Frame Averaging for Equivariant Shape Space Learning

A. Additional Details

Here we provide additional details for each experiment in section 5.

A.1. Global Euclidean Mesh $\rightarrow$ mesh.

Architecture. We start by describing in full details the backbone architectures for both the encoder ϕ and decoder ψ, for all the methods in table 1. Then, we describe the relevant hyperparmaeters for our Frame Averaging version. Note that the backbone architectures are similar to the ones in [5], and are described here again for completeness. The network consists of the following layers:

$$\begin{aligned}\mathrm{Conv}(n, d_{\text{in}}, d_{\text{out}}) &: \mathbb{R}^{n\times d_{\text{in}}} \rightarrow \mathbb{R}^{n\times d_{\text{out}}}\\ \mathrm{Pool}(n, n') &: \mathbb{R}^{n\times d} \rightarrow \mathbb{R}^{n'\times d}\\ \mathrm{Linear}(d_{\text{in}}, d_{\text{out}}) &: \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}.\end{aligned}$$

The $\mathrm{Conv}(n, d_{\text{in}}, d_{\text{out}})$ is the Chebyshev spectral graph convolutional operator [3] as implemented in [5, 12], where n denotes the number of vertices, and d_{in}, d_{out} are the layer's input and output features dimensions respectively. All the $\mathrm{Conv}(n, d_{\text{in}}, d_{\text{out}})$ layers have 6 Chebyshev polynomials. The $\mathrm{Pool}(n, n')$ layer is based on the mesh down/up sampling schemes defined in [12], where we used the same mesh samples as provided in [5] official implementation. The n, n' are the number of vertices in the input and the output, respectively. The linear layer is defined by $\boldsymbol{X}' = \boldsymbol{W}\boldsymbol{X} + \boldsymbol{b}$, where $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ and $\boldsymbol{b} \in \mathbb{R}^{d_{\text{out}}}$ are learnable parameters.

Moreover, the backbone architecture consists of the following blocks:

$$\begin{aligned}&\mathrm{EnBlock}(n, n', d_{\text{in}}, d_{\text{out}}) : \boldsymbol{X} \in \mathbb{R}^{n\times d_{\text{in}}} \mapsto \boldsymbol{X}' \in \mathbb{R}^{n\times d_{\text{out}}}\\ &\boldsymbol{X}' = \mathrm{Pool}(n, n', d_{\text{out}})\left(\sigma\left(\mathrm{Conv}(n, d_{\text{in}}, d_{\text{out}})\right)\right)\\ &\mathrm{DeBlock}(n, n', d_{\text{in}}, d_{\text{out}}) : \boldsymbol{X} \in \mathbb{R}^{n\times d_{\text{in}}} \mapsto \boldsymbol{X}' \in \mathbb{R}^{n\times d_{\text{out}}}\\ &\boldsymbol{X}' = \sigma\left(\mathrm{Conv}(n', d_{\text{in}}, d_{\text{out}})\left(\mathrm{Pool}(n, n', d_{\text{in}})(\boldsymbol{X})\right)\right)\end{aligned}$$

where σ is the ELU activation function. Then, the encoder $\phi : \mathbb{R}^{6890\times 3} \rightarrow \mathbb{R}^{72}$ consists of the following blocks:

$$\begin{aligned}&\mathrm{EnBlock}(6890, 3445, 3, 32) \rightarrow\\ &\mathrm{EnBlock}(3445, 1723, 32, 32) \rightarrow\\ &\mathrm{EnBlock}(1723, 862, 32, 32) \rightarrow\\ &\mathrm{EnBlock}(862, 431, 32, 64) \rightarrow \mathrm{Linear}(27584, 72) \rightarrow .\end{aligned}$$

Similarly, the decoder $\psi : \mathbb{R}^{72} \rightarrow \mathbb{R}^{n\times 3}$ is consisted of the following blocks:

$$\begin{aligned}&\mathrm{Linear}(72, 27584) \rightarrow \mathrm{DeBlock}(431, 862, 64, 64) \rightarrow\\ &\mathrm{DeBlock}(862, 1723, 64, 32) \rightarrow\\ &\mathrm{DeBlock}(1723, 3445, 32, 32) \rightarrow\\ &\mathrm{DeBlock}(3445, 6890, 32, 32) \rightarrow\\ &\mathrm{Conv}(6890, 6890, 32, 3) \rightarrow .\end{aligned}$$

For the Frame Averaging versions of these backbone architectures, $\Phi = \langle\phi\rangle_{\mathcal{F}}$, and $\Psi = \langle\psi\rangle_{\mathcal{F}}$, we set $m = 0$ and $d = 24$. For the frames construction of Φ and Ψ, $\mathcal{F} : V \rightarrow 2^{E(3)} \setminus \emptyset$, we use the construction defined in section 3.3, with $\boldsymbol{w} = \mathbf{1}$. Note that for Φ the frames are calculated with respect to the input vertices $\boldsymbol{V} = \boldsymbol{X} \in \mathbb{R}^{6890\times 3}$, and for Ψ the frames are calculated with respect to $\boldsymbol{V} = \boldsymbol{Z} \in \mathbb{R}^{24\times 3}$.

