A. Implementation details

A.1. Training details

In the following, we provide additional details on our training protocol and choice of parameters. Throughout our experiments, our model was trained on a single NVIDIA Quadro RTX 8000 graphics card with 48GB VRAM.

Data preprocessing For a given shape collection $\mathcal{S}$, we apply a few data standardization steps to ensure training stability. For each $\mathcal{X}^{(i)}$, we normalize the scale of the shape by setting the approximate geodesic diameter to a constant value $\sqrt{\text{area}(\mathcal{X}^{(i)})} = \frac{2}{3}$. The pose is further centered around the origin by setting the mean vertex position to $0 \in \mathbb{R}^3$. The eigenvalues and eigenvectors required for our method are precomputed prior to training our model. Otherwise, our method is directly applicable to any collection of shapes that fulfill the weak pose alignment as discussed in Section 3.1.

Training scheduling Our general training protocol is outlined in Section 3.4 and illustrated visually in Figure 2 of the main paper.

In each forward pass, we first query the DiffusionNet backbone Φ_{feat} to obtain sets of local features $\mathbf{F}^{(i)} := \Phi_{\text{feat}}(\mathcal{X}^{(i)})$ and $\mathbf{F}^{(j)} := \Phi_{\text{feat}}(\mathcal{X}^{(j)})$ for a pair of input shapes $\mathcal{X}^{(i)}$ and $\mathcal{X}^{(j)}$. In the second step, the matching module $(\mathbf{\Pi}^{(i,j)}, \mathbf{V}^{(i,j)}, \ell_{\text{match}}^{(i,j)}) := \Phi_{\text{match}}(\mathbf{F}^{(i)}, \mathbf{F}^{(j)})$ computes a set of putative correspondences $\mathbf{\Pi}^{(i,j)}$, registered vertices $\mathbf{V}^{(i,j)}$ and the corresponding matching loss $\ell_{\text{match}}^{(i,j)}$. Finally, Equation (6) produces the multi-shape correspondences $\mathbf{\Pi}_{\text{mult}}^{(i,j)}$, which then allows us to compute the loss $\ell_{\text{cyc}}^{(i,j)}$ through Equation (7).

As stated in Section 3.4, the shape graph $\mathcal{G}$ is updated regularly after a fixed number of epochs. This interval is chosen in dependence of the number of training shapes as a round figure that results in around $10k - 15k$ training iterations per update. To reduce the computational load, the pairwise correspondences between all pairs of training shapes $\mathbf{\Pi}_{\text{mult}}^{(i,j)}$ are precomputed and stored each time the shape graph is constructed. Additionally, we wait for 5 shape graph update cycles before activating the cycle-consistency loss. This burn-in period allows the feature extractor and putative correspondence modules to converge to a certain degree which facilitates a stable training and reduces stochasticity.

Learning Our model is trained in an end-to-end manner with the Adam optimizer [32], using standard parameters. All the learnable weights are contained in the DiffusionNet backbone **I**. The DeepShells pairwise matching module **II** and multi-matching module **III** convert the learned features into correspondences, but these maps themselves are fully deterministic. The backward pass updates the DiffusionNet weights both in terms of the pairwise alignment loss $\ell_{\text{match}}^{(i,j)}$ and the cycle-consistency loss $\ell_{\text{cyc}}^{(i,j)}$. The former stems from Equation (3) and the latter term is defined in Equation (7). The cycle-consistent correspondences themselves $\mathbf{\Pi}_{\text{mult}}^{(i,j)}$ are obtained from non-differentiable operations, since they are the quantized outputs from DeepShells concatenated via Dijkstra's algorithm in Equation (6a). Instead, the gradients from $\ell_{\text{cyc}}^{(i,j)}$ pass information back to the DeepShells layer **II** through the registrations $\mathbf{V}^{(i,j)}$.

