ARC-Flow : Articulated, Resolution-Agnostic, Correspondence-Free Matching and Interpolation of 3D Shapes Under Flow Fields

Supplementary Material

S1. Implementation Details

In this section, we provide implementation details that were omitted from the main text due to space constraints.

S1.1. ARC-Net Architecture

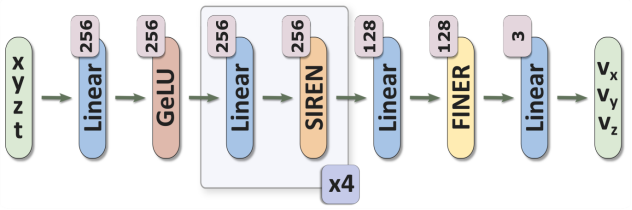

Figure 8. Overview of the ARC-Net Architecture.

The ARC-Net introduced in Section 4.1 comprises an MLP with 4 SIREN layers and one FINER layer, with widths of 256 and 128 respectively, as shown in Fig. 8. SIREN [64] uses a sine as a periodic activation function such that $\mathbf{z}_i : \mathbb{R}^{M_i} \rightarrow \mathbb{R}^{N_i}$ is the i-th layer of the network is defined as,

$$\mathbf{z}_i = \sin(\omega_0(\mathbf{W}_i\mathbf{z}_{i-1} + \mathbf{b}_i)) \,, \tag{18}$$

where $\mathbf{z}_{i-1}$ denotes the output of layer $i-1$ and ω_0 is a user defined parameter for controlling the frequency. For ARC-NET, we use $\omega_0 = 4.0$.

However, the standard formulation exhibits a well-documented spectral bias, wherein the network preferentially learns low-frequency components of the signal. While this bias can be advantageous for learning smooth flow fields, it potentially limits the network's ability to handle fine grain deviations required to handle intricate surface details on target surfaces. Thus, to address this limitation, we append a FINER layer [44] as the final layer, which replaces SIREN's conventional sine activation with a variable-periodic activation function,

$$\begin{aligned} \mathbf{z}_i &= \sin(\omega_0\alpha_i(\mathbf{W}_i\mathbf{z}_{i-1} + \mathbf{b}_i)) \,, \\ \text{where } \alpha_i &= |\mathbf{W}_i\mathbf{z}_{i-1} + \mathbf{b}_i| + 1 \end{aligned} \tag{3}$$

S1.2. Q-Net Architecture

Rotations in three-dimensional space can be represented through various mathematical formulations, including Euler angles, rotation matrices, axis-angle vectors, and unit quaternions. Our model employs unit quaternions due to their compact representation and straightforward geometric and algebraic properties. However, despite these advantages, it is known that naively attempting to learn large rotations via a neural network is problematic due to a critical limitation in the smoothness characteristics of unit quaternions, which can impede the network's ability to accurately learn the correct rotation.

Thus, the Q-Net discussed in Section 4.4 leverages the work of Peretroukhin *et al*. [53]. The network architecture comprises three fully connected layers, each with a width of 128 neurons using Tanh activation functions. Provided with a 4d input, (x, y, z, t), the output of the MLP produces a 10-dimension output which is used to construct the following 4×4 symmetric matrix,

$$\mathbf{A}(\theta) = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 \\ \theta_2 & \theta_5 & \theta_6 & \theta_7 \\ \theta_3 & \theta_6 & \theta_8 & \theta_9 \\ \theta_4 & \theta_7 & \theta_9 & \theta_{10} \end{bmatrix} , \tag{19}$$

This represents the set of real symmetric 4×4 matrices with a simple (i.e., non-repeated) minimum eigenvalue:

$$\mathbf{A} \in \mathbb{S}^4 : \lambda_1(\mathbf{A}) \neq \lambda_2(\mathbf{A}), \tag{1}$$

where λ_i are the eigenvalues of $\mathbf{A}$ arranged such that $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \lambda_4$, and $\mathbb{S}^n \triangleq \{\mathbf{A} \in \mathbb{R}^{n\times n} : \mathbf{A} = \mathbf{A}^\top\}$.

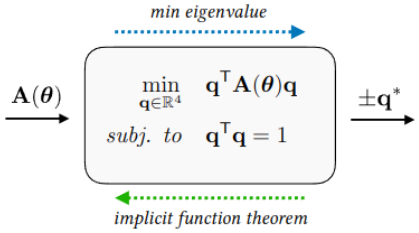

***Figure 9. QCQP layer** - Image Credit [53]*

$\mathbf{A}(\theta)$ is mapped to a unique rotation through a differentiable QCQP layer, illustrated in Figure 9. This layer predicts a quaternion as a solution derived from the minimum-eigenspace of $\mathbf{A}$ and the implicit function theorem allows for an analytic gradient to be computed for use as part of back-propagation,

$$\frac{\partial \mathbf{q}^*}{\partial \text{vec}(\mathbf{A})} = \mathbf{q}^* \otimes (\lambda_1 \mathbf{I} - \mathbf{A})^\dagger \,, \tag{9}$$

where $(\cdot)^\dagger$ denotes the Moore-Penrose pseudo-inverse, $\otimes$ is the Kronecker product, and $\mathbf{I}$ refers to the identity matrix.

S1.3. Skeleton Parameterisation

We utilise the skeleton provided with each dataset with a minor adjustment. To enhance realism, we extend the existing skeleton by adding bones at the extremities; specifically

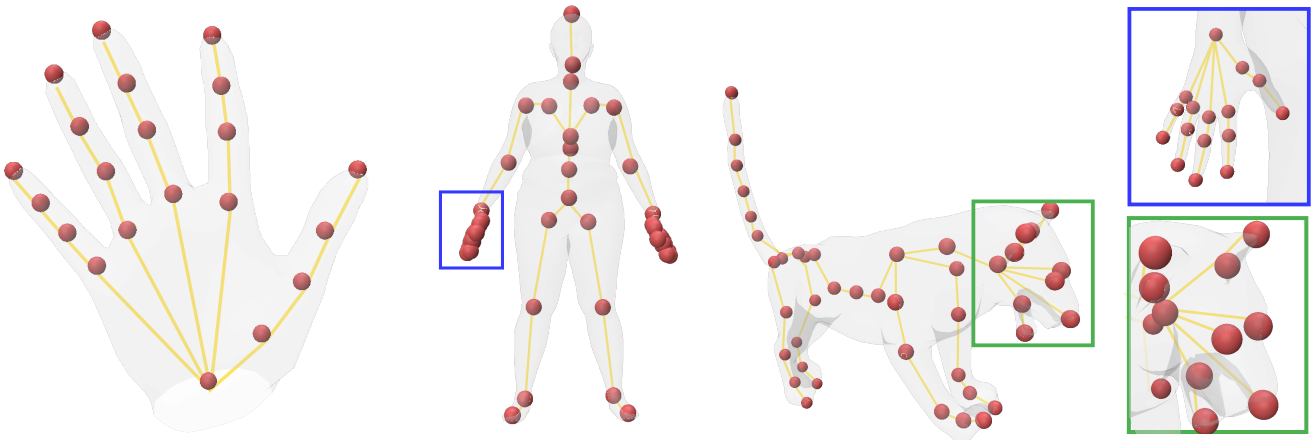

Figure 10. The simple skeletons, $\mathbb{S}_{\mathcal{X}}$, based upon skeleton provided with each dataset, used to augment the source shapes in our method. Left: MANO. *consisting of 21 joints and 21 bones.* ***Centre: DFAUST.*** *consisting of 53 joints and 52 bones.* ***Right: SMAL.*** *consisting of 44 joints and 43 bones.*

in the fingers, feet, and paws. This augmentation results in a more anatomically accurate representation, more closely mimicking real-life skeletal structures as shown in Fig. 10.

The exact regions defined as rigid / soft tissue and the number of samples used are summarised in Table 3. In general, for each bone, we use 50 samples for the bone, $\{\alpha_m\}$, and the soft tissue, $\{c_m\}$, components, while 500 samples are used for the surface points, $\{x_i\}$. All of which are randomly resampled in every epoch. We use a radius of 10% of the length of each bone and between 10% and 25% depending on the dataset for the soft tissue region. Human fingers are relative to the rest of a human body long and narrow, thus for DFAUST dataset they require significantly smaller regions to stay within the surface.

