4Deform: Neural Surface Deformation for Robust Shape Interpolation

Supplementary Material

https://4deform.github.io/

S1. Losses Derivations

Distortion Loss. If one breaks down the rate of deformation tensor in Eq. (9), $\mathbf{D}$ it is the symmetric part of the velocity gradient $\nabla\mathcal{V}$ plus its transpose. It is called the rate of deformation tensor which gives the rate of stretching of elements. Since $\mathcal{V}:\mathbb{R}^3\to\mathbb{R}^3$, $\mathbf{D}$ is a 3×3 matrix, it is also related to stress tensor in continuum mechanics. We adopt the second invariants of the deviatoric stress tensor [30]

$$\begin{aligned} J_2 &= \frac{1}{3}\operatorname{Tr}(\mathbf{D})^2 - \frac{1}{2}\big(\operatorname{Tr}(\mathbf{D})^2 - \operatorname{Tr}(\mathbf{D}\cdot\mathbf{D})\big) \\ &= \frac{1}{6}\operatorname{Tr}(\mathbf{D}\cdot\mathbf{D}) - \frac{1}{2}\operatorname{Tr}(\mathbf{D})^2 \,. \end{aligned} \tag{s.16}$$

The second invariant equal to zero implies that there is no shape-changing (distortional) component in the deformation or stress. In this case, all principal stresses or strains are equal, leading to a purely hydrostatic (isotropic) stress or strain state [16, 53].

Stretching Loss In fact, the term is related to the (right) Cauchy strain tensor and also related to distortion loss. As in Eq. (12), the deformation term $\mathbf{F}^\top\mathbf{F} := \mathbf{C}$ is called the Cauchy strain tensor [31]. The term $\mathbf{F}^\top\mathbf{F} - \mathbf{I} := \mathbf{E}$ is called Green-Lagrange strain tensor and used to evaluate how much a given displacement differs locally from a rigid body displacement. Write it in gradient tensor, *i.e.*, $\nabla\mathcal{V}$, we have

$$\mathbf{E} = \frac{1}{2}\big((\nabla\mathcal{V})^\top + \nabla\mathcal{V} + (\nabla\mathcal{V})^\top(\nabla\mathcal{V})\big) \,. \tag{s.17}$$

Therefore, Eq. (14) can be seen as projecting the rigid displacement to the tangent space of point $\mathbf{x}$. Even from a different perspective, our formulation coincides with the stretching loss in NFGP [55]. Different is we have an explicit formulation of deformation operator $\mathbf{F}$.

Normal Deformation Even though our method does not require an oriented point cloud as input. If normal information is available from the given point clouds, one could utilize the natural property of implicit representation to add normal deformation constraints. We follow the projection from our stretching loss, for any vector $\mathbf{t}_1$ and $\mathbf{t}_2$ in the tangent space of point $\mathbf{x}$ with normal $\mathbf{n}$, then we have $\mathbf{n}(\mathbf{x})\cdot\mathbf{t}_1 = 0$ and $\mathbf{n}(\mathbf{x})\cdot\mathbf{t}_2 = 0$. The deformation transform $\mathbf{t}_1$ to $\mathbf{t}_1' = \mathbf{F}\mathbf{t}$, and $\mathbf{t}_2' = \mathbf{F}\mathbf{t}_2$ the $\mathbf{F}$ is the same as in Eq. (12). Therefore, $\mathbf{t}_1'$ and $\mathbf{t}_2'$ lie in the tangent space of the deformed surface point $\mathbf{x}'$, thus, the normal in $\mathbf{x}'$, denoted as $\mathbf{n}'$ should be perpendicular to $\mathbf{t}_1'$ and $\mathbf{t}_2'$, that is,

$$\mathbf{n}'\cdot\mathbf{t}_1' = 0, \;\; \mathbf{n}'\cdot\mathbf{t}_2' = 0 \,. \tag{s.18}$$

Then we have

$$\mathbf{n}'\cdot\mathbf{F}\mathbf{t}_1 = 0, \;\; \mathbf{n}'\cdot\mathbf{F}\mathbf{t}_2 = 0 \,. \tag{s.19}$$