Implementation details. For all methods in table 1, training was done using the ADAM [6] optimizer, with batch size 64. The number of epochs was set to 150, with 0.0001 fixed learning rate. Training was done on 2 Nvidia V-100 GPUs.

Evaluation. The quantitative measure reported in table 1, the average per-vertex distance error (MSE), is based on the official implementation of [5]. Given a collection of N test examples $\{\boldsymbol{X}_i\}_{i=1}^N$, and corresponding predictions $\{\boldsymbol{Y}_i\}_{i=1}^N$, the metric is defined by

$$\text{MSE} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left\|\boldsymbol{X}_{ij} - \boldsymbol{Y}_{ij}\right\|, \tag{1}$$

where M is the number of vertices.

A.2. Global Euclidean: Point cloud $\to$ implicit

Architecture. In this experiment, the backbone encoder architecture is the same for the baseline VAE and our Frame Averaging version. The network is based on the design and implementation of OccupencyNetworks [9]. The encoder network consists of layers of the form

$$\begin{aligned}\mathrm{FC}(d_{\text{in}}, d_{\text{out}}) &: \boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{in}}} \mapsto \nu\left(\boldsymbol{X}\boldsymbol{W} + \mathbf{1}\boldsymbol{b}^T\right) \\ \mathrm{MaxPool} &: \boldsymbol{X} \in \mathbb{R}^{n \times d} \mapsto [\max \boldsymbol{X}\boldsymbol{e}_i] \\ \mathrm{Linear}(d_{\text{in}}, d_{\text{out}}) &: \boldsymbol{V} \in \mathbb{R}^{d_{\text{in}}} \mapsto \boldsymbol{W}'\boldsymbol{V} + \boldsymbol{b}'\end{aligned}$$

where $\boldsymbol{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, $\boldsymbol{W}' \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, $\boldsymbol{b} \in \mathbb{R}^{d_{\text{out}}}$, $\boldsymbol{b}' \in \mathbb{R}^{d_{\text{out}}}$ are the learnable parameters, $\mathbf{1} \in \mathbb{R}^n$ is the vector of all ones, $[\cdot]$ is the concatenation operator, $\boldsymbol{e}_i$ is the standard basis in $\mathbb{R}^d$, and ν is the ReLU activation. Then, we define the following block:

$$\begin{aligned}&\mathrm{EnBlock}(d_{\text{in}}, d_{\text{out}}) : \boldsymbol{X} \mapsto \\ &[\mathrm{FC}(d_{\text{in}}, d_{\text{out}}/2)(\boldsymbol{X}), \mathbf{1}\mathrm{MaxPool}(\mathrm{FC}(d_{\text{in}}, d_{\text{out}}/2)(\boldsymbol{X})],\end{aligned}$$

and the encoder ϕ consists of the following blocks:

$$\begin{aligned}&\mathrm{FC}(3, 510) \to \mathrm{EnBlock}(510, 510) \to \\ &\mathrm{EnBlock}(510, 510) \to \mathrm{EnBlock}(510, 510) \to \\ &\mathrm{FC}(510, 255) \to \mathrm{MaxPool} \overset{\times 2}{\to} \mathrm{Linear}(255, 255),\end{aligned}$$

where the final two linear layers outputs vectors $(\boldsymbol{\mu}, \boldsymbol{\eta})$, with $\boldsymbol{\mu} \in \mathbb{R}^{255}$ is the latent mean, and $\boldsymbol{\eta} \in \mathbb{R}^{255}$ is the latent log-standard-deviation.

