Supplementary Materials of TutteNet: Injective 3D Deformations by Composition of 2D Mesh Deformations

Bo Sun
UT Asutin
bosun@cs.utexas.edu

Thibault Groueix
Adobe Research
groueix@adobe.com

Chen Song
UT Austin
song@cs.utexas.edu

Qixing Huang
UT Austin
huangqx@cs.utexas.edu

Noam Aigerman
University of Montreal
noam.aigerman@umontreal.ca

1. Additional Results

1.1. Additional Results of NeRF Deformations (Section 4.1 in the main paper)

We produced more NeRF deformations, minimizing the elastic energy of the deformation (Section 4.1 in the main paper). Results are shown in Figure 1. We also show more qualitative results on real NeRFs we captured with a smartphone in Figure 2.

1.2. Additional Results for Fitting and Learning of Human Poses (Section 4.2 in the main paper)

More results on the fitting and learning experiments can be found in Figure 3.

1.3. Multi-view Videos for our NeRF Deformation Results (Section 4.1 in the main paper)

You can find the videos for 3 NeRF deformations shown in the main paper in the NeRF_videos.zip.

1.4. Intermediate Deformation Process

We further show the detailed intermediate deformations of our method (see attached intermediate_deformations.gif). Our method explicitly deforms the space by composing a sequence of 2D mesh deformations. The spatial points are deformed accordingly. By choosing different 3D orientations, we get our final 3D deformation.

2. Ablation Study

Since our method proposes a new representation, we ablate on the main parameters of our model: number of Tutte layers, and the mesh resolution while performing the fitting experiment from Section 4.2 - this experiment directly validates the capacity of our representation to fit to deformations as we modify the ablated parameters. We show results in Table 1 - we report the vertex and mesh gradient terms (as in Table 2 in the main paper), both multiplied by 10^3. We report average timings at the bottom. As expected, increasing any of these two parameters improves performance at the price of a slower computation.

The one additional parameter that could be ablated is the local coordinates $\mathbf{R}^i$. We ablate on them by running the learning experiment (Section 4.2 of the main paper) with different methods to choose $\mathbf{R}^i$. Table 2 shows the results. We show three variants for plane choices: regular triplane (alternating between 3 canonical coordinate systems on the 3 main axes); cubic (alternating between 8 vertex directions); and predicting $\mathbf{R}^i$ using a neural network. The best option is to let a neural network control the local coordinates.

Mesh Resolutions	7	11	17	25
Fitting Vert.	0.22	0.15	0.11	0.09
Fitting Grad.	5.6	4.4	2.9	2.1
Forward Time	0.088	0.094	0.118	0.166
Jacobian Time	0.02	0.02	0.02	0.02

(a) Ablation study on the Tutte mesh resolutions. We report the L_2 and Grad. errors on the fitting experiment with 8 tri-plane models.

Num of Layers	6	12	24	36
Fitting Vert.	1.29	0.24	0.15	0.12
Fitting Grad.	12.5	6.1	4.4	3.2
Forward Time	0.025	0.048	0.094	0.142
Jacobian Time	0.005	0.011	0.020	0.029

(b) Ablation study on increasing the number of layers (mesh resoution is fixed to 11×11 vertices) We report the L_2 and Grad. errors on the fitting experiment (Table 2 in the main paper).

Table 1. Ablation study on the Tutte mesh resolutions and number of Tutte layers.