This implies

$$\mathbf{F}^\top\mathbf{n}' = \lambda\mathbf{n} \,. \tag{s.20}$$

We normalized it and get the Normal Deformation Loss as

$$\mathcal{L}_n = \int_\Omega \left\| \mathbf{n}_t - \frac{\mathbf{F}^\top\mathbf{n}_{t+1}}{\|\mathbf{F}^\top\mathbf{n}_{t+1}\|} \right\|_{l^2} \mathrm{d}\mathbf{x} \,. \tag{s.21}$$

S2. Training Details

In this section, we summarize the training efficiency of our method and the comparison methods. We plot the average training time (per pair) in Fig. S.7. LipMLP [32] trains the fastest as they do not have discrete time steps during training. Our method trains as fast as [45] per pair. However, our method can directly train on temporal sequences without manually switching training pairs. In addition to that, NFGP [55] requires more than 75 hours to train a 5-step interpolation and LIMP [13] trains only on meshes with $2,500$ vertices and takes longer than our methods.

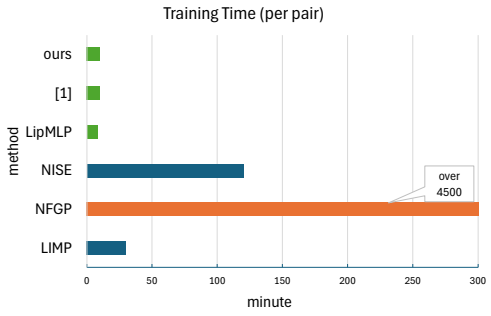

Figure S.7. **Training time visualization.** We plot the rough training time with comparison methods to show the efficiency of our methods.

LIMP Training Protocol. LIMP [13] learns a latent space of meshes and constructs an interpolation constrained under geometric properties. This method supports both isometric and non-isometric deformations. However, the input meshes are required to be in *pointwise correspondence* and *labeled based on stylistic classes*. Additionally, a pre-processing step is needed on the input meshes to reduce the number of vertices to 2500 and this step is done using iterative edge collapse [24]. The model supports sequence training and

training for 20,000 epochs takes about 30-40 minutes for pair training.

NISE Training Protocol. NISE [38] is a method that learns both isometric and non-isometric deformations between two input meshes. It relies on a pair of pre-trained SDF networks to linearly interpolate neural implicit surfaces, which form the foundation for modeling the deformation. In the paper of NISE [38], the author mentioned that the method can interpolate along a pre-defined linear path as well. However, this path needs to be defined per point and it can only interpolate linearly according to the Euclidean coordinates of the points. The method can only be trained on mesh pairs, and training each pair, including pre-training the SDF network to fit the input, requires approximately 4 hours for 20,000 epochs. Excluding the pre-training time is approximately 2 hours per pair.

NFGP Training Protocol. Training NFGP [55] requires first training an SDF network that fits the implicit field on the input shapes, which takes about 2 hours for 100 epochs. After that, a set of points is defined per deformation step as handles, along with the necessary rotation and translation parameters to transform these handle points into target points. To be able to use NFGP [55] as a time-dependent interpolation network that generates t intermediate shapes, one needs to train the network t times and decide how the gradual deformation at each time step should appear. Therefore, the process of defining handle points requires a thorough understanding of how to set rotations and translations to obtain physically plausible interpolation. Moreover, visualization is essential for selecting handles and targets from the vertices of the meshes reconstructed from their SDF network. The training for 500 epochs per deformation time step takes 8 hours. Thus, 50 hours — including the training for the implicit network — are required for deformation with 5 time steps.

S3. Visualizations of Quantitative Evaluated Sequences

In this section, we show the visual results of Tab. 2 on **4D-Dress** [54] in Fig. S.8. And the visual results of Tab. 3 on **SMAL** [56] dataset in Fig. S.9, to show the deformation of non-human objects.

S4. More Visualization

We show more visualization results of our method on real-world datasets. We show more sequences from **BeHave** [4] in Fig. S.10 and Fig. S.11. We also show more visualization of high-resolution real-world mesh interpolation on **4D-Dress** [54] in Fig. S.12.