The decoder, ψ, similarly to [1, 10], is a composition of 8 layers where the first layer is $\mathrm{FC}(255 + 3, 512)$, the second and third layer are $\mathrm{FC}(512, 512)$, the fourth layer is a concatenation of $\mathrm{FC}(512, 255)$ to the input, the fifth to seventh layers are $\mathrm{FC}(512, 512)$, and the final layer is $\mathrm{Linear}(512, 1)$. For the Frame Averaging architecture of Φ and Ψ, we use the same as the above backbone encoder ϕ and decoder ψ. We set $m = 0$ and $d = 85$. For the frames construction of Φ and Ψ, $\mathcal{F} : V \to 2^{E(3)} \setminus \emptyset$, we use the construction defined in section 3.3, with $\boldsymbol{w} = \mathbf{1}$. Note that for Φ the frames are calculated with respect to the input point cloud $\boldsymbol{V} = \boldsymbol{X} \in \mathbb{R}^{6400 \times 3}$, and for Ψ the frames are calculated with respect to $\boldsymbol{V} = \boldsymbol{Z} \in \mathbb{R}^{85 \times 3}$.

Implementation details. For all methods in table 2, training was done using the ADAM [6] optimizer, with batch size 64. The number of epochs was set to 4000, with 0.0001 fixed learning rate. Training was done on 4 Nvidia QUADRO RTX 8000 GPUs.

Evaluation. We used the following Chamfer distance metrics to measure similarity between shapes:

$$\mathrm{d_C}\left(\mathcal{X}_1, \mathcal{X}_2\right) = \frac{1}{2}\left(\mathrm{d_C^{\rightarrow}}\left(\mathcal{X}_1, \mathcal{X}_2\right) + \mathrm{d_C^{\rightarrow}}\left(\mathcal{X}_2, \mathcal{X}_1\right)\right) \qquad (2)$$

where

$$\mathrm{d_C^{\rightarrow}}\left(\mathcal{X}_1, \mathcal{X}_2\right) = \frac{1}{|\mathcal{X}_1|}\sum_{\boldsymbol{x}_1 \in \mathcal{X}_1} \min_{\boldsymbol{x}_2 \in \mathcal{X}_2} \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 \qquad (3)$$

and the sets $\mathcal{X}_i$ are point clouds. As the input test set are given as 3D point clouds, we use the test point clouds directly with this metric. However, as the reconstructions are represented as implicit surfaces, we first mesh the predictions zero levelset using the $\mathrm{MarchingCubes}$ [8] algorithm, and second, we generate a point cloud of 30000 points by uniformly sample from the predicted mesh.

A.3. Piecewise Euclidean: Mesh $\to$ mesh

Architecture. Here we provide additional details regarding the architectures for the methods in table 3 applied on the DFaust splits. The AE architecture is the same as the one detailed in section A.1 for the global Euclidean mesh $\to$ mesh. For ARAPReg [5], random split, we used the trained models provided with the official implementation. Note that ARAPReg is an auto-decoder, where in test time, latent codes are optimized to reconstruct test inputs. For the Frame Averaging version with the above encoder and decoder backbones, we describe next the design choices that have been made. We set the weights $\boldsymbol{W} \in \mathbb{R}^{6890 \times 24}$ according to the template skinning weights provided with SMPL [7]. For the random split, the latent space dimensions were set to $k = 24$, $m = 12$, $d = 20$. For both the unseen poses splits the latent space dimensions were set to $k = 24$, $m = 72$, $d = 24$. For the frames of Φ, $\mathcal{F}_i : V \to 2^{E(3)} \setminus \emptyset$, with $V = \mathbb{R}^{6890 \times 3}$ and $1 \leq i \leq 24$, we use the construction defined in section 3.3. We set the weights $\boldsymbol{w}$ used for the weighted PCA of the i-th frame with $\boldsymbol{W}\boldsymbol{e}_i$, where $\boldsymbol{W}$ is the template skinning weights and $\boldsymbol{e}_i$ is the i-th element in the standard basis of $\mathbb{R}^{24}$. The frames of $\langle\psi\rangle_{\mathcal{F}}$, $\mathcal{F} : V \to 2^{E(3)} \setminus \emptyset$ (where $V = \mathbb{R}^{20 \times 3}$), are calculated with $\boldsymbol{w} = \mathbf{1}$. Note that exactly the same Frame Averaging architecture was also used in the experiment in table 4.

