– Supplemental Document –
NPMs: Neural Parametric Models for 3D Deformable Shapes

Pablo Palafox[1] Aljaž Božič[1] Justus Thies[1,2] Matthias Nießner[1] Angela Dai[1]
[1]Technical University of Munich [2]Max Planck Institute for Intelligent Systems, Tübingen

In this appendix, we provide additional details about our proposed Neural Parametric Models. Specifically, we describe the training data preparation (Sec. 1), the network architectures (Sec. 2), as well as training details including the hyper-parameters (Sec. 3), and information about the inference-time optimization (Sec. 4). Additional qualitative evaluations and results are shown in the supplemental video.

1. Data Preparation

1.1. Waterproofing Canonical Shapes

We waterproof canonical shapes by first rendering depth maps of the non-watertight shape inside a virtual multi-view setup, then computing partial surface meshes from each depth map and finally running Poisson surface reconstruction [1] on the merged set of partial meshes. Although this approach is not guaranteed to produce watertight meshes, especially when applied to meshes with large holes, we found it to work well on our canonical shapes, allowing us to retain details present in the original (non-watertight) meshes.

1.2. Train Samples

Shape space. For each shape identity i in the train dataset, given their watertight mesh in canonical pose, we sample a total number of N_s points $\{\boldsymbol{x}_i^k\}_{k=1}^{N_\mathrm{s}} \in \mathbb{R}^3$, along with their corresponding ground truth SDF value $\{s_i^k\}_{k=1}^{N_\mathrm{s}} \in \mathbb{R}$ (see Fig. 1). As explained in the main paper, these samples come from two sources:

1. N_s^ns near-surface points sampled randomly within a distance of 0.05 from the surface of the shape.
2. N_s^u points uniformly sampled within the unit bounding box, such that $N_\mathrm{s} = N_\mathrm{s}^\mathrm{ns} + N_\mathrm{s}^\mathrm{u}$.

In our experiments, we set $N_\mathrm{s}^\mathrm{ns} = 300\mathrm{k}$ and $N_\mathrm{s}^\mathrm{u} = 100\mathrm{k}$, for a total of $N_\mathrm{s} = 400\mathrm{k}$.

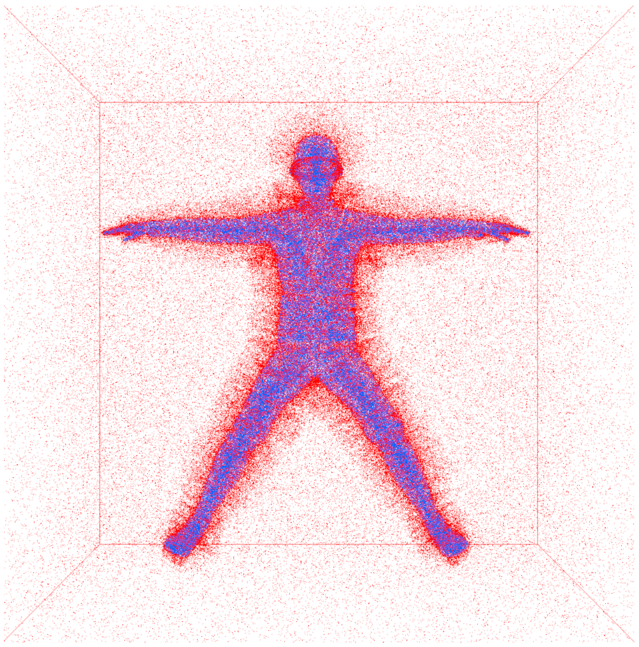
Figure 1: We visualize the N_s training points available for a given identity for training the shape MLP. Points in red are outside of the canonical shape, and so have a positive SDF value. Points in blue, on the other hand, have a negative SDF value, since they reside within the shape. Note that we sample densely near the surface ($N_\mathrm{s}^\mathrm{ns} = 300\mathrm{k}$). Additionally, we sample points in the unit bounding box of the shape ($N_\mathrm{s}^\mathrm{u} = 100\mathrm{k}$).