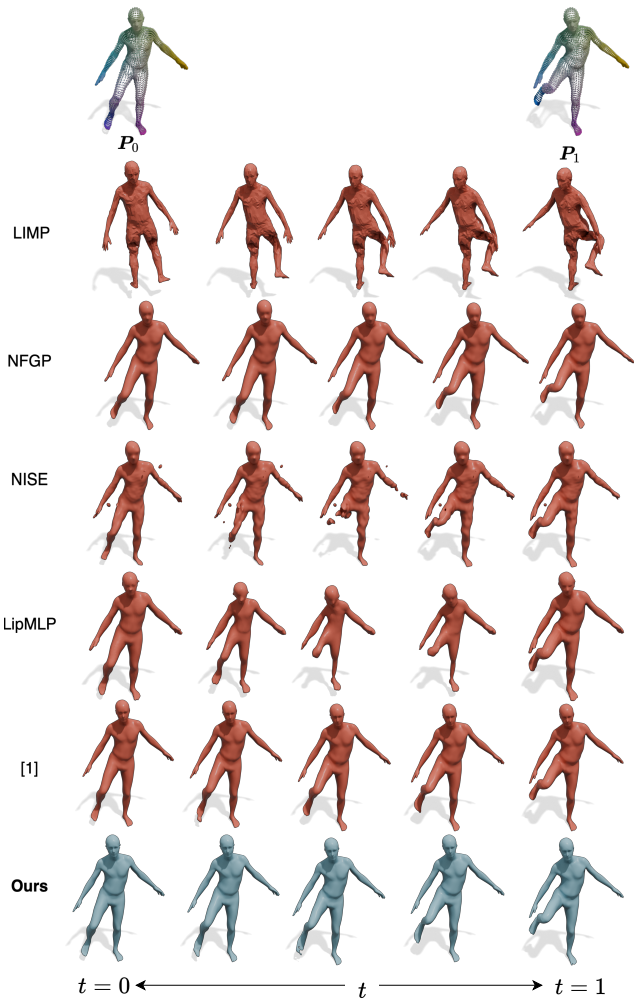

Figure S.8. **Visual results on human isometric deformation.** We show the visualization of our interpolated meshes on **4D-Dress** [54]. LIMP [13] can recover reasonable movement, however, it turns the leg in the wrong direction.

S5. Upsampling Video

We use our method to upsample sequences in **BeHave** [4] to 30FPS and render video for it. Please visit our project page https://4deform.github.io/.

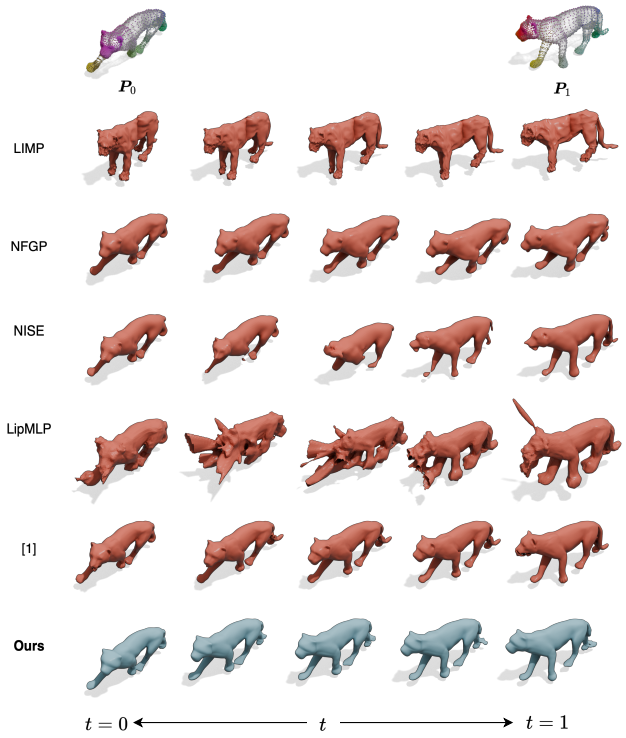

Figure S.9. **Visual results on non-human object deformation.** We show the visualization results for an animal data in **SMAL** [56]. LIMP [13] can only handle $2,500$ vertices, thus the interpolated mesh is low-quality.