Hyperparameters We set the cycle-consistency loss weight to $\lambda_{\text{cyc}} = 0.5$. The number of latent dimensions of the DiffusionNet encodings is chosen as $l = 128$ and the architecture comprises 4 consecutive DiffusionNet blocks. For the DeepShells matching layer, we directly use the hyperparameters specified by the original publication and the corresponding source code [21]. Specifically, the number of eigenfunctions used to compute the smooth shell product space poses (see Appendix A.2.2) is upsampled on a log-scale between $k_{\min} = 6$ and $k_{\max} = 21$.

A.2. Architecture details

A.2.1 Feature backbone

Our network proposed in Section 3.2 leverages the recent DiffusionNet [57] backbone for local feature extraction. We outline the basic architecture of this module here and refer to [57, Sec. 3] for further technical details.

The core motivation is to model feature propagation of signals $f(x,t)$ on the surface of a shape $\mathcal{X}^{(i)}$ as heat diffusion, governed by the standard heat equation

$$\frac{\partial}{\partial t} f(x,t) = \Delta f(x,t). \tag{9}$$

In this context, Δ is the intrinsic Laplace-Beltrami operator on the surface $\mathcal{X}^{(i)}$. In the discrete case, a common approximation is the cotangent Laplacian $\mathbf{L} := -\mathbf{M}^{-1}\mathbf{S} \in \mathbb{R}^{m \times m}$, where $\mathbf{M}$ and $\mathbf{S}$ are the mass matrix and stiffness matrix, respectively. We further consider the truncated basis of eigenfunctions $\mathbf{\Psi} \in \mathbb{R}^{m \times k}$ and corresponding diagonal matrix of eigenvalues $\mathbf{\Lambda} \in \mathbb{R}^{k \times k}$ of the discretized Laplacian. For a given signal $\mathbf{f} \in \mathbb{R}^m$ on $\mathcal{X}^{(i)}$, the heat propagation for a time-interval $t > 0$ according to Equation (9) then results in the approximate solution

$$H_t(\mathbf{f}) := \mathbf{\Psi} \exp(t\mathbf{\Lambda}) \mathbf{\Psi}^\dagger \mathbf{f}. \tag{10}$$

For a given feature matrix $\mathbf{F} \in \mathbb{R}^{m \times l}$, an individual operator H_t is applied separately to each channel with different learnable time step weights t. A key benefit of such propagation operators is that they are indifferent to the sampling

density and therefore robust to remeshing and local noise. On the other hand, pure heat diffusion is spatially isotropic and therefore not sufficiently expressive. To break radial symmetry, DiffusionNet additionally leverages a gradient-based feature refinement layer. At every point on the surface, it computes inner products between spatial gradients of the (scalar) feature signals on the tangent plane.

Putting everything together, an individual DiffusionNet block takes a set of features $\mathbf{F}$, propagates information both via a spatial diffusion and a spatial gradient layer, and feeds them to a per-point multilayer perceptron (MLP). The first layer is initialized by the input features, defined as the vertex coordinates $\mathbf{V}^{(i)}$. For further technical details, we refer the interested reader to the original publication [57].

A.2.2 Hierarchical pairwise matching

In Section 3.2, we further introduced our differentiable matching layer Φ_{match} based on DeepShells [21]. The final map is fully specified by Equation (3). In the following, we provide additional technical details required to derive the exact optimization steps and compute Φ_{match} in practice.

Following [21, Sec. 3], we first introduce the following latent feature representation of a shape $\mathcal{X}^{(i)}$ that is used within each DeepShells layer

$$\mathbf{F}^{(k)}_{\mathcal{X}^{(i)}} := \left(\mathbf{\Psi}^{(k)} \,,\; S^{(k)}\big(\mathbf{V}^{(i)}\big) \,,\; \mathbf{N}^{(k)} \right) \in \mathbb{R}^{m \times (k+6)} \tag{11}$$

where $\mathbf{\Psi}^{(k)}$ are the first k eigenfunctions of the intrinsic Laplace-Beltrami operator on $\mathcal{X}^{(i)}$, corresponding to the smallest eigenvalues. The operator $S^{(k)}$ is the smoothing map initially proposed in [19, Eq. (8)]. The matrix $\mathbf{N}^{(k)}$ denotes the outer normals of the (smoothed) input geometry.

