Neural Shape Deformation Priors – Supplementary Material –

Jiapeng Tang[1] **Lev Markhasin**[2] **Bi Wang**[2] **Justus Thies**[3] **Matthias Nießner**[1]

[1] Technical University of Munich [2] Sony Europe RDC Stuttgart
[3] Max Planck Institute for Intelligent Systems, Tübingen, Germany

Our Neural Shape Deformation Priors method is based on transformer-based deformation networks that represent the deformation as a composition of local surface deformations. The underlying architectures are discussed in Appendix A. The used evaluation metrics are detailed in Appendix B. Our notations are further explained in Appendix C. And more details about data-preprocessing are given in Appendix D. In addition to the results shown in the main paper, we conducted further experiments (see E). While our method exhibits good generalization to unseen poses and shapes, we discuss and show failure cases in Appendix F.

A Network Architectures

Vector Cross Attention: In Figure 1, we illustrate the architecture of vector cross attention [1] (VCA) which is a building block of our transformer-based deformation network (see Figure 3 in the main paper). The feature vectors $\mathbf{g}_i$ and $\mathbf{f}_i$ are transformed with three linear projectors $\varphi(\mathbf{g}_i)$, $\psi(\mathbf{f}_i)$ and $\alpha(\mathbf{f}_i)$, each of which is a fully-connected layer. To leverage relatively positional information of $\mathbf{f}_i$ and $\mathbf{g}_i$, $\mathbf{x}_i - \mathbf{y}_i$ is encoded by a positional embedding module [2, 3] $\delta := \theta(\mathbf{x}_i - \mathbf{y}_j)$ that consists of two linear layers with a single ReLU [4]. Then, the summation result of $\delta(\mathbf{x}_i - \mathbf{y}_j)$ and $\varphi(\mathbf{g}_j) - \psi(\mathbf{f}_i)$ will be further processed by a MLP γ. Next, a softmax function ρ is used to generate normalized attention scores that are used to calculate a weighted combination of $\alpha(\mathbf{f}_i) + \delta(\mathbf{x}_i)$ to obtain $\mathbf{f}'_i$.

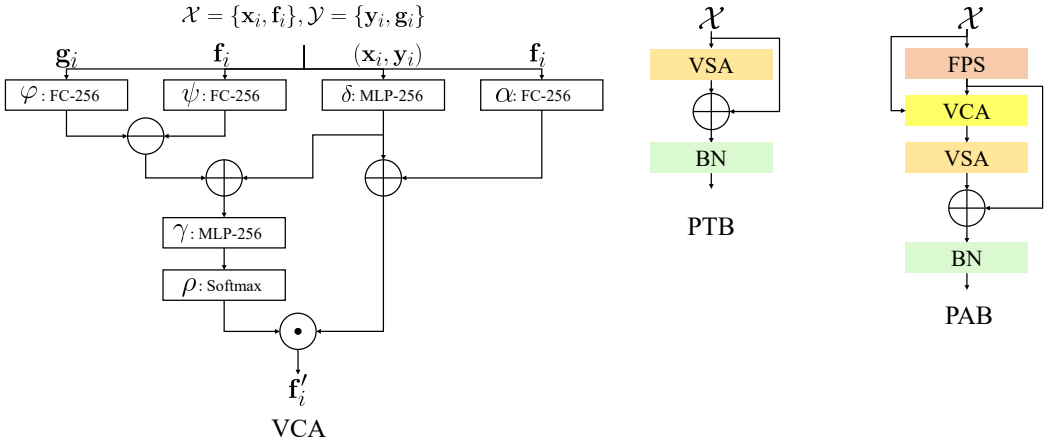


Figure 1: **Vector Cross Attention (VCA), Point Transformer Block (PTB), and Point Abstraction Block (PAB)**.

