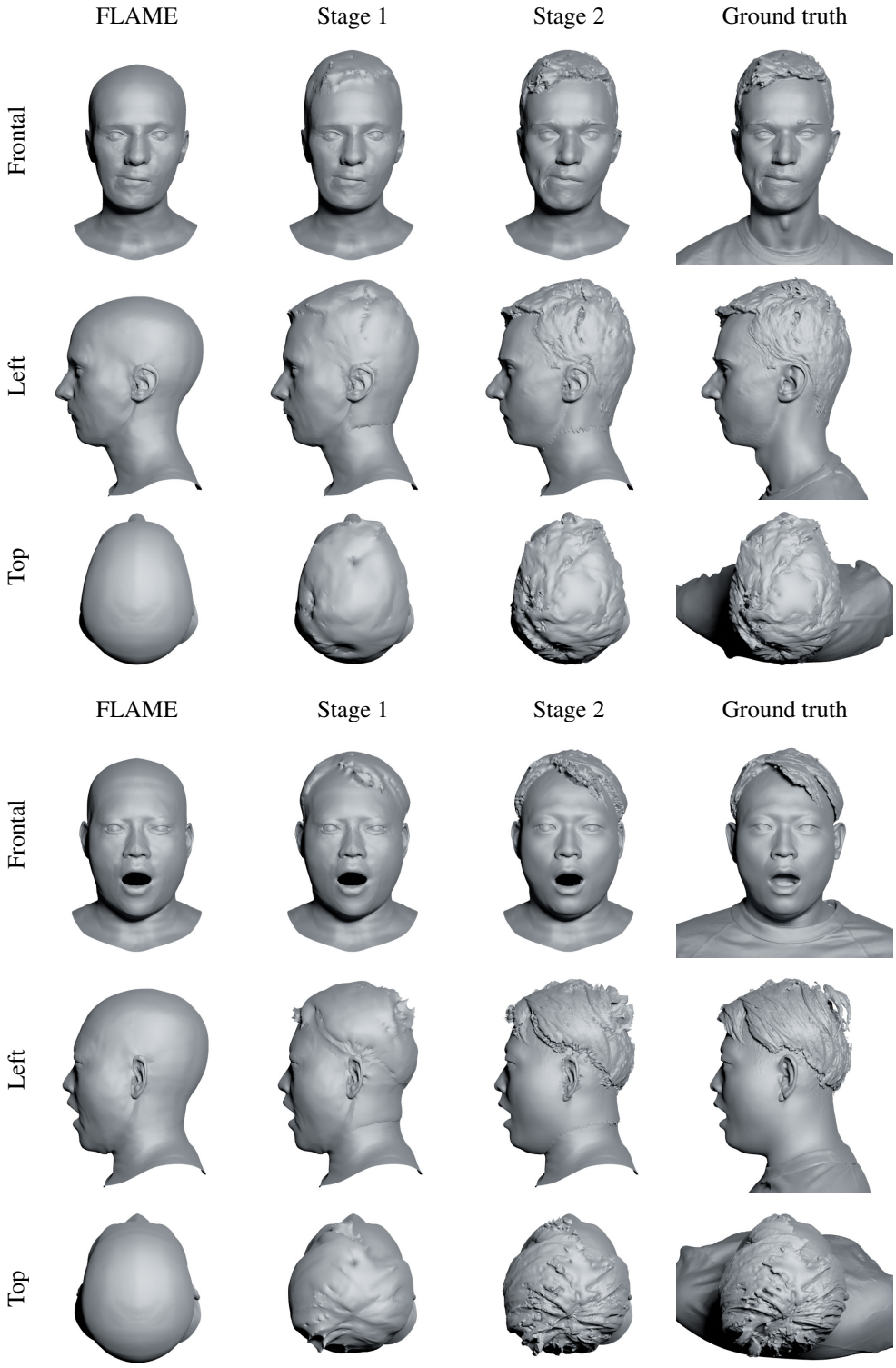

Figure 8. Additional demonstration of the two-stage registration. *Stage 1* corresponds to the vector displacements regression; *Stage 2* – to the refinement of the displacements along the normals. The second stage significantly improves the level of detail and allows us to match the high-frequency component of the scans, such as strands and subtle face features.