Overall, the resulting feature tensor $\mathbf{F}^{(k)}_{\mathcal{X}^{(i)}}$ yields a $(k+6)$-dimensional embedding per vertex $\mathbf{V}^{(i)} \in \mathbb{R}^{m \times 3}$, depending on the number of eigenfunctions k. In order to align two shapes in this embedding space, an affine transformation is proposed

$$\hat{\mathbf{F}}^{(k)}_{\mathcal{X}^{(i)}}\big(\mathbf{C}^{(k)}, \tau^{(k)}\big) := \left(\mathbf{\Psi}^{(k)} \mathbf{C}^{(k)^{\dagger}} \,,\; S^{(k)}\big(\mathbf{V}^{(i)}\big) + \mathbf{\Psi}^{(k)} \tau^{(k)} \,,\; \hat{\mathbf{N}}^{(k)} \right) \in \mathbb{R}^{m \times (k+6)}, \tag{12}$$

that deforms the input shape $\mathcal{X}^{(i)}$ in the $(k+6)$-dimensional embedding space. This deformation is parameterized with a functional map $\mathbf{C}^{(k)} \in \mathbb{R}^{k \times k}$ [44] and displacement coefficients $\tau^{(k)} \in \mathbb{R}^{k \times 3}$. The outer normals of the deformed pose are denoted as $\hat{\mathbf{N}}^{(k)}$.

As we discussed in Section 3.2, the correspondence task in our framework is fully specified by the optimal transport energy Equation (2). To make the resulting update steps differentiable, an additional entropy regularization term is added to the energy

$$E_{\text{match,reg}}\big(\mathbf{F}, \mathbf{G}; \tilde{\mathbf{\Pi}}\big) = E_{\text{match}}\big(\mathbf{F}, \mathbf{G}; \tilde{\mathbf{\Pi}}\big) + \lambda_{\text{ent}} \sum_{i',j'} \tilde{\mathbf{\Pi}}_{i',j'} \log \tilde{\mathbf{\Pi}}_{i',j'}. \tag{13}$$

This is a common approach that was initially proposed by the seminal work of Cuturi et al. [13]. One compelling implication is that $E_{\text{match,reg}}$ can be minimized efficiently with respect to the transport plans $\tilde{\mathbf{\Pi}} \in [0,1]^{m \times n}$ through Sinkhorn's algorithm. Moreover, each individual update step of this algorithm is differentiable which makes it viable for standard gradient-based optimization. The resulting map Φ_{match} is specified by the following alternating scheme of optimization steps

$$\big(\mathbf{C}^{(k)}, \tau^{(k)}\big) \mapsto \underset{\substack{\tilde{\mathbf{\Pi}}^{(k)} \in \\ \mathcal{T}(\mathcal{X}^{(i)}, \mathcal{X}^{(j)})}}{\arg\min} \; E_{\text{match,reg}}\left(\hat{\mathbf{F}}^{(k)}_{\mathcal{X}^{(i)}}, \mathbf{F}^{(k)}_{\mathcal{X}^{(j)}}; \tilde{\mathbf{\Pi}}^{(k)} \right), \tag{14a}$$

$$\tilde{\mathbf{\Pi}}^{(k)} \mapsto \underset{\mathbf{C}^{(k)}, \tau^{(k)}}{\arg\min} \; E_{\text{match,reg}}\left(\hat{\mathbf{F}}^{(k)}_{\mathcal{X}^{(i)}}, \mathbf{F}^{(k)}_{\mathcal{X}^{(j)}}; \tilde{\mathbf{\Pi}}^{(k)} \right), \tag{14b}$$