S1.4. Training details

In this section, we provide further details of the training procedure and parameters used to fit our model to the various datasets tested, a summary which is provided in Table 4.

As discussed in the main text, the loss function, Equation (17), is optimised using the VectorAdam [41] algorithm and the ODE modelling of the flow field is solved using a probabilistic ODE solver, specifically the Kronecker EK0 formulation with a single derivative operating with a smoother strategy. We use a variable learning rate, which is controlled via a warmup cosine decay schedule, in which 50 epochs are assigned to the warm up stage and the initial and final rates for different datasets are shown in the aforementioned table.

The model is optimised in two stages; main and fine-tuning (FT). During the main phase, after 1k, 2k and 3k epochs, the length scales of the Varifold kernels are adjusted in a coarse to fine manner, after which it is held constant for the remainder of the optimisation. Although many parameters vary slightly depending on the dataset, the weightings of $\mathcal{L}_{\text{bone}}$, $\mathcal{L}_{\text{soft}}$, and $\mathcal{L}_{\text{surf}}$, represented by $\boldsymbol{\lambda_1}$, $\boldsymbol{\lambda_2}$, and $\boldsymbol{\lambda_3}$ respectively, are consistent across all datasets.

Following the completion of the main stage, the values of $\boldsymbol{\lambda_1}$ and $\boldsymbol{\lambda_2}$ are increased (all other parameters are held constant), and the network is trained for an additional 2k epochs with these increased values. This additional fine-tuning step was found to improve the quality of the interpolation, both qualitatively and quantitatively (via the conformal metric). Starting initially with these higher values was problematic, as they place a high cost on any initial deformation of the source surface towards the target.

Dataset	Rigid $\{\alpha_m\}$		Soft Tissue $\{c_m\}$	
	Radius	# Samples	Radius	# Samples
MANO	0.1	50	0.25	50
DFAUST (Body)	0.1	50	0.25	25
DFAUST (Hands)	0.01	50	0.02	25
SMAL	0.1	50	0.15	50

Table 3. Parameters for Rigid & Soft Tissue Sampling: *The radii as defined in terms of the percentage of the bone length and used for both the bone and soft tissue regions.*

S2. Quaternion Interpolation Derivation

Section 4.3 discusses how the locations of samples modelling bones are interpolated under quaternion rotations. In this section, we provide additional mathematical background on the derivation of these formulae.

Given a point $\mathbf{p}_0$ and a quaternion representing rotation $\mathbf{q}$, the position of the point after the rotation has been applied is:

$$p_1 = \mathbf{q} \cdot \mathbf{p}_0 \cdot \mathbf{q}^* \tag{20}$$

Dataset	Epochs		LR		ODE	Initial		Epoch 1k		Epoch 2k		Epoch 3k	
	Main	FT	Init	Final	Steps	ℓ_{κ_x}	ℓ_{κ_n}	ℓ_{κ_x}	ℓ_{κ_n}	ℓ_{κ_x}	ℓ_{κ_n}	ℓ_{κ_x}	ℓ_{κ_n}
MANO	4k	2k	1e-2	1e-3	10	0.5	0.5	0.1	0.5	0.1	0.4	0.1	0.3
DFAUST	5k	2k	5e-3	1e-4	15	0.5	0.5	0.25	0.5	0.1	0.4	0.1	0.3
SMAL	4k	2k	5e-3	1e-4	10	0.5	0.5	0.25	0.5	0.1	0.4	0.1	0.3

Opt	λ_1	λ_2	λ_3
Main	2e2	1e1	5e3
FT	1e3	1e2	5e3

Table 4. Left: *Hyper-parameters & Varifold settings used for training on each dataset.* ***Right:*** *Loss function weightings for the two stages of the optimisation; main & fine tuning, across all datasets.*

If we assume that over $t \in [0,1]$ we apply a rotation of $\mathbf{q}$ and translation of $\mathbf{T}$, then the path traced out will be

$$\mathbf{p}(t) = t\mathbf{T} + \mathbf{q}(t) \cdot \mathbf{p}_0 \cdot \mathbf{q}^*(t) \tag{21}$$

Now we want to find the velocity field $\mathbf{f}_{\mathbf{q}}(\mathbf{p}, t)$ that will trace out the same trajectory for any initial $\mathbf{p}_0$. Thus we have

$$\mathbf{p}_T = \mathbf{p}_0 + \int_0^t \mathbf{f}(\mathbf{p}, t)\, dt \tag{22}$$

Therefore,

$$\int_0^t \mathbf{f}(\mathbf{p}, t)\, dt = \mathbf{p}(t) - \mathbf{p}_0 = t\mathbf{T} + \mathbf{q}(t) \cdot \mathbf{p}_0 \cdot \mathbf{q}^*(t) \tag{23}$$

And hence,

$$\frac{\partial}{\partial t} \Rightarrow \mathbf{f}(\mathbf{p}, t) = \mathbf{T} + \frac{\partial}{\partial t}\Big[\mathbf{q}(t) \cdot \mathbf{p}_0 \cdot \mathbf{q}^*(t)\Big] \tag{24}$$

By product rule,

$$\frac{\partial}{\partial t}\Big[\mathbf{q}(t)\cdot\mathbf{p}_0\cdot\mathbf{q}^*(t)\Big] = \frac{\partial \mathbf{q}\,(t)}{\partial t}\cdot\mathbf{p}_0\cdot\mathbf{q}^*(t) + \mathbf{q}(t)\cdot\mathbf{p}_0\cdot\frac{\partial \mathbf{q}^*(t)}{\partial t} \tag{25}$$

Now, interpolating between between a unit identity quaternion and some new quaternion $\mathbf{q}$,

$$\mathbf{q}(t) = \text{SLERP}(\mathbf{1}, \mathbf{q}_0, t) = \left(\cos\frac{\alpha t}{2}, \vec{\mathbf{n}}\sin\frac{\alpha t}{2}\right) = \mathbf{q}_0^t\,, \tag{26}$$

where $0 \leq t \leq T$.

For a unit quaternion,

$$\mathbf{q}^t = \exp(\ln(\mathbf{q}) * t) \tag{27}$$

The derivative of the function $\mathbf{q}$, where $\mathbf{q}$ is a constant unit quaternion is,

$$\frac{\partial}{\partial t}\mathbf{q}^t = \ln(\mathbf{q}) \cdot \mathbf{q}^t = \ln(\mathbf{q}) \cdot \exp(\ln(\mathbf{q}) * t) \tag{28}$$

, where the quaternion forms of exp, log and to a power are,

$$\exp(\mathbf{q}) = \exp(s)\begin{pmatrix} \cos(|\mathbf{v}|) \\ \frac{\mathbf{v}}{|\mathbf{v}|}\sin(|\mathbf{v}|) \end{pmatrix}. \tag{29}$$

$$\ln(\mathbf{q}) = \begin{pmatrix} \ln(|\mathbf{q}|) \\ \frac{\mathbf{v}}{|\mathbf{v}|}\arccos\left(\frac{s}{|\mathbf{q}|}\right) \end{pmatrix}. \tag{30}$$

$$\mathbf{q}^{\mathbf{P}} = \exp(\ln(\mathbf{q}) \cdot \mathbf{p})\,. \tag{31}$$

Note that if p is in fact scalar, then the power is

$$\mathbf{q}^p = \exp(\ln(\mathbf{q}) * p)\,. \tag{32}$$

S3. Sparse Nyström Varifold Approximation

This section details the Sparse Nyström Approximation algorithm introduced by Paul *et al*. [52]. We recall Sec. 4.2 where we describe the varifold matching loss

$$\begin{aligned} \mathcal{L}_{\text{var}}(\theta) &:= d(\boldsymbol{\mathcal{X}}^{(T)}, \boldsymbol{\mathcal{Y}}) \\ &= \langle \mu_{\boldsymbol{\mathcal{X}}^{(T)} - \mu_{\boldsymbol{\mathcal{Y}}}}, \mu_{\boldsymbol{\mathcal{X}}^{(T)}} - \mu_{\boldsymbol{\mathcal{Y}}} \rangle_{\mathbb{V}^*} \\ &= \langle \mu_{\boldsymbol{\mathcal{X}}^{(T)}}, \mu_{\boldsymbol{\mathcal{X}}^{(T)}} \rangle_{\mathbb{V}^*} - 2\langle \mu_{\boldsymbol{\mathcal{X}}^{(T)}}, \mu_{\boldsymbol{\mathcal{Y}}} \rangle_{\mathbb{V}^*} \\ &\quad + \langle \mu_{\boldsymbol{\mathcal{Y}}}, \mu_{\boldsymbol{\mathcal{Y}}} \rangle_{\mathbb{V}^*}\,. \end{aligned} \tag{33}$$

Again, $\boldsymbol{\mathcal{X}}^{(T)}$ denotes the set of vertices, normals and differential surface areas that are obtained at time $t = T$ from pushing $\boldsymbol{\mathcal{X}}$ through the $\mathsf{ODE}_{\mathsf{Solve}}$ in Sec. 4.1. In practice, we do not need to calculate $\langle \mu_{\boldsymbol{\mathcal{Y}}}, \mu_{\boldsymbol{\mathcal{Y}}} \rangle_{\mathbb{V}^*}$ as it is a constant due to the target $\boldsymbol{\mathcal{Y}}$ remaining unchanged.