3. Detailed Timing Analysis

We show a detailed timing analysis for our method in Figure 4. All numbers are averaged on 200 pairs in the fitting experiments (Section 4.2 in the main paper). With an in-

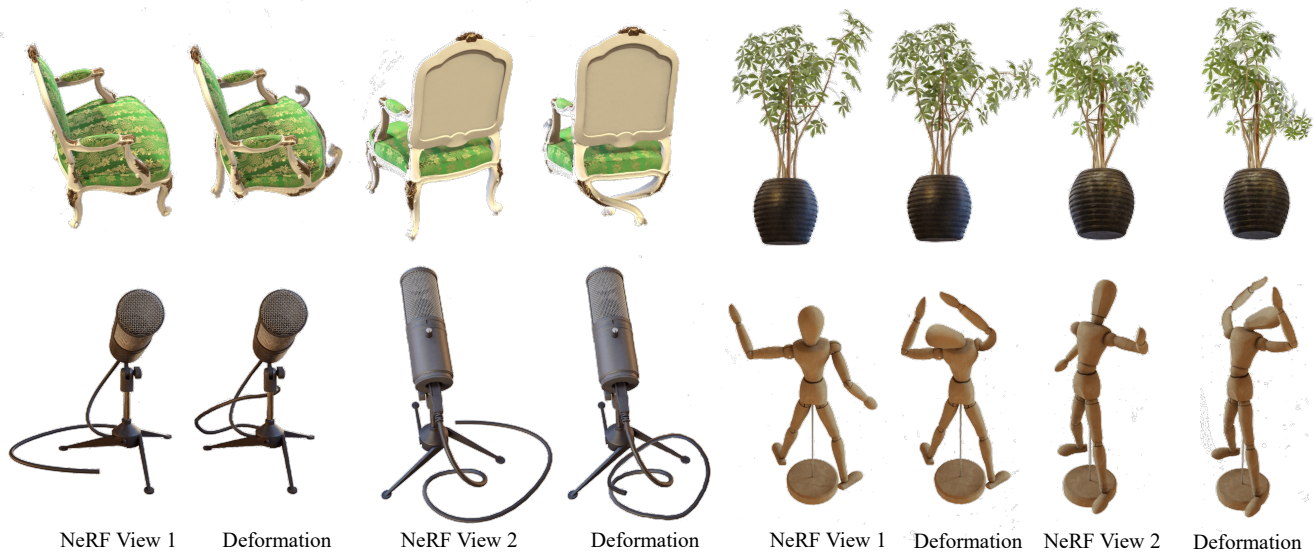

Figure 1. Additional elastic deformations of various NeRFs [7] via our representation, as described in Section 4.1 in the main paper.

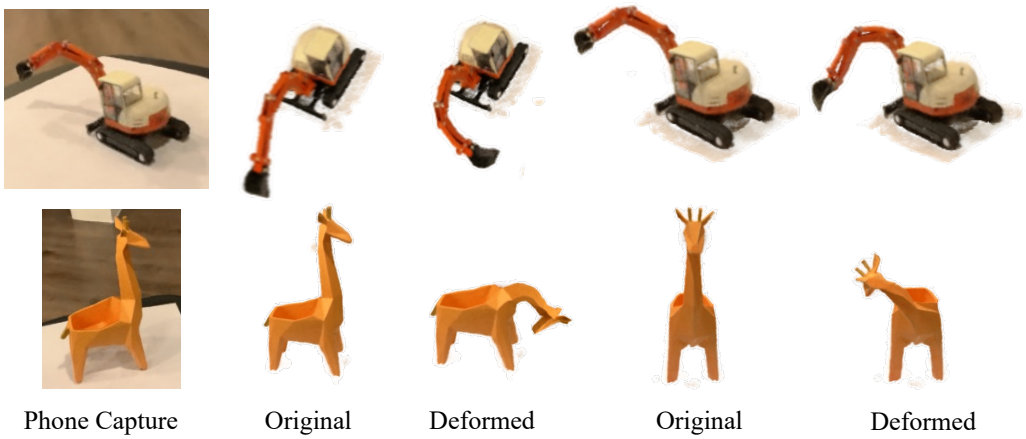

Figure 2. Additional elastic deformations of phone-captured NeRFs [7] via our representation.