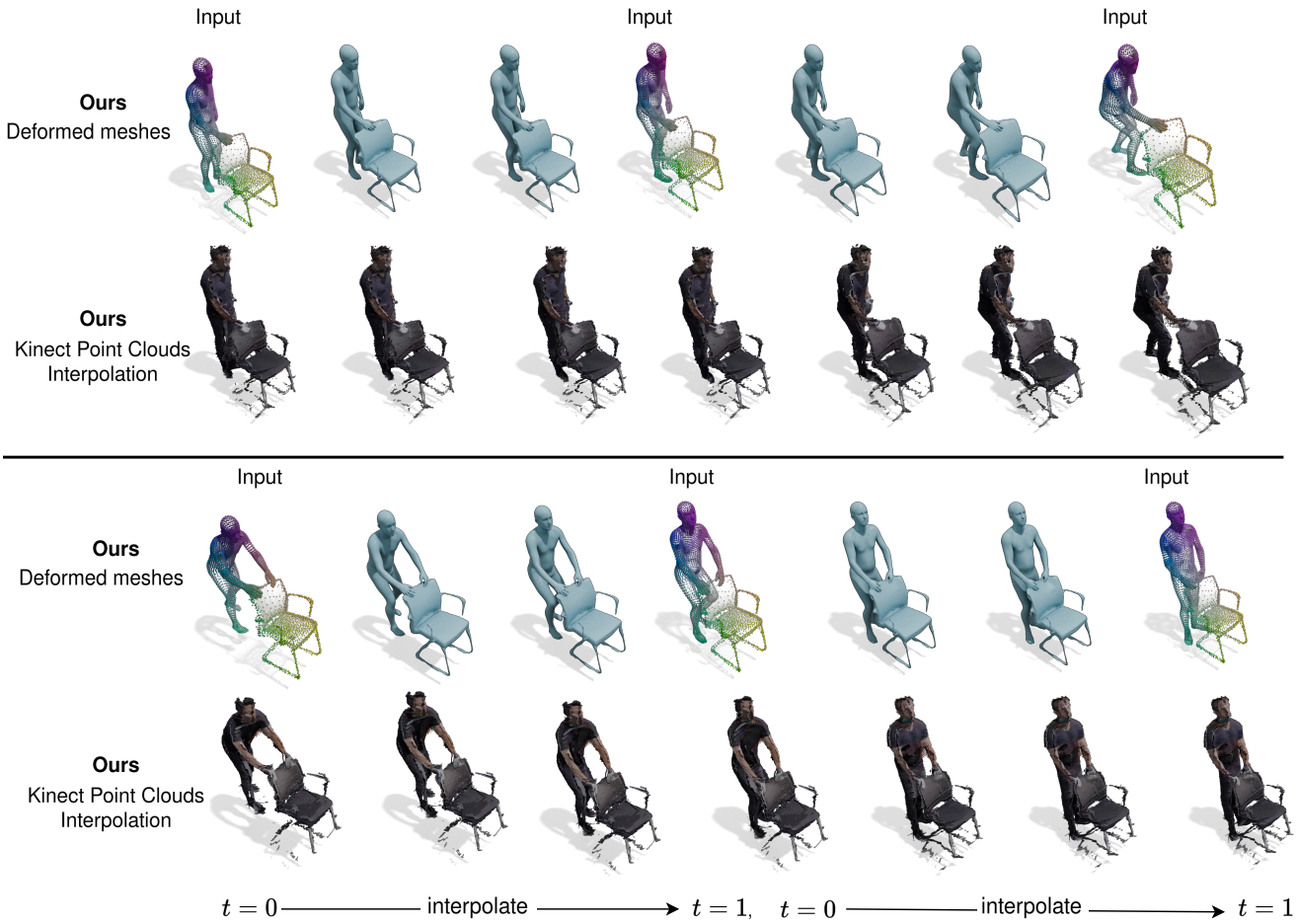

Figure S.10. **Upsampling on real-world data.** We show examples of the **BeHave** [4] sequence. Starting from a sparse set of keyframes (1fps, colored point clouds), our method lets us interpolate the registration (first row), as well as the real Kinect point clouds (second row) between keyframes at an arbitrary continuous resolution.