Point Transformer Block (PTB): As illutrated in Figure 1, we introduce the architecture of point transformer block. The point transformer block is used to encapsulate the information from k_{enc} = 16

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

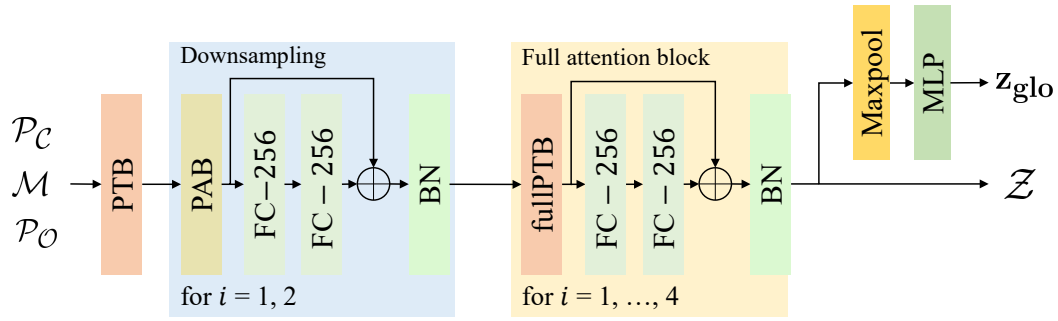


Figure 2: **Point Transformer Encoder**.

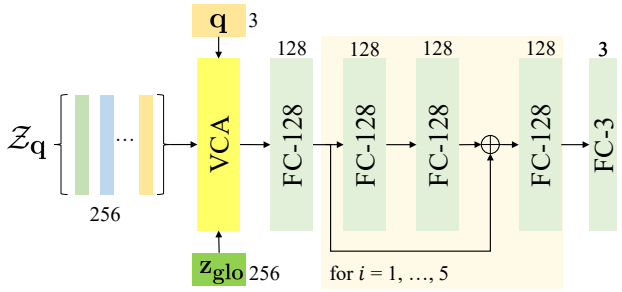


Figure 3: **Attentive Deformation Decoder**.

nearest neighborhoods while keeping the position of a point $\mathcal{X}_i$ unchanged. The input $\mathcal{X}_i$ is fed into a vector attention block (VSA) and through a BatchNorm (BN) [5] (including a residual connection from the input $\mathcal{X}_i$).

Point Abstraction Block (PAB): The point abstraction block consists of a farthest point sampling module (FPS), a VCA module, a VSA module, followed by a BN layer. The farthest point sampling (FPS) is used to downsampled $\mathcal{X}$ which is then fed into a VCA followed by a VSA module. We employ a skip connection from the original $\mathcal{X}$ to the VCA module. The output of the FPS and the VSA module are fed into a batchnorm layer which computes the output of the point abstraction block.

Point Transformer Encoder As shown in Figure 2, a PTB is used to obtain an initial feature encoding $\mathcal{Z}_0 = \{\mathbf{c}_i^0, \mathbf{z}_i^0\}_{i=1}^{n_0}, n_0 = 5000$. Two consecutive point abstraction blocks (PABs) with intermediate set size of $n_1 = 500$ and $n_2 = 100$, are used to obtain downsampled feature point clouds $\mathcal{Z}_1 = \{\mathbf{c}_i^1, \mathbf{z}_i^1\}_{i=1}^{n_1}$ and $\mathcal{Z}_2 = \{\mathbf{c}_i^2, \mathbf{z}_i^2\}_{i=1}^{n_2}$. To enhance global deformation priors, we stack 4 point transformer block with full self-attention whose k_{enc} is set to 100 to exchange the global information in the whole set of $\mathcal{Z}_2$. By doing so, we can obtain a sparse set of local deformation descriptors $\mathcal{Z} = \{\mathbf{c}_i, \mathbf{z}_i\}_{i=1}^{100}$ that are anchored in $\{\mathbf{c}_i\}$. Finally, a global max-pooling operation followed by two linear layers is used to obtain the global latent vector $\mathbf{z}_{\text{glo}}$.