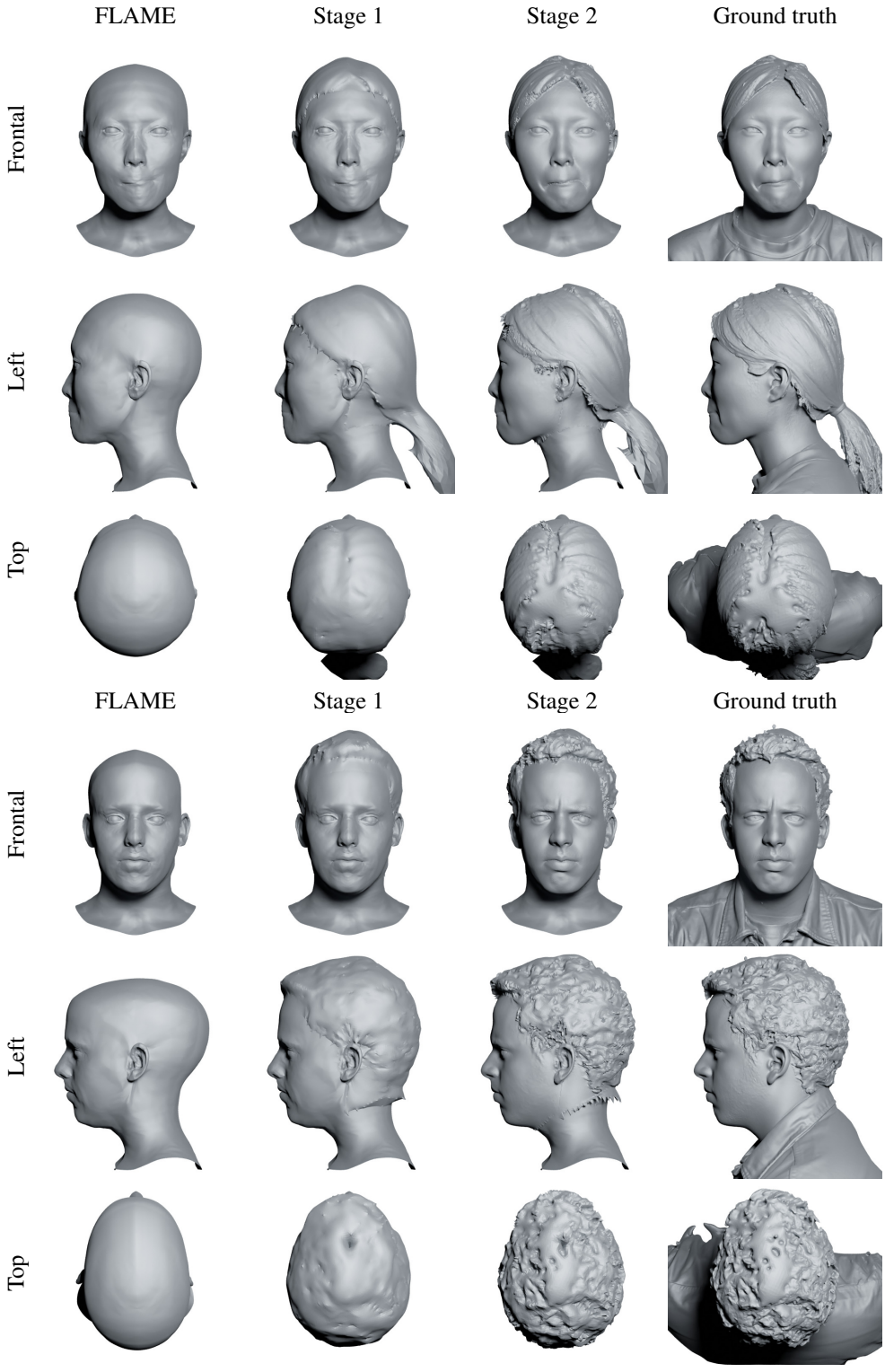

Figure 9. Additional demonstration of the two-stage registration. *Stage 1* corresponds to the vector displacements regression; *Stage 2* – to the refinement of the displacements along the normals. The second stage significantly improves the level of detail and allows us to match the high-frequency component of the scans, such as strands and subtle face features.