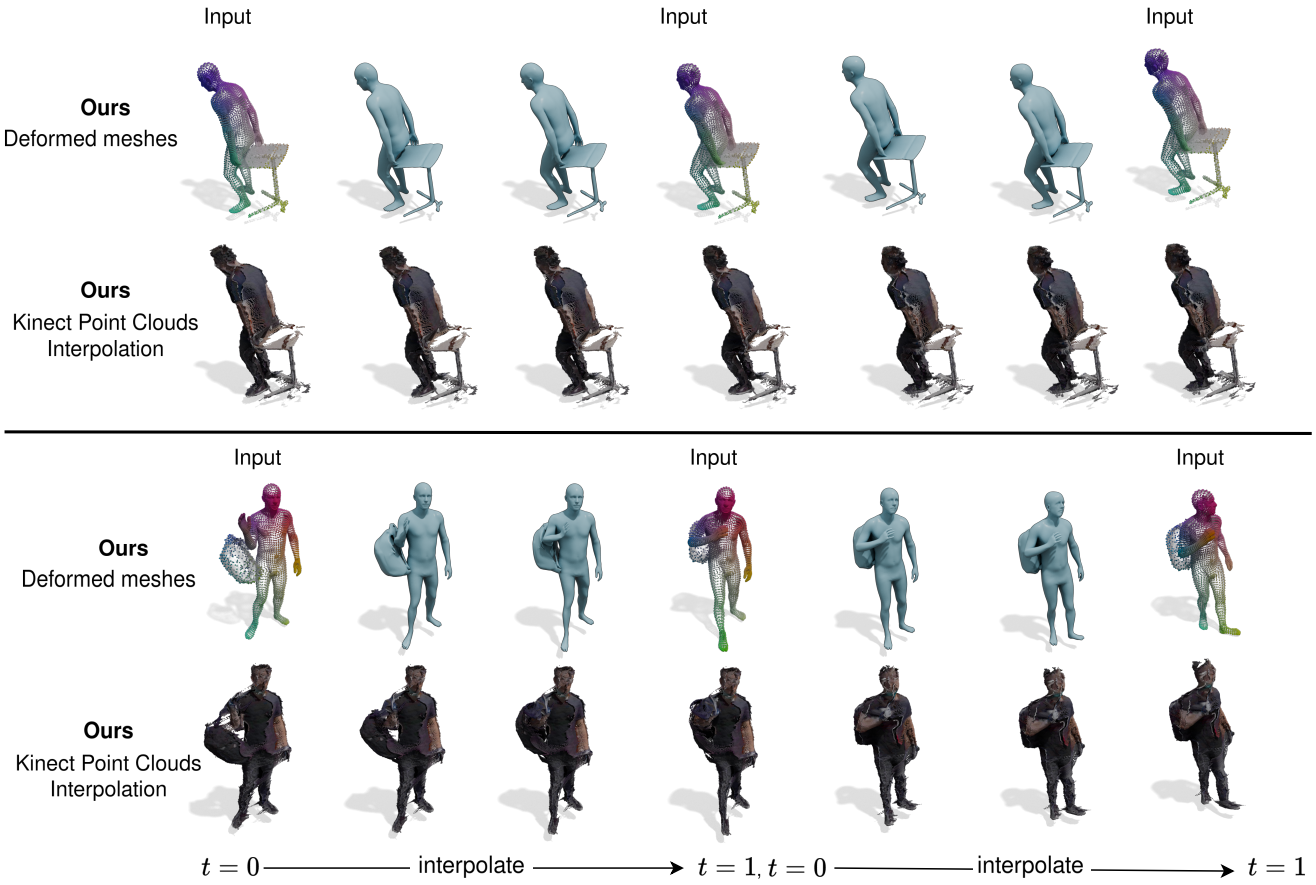

Figure S.11. **Upsampling on real-world data.** We show examples of the **BeHave** [4] sequence. Starting from a sparse set of keyframes (1fps, colored point clouds), our method lets us interpolate the registration (first row), as well as the real Kinect point clouds (second row) between keyframes at an arbitrary continuous resolution.