Attentive Deformation Decoder The detailed architecture of attentive deformation decoder is shown in Figure 3. It fuses near-by local latent codes $\mathcal{Z}_{\mathbf{q}}$ of $\mathbf{q}$ under the guidance of a global latent code $\mathbf{z}_{\mathbf{glo}}$ into $\mathbf{z}$, and feeds $\mathbf{z}$ into an MLP consisting of five stacked Res-FC blocks to estimate the displacement vector of $\mathbf{q}$.

B Evaluation Metrics

For defining the evaluation metrics, we assume two meshes $\mathcal{T} = \{\mathcal{V}, \mathcal{F}\}$ and $\mathcal{T}' = \{\mathcal{V}', \mathcal{F}\}$ being the ground-truth and deformed mesh respectively, sharing the same connectivity.

Vertex ℓ_2 error: The vertex ℓ_2 distance error is the mean square distance between ground-truth vertices $\mathcal{V} = \{\mathbf{v}_i\}$ and deformed vertices $\mathcal{V}' = \{\mathbf{v}'_i\}$:

$$\ell_2(\mathcal{T}', \mathcal{T}) := \frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} \|\mathbf{v}_i - \mathbf{v}'_i\|_2^2,$$

where $|\mathcal{V}|$ denotes the number of mesh vertices.

Chamfer distance: To calculate the chamfer distance between $\mathcal{T}'$ and $\mathcal{T}$, we firstly sample two point set $\mathcal{P}_{\mathcal{T}'}$ and $\mathcal{P}_{\mathcal{T}}$ from $\mathcal{T}'$ and $\mathcal{T}$ individually. Then, the Chamfer distance of two point sets is defined as:

$$\mathrm{CD}(\mathcal{T}', \mathcal{T}) := \mathrm{CD}(\mathcal{P}_{\mathcal{T}'}, \mathcal{P}_{\mathcal{T}}) = \sum_{x \in \mathcal{P}_{\mathcal{T}'}} \min_{y \in \mathcal{P}_{\mathcal{T}}} \|x - y\|_2^2 + \sum_{y \in \mathcal{P}_{\mathcal{T}}} \min_{x \in \mathcal{P}_{\mathcal{T}'}} \|x - y\|_2^2.$$

Face Normal Consistency The face normal consistency describes the mean cosine similarity score of the triangle normals of two meshes. Let $\mathcal{N}$ and $\mathcal{N}'$ denote the set of face normals of $\mathcal{T}$ and $\mathcal{T}'$ respectively. We define Face Normal Consistency as:

$$\mathrm{FNC}(\mathcal{T}', \mathcal{T}) := \frac{1}{|\mathcal{N}|} \sum_{i=1}^{|\mathcal{N}|} |\mathbf{n}' \cdot \mathbf{n}|,$$

where $|\mathcal{N}| = |\mathcal{F}|$ denotes the number of triangle faces and $\cdot$ denotes the dot product of two vectors.