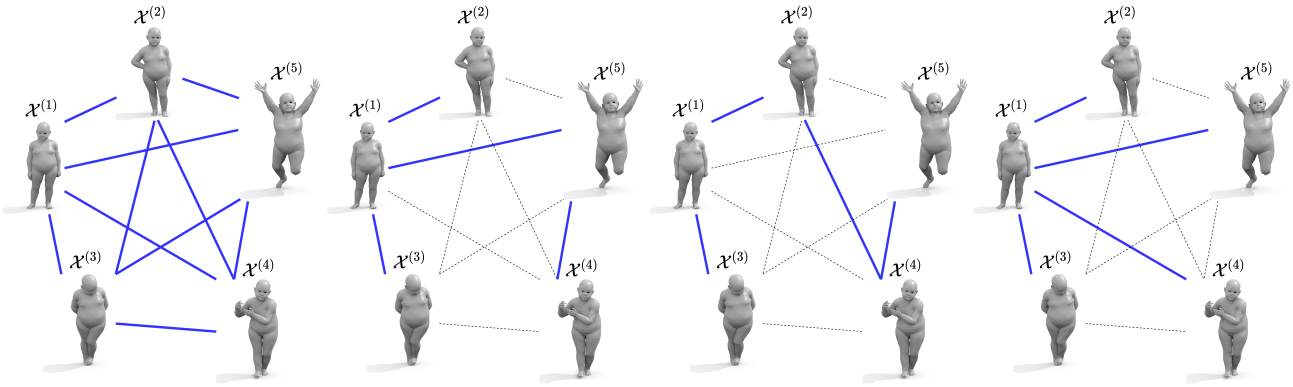
Through this scheme, the minimization of the energy $E_{\text{match,reg}}$ is decoupled into two separate update steps, each of which can be solved efficiently in closed form. The first expression is minimized via Sinkhorn's algorithm to obtain an optimal transport plan $\tilde{\mathbf{\Pi}}^{(k)}$ from the transportation polytope $\mathcal{T}(\mathcal{X}^{(i)}, \mathcal{X}^{(j)})$. The second update results in a standard linear least squares problem, see [21, Sec. 3] for additional details. For the initial step, we replace the $(k+6)$-dimensional feature embeddings with the learned features $\mathbf{F}^{(i)}$ and $\mathbf{F}^{(j)}$ produced by the DiffusionNet backbone. The map Φ_{match} then alternates between the minimization steps Equation (14a) and Equation (14b) while increasing the number of eigenfunctions k after each step. The final outputs are defined as

$$\mathbf{\Pi}^{(i,j)} := \underset{\mathbf{\Pi}}{\arg\min} \; E_{\text{match}}\left(\hat{\mathbf{F}}^{(k_{\max})}_{\mathcal{X}^{(i)}}, \mathbf{F}^{(k_{\max})}_{\mathcal{X}^{(j)}} \,;\, \mathbf{\Pi} \right) \tag{15a}$$

$$\mathbf{V}^{(i,j)} := \mathbf{V}^{(i)} + \mathbf{\Psi}^{(k_{\max})} \tau^{(k_{\max})}, \tag{15b}$$

$$\ell^{(i,j)}_{\text{match}} := \sum_{k} E_{\text{match}}\left(\hat{\mathbf{F}}^{(k)}_{\mathcal{X}^{(i)}}, \mathbf{F}^{(k)}_{\mathcal{X}^{(j)}} \,;\, \tilde{\mathbf{\Pi}}^{(k)} \right). \tag{15c}$$

The matches $\mathbf{\Pi}^{(i,j)} \in \{0,1\}^{m \times n}$, with $\mathbf{\Pi}^{(i,j)} \mathbf{1}_n = \mathbf{1}_m$, produced by Φ_{match} are thereby the outputs of the final optimization layer $k_{\max}$. In practice, they are obtained as the hard nearest-neighbor assignment between the final obtained shape embeddings $\hat{\mathbf{F}}^{(k_{\max})}_{\mathcal{X}^{(i)}}$ and $\mathbf{F}^{(k_{\max})}_{\mathcal{X}^{(j)}}$.



(i) Full graph. (ii) MST graph. (iii) TSP graph. (iv) Star graph.

Figure 6. An overview of different shape graph topologies $\mathcal{G}$. We consider graphs that are (i) fully connected, (ii) minimal spanning trees, (iii) minimal Hamiltonian paths, specified by the traveling salesman problem and (iv) star graphs centered around one canonical pose. We provide a detailed discussion in Appendix B. Unless stated otherwise, the full graph (i) is the default for our approach.