Pose space. In Sec. 3.3 of the main text, we explain the procedure for generating flow samples for training the pose MLP; here, we provide additional details. First, we sample N_p surface points $\{\boldsymbol{x}_i^k\}_{k=1}^{N_\mathrm{p}}$ on the normalized canonical shapes for each i-th identity in the dataset; we also store the barycentric weights for each sampled point. Each point is then randomly displaced a small distance $\boldsymbol{\delta n} \sim \mathcal{N}(0, \boldsymbol{\Sigma}^2)$ along the normal direction of the corresponding triangle in the mesh, with $\boldsymbol{\Sigma} \in \mathbb{R}^3$ a diagonal covariance matrix with entries $\boldsymbol{\Sigma}_{ii} = \sigma$. Then, for each j-th posed shape available for the identity, we compute corresponding points $\{\boldsymbol{x}_j^k\}_{k=1}^{N_\mathrm{p}}$

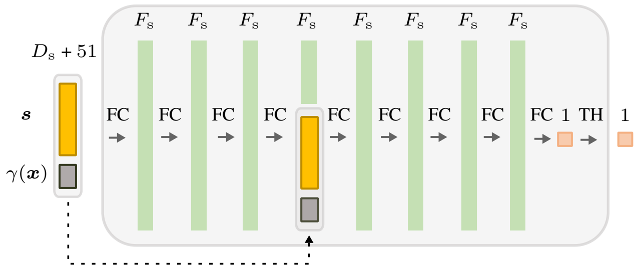

Figure 2: A detailed visualization of our shape MLP.

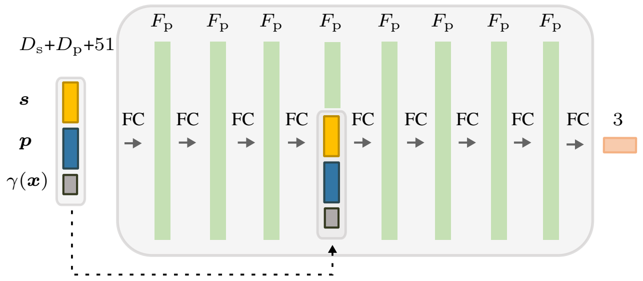

Figure 3: A detailed visualization of our pose MLP.

in the posed shape by using the same barycentric weights and $\boldsymbol{\delta n}$ to sample the posed mesh. This approach gives us a deformation field (defined near the surface) between the canonical pose of a given identity i and a deformed pose j of the same identity.

In our experiments, we pre-compute $N_p = 200$k correspondences for every j-th posed shape available in the dataset. We obtain good results by sampling 50% of these points with $\sigma = 0.01$, and 50% with $\sigma = 0.002$ (in normalized coordinates).

2. Network Architecture

2.1. Shape and Pose Auto-decoders

Figures 2 and 3 detail our feed-forward networks for learning our latent shape and pose spaces. Both MLPs are composed of 8 fully connected layers, applied with weight-normalization [6]. We use ReLU activations as non-linearities after every intermediate layer. After the final layer of our shape MLP we use tanh to regress an SDF value, whereas for our pose MLP we directly regress a 3-dimensional flow vector. For both networks, a skip connection is used at the fourth layer.

To learn our human NPM, we set the feature size F_s of each layer in our shape MLP to 512, and $D_s = 256$. For learning a hand NPM, we use $F_s = 64$ and $D_s = 16$. When learning the latent pose space, we set $F_p = 1024$ and $D_p = 256$ in the case of humans, and $F_p = 256$ and $D_p = 64$ for hands. We use the positional encoding proposed in [5] to encode the query point $\boldsymbol{x}$ for both our shape and pose MLPs, and use 8 frequency bands. Positional encoding is denoted by $\gamma(\boldsymbol{x})$ in Figures 2 and 3.

2.2. Shape and Pose Encoders for Initialization

As presented in the main paper, to provide a good initialization for our latent-code optimization at test time, we train two 3D convolutional encoders f_{Ω_s} and f_{Ω_p} to predict initial estimates of the latent shape and pose codes, respectively. Both encoders take as input the back-projected depth observation in the form of a partial voxel grid V. We then employ 3D convolutions and a final fully-connected layer to output a latent code estimate.

Shape encoder. In particular, given the list of shape codes $\{\boldsymbol{s}_i\}_{i=1}^S$ learned from the S identities in the train dataset, and a set of P voxel grids of the available P posed shapes in the training dataset $\{V_j\}_{j=1}^P$, we train f_{Ω_s} to predict the mapping from the voxel grid V_j to the corresponding shape code of the underlying identity.