C Notation

We will explain our notation in more detail after having briefly defined it in Section 3. By $\mathcal{S}$, $\mathcal{C}$, $\mathcal{T}$, $\mathcal{T}'$ we denote meshes of the considered shapes. $\mathcal{S} = \{\mathcal{V}, \mathcal{F}\}$ is the source mesh and $\mathcal{V}$ is the set of vertices of $\mathcal{S}$ while $\mathcal{F}$ is the set of faces of $\mathcal{S}$. $\mathcal{S}$ is deformed in a 2-step approach. By $\mathcal{C}$ we denote the canonical shape and $\mathcal{T}$ is the target shape. We select a sparse set of handles $\mathcal{H} = \{\mathbf{h}_i\}_{i=1}^{\ell}$ of the original shape. The handles can be dragged to new target locations $\mathcal{O} = \{\mathbf{o}_i\}_{i=1}^{\ell}$ which define the target mesh $\mathcal{T}$. The continuous deformation field learnt in our work is denoted by $\mathbf{D}$. We apply $\mathbf{D}$ to deform the vertices of $\mathcal{S}$ to obtain the deformed mesh $\mathcal{T}' = \{\mathcal{V} + \mathbf{D}(\mathcal{V}), \mathcal{F}\}$ where $\mathcal{V} + \mathbf{D}(\mathcal{V})$ are the vertices of the deformed mesh. We denote the backward deformation field by $\mathbf{D}_b$ and the forward deformation field by $\mathbf{D}_f$. It holds $\mathbf{D}_f(\mathbf{D}_b(\cdot))$. Since our method performs operations in the point cloud domain, we sample point clouds from the surface meshes. $\mathcal{P}_{\mathcal{S}} = \{\mathbf{p}_i\}_{i=0}^{n}$ is a surface point cloud of canonical mesh $\mathcal{S}$ with size $n = 5000$. We define the binary user handle mask as $\mathcal{M} = \{b_i \mid b_i = 1 \text{ if } \mathbf{p}_i \text{ is a handle or } b_i = 0 \text{ else}, i = 1, \ldots, n\}$. The point cloud $\mathcal{P}_{\mathcal{S}}$ is passed through the backward transformation network Ω_b and mapped into the canonical pose $\mathcal{P}'_{\mathcal{C}}$, i.e. $\mathcal{P}'_{\mathcal{C}} = \mathcal{P}_{\mathcal{S}} + \mathbf{D}_b(\mathcal{P}_{\mathcal{S}})$. Then the point cloud $\mathcal{P}'_{\mathcal{C}}$ is passed through the forward transformation network Ω_f and mapped into the target pose $\mathcal{P}'_{\mathcal{T}}$, i.e. $\mathcal{P}'_{\mathcal{T}} = \mathcal{P}'_{\mathcal{C}} + \mathbf{D}_f(\mathcal{P}'_{\mathcal{C}})$. Further, consult Table 1 for the definition of all items.

D Data

To train and evaluate our method, we use the DeformingThing4D [6] dataset, which is available under a non-commercial academic license. It does not contain personally identifiable information or offensive contents. We have obtained the consent to use the dataset.

Train/test split The DeformingThing4D consists of a large number of quadruped animal animations with various motions, such as “bear3EP Jump”, “bear9AK Jump”, or “bear3EP Lie” where "bear3EP" and "bear9AK" are identity names, and "Jump" and "Lie" are motion names. Similar to the D-FAUST [7] used in OFlow [8], the train/test split is based on these identity and motion names of deforming sequences. We firstly divide the animations of the dataset into two parts, seen identities and unseen identities. For the animations of seen identities, we further divide it into seen motions of seen identities (used as training set), and unseen motions of seen identities (used as the test set of S1). The animations of unseen identities are used as the test set of S2. Finally, the train, test S1, and test S2 datasets individually contains 1296, 143, and 55 deforming sequences.