	Triplane	Cubic	Predicted
Learning Vert.	0.25	0.21	0.16
Learning Grad.	8.9	8.4	7.7

Table 2. Ablation on choices of orientations in the learning experiment. In a triplane manner, we alternate orientations among x, y, z axes. In a cubic manner, we chose the orientations of 8 vertices of the unit cube and alternated among those 8 directions. In a predicted manner, we use an MLP to predict orientations for all layers. In this experiment, we use 24 layers with a mesh resolution 11×11.

crease of number of Tutte layers, the time increases linearly as each layer's computation takes a fixed amount of time. The 2D mesh resolution, on the other hand, can be increased while both the Jacobian computation time and the inference time remain close to constant, as their computation per layer is not affected by mesh resolution significantly. Of course, computing the Tutte embedding (solving the linear system Eq. 3 in the main paper), the inverse time and the back-propagation time increase as the mesh becomes denser.

4. Limitation: Non-localized Effect of the TutteNet Representation (Section 5 in the main paper)

As mentioned in Section 5 in the main paper, one limitation of our method (that all other injective methods share) is that deforming one part of space may have an effect on another part, and it is non-trivial to completely localize deformations to one part of a shape. We show an example of this issue in Figure 5. On the left, we show the source model and the constraint (green) dragging the hand to a new po-