Pose encoder. Similarly, given the list of pose codes $\{\boldsymbol{p}_j\}_{j=1}^P$ learned from the P posed shapes in the dataset, and the set of P voxel grids $\{V_j\}_{j=1}^P$, we train f_{Ω_p} to predict the mapping from the voxel grid V_j of a posed shape to the corresponding pose code.

Fig. 4 visualizes the encoder architecture employed for f_{Ω_s}, with D the output latent code dimension. A similar architecture is employed to learn f_{Ω_p}, differing only in the output channel dimension of the 3D convolution operations. In particular, in our pose encoder we employ output channel dimensions of 16, 32, 64, 128, 256, 256, respectively for each 3D convolutional block in Fig 4.

3. Training Details

We train both our shape and pose MLPs until convergence; in practice, this required 4000 epochs for the shape MLP, and 150 epochs for the pose MLP, which amounts to a similar number of iterations for each. We use a batch size of 4 in both cases. Our human NPM was trained on a GeForce RTX 3090 for approximately 8 days in total, which could be accelerated by parallelizing training on multiple GPUs.

Shape space. For each identity in a mini-batch, out of the total N_s samples available for a given identity in its canonical shape, we randomly sub-sample 50k training points, 70% of which are drawn from the available N_s^{ns} near-surface points, with the remaining 30% drawn from the set of N_s^{u} uniform samples.

Pose space. For each posed shape in a mini-batch, out of the total N_p flow samples available, we randomly sub-sample 50k correspondences.

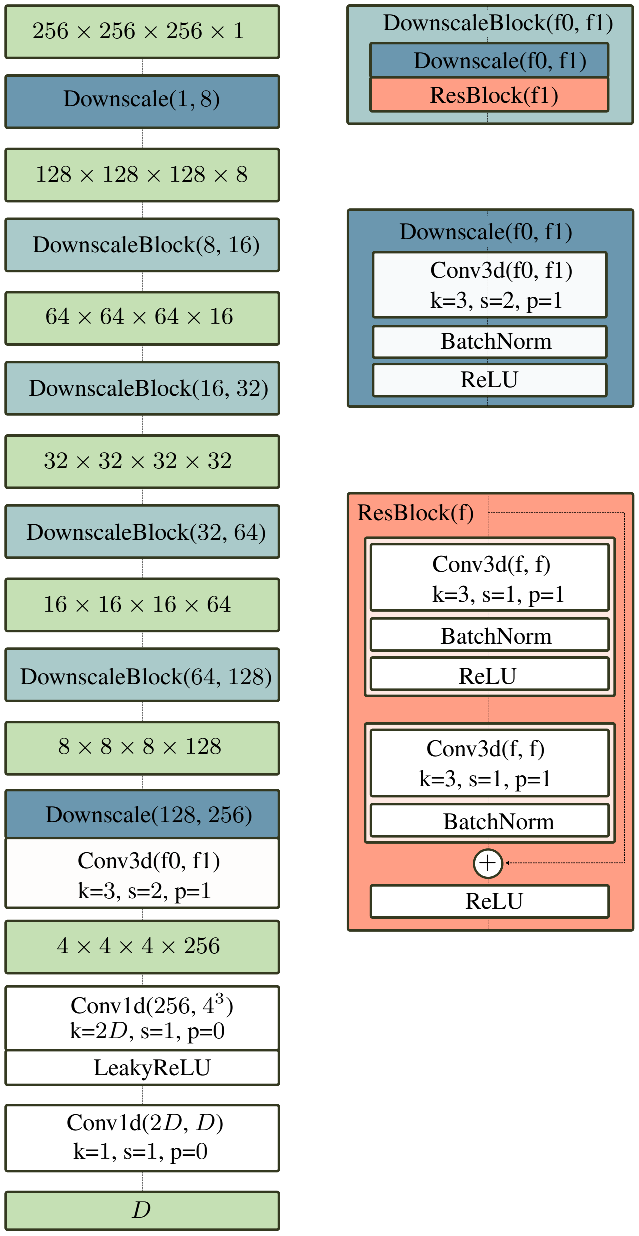

Figure 4: A visualization of our 3D shape encoder for latent-code initialization at test time. Green blocks represent tensors, while the remaining blocks depict operations.