Notations	Meaning
$\mathcal{S}, \mathcal{C}, \mathcal{T}, \mathcal{T}'$	Source mesh, canonical mesh, target mesh, deformed mesh
$\mathcal{V}, \mathcal{F}$	Vertices, faces of source mesh $\mathcal{S}$
$\mathcal{H}, \mathbf{h}_i$	Set of handles, i-th handle location
$\mathcal{O}, \mathbf{o}_i$	Set of target locations of handles, i-th target location
$\mathcal{M}, \mathbf{b}_i$	Binary user handle mask, i-th element of $\mathcal{M}$
$\mathcal{P}_\mathcal{S}, \mathcal{P}_\mathcal{C}, \mathcal{P}_\mathcal{T}$	Surface point clouds of size n sampled from the surface of $\mathcal{S}, \mathcal{C}, \mathcal{T}$
$\mathcal{P}_\mathcal{O}$	Target handle point locations
$\mathcal{Q}_\mathcal{S}, \mathcal{Q}_\mathcal{C}, \mathcal{Q}_\mathcal{T}$	Non-surface point clouds of size m sampled from the 3D space of $\mathcal{S}, \mathcal{C}, \mathcal{T}$
$\mathbf{q}_i$	i-th non-surface querying point
n	Size of surface point clouds $\mathcal{P}_\mathcal{S}, \mathcal{P}_\mathcal{C}, \mathcal{P}_\mathcal{T}$
m	Size of non-surface point clouds $\mathcal{Q}_\mathcal{S}, \mathcal{Q}_\mathcal{C}, \mathcal{Q}_\mathcal{T}$
$\mathbf{p}_i$	i-th point from $\mathcal{P}_\mathcal{S}$
$\mathcal{P}'_\mathcal{C}, \mathcal{P}'_\mathcal{T}$	Mapping of $\mathcal{P}_\mathcal{S}$ in canonical pose, target pose
$\mathbf{D}_b, \mathbf{D}_f$	Backward deformation field, forward deformation field
$\mathbf{D}$	Deformation field between two arbitrary poses, i.e. $\mathbf{D}_f(\mathbf{D}_b(\cdot))$
Ω_b, Ω_f	Backward transformation network, forward transformation network
$\mathcal{X}, \mathcal{Y}$	Query sequence, key-value sequence
$\mathbf{x}_i, \mathbf{f}_i, \mathbf{f}'_i$	Coordinate of i-th query point, corresponding feature vector, aggregated feature
$\mathbf{y}_j, \mathbf{g}_j$	Coordinate of j-th key-value point, corresponding feature vector
VCA	Vector cross attention
φ, ψ, α	Fully-connected layers
γ	Attention weight normalization function, e.g. *softmax* function
δ	Positional embedding module
VSA	Vector self-attention operator
PTB, PAB	Point transformer block, point abstraction block
BN	BatchNorm Layer
$\mathcal{Z}$	Set of local deformation descriptors
$\mathbf{q}, \mathbf{z}_\mathbf{q}$	A point in $\mathcal{C}$, corresponding feature vector
$\mathbf{c}_i, \mathbf{z}_i$	Coordinates and feature vector of i-th deformation descriptor
$\mathbf{z}_{\text{glo}}$	Global latent vector
$L_b, L_f, L_{\text{total}}$	Backward loss function, forward loss function, end-to-end loss function

Table 1: Notations in order of appearance in the main paper.

Data preparation In Section 3.3 of the main text, we mentioned that our method utilizes a set of triplets including source $\mathcal{S}$, canonical $\mathcal{C}$, and target mesh $\mathcal{T}$ with dense correspondence for training. The point clouds $\mathcal{P}_\mathcal{S}, \mathcal{P}_\mathcal{C}, \mathcal{P}_\mathcal{T}$ of size n with one-to-one correspondence are sampled from the surfaces of $\mathcal{S}$, $\mathcal{C}$, $\mathcal{T}$. And the non-surface point sets $\mathcal{Q}_\mathcal{S}, \mathcal{Q}_\mathcal{C}, \mathcal{Q}_\mathcal{T}$ of size m are sampled from their 3D space. Here, we provide the details of data preparation. Firstly, we sample N_p surface points $\{\mathbf{x}_i\}_{i=1}^{i=N_p}$ from the canonical mesh $\mathcal{C}$; we also store the corresponding barycentric weights of sample points. Then, each point is randomly permuted by a small displacement vector $\delta_{\mathbf{n}_i} = \mathbf{x}_i + \beta * \mathbf{n}_i$ along the normal direction $\mathbf{n}_i$ of the corresponding triangle. The displacement distance β is from a Gaussian distribution $N(0, \sigma^2)$. Next, for source $\mathcal{S}$ and target $\mathcal{T}$ meshes, we use the same barycentric weights to obtain $\mathcal{P}_\mathcal{S}, \mathcal{P}_\mathcal{T}$ with correspondences, and use the same displacements $\delta_\mathbf{n}$ to obtain $\mathcal{Q}_\mathcal{S}, \mathcal{Q}_\mathcal{T}$ with correspondences. Concretely, we pre-compute $N_p = 20{,}000$ points from each canonical surface mesh, and get the non-surface points with 50% of surface points permuted by $\sigma = 0.02$, with 50% of surface points permuted by $\sigma = 0.1$. During training, we down-sample $n = 5000$ points of $\mathcal{P}_\mathcal{S}, \mathcal{P}_\mathcal{C}, \mathcal{P}_\mathcal{T}$, and down-sample $m = 5000$ of $\mathcal{Q}_\mathcal{S}, \mathcal{Q}_\mathcal{C}, \mathcal{Q}_\mathcal{T}$. To maintain one-to-one correspondence, we use the same sampling indices for $\mathcal{S}, \mathcal{C}, \mathcal{T}$.