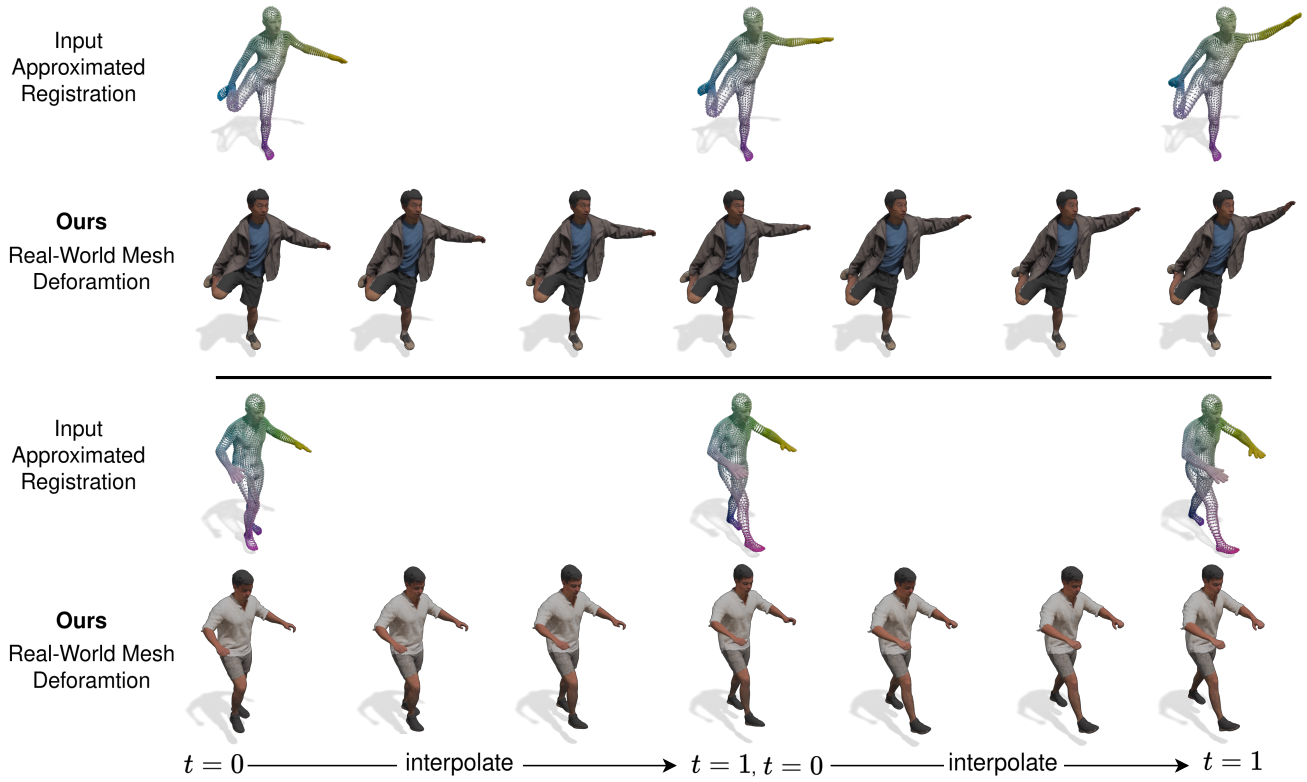

Figure S.12. **Deformation on real-world mesh.** We examples of the **4D-Dress** [54] sequence. Starting from a sparse set of approximated registration of SMPL model [33], our method lets us interpolate the real-world, high-resolution meshes (second row, around $40,000$ vertices) between keyframes.