For the experiment with SMAL in table 3, we next provide the baseline AE architecture based on the one provided in ARAPReg. The encoder, ϕ, is consisted of the following blocks, where each block is defined as in section A.1:

$$\begin{aligned}&\mathrm{EnBlock}(3889, 1945, 3, 32) \to \\ &\mathrm{EnBlock}(1945, 973, 32, 32) \to \\ &\mathrm{EnBlock}(973, 973, 32, 32) \to \\ &\mathrm{EnBlock}(973, 973, 32, 64) \to \mathrm{Linear}(62272, 96) \to .\end{aligned}$$

The decoder, ψ, consists of the following blocks:

$$\begin{aligned}&\text{Linear}(96, 62272) \rightarrow \text{DeBlock}(973, 973, 64, 64) \rightarrow \\ &\text{DeBlock}(973, 973, 64, 32) \rightarrow \\ &\text{DeBlock}(973, 1945, 32, 32) \rightarrow \\ &\text{DeBlock}(1945, 3889, 32, 32) \rightarrow \\ &\text{Conv}(3889, 3889, 32, 3) \rightarrow .\end{aligned}$$

For our Frame Averaging architecture we set $k = 33$, $m = 12$, $d = 28$. The weights $\boldsymbol{W} \in \mathbb{R}^{3889\times 33}$ are set according to the template skinning weights provided with the dataset. The frames for the encoder and the decoder are constructed in the same fashion as the one described above for the Frame Averaging DFaust architecture.

Implementation details. For our Frame Averaging method in tables 3 and 4, training was done using the ADAM [6] optimizer, with batch size 16. The number of epochs was set to 100, with 0.0001 fixed learning rate. Training was done on 4 Nvidia QUADRO RTX 8000 GPUs. For ARAPReg, the numbers reported for the random split and SMAL in table 3 are from the original paper [5]. For the unseen pose and unseen global pose splits in table 3, we retrained the ARAPReg and AE using the official implementation training details, where the only change is that the latent size was set to 144 from 72 (to be equal to ours). The numbers reported in table 4 for the baselines SNARF [2] and NASA [4] are from table 1 in [2]. The qualitative results for SNARF in figure 5 were produced using SNARF official implementation. Note that the meshes for the qualitative results in the original SNARF paper were extracted using a *Multiresolution IsoSurface Extraction* [9], whereas the meshing in the official SNARF implementation (used in this paper) is based on the standard MarchingCubes [8] algorithm, result in with more severe staircase artifacts.

Evaluation. The definition for the MSE metric reported in 3 is the same as defined above in equation 1. The numbers reported for our method in table 3, were produced with the evaluation code from the official implementation of SNARF. For completeness, we repeat the procedure for the calculation of the IoU metric here: Let $\boldsymbol{X}$ be a ground truth shape and $\boldsymbol{Y}$ is the corresponding prediction. Then, two samples of points in space, bbox (bounding box) and near surface, are constructed. The near surface sample is constructed by adding a Gaussian noise with $\sigma = 0.01$ to a sample of 200000 points from the shape $\boldsymbol{X}$. The bbox is constructed by uniformly sampling of 100000 points in a the bounding box of all shapes in the dataset. Then, the IoU is computed by

$$\text{IoU} = \frac{1}{|S|}\sum_{\boldsymbol{x}\in S} o(\boldsymbol{x};\boldsymbol{X}) * o(\boldsymbol{x};\boldsymbol{Y})$$

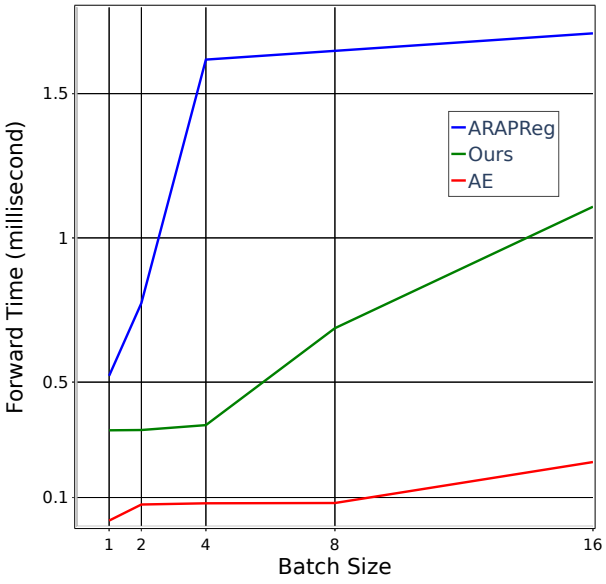

Figure A1. Timings for the methods in table 3 for various batch sizes.

where S is the prepared sample (bbox or near surface), and $o(\cdot;\boldsymbol{X}) \in \{0,1\}$, $o(\cdot;\boldsymbol{Y}) \in \{0,1\}$ are the occupancy functions of the shapes $\boldsymbol{X}$ and $\boldsymbol{Y}$.