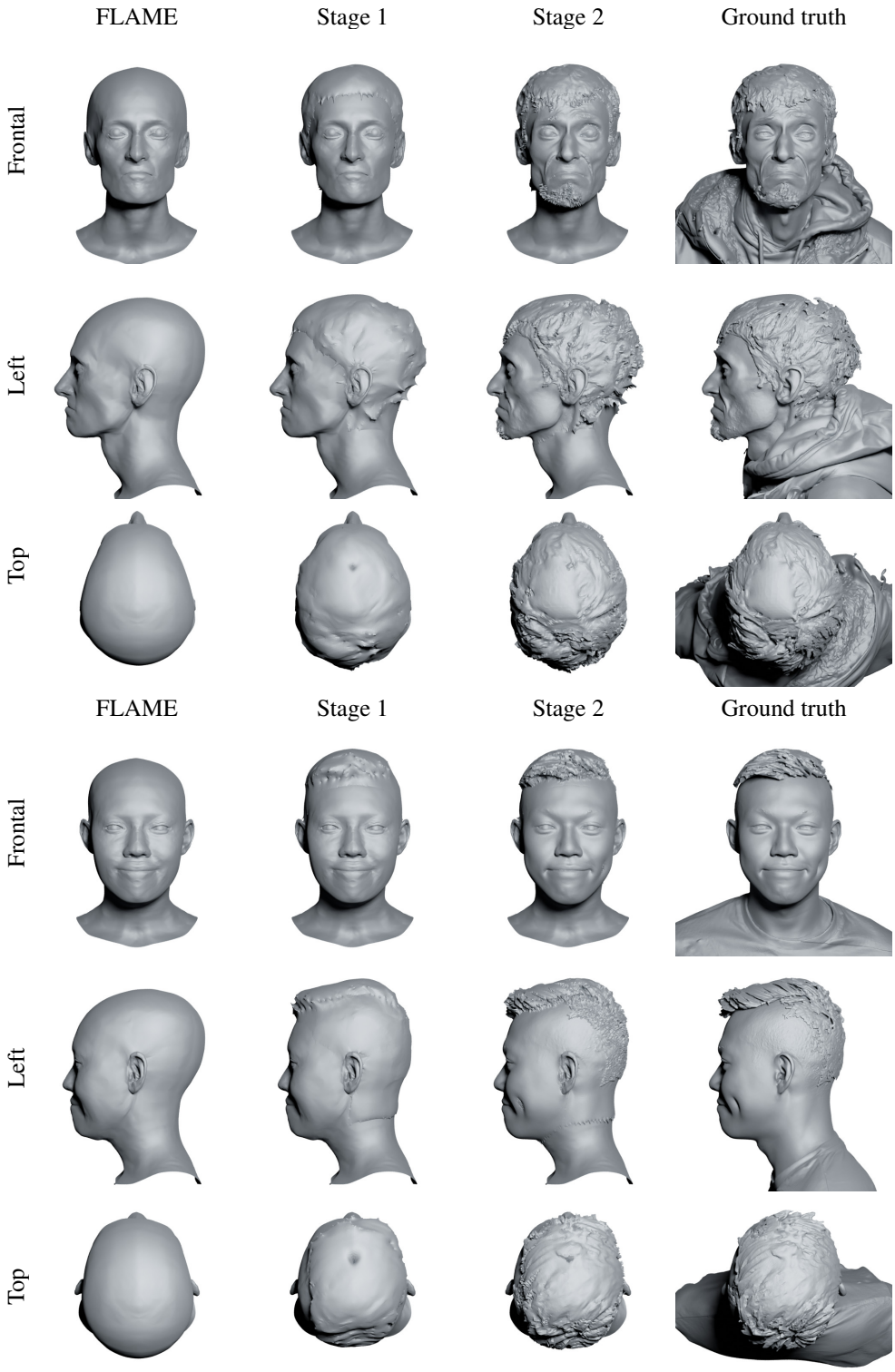

Figure 10. Additional demonstration of the two-stage registration. *Stage 1* corresponds to the vector displacements regression; *Stage 2* – to the refinement of the displacements along the normals. The second stage significantly improves the level of detail and allows us to match the high-frequency component of the scans, such as strands and subtle face features.