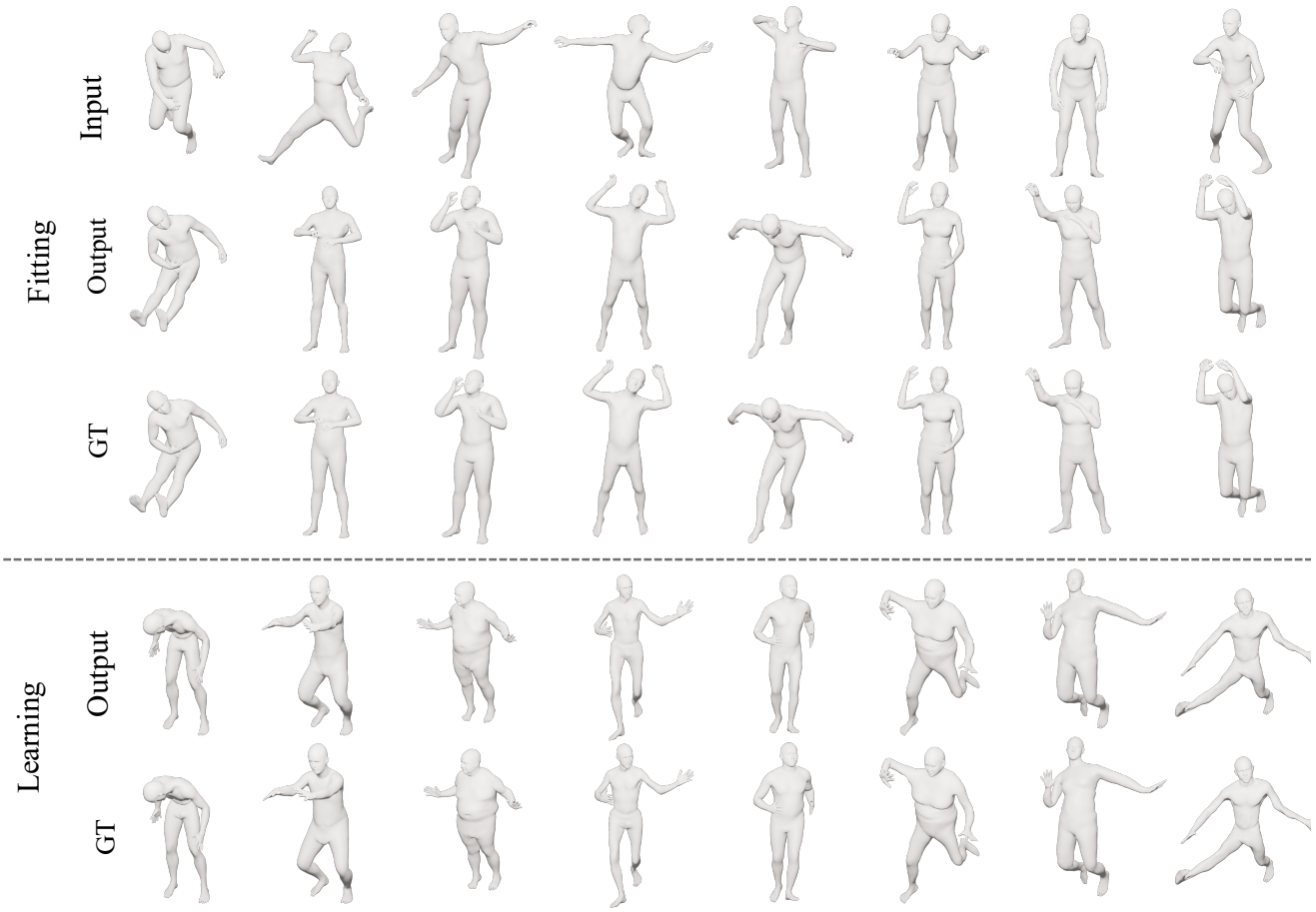

Figure 3. Additional results on the learning (bottom) and fitting (top) human poses experiment (section 4.2).

sition. Second, from left, we show the result of optimizing for the fitting of the constraint, without applying any regularization to other parts of the shape; the unconstrained part moves as the changes the TutteNet performs to fit the constraint have global effect. Third from the left: once we add a distortion minimization regularizer to every other part of the human, the hand goes to place and the entire TutteNet converges into emulating the deformation which matches the constraint and minimizes the elastic energy: a global rotation. Right: The result of applying the same constraint, but regularizing to keep the entire body of the human (blue) static, allowing only a small potion of the hand to bend.

5. Computation of the Jacobian of the Deformation

The deformation's Jacobian can be computed in a quick and straightforward manner. Let $\mathbf{p} \in \mathbb{R}^3$, and define a prismatic map Φ^i with respect to $\mathbf{R}^i$, Ψ^i as in Section 3.2. Then the Jacobian of Φ^i at point $\mathbf{p}$ is

$$D_{\mathbf{p}}\Phi^i \equiv \mathbf{R}^i \tilde{A}_{\mathbf{t}} {\mathbf{R}^i}^T, \tag{1}$$

where $\mathbf{t}$ is the triangle the point lies in (per Algorithm 1), and $\tilde{A}_{\mathbf{t}} = \begin{pmatrix} A_{\mathbf{t}} & \begin{matrix}0\\0\end{matrix} \\ \begin{matrix}0 & 0\end{matrix} & 1 \end{pmatrix}$ is the 2D Jacobian of the 2D mesh deformation at point $\mathbf{p}$, lifted to 3D. Finally, the Jacobian of the map f at point $\mathbf{p}$ can be computed by applying the chain rule to equation (6),

$$D_{\mathbf{p}} f = \Pi_{i=o}^{n} D_{\mathbf{p}}\Phi^i. \tag{2}$$

6. Training Details of the Fitting and Learning Experiments (Section 4.2 in the main paper)

Choices of Shape Pairs in the Shape Fitting Experiment. We randomly selected pairs of shapes from the AMASS training set [6]. In order to focus our fitting experiments solely on pose changes, we aligned the shape parameters of the source shapes with those of their corresponding target