4. Inference-time Optimization Details

When fitting an NPM to an input monocular depth sequence of L frames, we optimize for the shape code $\boldsymbol{s}$ and L pose codes $\{\boldsymbol{p}_j\}_{j=1}^{L}$ by minimizing Eq. 5 from the main text. In our experiments, we optimize for a total of $I = 1000$ iterations. We use the Adam optimizer [2] and learning rates of 5×10^{-4} and 1×10^{-3} for the shape and pose codes, respectively. We decrease these learning rates by a factor of 0.5 after every 250 iterations, and control the temporal regularization loss $\mathcal{L}_t$ in Eq. 5 with a weight of 100.

During test-time optimization, each element in a mini-batch consists of one frame from the sequence; we use a batch size of 4. Before optimization begins, we make initial estimates of the shape code and the L pose codes. For the former, we use our shape encoder f_{Ω_s} to make shape code estimates for all L frames in the input sequence. We then compute the average vector and use this as initial shape code estimate. As for the pose codes, we employ our pose encoder f_{Ω_p} to estimate a pose code for each frame in the sequence.

At this point, given the initial shape code estimate, we can extract an initial canonical shape by querying our learned shape MLP on a 3D grid, and then running Marching Cubes to extract the surface [3], as explained in Sec. 3.2. We then pre-sample $N_t = 500$k points, $\{\boldsymbol{x}_k\}_{k=1}^{N_t}$, around this initial estimate of the canonical shape; during optimization, for each frame in a mini-batch, we sub-sample $N_b = 20$k points –out of the available N_t– to minimize Eq. 5.

As presented in the main text, we employ an additional ICP-like (Iterative Closest Point) term –denoted by $\mathcal{L}_\mathrm{icp}$ in Eq. 5– that enforces surface points to be close to the input point cloud Q_j, which results from back-projecting the observed depth map. In particular, at the beginning of every iteration, out of the N_b points $\boldsymbol{x}_k$ sampled in the canonical space for each element in a mini-batch, we select those points $\boldsymbol{x}_k^\mathrm{ns}$ within a distance ϵ_icp from the implicitly represented surface of the canonical shape, i.e., $\boldsymbol{x}_k^\mathrm{ns} = \boldsymbol{x}_k \mid |f_{\theta_\mathrm{s}}(\boldsymbol{s}, \boldsymbol{x}_k)| < \epsilon_\mathrm{icp}$ (in our experiments, $\epsilon_\mathrm{icp} = 0.001$ in normalized coordinates). Then, for every observed point $\boldsymbol{q} \in Q_j$, we minimize the distance to its nearest neighbor in the set of predicted model points in the j-th frame, denoted by $\mathcal{R} = \{\boldsymbol{x}_k^\mathrm{ns} + f_{\theta_\mathrm{p}}(\boldsymbol{s}, \boldsymbol{p}_j, \boldsymbol{x}_k^\mathrm{ns})\}$:

$$\mathcal{L}_\mathrm{icp} = \lambda_\mathrm{icp} \sum_{\boldsymbol{q} \in Q_j} \|\boldsymbol{q} - \mathrm{NN}_\mathcal{R}(\boldsymbol{q})\|_2 . \tag{1}$$

In the above equation, $\mathrm{NN}_\mathcal{R}(\cdot)$ denotes a function that queries the nearest neighbor of a 3D point in a set of points $\mathcal{R}$. We control the importance of this loss with λ_icp, which we set to 0.0005 in our experiments. We found this loss to be specially beneficial at the beginning of the optimization, in order to avoid falling in local minima. We then disable it, i.e., $\lambda_\mathrm{icp} = 0$, after $I/2$ iterations. In Tab. 1 we show the contribution of our ICP term.

As a reference, optimizing over an input sequence of 100 frames takes approximately 4 hours on a GeForce RTX 3090 with our unoptimized implementation.

Method	**IoU** $\uparrow$	**C-ℓ_2** $(\times 10^{-3}) \downarrow$	**EPE** $(\times 10^{-2}) \downarrow$
Ours (w/o Positional Enc.)	0.76	0.142	1.29
Ours (code size 128)	0.81	0.052	0.95
Ours (code size 384)	0.82	0.076	1.05
Ours (w/o temp. consist.)	0.82	0.037	0.81
Ours (w/ code consist.)	0.82	0.034	0.75
Ours (w/o ICP loss)	0.83	0.031	0.79
Ours	**0.83**	**0.022**	**0.74**

Table 1: Ablation study on CAPE [4] to evaluate the contribution of our different modules and network choices.