B. Shape graph topology

We explore the following classes of graph topologies for the shape graph $\mathcal{G}$, see also Figure 6 for a visualization:

i. **Full graph**. The default setting for our method is the fully connected graph with the edge weights defined in Equation (5).

ii. **MST graph**. We consider the minimal spanning tree corresponding to the full graph $\mathcal{G}$. This graph topology is a minimal choice, in the sense that it has the smallest total edge weight among all subgraphs of $\mathcal{G}$ that span the set of nodes $\mathcal{S}$.

iii. **TSP graph**. Based on the traveling salesman problem, we predict a Hamiltonian path of minimal total edge weight. This effectively defines an optimal ordering of the input set $\mathcal{S}$.

iv. **Star graph**. We define a complete, bipartite graph which connects one specific center node with all $N-1$ remaining nodes. The idea is to imitate template-based shape matching methods such as [4, 23] where all training poses are matched to a canonical shape.

Unless stated otherwise, we use the fully connected graph (i) by default in our experiments in Section 4. Choosing the number of retained edges is generally subject to a trade-off between accuracy and efficiency. Using (i) all $\frac{N(N-1)}{2}$ edges often yields the most accurate matching $\mathbf{\Pi}_{\text{mult}}^{(i,j)}$, since this leads to the shortest possible path lengths in Equation (6a). Nevertheless, the sparse graph topologies (ii)-(iv) might be preferable for specific applications. All three definitions (ii)-(iv) specify variants of spanning trees with exactly $N-1$ edges. This means, that the memory complexity for storing the graph, as well as the full query runtime cost is in $\mathcal{O}(N)$, see Appendix C for a cost analysis. Thus, they are more suitable for very large training sets or in scenarios where computational resources are scarce. See Section 4.4 in the main paper for an empirical comparison of the different topologies (i)-(iv).

Discussion Each of the variants (ii)-(iv) has interesting properties that might give rise to potential new avenues of applications in future work: (ii) Removing the (k-1) largest edges in the MST graph yields a subdivision of $\mathcal{G}$ into k optimal clusters (MST clustering). (iii) The TSP graph orders the input shapes into a sequence, i.e., predicts a canonical ordering. (iv) Choosing an optimal star graph automatically selects one of the shapes in the collection $\mathcal{S}$ as the canonical pose. By comparing the set of all possible star graphs, we can in principle rank all input poses in terms of how representative they are of the underlying shape manifold.

C. Empirical computation cost

Training We empirically measure the computation cost of our full pipeline. To this end, we choose a training set of $\{10^2, 20^2, 50^2, 100^2\}$ pairs from the SURREAL [61] dataset with a fixed mesh resolution of 6890 vertices, which is common for SMPL [37] meshes. The resulting training runtime and memory costs are summarized in Table 4. Averaged over all samples, our model takes around $\approx 0.4s$ per training pair. The majority of the cost for constructing the graph $\mathcal{G}$ results from querying all sample pairs, which is equivalent to the epoch training cost minus the backward pass. The remaining cost stems from precomputing the concatenated, pairwise matches as discussed in Appendix A.1.

	#pairs = 10^2	20^2	50^2	100^2
Epoch training time (s)	37.85 ± 2.48	174.82 ± 8.31	921.23 ± 25.56	4192.91 ± 142.03
Graph construction (s)	36.04 ± 1.95	157.30 ± 7.04	893.23 ± 9.05	3814.96 ± 44.48
Required RAM (GB)	3.57 ± 0.02	3.96 ± 0.04	5.81 ± 0.28	9.02 ± 1.73

Table 4. **Empirical training cost.** We quantify the computation cost of our pipeline for different training set sizes. For a given number of shapes $N = |\mathcal{S}|$, one epoch consists of #pairs = $N^2 \in \{10^2, \dots, 100^2\}$ optimization steps that each match a pair of shapes $\mathcal{X}^{(i)}, \mathcal{X}^{(j)} \in \mathcal{S}$.