A.4. Interpolation in shape space

Here we provide additional details regarding the interpolation in shape space. Let $\boldsymbol{Z}^{(j)} = (\boldsymbol{q}^{(j)}, \boldsymbol{Q}^{(j)}) \in Z$, $j = 0, 1$, be two latent codes. Here we describe the case $k = 1$, as for $k > 1$ the following scheme can be applied in the same manner for each part. Note that $\boldsymbol{q}^{(j)}$ is the invariant features, whereas $\boldsymbol{Q}^{(j)}$ is the equivariant features. Let $\boldsymbol{R}$ be the optimal rotation between $\hat{\boldsymbol{Q}}^{(0)}$ to $\hat{\boldsymbol{Q}}^{(1)}$, where $\hat{\boldsymbol{Q}}^{(j)}$ denotes the centered version of $\boldsymbol{Q}^{(j)}$. Then,

$$\begin{aligned}\boldsymbol{Q}_t =& \text{slerp}(t, \boldsymbol{I}, \boldsymbol{R})(\hat{\boldsymbol{Q}}^{(0)} + t\boldsymbol{D}) + \\ & t\frac{1}{d}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{Q}^{(0)} + (1-t)\frac{1}{d}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{Q}^{(1)}\end{aligned}$$

where $\boldsymbol{D} = \boldsymbol{R}^T\boldsymbol{Q}^{(1)} - \boldsymbol{Q}^{(0)}$, and $\text{slerp}(t, \boldsymbol{I}, \boldsymbol{R})$ denotes the spherical liner interpolation between $\boldsymbol{I}$ to $\boldsymbol{R}$. For the invariant features, we perform regular linear interpolation $\boldsymbol{q}_t = t\boldsymbol{q}^{(0)} + (1-t)\boldsymbol{q}^{(1)}$. Then, the interpolant at time t is

$$\boldsymbol{Z}_t = (\boldsymbol{q}_t, \boldsymbol{Q}_t).$$

A.5. Timings

In figure A1 we report the training forward time for various batch sizes for the methods in table 3. Note that the increase in forward time for our method with respect to the baseline AE, stems from the computation of the backbone network for all the different parts and possible elements in

the frame. Note that for ARAPReg, the forward time in test reduces significantly as the computation of the ARAP regularizer is not needed. However, note that in the case of ARAReg with an AutoDecoder, prediction time increases significantly as it requires optimizing the latent codes.

B. Proofs

Proof of Proposition 1. The proof basically repeats the proof in [11] but for the weighted case. Let $g = (\boldsymbol{S}, \boldsymbol{u})$ be an element in $E(3)$. We need to show that $\mathcal{F}(\rho_1(g)\boldsymbol{V}) = g\mathcal{F}(\boldsymbol{V})$. The above equality for the translations part $\mathbb{R}^3$ in the group $E(3) = O(3) \ltimes \mathbb{R}^3$ is trivial. For showing the above equality for $O(3)$, Let $\boldsymbol{C_V}$ be the covariance matrix of $\boldsymbol{V}$. That is $\boldsymbol{C_V} = (\boldsymbol{V} - \frac{\mathbf{1}\boldsymbol{w}^T}{\mathbf{1}^T\boldsymbol{w}}\boldsymbol{V})^T \mathrm{diag}(\boldsymbol{w})(\boldsymbol{V} - \frac{\mathbf{1}\boldsymbol{w}^T}{\mathbf{1}^T\boldsymbol{w}}\boldsymbol{V})$. A direct calculation shows that

$$\boldsymbol{C_V} = \boldsymbol{V}^T \left[\mathrm{diag}(\boldsymbol{w}) \left(\boldsymbol{I} - \frac{\mathbf{1}\boldsymbol{w}^T}{\mathbf{1}^T\boldsymbol{w}}\right)\right] \boldsymbol{V}.$$

Then, the covariance matrix of $\rho_1(g)\boldsymbol{V}$ satisfies

$$\boldsymbol{C}_{\rho_1(g)\boldsymbol{V}} = \boldsymbol{S}^T\boldsymbol{V}^T \left[\mathrm{diag}(\boldsymbol{w}) \left(\boldsymbol{I} - \frac{\mathbf{1}\boldsymbol{w}^T}{\mathbf{1}^T\boldsymbol{w}}\right)\right] \boldsymbol{V}\boldsymbol{S}.$$

Thus, $\boldsymbol{r}$ is an eigen vector of $\boldsymbol{C_V}$ if and only if $\boldsymbol{S}\boldsymbol{r}$ is an eigen vector of $\boldsymbol{C}_{\rho_1(g)\boldsymbol{V}}$. To finish the proof, notice that by definition $g\mathcal{F}(\boldsymbol{V}) = \{(\boldsymbol{S}\boldsymbol{R}, \boldsymbol{S}\boldsymbol{t} + \boldsymbol{u}) \mid \boldsymbol{R} = [\pm\boldsymbol{r}_1, \pm\boldsymbol{r}_2, \pm\boldsymbol{r}_3]\}$.