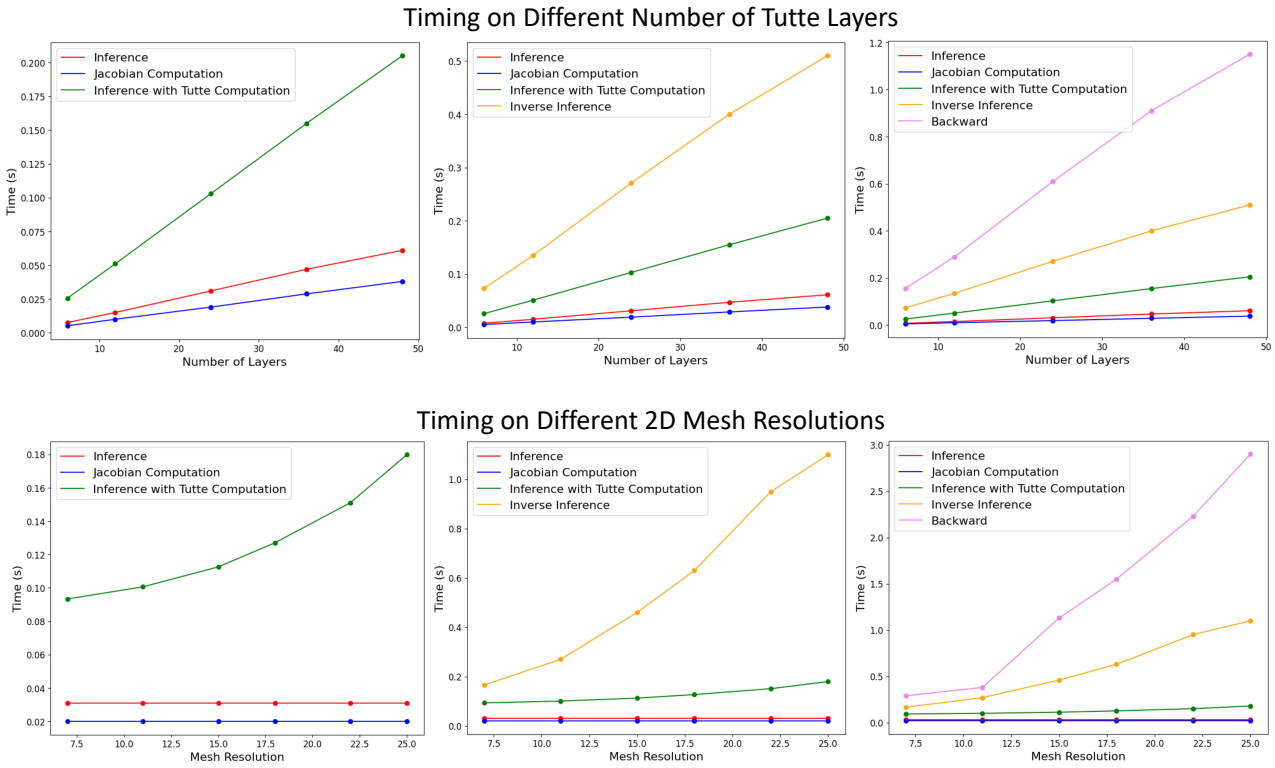

Figure 4. Timing analysis on number of Tutte layers and 2D mesh resolutions. Top: timing w.r.t. number of Tutte layers. Bottom timing w.r.t. the 2D mesh's resolutions, with the fixed number of layers as 24.