For the first two terms in Eq. (33), we use a discrete approximation of the inner product integrals in Eq. (5)

$$\langle \mu_{\boldsymbol{\mathcal{X}}}, \mu_{\boldsymbol{\mathcal{Y}}} \rangle \approx \sum_{i=1}^{I_{\boldsymbol{\mathcal{X}}}} \sum_{i'=1}^{I_{\boldsymbol{\mathcal{Y}}}} \kappa_{\mathbf{x}}(\mathbf{x}_i, \mathbf{y}_{i'})\, \kappa_{\mathbf{n}}(\hat{\mathbf{n}}_{\boldsymbol{\mathcal{X}}_i}, \hat{\mathbf{n}}_{\boldsymbol{\mathcal{Y}}_{i'}}) s_{\boldsymbol{\mathcal{X}}_i}\, s_{\boldsymbol{\mathcal{Y}}_{i'}}\,, \tag{34}$$

where the summation has the computational complexity $\mathcal{O}(I_{\boldsymbol{\mathcal{X}}}\, I_{\boldsymbol{\mathcal{Y}}})$; this cost becomes very expensive when fitting to a very high resolution target (e.g. fitting a template to a raw point cloud scan) where the number of vertices on the target is far larger than the template, $I_{\boldsymbol{\mathcal{Y}}} \gg I_{\boldsymbol{\mathcal{X}}}$ and we seek to reduce this cost.

Sparse Approximation: The algorithm of Paul *et al*. [52] creates sparse approximations of Varifold representations significantly smaller than the input data while maintaining high accuracy. It employs a Ridge Leverage Score (RLS) approach to assess a data point's importance, which is used to create a Nyström approximation for the in the Reproducing Kernel Hilbert Space (RKHS), offering a computationally efficient and accurate approach to shape compression. For a comprehensive understanding of the theoretical foundations, including detailed mathematical proofs, readers are referred to the original paper.

Application to the target, results in a compressed approximation $\boldsymbol{\mathcal{Y}}^c = \{V_{\boldsymbol{\mathcal{S}}_{\boldsymbol{\mathcal{Y}}}\rfloor}, N_{\boldsymbol{\mathcal{S}}_{\boldsymbol{\mathcal{Y}}}\rfloor}, \boldsymbol{\beta}\}$ containing a subset of $I^c_{\boldsymbol{\mathcal{Y}}}$

of the original vertices and normals with a corresponding set of weights $\boldsymbol{\beta}$. The Varifold matching representation from Eq. (34) to a compressed target becomes

$$\langle \mu_{\boldsymbol{\mathcal{X}}}, \mu_{\boldsymbol{\mathcal{Y}}^c} \rangle \approx \sum_{i=1}^{I_{\boldsymbol{\mathcal{X}}}} \sum_{i'=1}^{I_{\boldsymbol{\mathcal{Y}}^c}} \kappa_{\mathbf{x}}(\mathbf{x}_i, \mathbf{y}_{i'}^c) \kappa_{\mathbf{n}}(\hat{\mathbf{n}}_{\boldsymbol{\mathcal{X}}_i}, \hat{\mathbf{n}}_{\boldsymbol{\mathcal{Y}}_{i'}^c}) s_{\boldsymbol{\mathcal{X}}_i} \beta_{\boldsymbol{\mathcal{Y}}_j^c} , \tag{35}$$

where $I_{\boldsymbol{\mathcal{Y}}}^c \ll I_{\boldsymbol{\mathcal{Y}}}$ dramatically reducing the computational cost of calculating the Varifold loss required.

ALGORITHM 1: Varifold Compression Algorithm

Input:
$\boldsymbol{\mathcal{Y}}$: Uncompressed Data - $\{V_{\boldsymbol{\mathcal{Y}}}, N_{\boldsymbol{\mathcal{Y}}}, dS_{\boldsymbol{\mathcal{Y}}}\}$
n: Number of Samples in $\boldsymbol{\mathcal{Y}}$ $(= I_{\boldsymbol{\mathcal{Y}}})$
m: Desired Compressed Size $(= I_{\boldsymbol{\mathcal{Y}}^c})$
λ: Regularisation parameter
$\kappa_{\mathbf{x}}(\cdot,\cdot)$: Positional Kernel
$\kappa_{\mathbf{n}}(\cdot,\cdot)$: Normal Kernel
Output:
$\boldsymbol{\mathcal{Y}}^c$: Compressed Representation - $\{V_{\boldsymbol{\mathcal{Y}}^c}, N_{\boldsymbol{\mathcal{Y}}^c}, \boldsymbol{\beta}\}$

► **Compute leverage scores:**
$b_s \leftarrow \lfloor \sqrt{n} \rfloor$;
$b \leftarrow \lfloor n/s \rfloor$;
Split $\boldsymbol{\mathcal{Y}}$ into b random batches $\{\boldsymbol{\mathcal{Y}}_1, \ldots, \boldsymbol{\mathcal{Y}}_b\}$ of size b_s;
for $j = 1$ **to** b **do**
  for $i = 1$ **to** b_s **do**
    $\boldsymbol{\Lambda}_i \leftarrow \mathbf{K}_{i,\boldsymbol{\mathcal{Y}}_j} (\mathbf{K}_{i,\boldsymbol{\mathcal{Y}}_j} + \lambda \mathbf{I})^{-1}$;
    ► $\mathbf{K}_{i,\boldsymbol{\mathcal{Y}}_j} = \sum_{i' \in \boldsymbol{\mathcal{Y}}_j} \kappa_{\mathbf{x}}(\mathbf{y}_i, \mathbf{y}_{i'}) \kappa_{\mathbf{n}}(\hat{\mathbf{n}}_{\boldsymbol{\mathcal{Y}}_i}, \hat{\mathbf{n}}_{\boldsymbol{\mathcal{Y}}_{i'}})$
  end
end

► **Draw weighted samples:**
Define sampling distribution:
Let $X_i \sim p(W)$ where $p(X_i = s_j) = \frac{\boldsymbol{\Lambda}_j}{\sum_{k=1}^n \boldsymbol{\Lambda}_k}$;
$\boldsymbol{\mathcal{C}} \leftarrow \{\,\}$;
for $i = 1$ **to** m **do**
  $\{\mathbf{x}_s, \hat{\mathbf{n}}_{\boldsymbol{\mathcal{Y}}_s}\} \leftarrow$ Draw a sample from $\boldsymbol{\mathcal{Y}}$ acc. to $p(W)$;
  Add $\{\mathbf{x}_s, \hat{\mathbf{n}}_{\boldsymbol{\mathcal{Y}}_s}\}$ to $\boldsymbol{\mathcal{C}}$;
end