□

Proof of Proposition 2. We need to show that

$$\Psi(\rho_Z(g)\boldsymbol{Z}) = \rho_Y(g)\Psi(\boldsymbol{Z})$$

which is equivalent to showing that for all $\boldsymbol{x} \in \mathbb{R}^3$

$$\hat{\Psi}(\rho_Z(g)\boldsymbol{Z}, \boldsymbol{x}) = \hat{\Psi}(\boldsymbol{Z}, \boldsymbol{R}^T(\boldsymbol{x} - \boldsymbol{t}))$$

which is again equivalent to showing that for all $\boldsymbol{x} \in \mathbb{R}^3$

$$\hat{\Psi}(\rho_Z(g)\boldsymbol{Z}, \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}) = \hat{\Psi}(\boldsymbol{Z}, \boldsymbol{x})$$

which by definition of ρ_V is finally equivalent to showing for all $\boldsymbol{x} \in \mathbb{R}^3$

$$\hat{\Psi}(\rho_P(g)(\boldsymbol{Z}, \boldsymbol{x})) = \rho_{\mathbb{R}}(g)\hat{\Psi}(\boldsymbol{Z}, \boldsymbol{x})$$

as required. □

Proof of Theorem 1. If $g \in \mathcal{F}_j(\boldsymbol{X})$ then $g = (\boldsymbol{R}, \boldsymbol{t})$, where $\boldsymbol{t} = (\mathbf{1}^T\boldsymbol{w}_j)^{-1}\boldsymbol{X}^T\boldsymbol{w}_j$ as defined in equation 7. Therefore

$$\begin{aligned}\rho_X(g)^{-1}\boldsymbol{X}_j &= ((\mathbf{1} - \boldsymbol{w}_j)\boldsymbol{t}^T + \boldsymbol{w}_j \odot \boldsymbol{X} - \mathbf{1}\boldsymbol{t}^T)\boldsymbol{R} \\ &= \boldsymbol{w}_j \odot (\boldsymbol{X} - \mathbf{1}\boldsymbol{t}^T)\boldsymbol{R} \\ &= \boldsymbol{w}_j \odot \rho_X(g)^{-1}\boldsymbol{X}\end{aligned}$$

Furthermore, it can be directly checked that for hard weights $\mathcal{F}_j(\boldsymbol{X})$ is an equivariant frame that is defined only in terms of the rows of $\boldsymbol{X}$ that belong to the j-th part. Now, $\boldsymbol{Z}_j = \langle\phi\rangle_{\mathcal{F}_j}(\boldsymbol{X}_j)$ where

$$\begin{aligned}\langle\phi\rangle_{\mathcal{F}_j}(\boldsymbol{X}_j) &= \frac{1}{|\mathcal{F}_j(\boldsymbol{X})|} \sum_{g \in \mathcal{F}_j(\boldsymbol{X})} \rho_{Z_j}(g)\phi(\rho_X(g)^{-1}\boldsymbol{X}_j) \\ &= \frac{1}{|\mathcal{F}_j(\boldsymbol{X})|} \sum_{g \in \mathcal{F}_j(\boldsymbol{X})} \rho_{Z_j}(g)\phi(\boldsymbol{w}_j \odot \rho_X(g)^{-1}\boldsymbol{X}).\end{aligned}$$

Therefore Frame Averaging now implies that $\langle \boldsymbol{X}_j \rangle_{\mathcal{F}}$ is equivariant to Euclidean motions of the j-th part. For the decoder, $\langle \boldsymbol{Z}_j \rangle_{\mathcal{F}}$ is equivariant to Euclidean motions of $\boldsymbol{Z}_j$ by Frame Averaging, and $\boldsymbol{w}_j \odot \rho_Y(g_j)\boldsymbol{Y}$ transforms the decoded j-th part accordingly. □

C. Additional results

We provide additional qualitative results for some of the experiments in section 5. Figure A2 shows additional qualitative results for the comparison with an implicit pose-conditioned method (SNARF). Figure A3 shows typical test reconstructions on the SMAL dataset. Note that the test set here does not include out of distribution poses, hence relatively easy. Note that our method is able to capture the pose better than the baselines (see the animal's tail for example), while achieving high fidelity (see base of the feet).