E Additional Results

Effects of point cloud sampling density To study the effect of sampling density of input point cloud, we individually train our model by using point clouds of size 2500, 5000, 7500 as input. Quantitative results are shown in Table 2. We can observe that the results of different evaluation

metrics only show a slightly small variance. To balance accuracy and computational cost, we use 5000 points in our final model.

#sampling points	New motions (S1)			Unseen identities (S2)		
	$\ell_2 \downarrow$	CD $\downarrow$	FNC $\uparrow$	$\ell_2 \downarrow$	CD $\downarrow$	FNC $\uparrow$
Ours-2500	0.789	1.008	96.27	0.905	1.285	96.57
Ours-5000	0.752	0.948	**96.59**	0.795	**1.241**	**96.68**
Ours-7500	**0.732**	**0.944**	96.39	**0.789**	1.251	96.66

Table 2: Quantitative results of different input point cloud density on the S1 and S2 test sets of DeformingThing4D [6] dataset.

Robustness to noisy source mesh To analyze the robustness of noise effects, we individually train our model by adding gaussian noise permutations to the source meshes. The standard deviation of gaussian noise is set to 0, 0.0025 or 0.005. The comparison in Table 3 shows that with the noise becoming larger, the performance of our method experiences only slight variation; however, this demonstrates the robustness of our method to noisy source meshes.

#standard deviation	New motions (S1)			Unseen identities (S2)		
	$\ell_2 \downarrow$	CD $\downarrow$	FNC $\uparrow$	$\ell_2 \downarrow$	CD $\downarrow$	FNC $\uparrow$
Ours-0	**0.752**	**0.948**	**96.59**	**0.795**	**1.241**	**96.68**
Ours-0.0025	0.774	0.973	95.90	0.808	1.278	96.65
Ours-0.0050	0.851	1.017	96.50	0.911	1.392	96.16

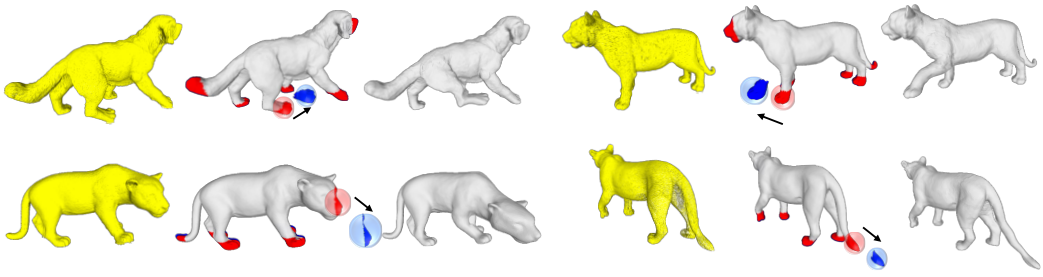
Table 3: Quantitative results of source meshes with different noise intensities on the S1 and S2 test sets of DeformingThing4D [6] dataset.