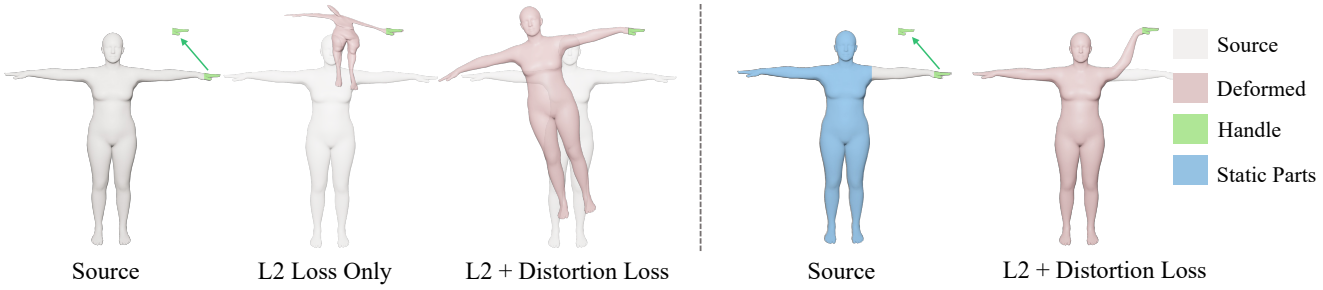

Figure 5. Our deformation is not localized: we show the *source* model and the constraint (green) dragging the hand to a new position. Second from left, we show the result of optimizing for the fitting of the constraint, without applying any regularization to other parts of the shape; the unconstrained part moves as the changes the TutteNet performs to fit the constraint have global effect, modifying the deformation in every part of the space. Third from the left: once we add a distortion minimization regularizer to every other part of the human, the hand goes to place, and the entire TutteNet converges into emulating the deformation which matches the constraint and minimizes the elastic energy: a global rotation. Right: The result of applying the same constraint, but while regularizing to keep the entire body of the human (blue) static, allowing only a small portion of the hand to bend.

shapes. However, this alignment could potentially result in self-intersections in the source shape due to the modification of its parameters. To address this, we excluded pairs with self-intersecting source shapes and retained 200 pairs for the evaluation set in our fitting experiment

Dataset Generation of the Learning Experiment. Our training set for the learning experiment is derived from the AMASS dataset [6]. Rather than directly utilizing the training set provided by AMASS, which contains numerous repetitive and closely related poses, we opt to construct our dataset by randomly sampling from a Gaussian distribution based on the pose and shape distributions observed in the AMASS dataset. To achieve this, we calculate the mean (μ_s, μ_p) and variance (σ_s, σ_p) for all shape and pose

parameters in the AMASS training dataset. Subsequently, we sample our dataset's shape parameters with a mean of μ_s and a variance of $2\sigma_s$, while pose parameters are sampled with a mean of μ_p and a variance of $1.5\sigma_p$. Our model is then trained on this randomly sampled dataset, and its performance is evaluated on the AMASS validation set.

Network Architecture of the Learning Experiment. The detailed process of data preparation and the model architecture are illustrated in Figure 6. For data preparation, we generate eight depth images for the input shapes and feed them into the CLIP [9] and DINO-V2 [8] image backbones to extract image features. These features, along with the target pose parameters, serve as input for the deformation model. In the comparisons presented in the main paper, we consistently set the source pose parameters to the canonical pose. The source shape parameters and target pose parameters are sampled following Section 6. During inference, the input is not necessarily restricted to the SMPL model. Instead, we directly take the 3D model as input and render images onto it. During model forwarding, the image features are initially encoded in smaller feature vectors. These encoded features, along with the target pose parameters, act as conditioning vectors, guiding the prediction of Tutte parameters. Three networks are used to predict edge weights, boundary angles, and plane orientations. The edge MLP takes the positional encoding of each edge center, along with the conditioning vectors, as input and outputs the edge weights for all layers of each edge. Similarly, the boundary MLP takes the positional encoding of each boundary vertex position and the conditioning vectors as input, producing the boundary angle for all layers of each boundary vertex. The orientation MLP takes only the conditioning vector as input and outputs the orientations for all layers. In our experiments, we set the number of layers to 24, the resolution of the mesh to 11×11, and the positional encoding frequency to 50. Additional details on channel dimensions can be found in Figure 6