		#pairs = 10^2	20^2	50^2	100^2
Total query time (s)	Full graph	41.00 ± 3.32	166.91 ± 9.54	1077.06 ± 28.35	4370.36 ± 115.00
	MST graph	3.62 ± 0.29	8.10 ± 0.36	22.48 ± 0.90	50.98 ± 2.57
Graph storage (MB)	Full graph	10.66 ± 0.00	42.64 ± 0.01	266.47 ± 0.08	1065.89 ± 0.33
	MST graph	0.96 ± 0.00	2.03 ± 0.00	5.22 ± 0.00	10.55 ± 0.00

Table 5. **Total query cost.** We report the computation cost of our pipeline at test time. Note, that these results only apply to the specific setting where all pairs of a given set of shapes are queried and the graph $\mathcal{G}$ is *precomputed*. Under these circumstances, the MST graph proves to be superior as its computation cost increases linearly $\mathcal{O}(N)$ in the number of shapes N. For pairwise matching at test time, or when the graph $\mathcal{G}$ needs to be extracted on the fly, the advantages of MST are less prominent.

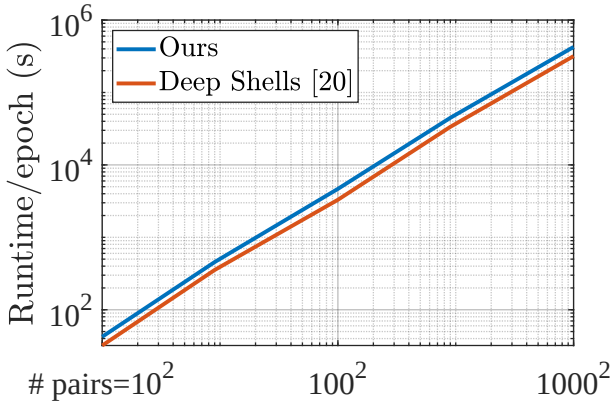


Figure 7. **Total training time** We compare the total training time per epoch of our approach to Deep Shells [21] for different training set sizes on the SURREAL [61] dataset.

Query time We additionally compare the required cost for querying our model. Aside from our main pipeline, we also consider the sparse 'MST' graph type introduced in Appendix B. The resulting costs are summarized in Table 5. For a given set of query shapes, one has to distinguish whether the graph $\mathcal{G}$ is precomputed or needs to be predicted on the fly. In the latter case, an additional cost of constructing the graph is added, see the second row of Table 4. Notably, this cost does not depend on the test graph topology. This also means that the main computational advantages of the MST graph are less prevalent when no offline graph precomputation is possible, e.g., for a pair of unseen test poses. When the unseen pose is supposed to be added to an existing graph in an online fashing, only N pairs between the old training set and the new pose need to be computed. This, however, again entails the same cost for either the full graph or MST. On the other hand, MST is much faster for precomputed graphs. Also, the storage cost of the MST graph is always more efficient than the dense 'full' setting. This makes sparse graph topologies relevant when memory is limited or for very large training sets, since the required memory of the full graph grows quadratically $\mathcal{O}(N^2)$ with the number of training shapes $N = |\mathcal{S}|$.

Comparison to [21] We show a comparison of the total training time per epoch to the baseline approach Deep Shells [21] in Figure 7. The runtime of our approach is on par with [21], while leading to significantly more accurate correspondences, see Table 1. The comparison further demonstrates that, in practice, the training time of both approaches increases quadratically in the number of training shapes. Thus, they feasibly scale to a large training set of 1000^2 training pairs from the SURREAL [61] dataset.

D. Qualitative results

For a more complete picture, we provide several additional qualitative comparisons. Figure 8 and Figure 9 show results corresponding to the benchmark comparisons from Section 4.2 and Section 4.3 of the main paper.