► **Calculate sample weights:**
$\boldsymbol{\beta} \leftarrow \mathbf{K}_{\boldsymbol{\mathcal{C}},\boldsymbol{\mathcal{C}}}^{-1} \mathbf{Y}$;
► $\mathbf{K}_{\boldsymbol{\mathcal{C}},\boldsymbol{\mathcal{C}}} = \sum_{i=1}^{I_{\boldsymbol{\mathcal{C}}}} \sum_{i'=1}^{I_{\boldsymbol{\mathcal{C}}}} \kappa_{\mathbf{x}}(\mathbf{c}_i, \mathbf{c}_{i'}) \kappa_{\mathbf{n}}(\hat{\mathbf{n}}_{\boldsymbol{\mathcal{C}}_i}, \hat{\mathbf{n}}_{\boldsymbol{\mathcal{C}}_{i'}})$
► $\mathbf{Y} = \sum_{i=1}^{I_{\boldsymbol{\mathcal{C}}}} \sum_{i'=1}^{I_{\boldsymbol{\mathcal{Y}}}} \kappa_{\mathbf{x}}(\mathbf{c}_i, \mathbf{x}_{i'}) \kappa_{\mathbf{n}}(\hat{\mathbf{n}}_{\boldsymbol{\mathcal{C}}_i}, \hat{\mathbf{n}}_{\boldsymbol{\mathcal{Y}}_{i'}}) s_{\boldsymbol{\mathcal{Y}}_i} s_{\boldsymbol{\mathcal{Y}}_{i'}}$

return $\{V_{\boldsymbol{\mathcal{Y}}^c}, N_{\boldsymbol{\mathcal{Y}}^c}, \boldsymbol{\beta}\}$

Compression Process: The compression process, as outlined in Algorithm 1, consists of three main steps:

1. The RLS values for each input element are generated efficiently using a sampling method that avoids calculating the full all-pairwise matrices.
2. Control points are then selected using a weighted sampling approach, with the RLS score determining the probability of selection.
3. Updated weights are calculated for each selected control point.

Several parameters are required as input for the compression process. Firstly, the desired compression size $m < I_{\boldsymbol{\mathcal{Y}}}$ determines the final number of control points. Additionally, the length scales of the kernels, $\kappa_{\mathbf{x}}$ and $\kappa_{\mathbf{n}}$, need to be set (they are the same as in the matching algorithm $\ell_{\mathbf{x}}$ and $\ell_{\mathbf{n}}$). Finally, an approximation parameter λ is required; we used a default value of 1 for all our experiments.

S4. Additional results

This section presents additional results that were omitted from the main results section due to space limitations.

S4.1. More Quantitative Results

In Figures 11 to 13 the interpolation results for our method against the state-of-the-art SMS [12] and ESA [28] methods for the three datasets; MANO, DFAUST and SMAL, showing the mean and confidence intervals for all metrics.

Our method demonstrates improvement across all metrics for each dataset, showing both higher mean values and reduced variance in results.

The reduction in variance of our method can be attributed primarily to our method's superior performance on more challenging problems. This is illustrated in Fig. 14, where we plot individual curves for interpolations where we colour each line with a "difficulty rating" calculated based on the average vertex displacement (normalised to one). When dealing with shapes that undergo a larger degree of deformation, competing methods show a significant drop in the quality of their results; in contrast, our approach maintains its effectiveness, leading to more consistent performance across varying levels of problem difficulty. All approaches show a roughly monotone increase in performance as the difficulty rating decreases.

S4.2. Further Qualitative Results

In this section we provide additional qualitative results, highlighting some of the advantages of our method which may not be accounted for by performance metrics alone.

S4.3. Non-Intersection of Surfaces

In Fig. 15, we demonstrate how our method handles a leg lift scenario where the leg comes into contact with the stomach of the individual. Since we model deformation as a diffeomorphism, represented by a time-varying vector flow field, our approach guarantees non-intersection by construction. In contrast, the interpolation produced by the ESA method fails to maintain surface integrity, resulting in unrealistic overlapping and severe mesh distortions.

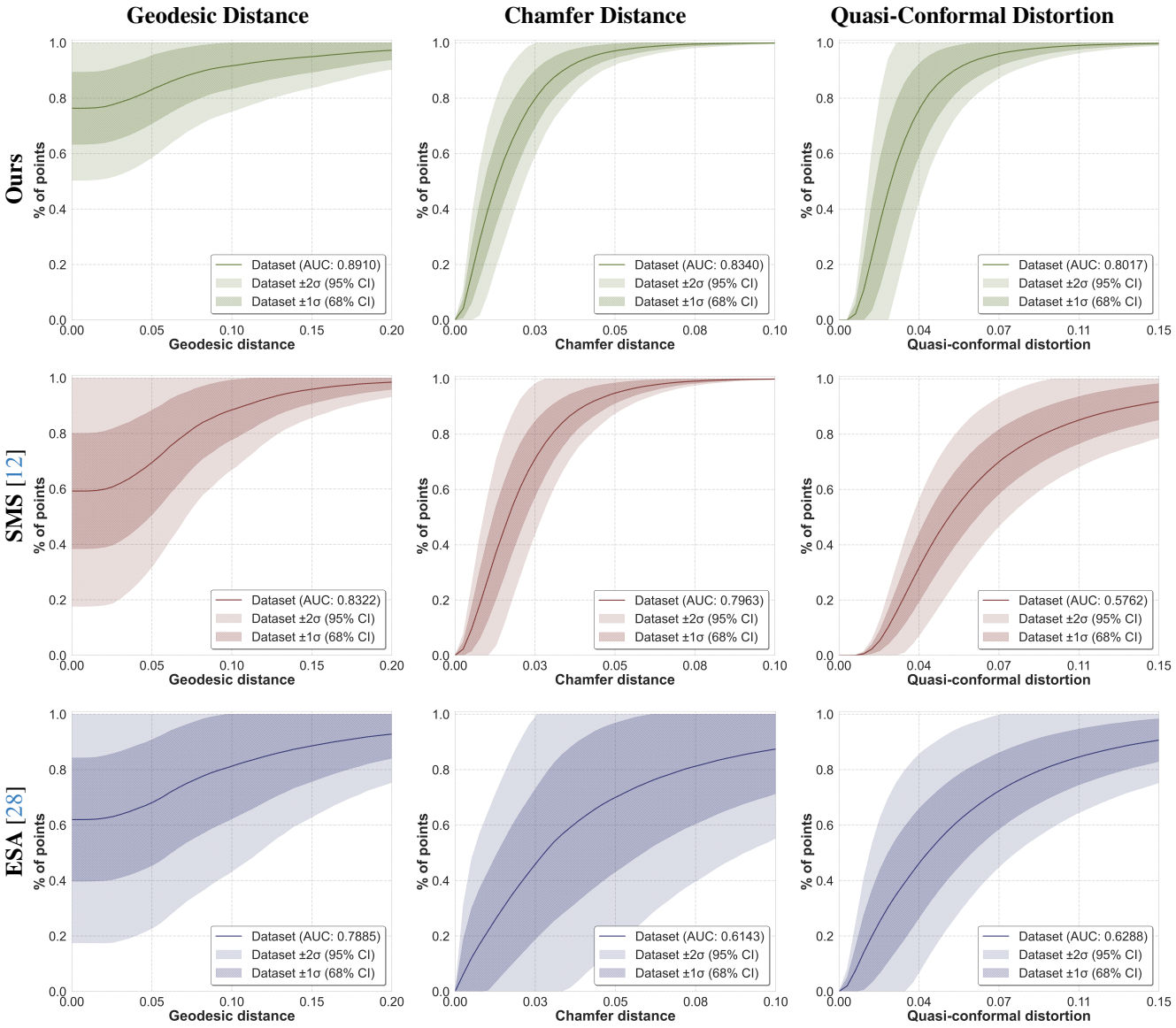

Figure 11. Interpolation results for MANO: Ours vs SMS [12] & ESA [28]. *Mean and confidence intervals for the three metrics are shown; top row has our results, middle SMS & bottom row ESA.*