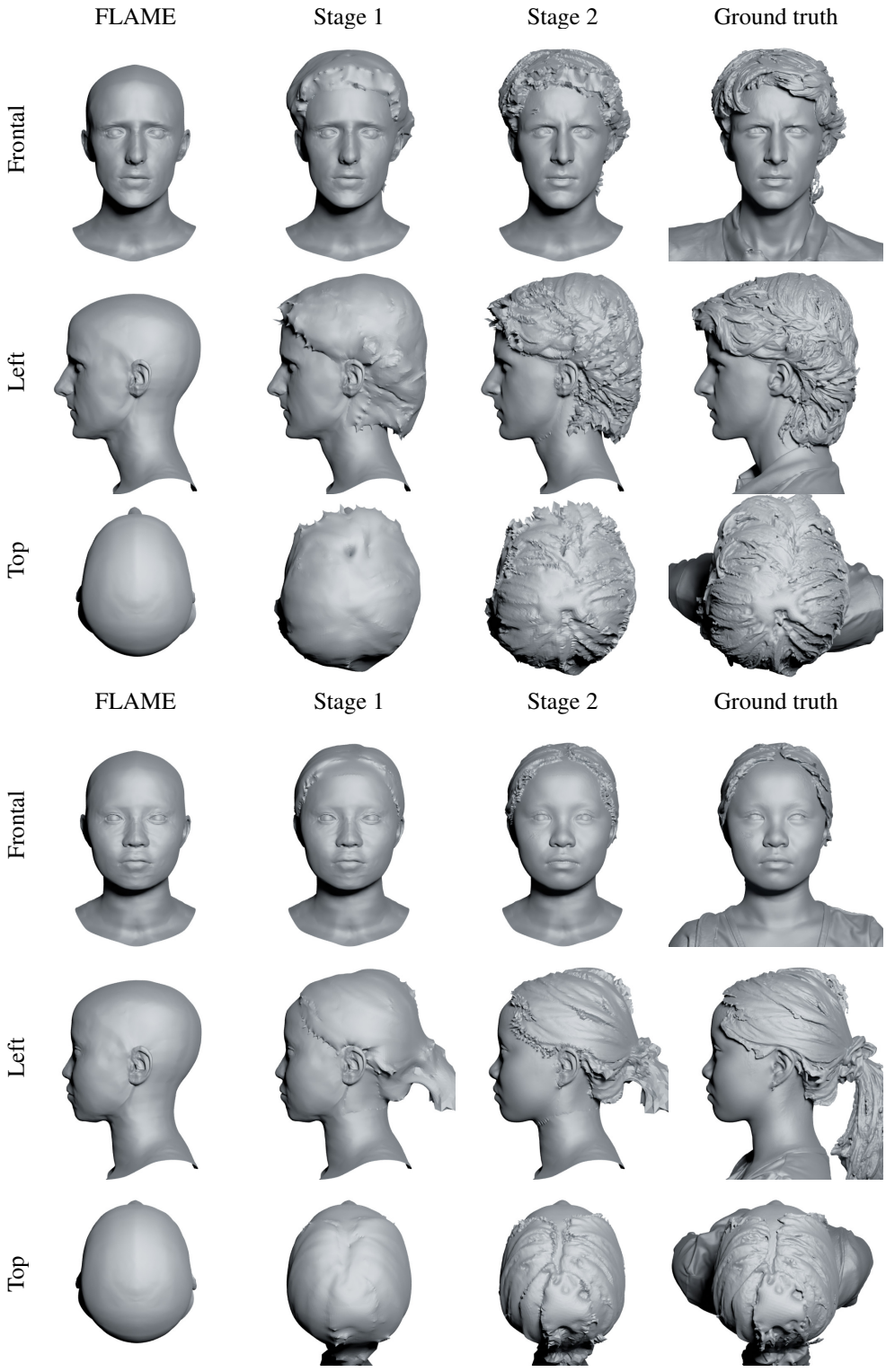

Figure 11. Additional demonstration of the two-stage registration. *Stage 1* corresponds to the vector displacements regression; *Stage 2* – to the refinement of the displacements along the normals. The second stage significantly improves the level of detail and allows us to match the high-frequency component of the scans, such as strands and subtle face features.