7. Detailed Baseline Settings

7.1. NeRF Deformation Baselines (Section 4.1 in the main paper)

- **NeRF-Editing [11]** We adhere to the procedures outlined in the official GitHub repository at https://github.com/IGLICT/NeRF-Editing. For the Lego data set, we utilize the checkpoint and cage data provided by the manufacturer. In the case of the Trex and Robot datasets, we follow the instructions on GitHub and receive direct guidance from the authors to train the model and generate the cage. During the editing phase, we input their extracted mesh into our pre-optimized model, incorporating specified handle constraints to obtain the deformed mesh. Subsequently, we follow their prescribed steps to achieve the final rendering results.
- **Deforming-nerf [10]** We closely follow the procedures outlined in the official GitHub repository at https://github.com/xth430/deforming-nerf. This method necessitates an initial deformation of the cage vertices, with subsequent harmonic coordinate interpolation employed to determine the corresponding deformed positions for ray points. Typically, users manually perform the deformation of the cage vertex. However, in our case, we lack explicit instructions on how to manipulate cage vertices to meet handle constraints. Instead, we employ a different approach. Initially, our pre-optimized model is used to obtain the deformed positions for the shape mesh. Leveraging the differentiability of barycentric interpolation, we optimize the cage vertices so that their interpolation leads to the deformed shape positions. In this optimization process, we consider the cage vertices as variables subject to optimization. The procedure takes the undeformed shape points as input, utilizes the cage vertices to derive the deformed shape points, and optimizes the L_2 loss between the resulting points and the ground truth (GT) deformed points, those generated by our deformed model. We sample 10,000 points from the shape and iteratively optimize the L_2 loss until stability is reached and the loss is lower than 1×10^{-5}. For the Lego and Robot datasets, the author has generously provided pre-trained models. However, we trained the Trex model from scratch following the provided instructions.
- **SPIDR [5]** We adhere to the guidelines presented in the official GitHub repository available at https://github.com/nexuslrf/SPIDR. For elastic deformation, we employ the notebook accessible at this link. With handle constraints specified, the original method utilizes open3d for mesh deformation, a process that occasionally yields unsatisfactory outcomes due to non-injectivity issues. To ensure a fair comparison, we substitute the open3d deformation function with our pre-optimized model, seamlessly integrating it into the remaining steps outlined in their methodology. In particular, the checkpoints provided are exclusively available for the Lego and Trex datasets.

7.2. Injective Baselines (Section 4.2 in the main paper)

- **i-ResNet [1]** We adopt the implementation provided at https://github.com/stevenygd/NFGP. The chosen hyperparameters align with the configuration specified in the deformation settings, available at this link. Specifically, we configure the model with six layers, a positional encoding frequency of 5, and a latent dimension of 256. During the learning experiment, we condition the generation by appending the conditional feature vector to the positional encoding.