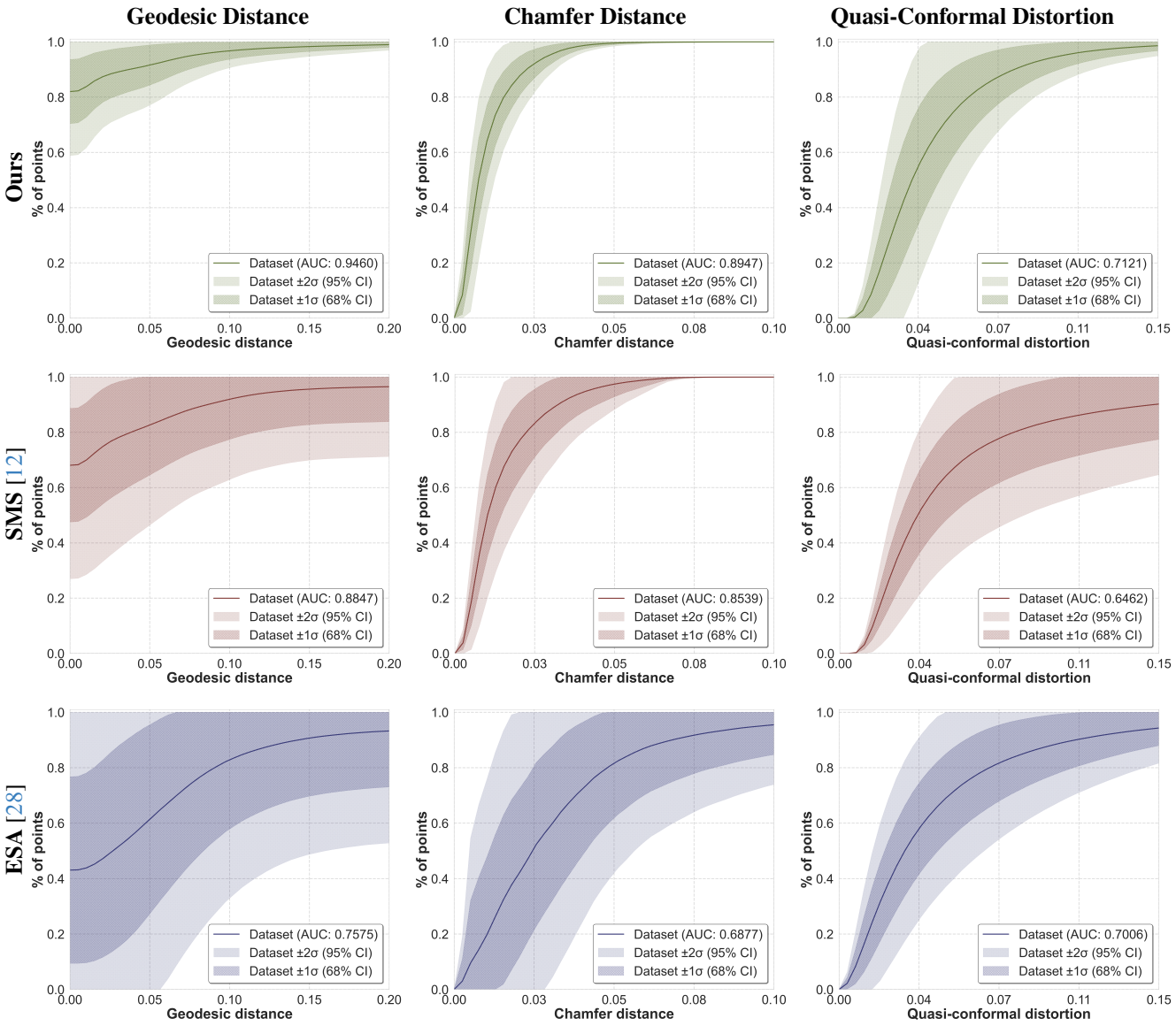

Figure 12. Interpolation results for DFAUST: Ours vs SMS [12] & ESA [28]. *Mean and confidence intervals for the three metrics are shown; top row has our results, middle SMS & bottom row ESA.*

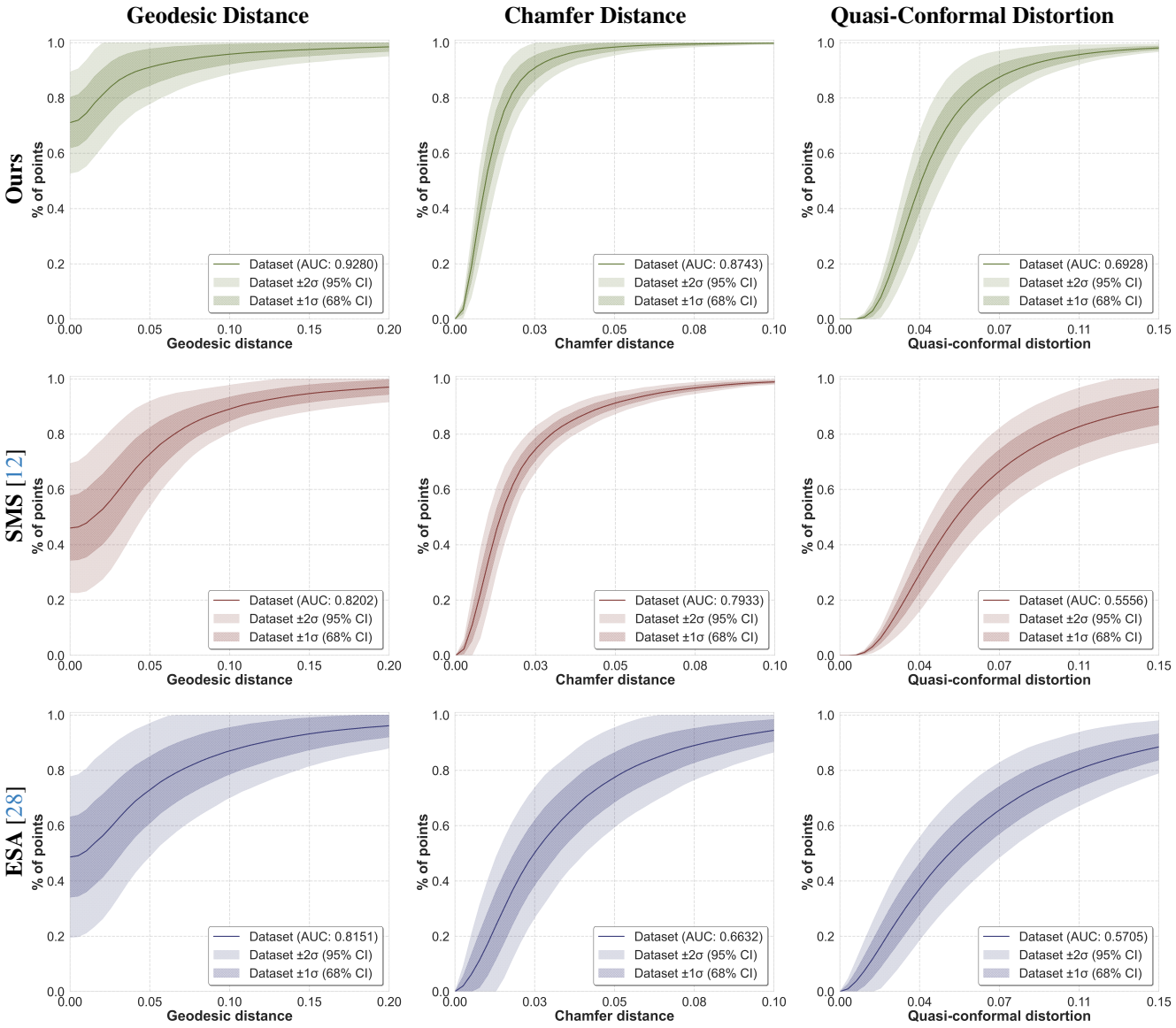

Figure 13. Interpolation results for SMAL: Ours vs SMS [12] & ESA [28]. *Mean and confidence intervals for the three metrics are shown; top row has our results, middle SMS & bottom row ESA.*