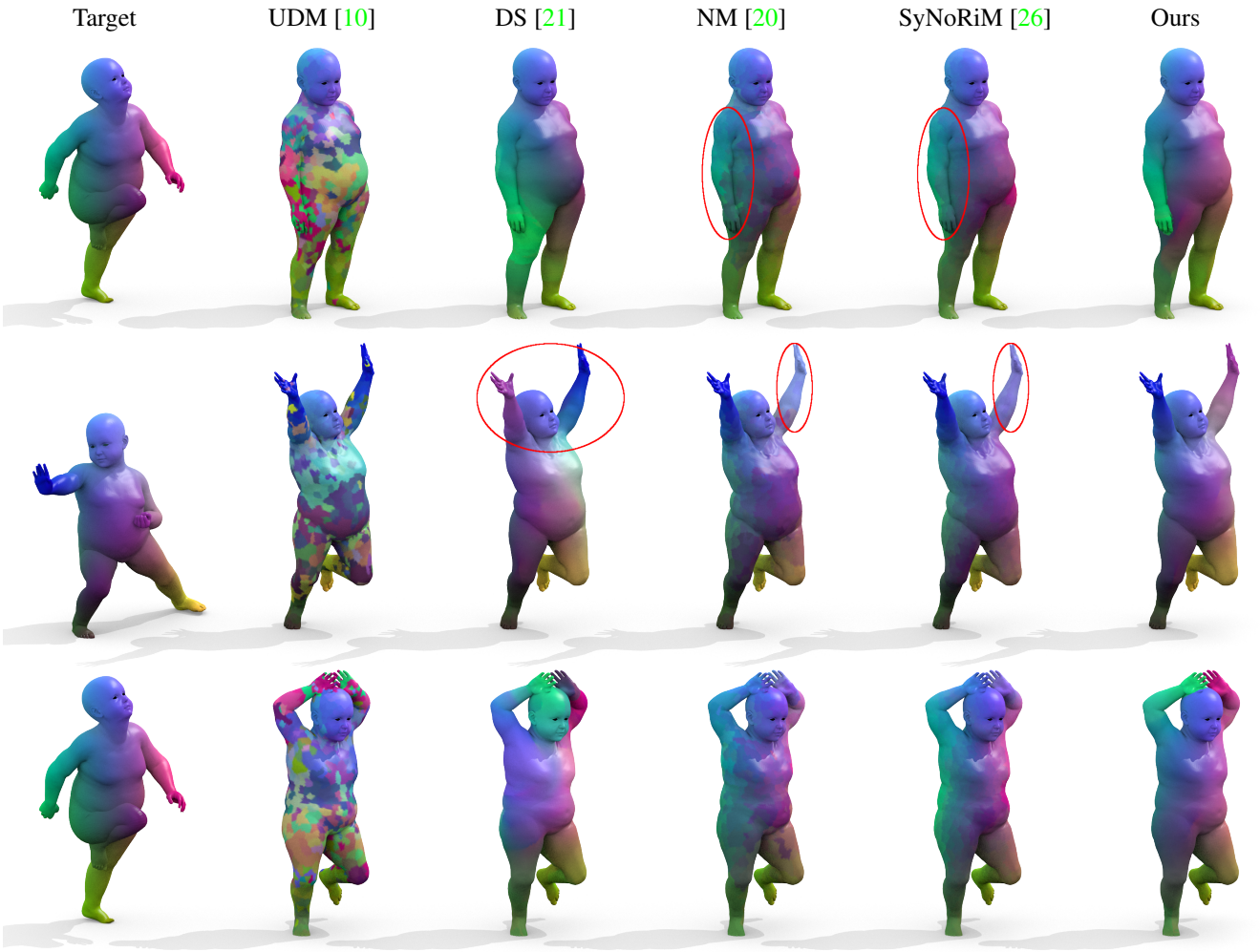


Figure 8. **Topological noise qualitative.** We compare the quality of the predicted correspondences on three pairs from TOPKIDS [33], corresponding to the quantitative results from Figure 3. All three pairs are corrupted by topological noise in places of self-contact, e.g., where the child's arms touch its upper body or head.