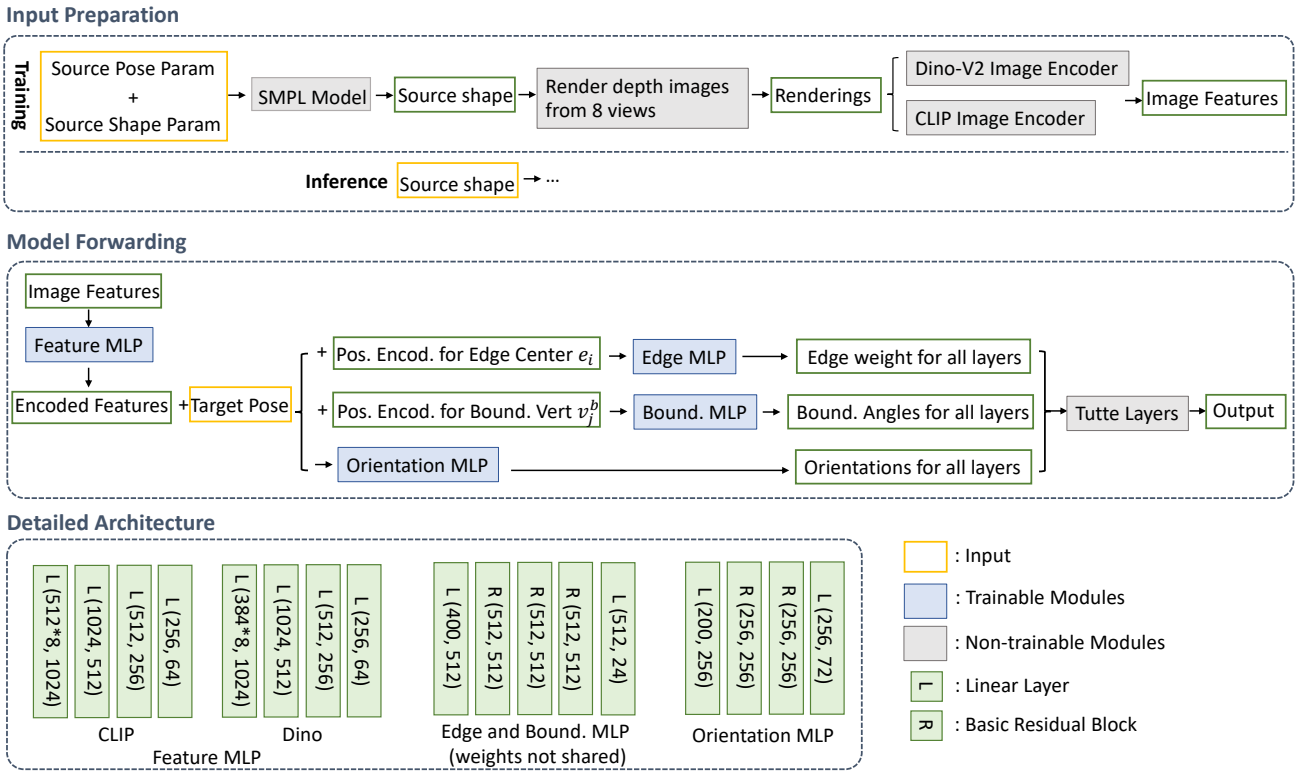

Figure 6. Detailed model architecture and data preparation for the learning experiment (Section 4.2 in the main paper).

- **RealNVP [3]** We adopt the implementation available at https://github.com/ikostrikov/pytorch-flows and perform a thorough hyperparameter search to optimize performance. Regarding the mask selection, given our three-dimensional input and output, we employ an alternating approach across the three dimensions. This involves masking out one dimension at a time during each iteration to facilitate the RealNVP layer forward pass. Based on the best performance observed and considering the compatibility with our model size, we set the number of layers to 6 and the hidden dimension to 32. In the learning phase, we seamlessly integrate the conditional input, following the approach outlined in their code.
- **NeuralODE [2]** We align with the implementations available at https://github.com/hjwdzh/MeshODE and https://github.com/maxjiang93/ShapeFlow, both of which utilize NeuralODE for mesh deformation. To accommodate the size of our model, we employ four Linear layers with a latent dimension of 120. During the learning experiment, we adhere to the ShapeFlow [4] configuration, incorporating a conditional vector for every sample in the ODE function. We set the absolute and relative tolerance in odeint at 10^{-4}. In the elastic deformation experiment, we leverage the pytorch gradient function to invoke their built-in Jacobian computation in the ODE solver. Subsequently, using the same handle constraints, we perform deformation. To achieve optimal results and match our model size (24 layers with mesh resolution 25×25), we set the latent dimension to 200 and employ the Adam optimizer for 12,000 steps with a learning rate of 10^{-3}.

References

[1] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International conference on machine learning*, pages 573–582. PMLR, 2019. 5

[2] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018. 6

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 6

[4] Chiyu Jiang, Jingwei Huang, Andrea Tagliasacchi, and Leonidas J Guibas. Shapeflow: Learnable deformation flows among 3d shapes. *Advances in Neural Information Processing Systems*, 33:9745–9757, 2020. 6

[5] Ruofan Liang, Jiahao Zhang, Haoda Li, Chen Yang, Yushi Guan, and Nandita Vijaykumar. Spidr: Sdf-based neural

point fields for illumination and deformation. *arXiv preprint arXiv:2210.08398*, 2022. 5

[6] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision*, pages 5442–5451, 2019. 3, 4

[7] 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. *Communications of the ACM*, 65(1):99–106, 2021. 2

[8] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023. 5

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 5

[10] Tianhan Xu and Tatsuya Harada. Deforming radiance fields with cages. In *ECCV*, 2022. 5

[11] Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. Nerf-editing: geometry editing of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18353–18364, 2022. 5