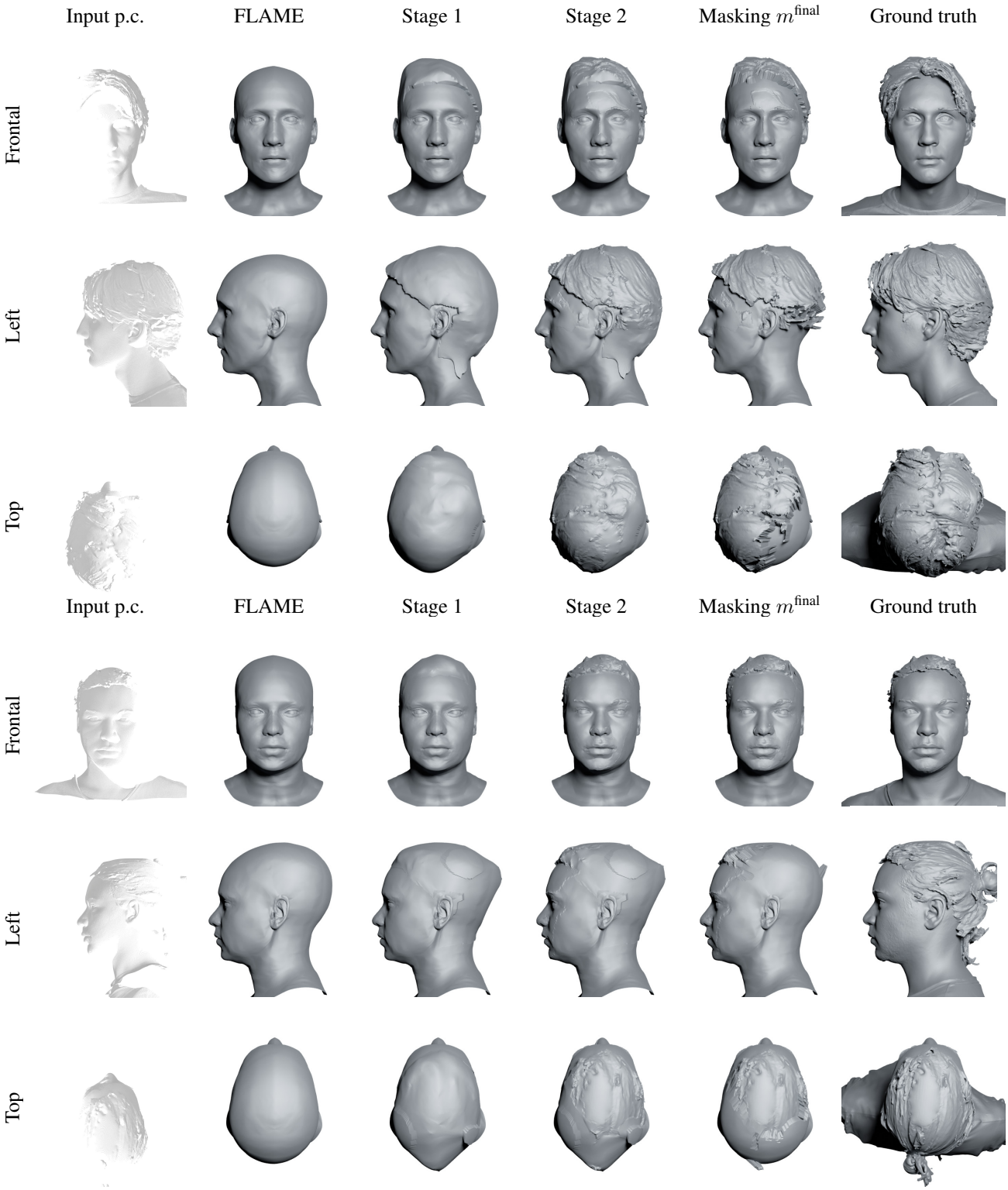

Figure 12. Demonstration of the stages of the partial registration procedure required to fit a part of the scan. The key difference between this procedure and the standard registration used to generate training data for HeadCraft is in the presence of only a part of the scan, e.g. a point cloud coming from the depth map. To overcome that obstacle, the displacements are being estimated only in the convex hull of the point cloud, and are subsequently filtered out by a separate mask m^{final}, leaving only the displacements close enough the ground truth scan (others are nullified in this visualization).