Robustness to partial source mesh To investigate the robustness to incomplete source meshes, we randomly sample 5 seeds from the source mesh surface, and then remove the corresponding k_r nearest vertices and corresponding faces. The k_r is calculated by $k_r = p_r * |\mathcal{V}|$, where p_r is the incompleteness ratio and $|\mathcal{V}|$ is the number of source mesh vertices. Again, our model is directly evaluated under two different settings of $p_r = 0.05$ and $p_r = 0.1$. The quantitative results are provided in Table 4. As seen, there are not significant numerical variations between different incompleteness ratios. This clearly demonstrates the robustness of our approach to incomplete source meshes.

#incompleteness ratio	New motions (S1)			Unseen identities (S2)		
	$\ell_2 \downarrow$	CD $\downarrow$	FNC $\uparrow$	$\ell_2 \downarrow$	CD $\downarrow$	FNC $\uparrow$
Ours-0.0	**0.752**	**0.948**	**96.59**	**0.795**	**1.241**	**96.68**
Ours-0.05	0.770	0.957	95.80	0.804	1.244	96.66
Ours-0.10	0.823	1.002	96.44	0.858	1.261	96.55

Table 4: Quantitative results of source meshes with different incomplete ratios on the S1 and S2 test sets of DeformingThing4D [6] dataset. Note that our model is directly evaluated on partial meshes without fine-tuning.

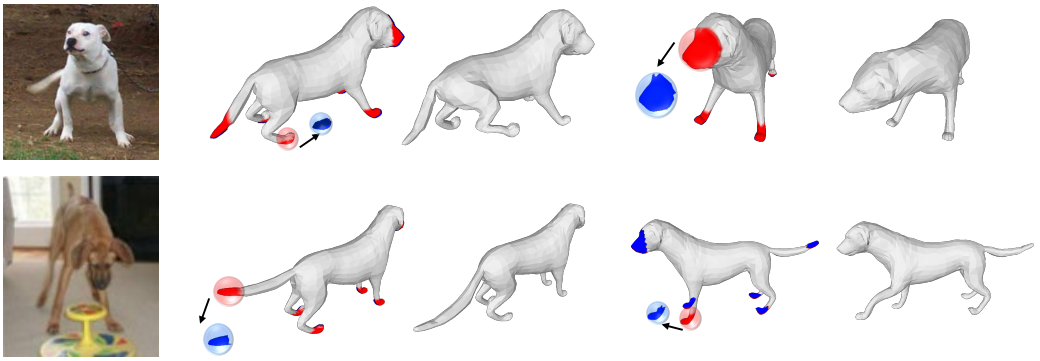
Evaluations on real animals scans. We evaluate our pre-trained model on the real animal scans captured by ourselves. As show in Figure 4, our method can still learn realistic shape deformations, which demonstrates the generalization ability of our approach to real captured models.



(a) (b) (c) (a) (b) (c)

Figure 4: Evaluation on real animal scans. (a) Real animal scans (b) Source meshes obtained via the Screened PSR [9] and handles. (c) Ours.

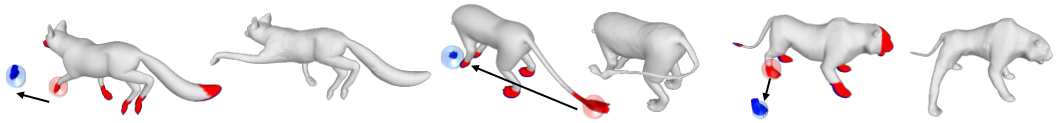
Evaluations on reconstructed animals from real images. In addtion, we evaluate our pre-trained model on the reconstructed animals from real RGB images using the BARC [10] method. As shown in Figure 5, our method estimates realistic deformations for reconstructed animals from natural images. This also demonstrates the generalization ability of our method.



(a) (b) (c) (d) (e)

Figure 5: Evaluation on reconstructed animals from real RGB images using the method of BARC [10] (a) Real images. (b) Reconstructed source meshes and handles. (c) Ours. (d) Reconstructed source meshes and handles. (e) Ours.