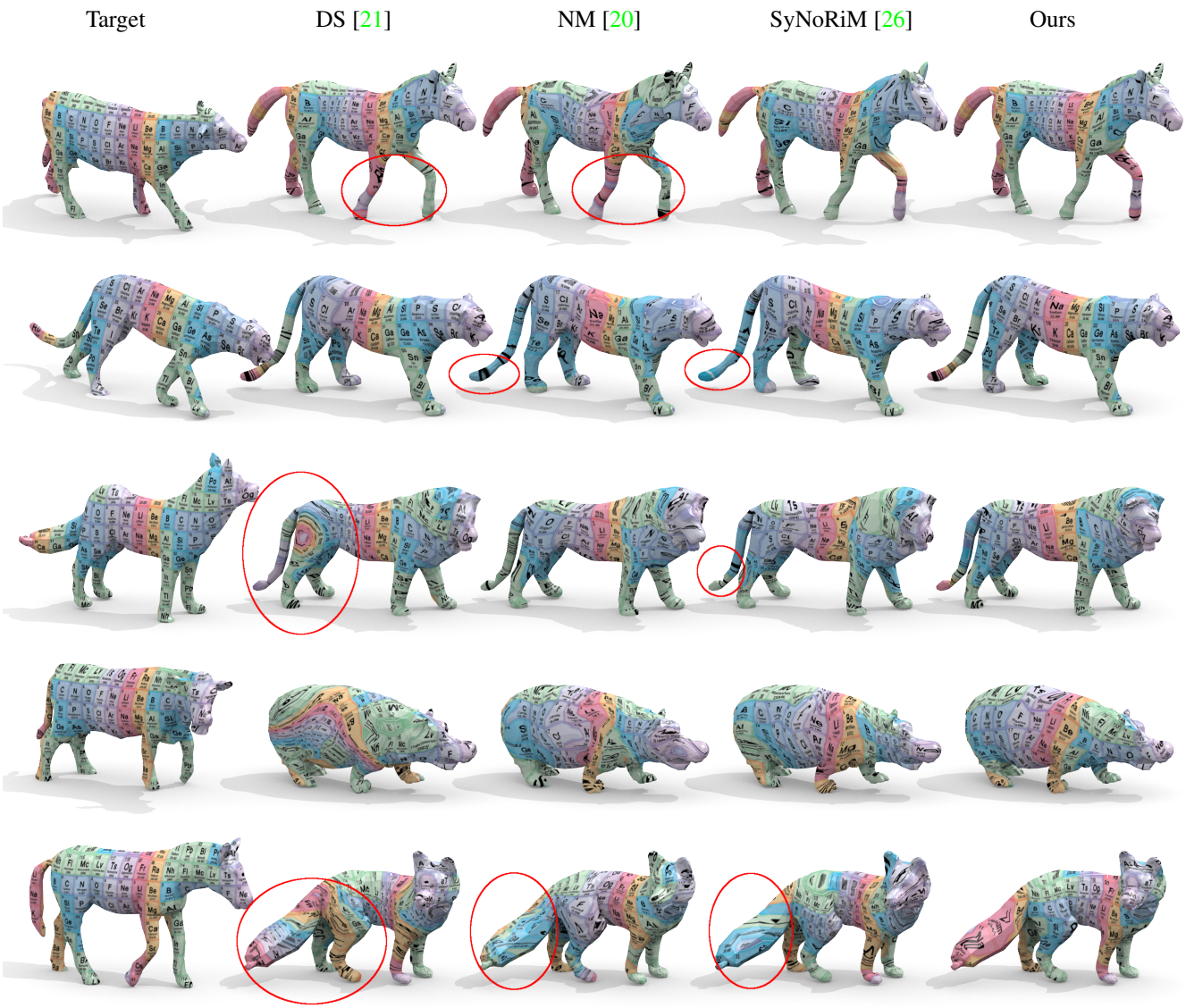


Figure 9. **Inter-class texture transfers.** We assess the map smoothness of several baseline approaches, in comparison to our proposed method. The five sample pairs are taken from the SMAL [65] test set corresponding to our benchmark comparison in Figure 5. In each case, the obtained matches are visualized via a texture map.