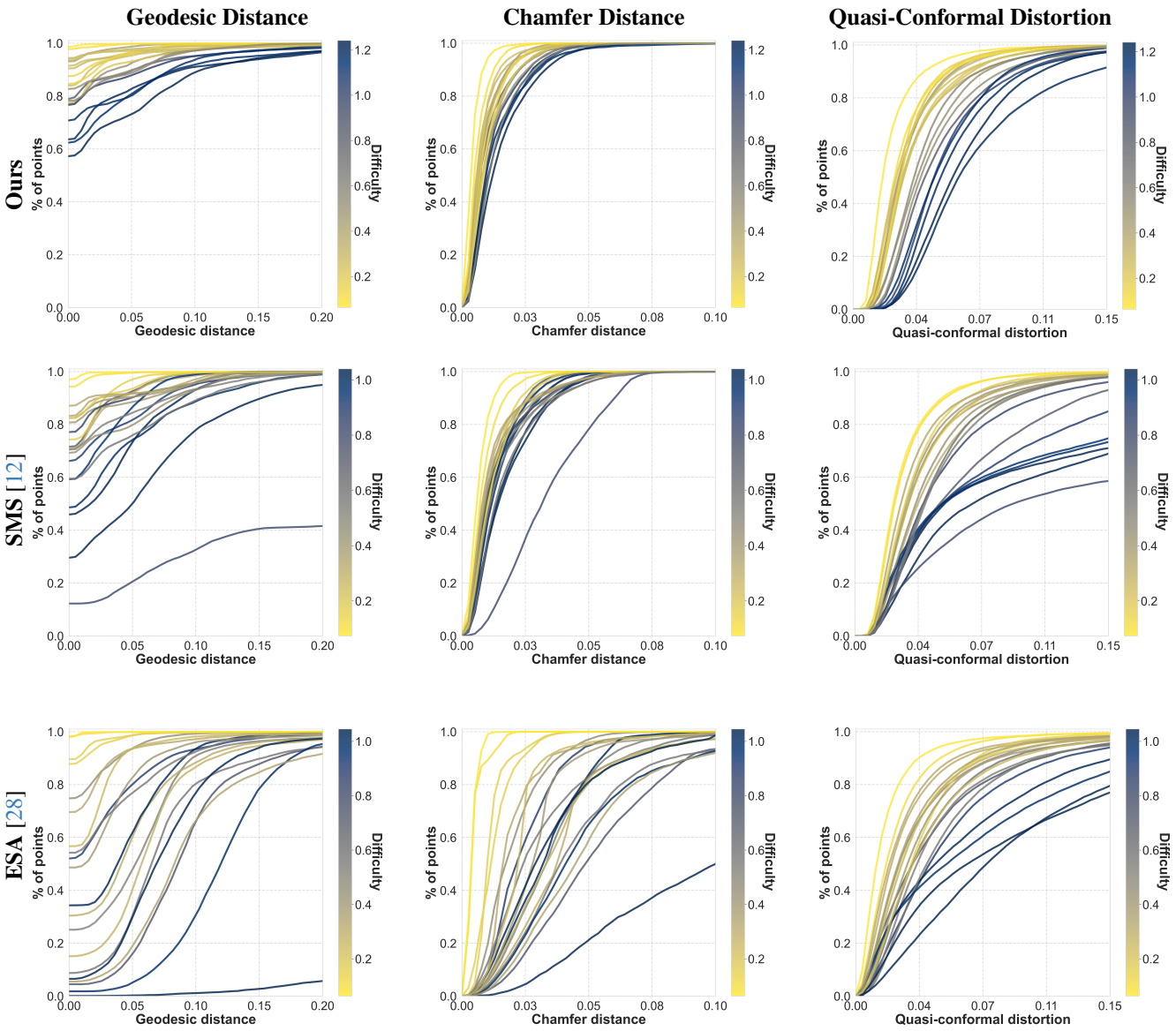

Figure 14. Interpolation results for DFaust: Ours vs SMS [12] & ESA [28]. *Plotting individual interpolations in which the difficulty of the problem is colour-coded; top row has our results, middle SMS & bottom row ESA. The difficulty rating is determined by the average vertex displacement for each interpolation task (from the ground-truth) normalised to one.*

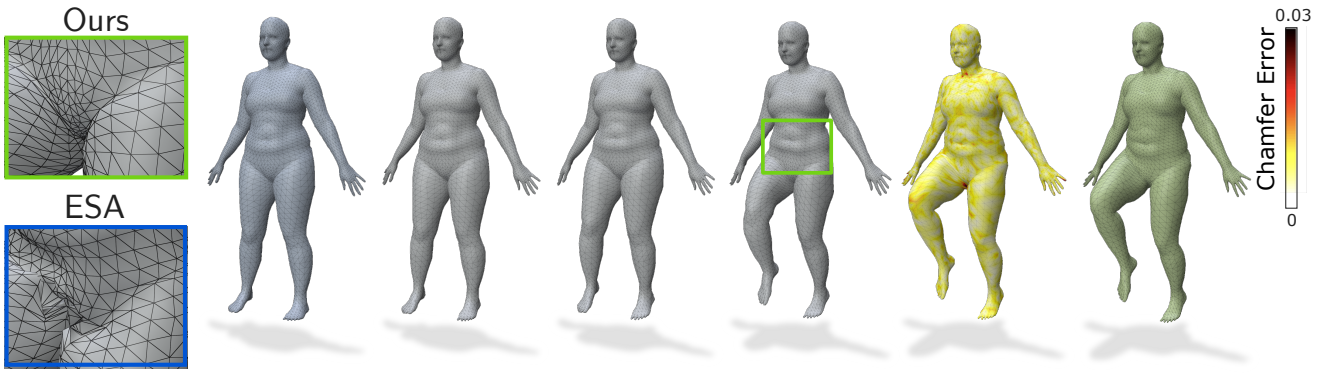

Figure 15. Non-Intersection of Surfaces: *We show interpolants from our method from the source on the left to target on the right (with final chamfer error on the second to right). As the leg is raised the interpolation from ESA fails to maintain surface integrity where the top of the leg meets the stomach resulting in unrealistic mesh distortions and overlapping (see closeup on bottom-left). Our diffeomorphic approach preserves topology by construction and guarantees non-intersection of the mesh (see closeup on top-left).*

S4.4. Fitting a Template to Topologically Noisy Data

A common approach in statistical shape analysis is to first bring all the raw data into correspondence by fitting a template to each sample. This also has the advantage that it removes noise and partial surfaces. However, this is a tricky task that often requires manual input.

We present an example demonstrating the potential for our method to automate this task. We select a neutral pose from the MANO dataset as our "template" and attempt to register this to a raw hand scan consisting of 38k vertices. Using the compression of Paul *et al*. [52], we form a 5k compressed representation as the target (in increasing computational efficiency).

Figure 16 illustrates the result of applying our approach, resulting in a high-quality registration. Although the majority of the surface fit has a very low Chamfer error, an area of higher error can be observed on the ring finger. This is due to the noise in the raw scan, where an unnatural bulge is clearly visible on one of the fingers. The use of a volume-preserving constraint by construction allows our method to fit the template despite this noise in the raw data.

Overall, as previously demonstrated there is no difference in the quality of the fit between using the full raw data and a Varifold compressed representation.

S4.5. Automatic Skeleton Transfer

The animation community has spent significant effort trying to ease rigging procedures. This is necessitated because the increasing availability of 3d data makes manual rigging infeasible. However, object animations involve understanding elaborate geometry and dynamics, and such knowledge is hard to infuse even with modern data-driven techniques. Automatic rigging methods do not provide adequate control and cannot generalise in the presence of unseen artefacts. An alternative approach is to learn to transfer an existing rig to a target using a dataset of known target poses to train a neural network.