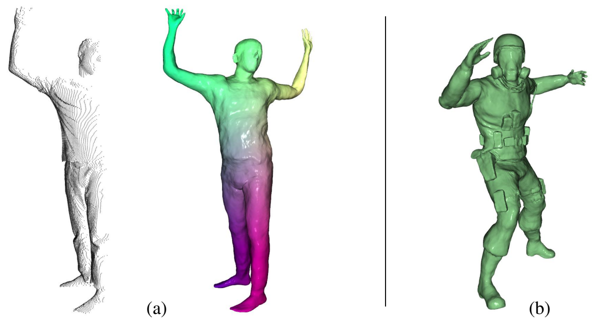

Figure 5: (a) Robustness in the presence of strong occlusions. (b) Example of complex topology and clothing.

5. Ablation Studies

Table 1 shows an ablation study on the effect of training the shape and pose MLPs without positional encoding, on the latent code size, as well as on the effect of several test-time hyperparameters: not using any kind of temporal consistency, enforcing temporal consistency on the pose codes (instead of on the flow predictions), and, finally, on the effect of our ICP loss, as already mentioned in Sec. 4.

6. Robustness to Initialization

Latent code initialization via learned encoders offers additional robustness against local minima, but even without such initial code estimates, our method achieves competitive performance (Table 1 in the main text). Additionally, methods such as SMPL rely on strong initialization, which we provide with human joint predictions computed with OpenPose. Without this additional guidance, optimizing over SMPL parameters becomes ill-posed, particularly for the task of monocular depth fitting. In contrast, our proposed model fitting is robust to self-occlusions caused by rotations (Fig. 5a).

7. Consistency of Shape Fitting

To evaluate the consistency of shape fitting in NPMs, we reconstructed two consecutive sub-sequences of 50 frames each, and measured the similarity between the corresponding shape estimates, which matched almost perfectly (0.96 IoU and 1.61×10^{-6} chamfer).

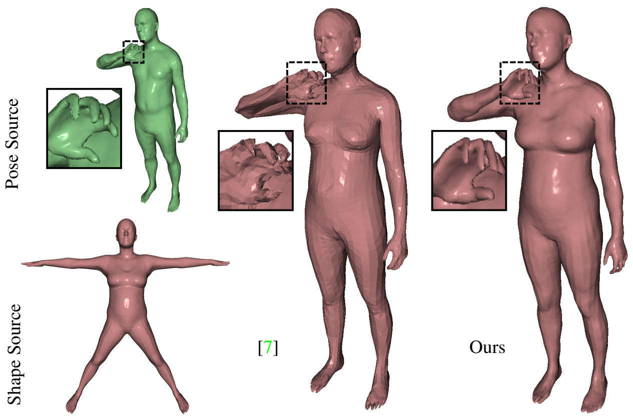

Figure 6: Comparison with [7] on the task of pose transfer.

8. Additional Examples

To further showcase the expressiveness and potential of NPMs, in Figure 5b we show an additional identity drawn from the latent space featuring complex topologies and clothing.

9. Comparison with Zhou et al. [7]

Additionally, we compare with [7] on the task of pose transfer. In contrast to [7], our focus is to construct disentangled spaces which enable test-time fitting to new observations. Furthermore, [7] uses a mesh auto-encoder, limiting the approach to only representing a fixed topology. Our approach naturally deals with complex topology, as we have shown in Sec. 4 in the main paper. For this experiment, we trained an NPM on the same human body dataset as [7], and show a comparison in Fig. 6.

References

[1] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. 1

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[3] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. 3

[4] Qianli Ma, Jinlong Yang, Anurag Ranjan, Sergi Pujades, Gerard Pons-Moll, Siyu Tang, and Michael J. Black. Learning to dress 3d people in generative clothing. In *Computer Vision and Pattern Recognition (CVPR)*, June 2020. 4

[5] 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. 2

[6] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016. 2

[7] Keyang Zhou, Bharat Lal Bhatnagar, and Gerard Pons-Moll. Unsupervised shape and pose disentanglement for 3d meshes. In *European Conference on Computer Vision*, pages 341–357. Springer, 2020. 4