References

[1] Matan Atzmon and Yaron Lipman. SALD: sign agnostic learning with derivatives. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[2] Xu Chen, Yufeng Zheng, Michael J Black, Otmar Hilliges, and Andreas Geiger. Snarf: Differentiable forward skinning for animating non-rigid neural implicit shapes. *arXiv preprint arXiv:2104.03953*, 2021.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

[4] Boyang Deng, John P Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. Nasa neural articulated shape approximation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 612–628. Springer, 2020.

[5] Qixing Huang, Xiangru Huang, Bo Sun, Zaiwei Zhang, Junfeng Jiang, and Chandrajit Bajaj. Arapreg: An as-rigid-as possible regularization loss for learning deformable shape generators. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5815–5825, 2021.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, Oct. 2015.

[8] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[9] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[10] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[11] Omri Puny, Matan Atzmon, Heli Ben-Hamu, Edward J Smith, Ishan Misra, Aditya Grover, and Yaron Lipman. Frame averaging for invariant and equivariant network design. *arXiv preprint arXiv:2110.03336*, 2021.

[12] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J Black. Generating 3d faces using convolutional mesh autoencoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 704–720, 2018.

[13] Silvia Zuffi, Angjoo Kanazawa, David W Jacobs, and Michael J Black. 3d menagerie: Modeling the 3d shape and pose of animals. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6365–6373, 2017.

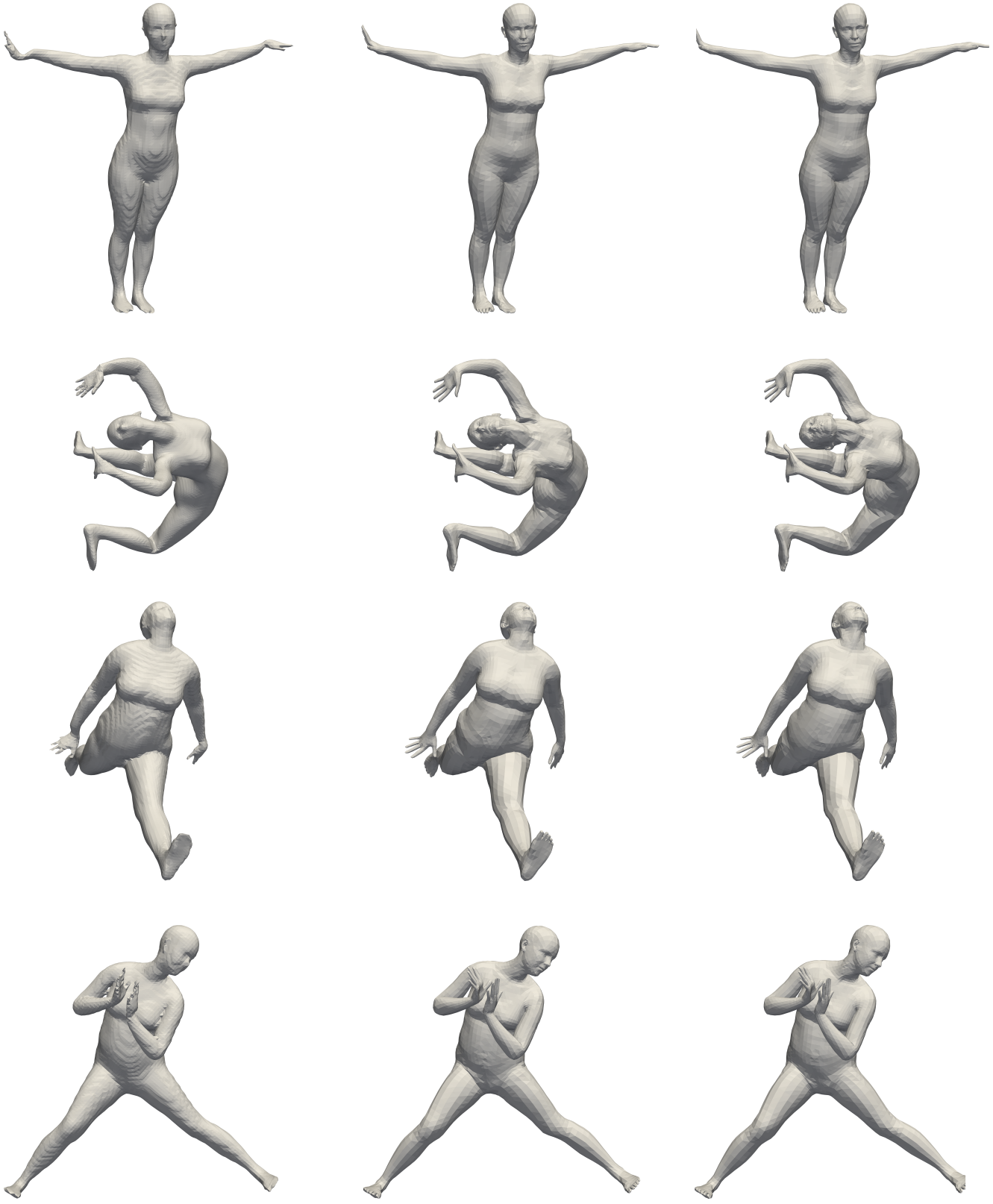

Figure A2. Comparison to SNARF on ”out of distribution” test.