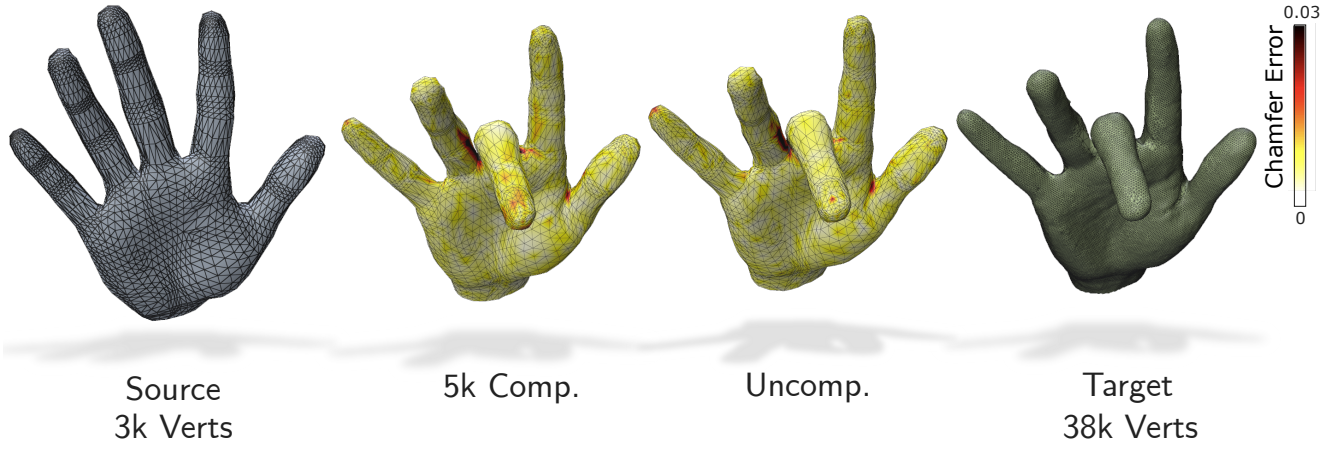

Figure 16. Fitting Template to Raw Scan Data. *We use a high-resolution noisy scan as the target to illustrate use of our method for template fitting. Varifold compression improves the efficiency of our approach with negligible change in final quality to using the full (uncompressed) target. Our method is robust to the noisy and partial data found in the dense target scan.*

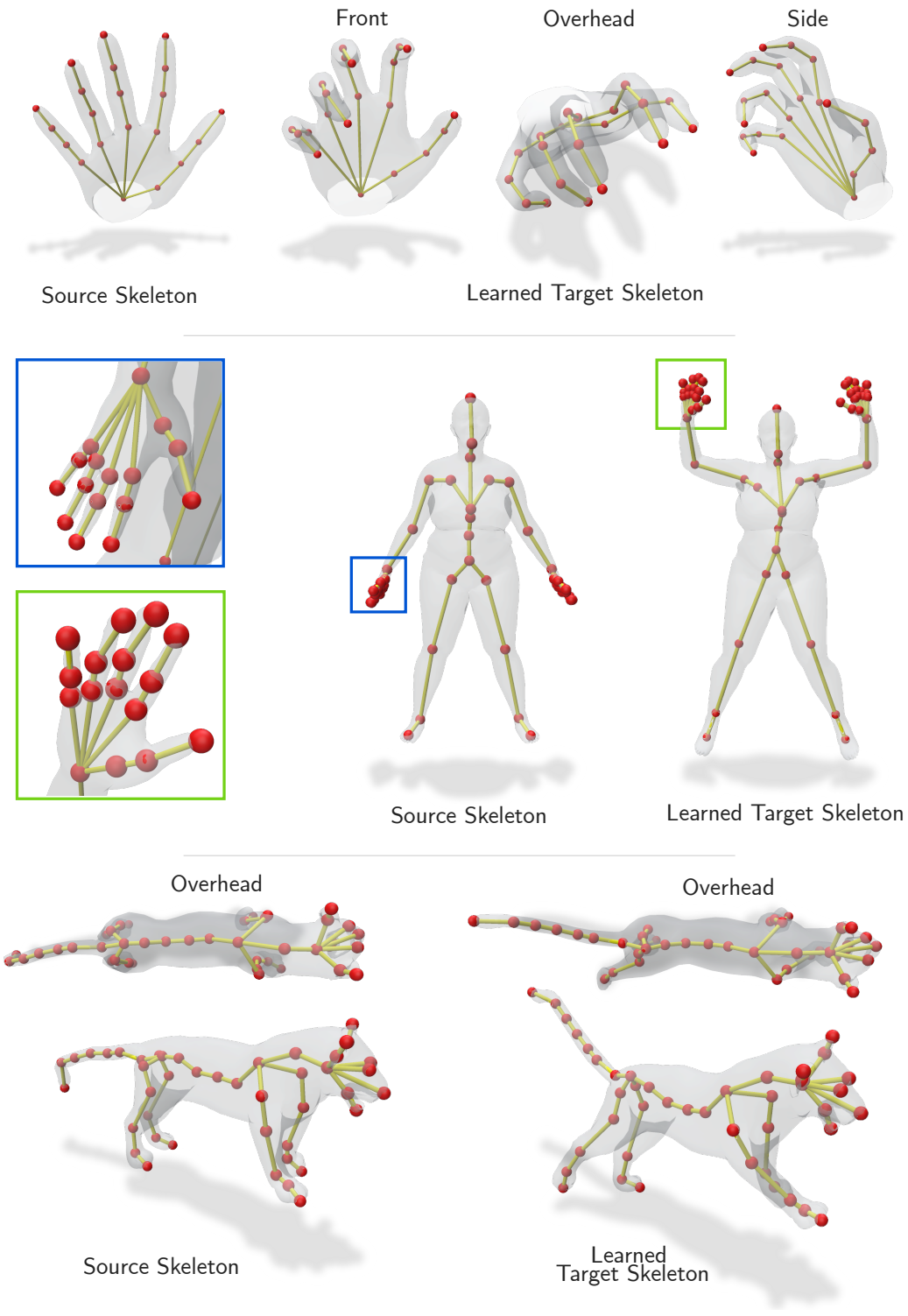

Figure 17. Learned Target Skeleton Examples: *Skeletons resulting from interpolations between poses involving* ***Top:*** *MANO,* ***Middle:*** *DFAUST &* ***Bottom:*** *SMAL datasets. The target skeleton is learned as a by-product of our method without any prior knowledge of the ground truth.*

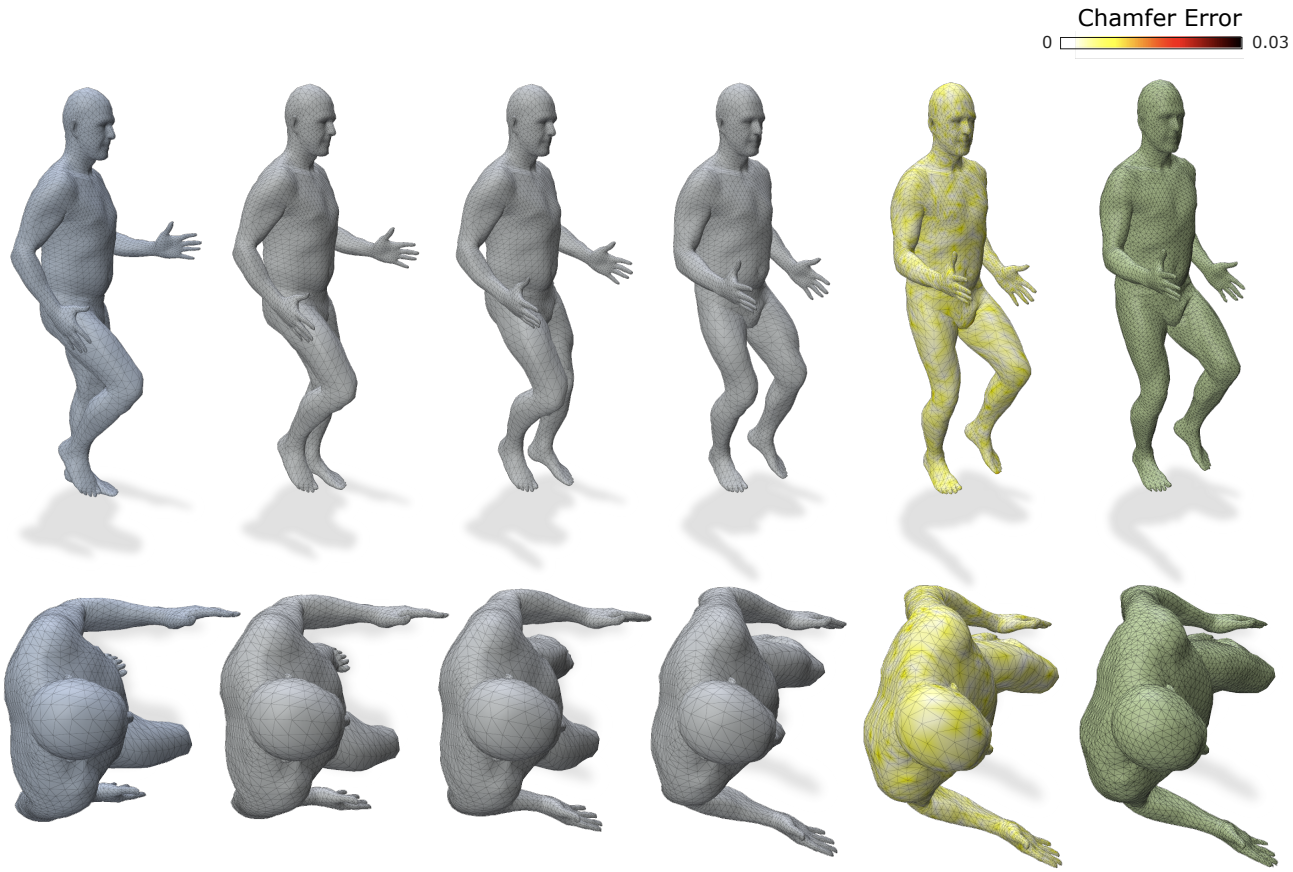

Figure 18. Interpolation Between Frames Capturing A Man Running on the Spot From DFAUST Dataset: *We show interpolants from our method from the source on the left to target on the right (with final chamfer error on the second to right).*