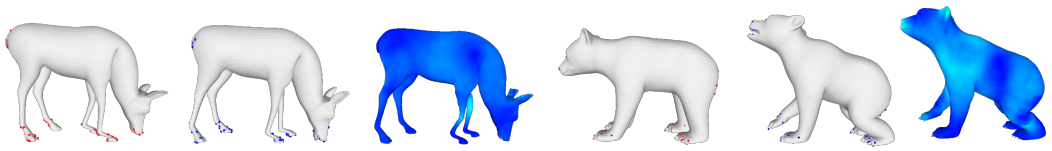
Evaluations on non-realistic user-specified handles. While our goal of data-driven deformation priors is to obtain deformations that are as realistic as possible, we also evaluate our method on non-realistic or non-physical-aware handles. As shown in Figure 6, our method will try to find the closest deformation of animals that can best explain the provided handle displacements. However, our method could be easily trained on non-realistic or non-physical-aware samples and learn the respective deformation behavior.



(a) (b) (a) (b) (a) (b)

Figure 6: Evaluation on non-realistic user-specified handles. (a) Source meshes and handles. (b) Ours.

Without dense correspondence While our current method uses an existing dataset where dense correspondences between temporal mesh frames are available, our framework can also be trained on datasets without dense correspondences through some adjustments on inputs and loss functions. Concretely, we change our method to receive sparse handle correspondences as inputs, and utilize Chamfer distance as the loss function that does not require ground-truth meshes with dense correspondences as supervision. In Figure 7, we visualize several test results of such a modified framework. As seen, without dense correspondences for training, our method can still obtain accurate deformations.

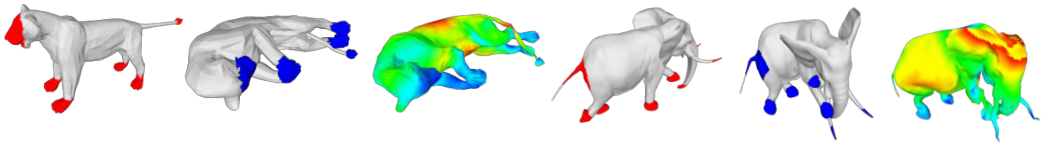


(a) (b) (c) (a) (b) (c)

Figure 7: The evaluation results of our modified framework that uses sparse handles as input and does not require dense correspondences as supervision. (a) Source meshes and handles. (b) Target meshes and handles. (c) Our results with vertex error map.

Video animations To visualize the deformation behaviours of the different approaches, we use a sequence of handle movements as inputs, and run our model frame by frame to obtain a deformation motion sequence. We refer to the supplemental video for an animated sequence.

F Limitations



(a) (b) (c) (a) (b) (c)

Figure 8: The failure cases. (a) Source meshes and handles. (b) Target meshes and handles. (c) Our results with vertex error map.

While compelling results have been demonstrated for shape manipulation, a few limitations still exist in our approach that can be addressed in future work. Two representative failure cases are depicted in Figure 8. We can see that our method cannot well address extreme shape deformations (e.g. left of Figure 8) or manipulate unseen identities that are far from the training data distribution (e.g. the elephant in the right of Figure 8). We believe this issue can be alleviated by a larger training dataset, a richer data augmentation strategy, and/or few shot generalization techniques in the future.

References

[1] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10076–10085, 2020.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[3] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[4] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[6] Yang Li, Hikari Takehara, Takafumi Taketomi, Bo Zheng, and Matthias Nießner. 4dcomplete: Non-rigid motion estimation beyond the observable surface. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12706–12716, 2021.

[7] Federica Bogo, Javier Romero, Gerard Pons-Moll, and Michael J Black. Dynamic faust: Registering human bodies in motion. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6233–6242, 2017.

[8] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5379–5389, 2019.

[9] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.

[10] Nadine Rüegg, Silvia Zuffi, Konrad Schindler, and Michael J Black. Barc: Learning to regress 3d dog shape from images by exploiting breed information. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3876–3884, 2022.