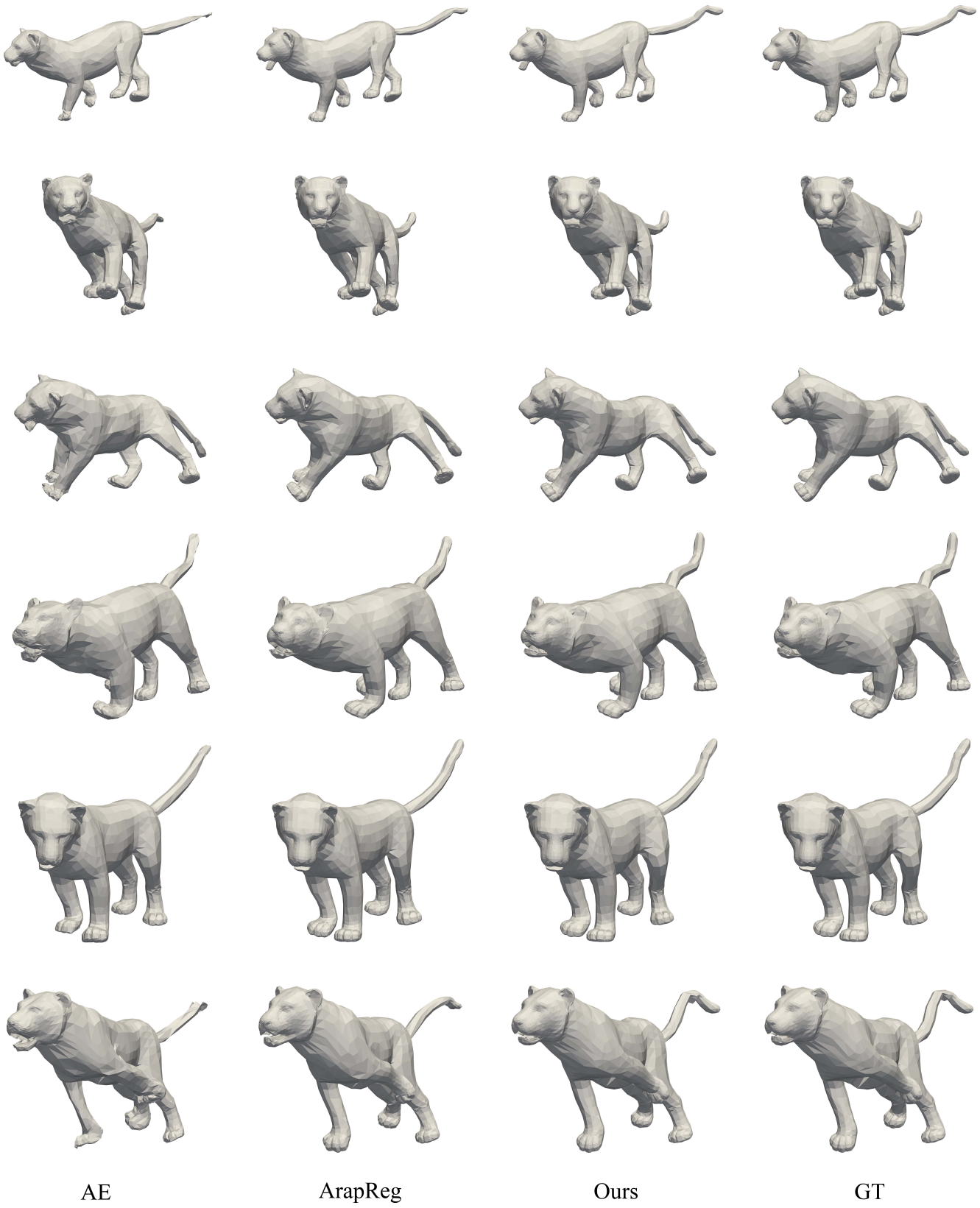

Figure A3. Piecewise Euclidean mesh → mesh, qualitative results; SMAL [13] dataset.