As part of our method the forward kinematics of the final skeletal pose (the global translation $\tilde{\mathbf{s}} \in \mathbb{R}^3$ and quaternion joint angles $\tilde{\mathbf{r}}_k \in \mathbb{Q}$ are optimised. As a result, we learn to transfer the source skeleton to the target as a by-product of our method. In Figure 17 we provide examples of these transferred skeletons, which appear in plausible configurations, and notably were achieved without any prior knowledge of ground truth target configurations.

S4.6. Additional Interpolations Examples

To further illustrate our findings, in Figures 18 to 20 we present additional interpolation examples generated using our method as a comprehensive showcase of our technique's capabilities.

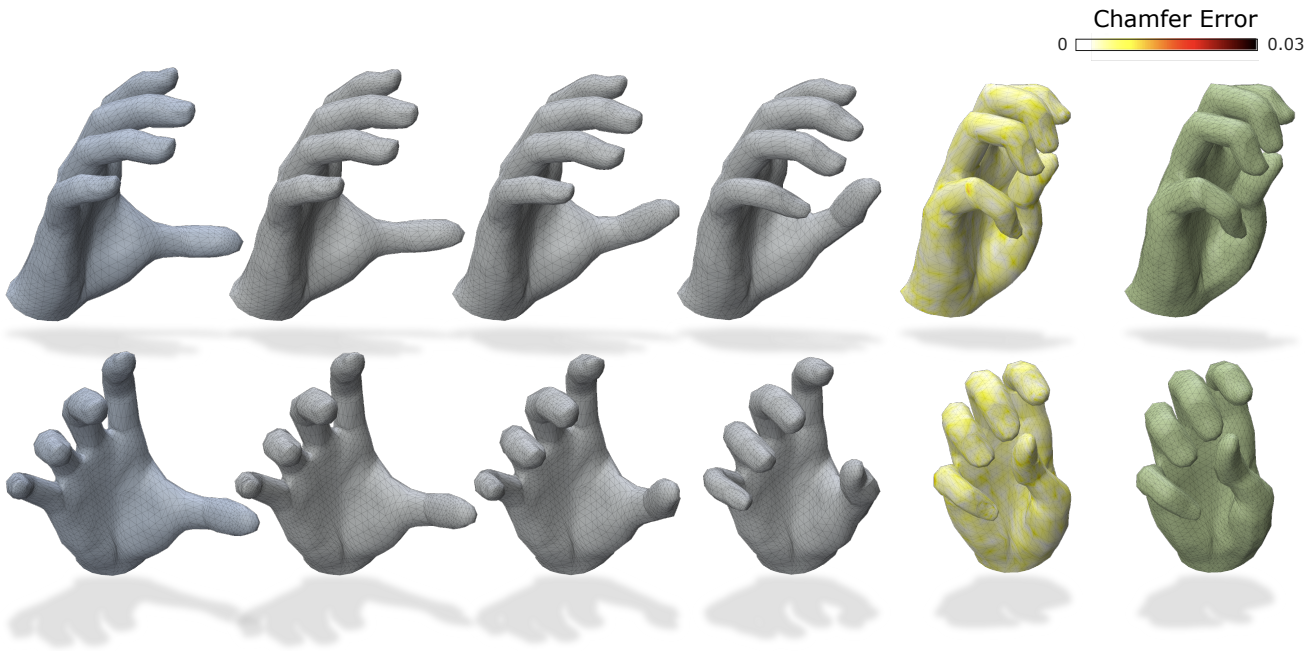

Figure 19. Interpolation Between An Open & Closed Hand Poses from the MANO Dataset: *We show interpolants from our method from the source on the left to target on the right (with final chamfer error on the second to right).*

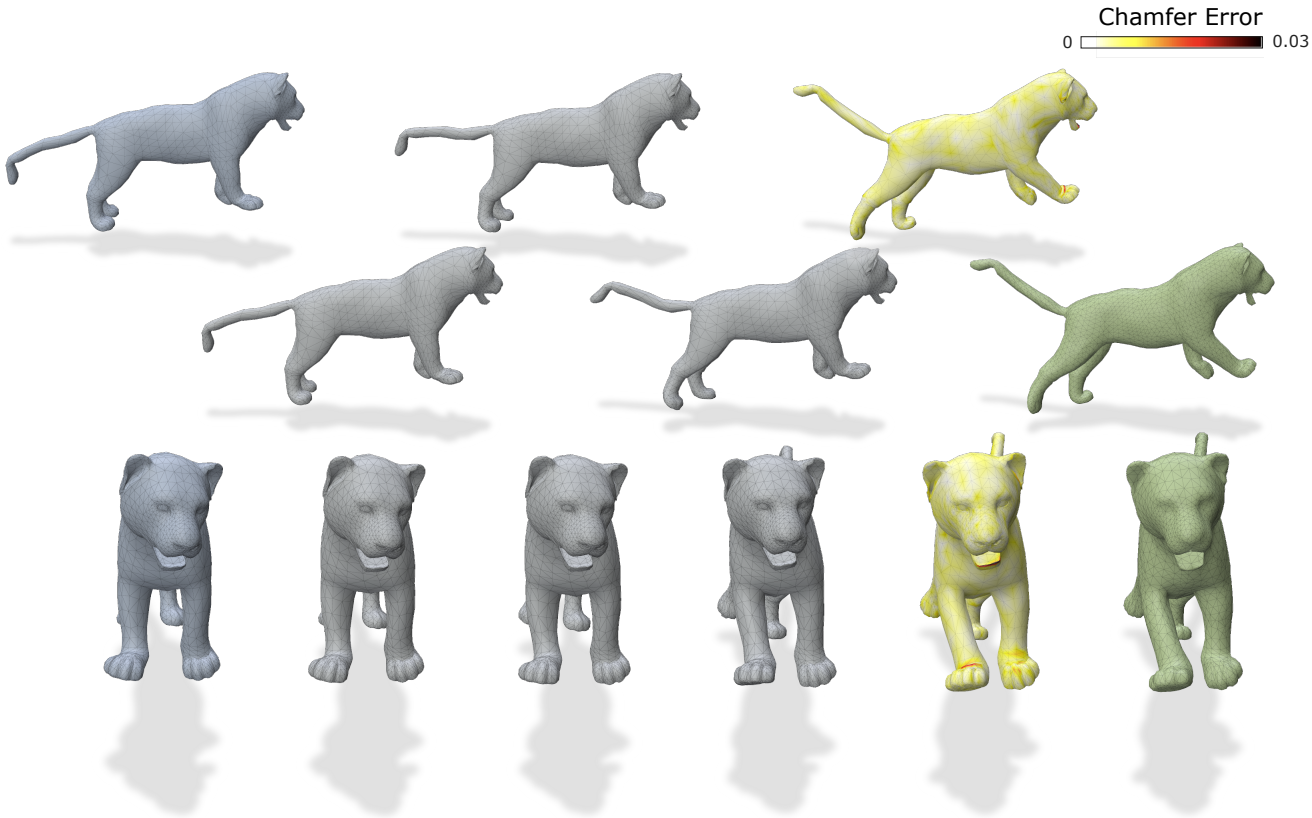

Figure 20. Interpolation Between Two Running Poses from the SMAL Dataset: *We show interpolants from our method from the source on the left to target on the right (with final chamfer error on the second to right).*