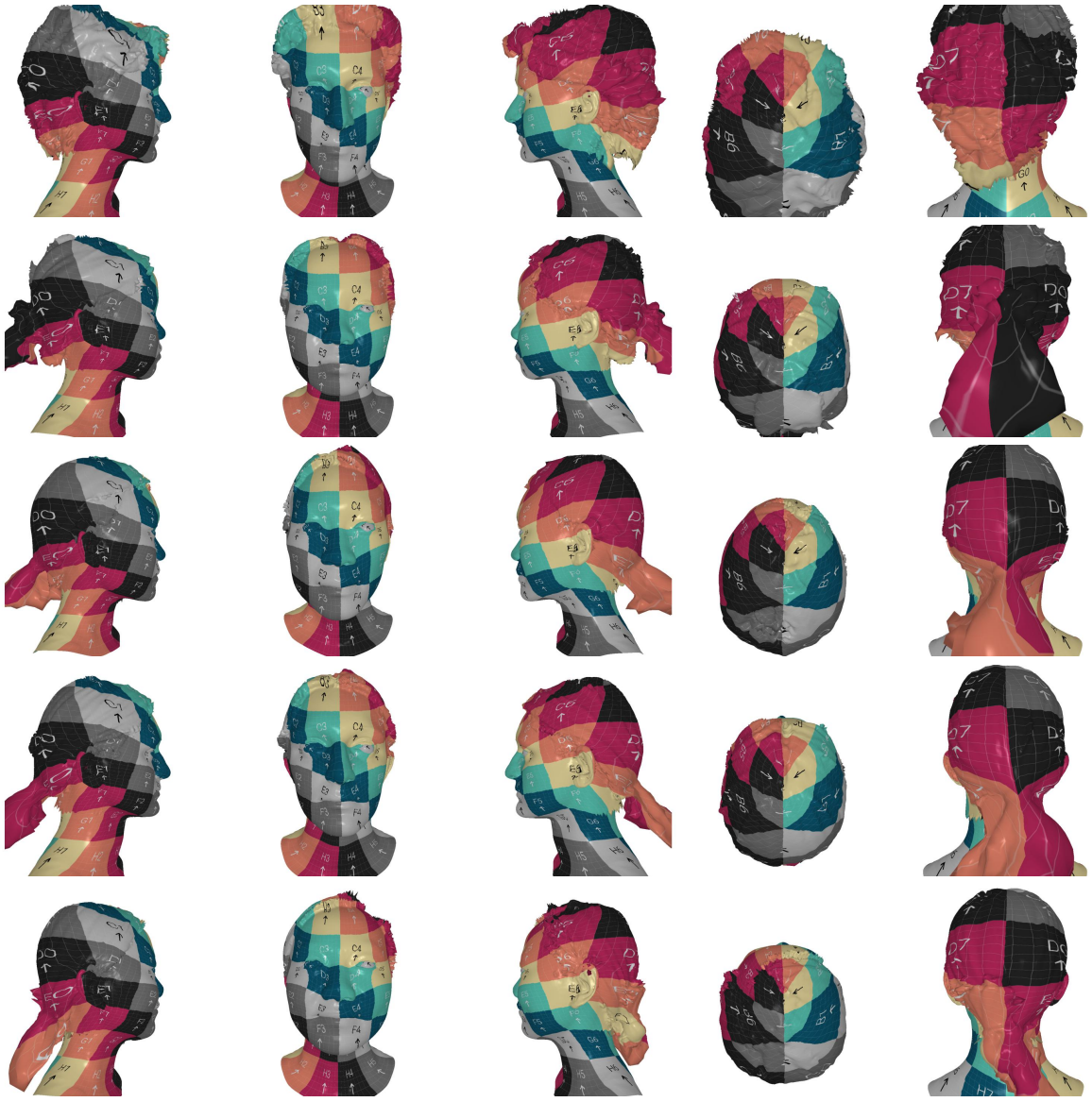

Figure 13. Consistency analysis of the registration. We demonstrate which template vertices are offset with the registration procedure to cover various regions of different meshes. Since we know the UV coordinates of all template vertices, this can be done by rendering the meshes with a *UV checker* texture image. For clarity of the visualization, the texture is applied to the standard FLAME layout. Note that the long hair parts, such as pony tails, are mostly explained by the same regions of the layout as the vertices they originate from.