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

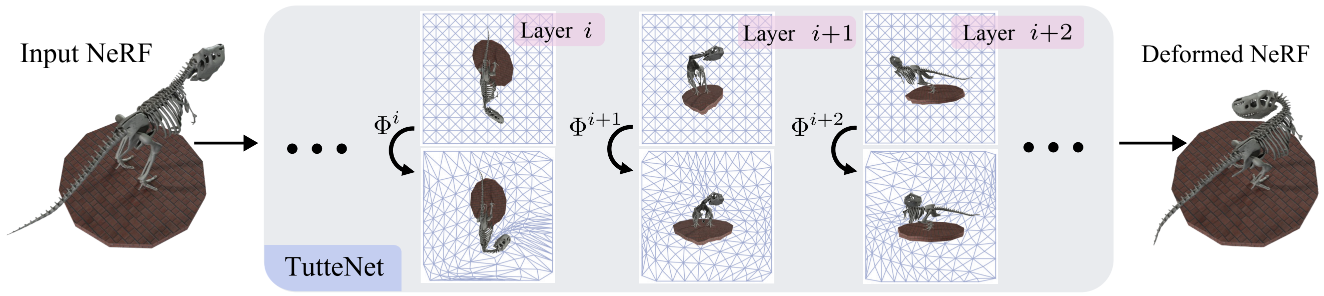

Figure 1. Elastically deforming a NeRF [55] based on user-designated positioning of the head (turned) tail (bent) and body (lowered), and optimizing the degrees of freedom of TutteNet to minimize the elastic energy of the deformation. TutteNet guarantees an injective (1-to-1) deformation of the ambient 3D space surrounding the T-Rex, ensuring the NeRF is rendered correctly without artifacts by enabling "pulling back" points and view directions from deformed space. Each layer within TutteNet views the T-Rex over a different 2D plane (in this case, alternating between the 3 main axes in a tri-plane manner). In each layer, the T-Rex is enveloped with a regular 2D mesh of the unit square (top row). The 2D mesh is deformed using the layer's optimizeable parameters which define a Tutte's embedding [19, 80] (bottom row). This defines an injective 2D piecewise-linear map, which can be applied to the 3D T-Rex, without modifying the normal direction to the plane, resulting in an injective 3D deformation Φ^i. Composition of these layers yields the final expressive 3D injective deformation.

Abstract

This work proposes a novel representation of injective deformations of 3D space, which overcomes existing limitations of injective methods, namely inaccuracy, lack of robustness, and incompatibility with general learning and optimization frameworks. Our core idea is to reduce the problem to a "deep" composition of multiple 2D mesh-based piecewise-linear maps. Namely, we build differentiable layers that produce mesh deformations through Tutte's embedding (guaranteed to be injective in 2D), and compose these layers over different planes to create complex 3D injective deformations of the 3D volume. We show our method provides the ability to efficiently and accurately optimize and learn complex deformations, outperforming other injective approaches. As a main application, we produce complex and artifact-free NeRF and SDF deformations.

1. Introduction

This work concerns computation and learning of 3D deformations. As the most immediate mode of manipulation and interaction with 3D shapes, deformations play a crucial role in various fields such as vision [81], medical imaging [52], 3D registration [13], and graphics [32]. In many real-world applications, it is crucial that the deformation does not create any self-overlaps, i.e., is *injective* (a 1-to-1 mapping). An example that is a key motivation for this work, is deformation of Neural Radiance Fields (NeRFs) [55]: when deforming NeRFs, lack of injectivity can easily cause severe rendering artifacts due to intersecting "deformed" rays during the ray tracing process, see Figure 3.

Unfortunately, current approaches do not provide an injective deformation method that is both sufficiently expressive and robust, as well as lending itself to practical optimization and learning:

– On one hand, within geometry processing and graphics, 3D deformations are heavily-researched through triangular/tetrahedral *mesh* deformations, i.e., modifying the position of each vertex of the mesh. Mesh deformations provide a finite set of meaningful geometric degrees of freedom leading to stable, quick, and straightforward computation, as well as access to geometric quantities such as the deformation gradients (*Jacobians*), critical in most mesh-deformation approaches [87]. However, mesh deformations cannot be *learned* while ensuring injectivity, nor are di-

	Learnable	Analytical inverse	Fast Det. Jacobian	Fast full Jacobian	Robustness
i-ResNet [6]	✓	✗	✗	✗	✗
RealNVP [16]	✓	✓	✓	✗	✗
NeuralODE [10]	✓	✓	✓	✓	✗
Ours	✓	✓	✓	✓	✓

Table 1. **Properties of injective deformation methods.** Beyond superiority in accuracy and efficiency, our method holds unique properties compared to other injective methods, see Sec. 3.3.

rectly applicable when an explicit triangulation of the shape is not given.

– On the other hand, the ML community has heavily-researched *functional* representations of injective maps via neural networks, such as normalizing flows [16] and solutions to ODEs [10]. These methods were mainly designed for high-dimensional mappings, e.g., for generative tasks [24], but have recently been successfully adapted to injective 3D deformations [30, 36, 81]. The functional representation which they provide is not geometric, but rather embedded abstractly within the network's weights, resulting in many cases in slow, cumbersome and unstable computation, possibly leading to a less-accurate prediction, or practical intangibility of critical geometric quantities such as the aforementioned deformation Jacobians.

In this work, we aim to resolve these issues and gain the benefits of both worlds: we propose a novel computational representation for 3D injective deformations, which *combines* the geometric representation of mesh-based deformations with the standard deep-learning approach of functional composition.

Our core observation is that sequentially composing multiple 2D mesh deformations, over different 3D planes, achieves two critical goals simultaneously: 1) similarly to other "deep" representations, compositionality leads to an expressive representation, able to capture complex 3D deformations accurately, while simultaneously using mesh deformations for its layers, providing virtues such as numerical stability and accuracy; 2) while injectivity is not directly tractable for 3D mesh deformations, it is in 2D. Hence, by reducing each deformation "layer" to 2D, we can leverage recent observations for 2D injective mesh deformations [1], which show how 2D Tutte embeddings [80] can yield a differentiable parameterization of *all* injective mesh deformations into a convex domain, enabling unconstrained learning and optimization. Composing multiple 2D injective deformations from different viewpoint defines a family of injective volumetric 3D deformations.

We show through experiments that our method can be used both for accurately *learning* injective deformations (e.g., learning to repose a given human model to arbitrary poses), as well as *optimizing* volumetric deformations in tasks in which injectivity is critical, e.g., elastically deforming a NeRF with respect to user interactions. Through comparisons, we show that our method significantly outperforms other injective deformation techniques. Furthermore, through comparisons to previous (non-injective) NeRF-deformation techniques, we both exhibit the importance of injectivity, as well as show that in many cases our method is still more expressive than those competing techniques, in spite of them facing a less-constrained problem.

2. Related Work

Invertible neural networks. Injective maps are critical for generative modeling, in order to map between distributions. This has fueled the development of families of invertible neural functions. Normalizing Flows [15, 16, 23, 40, 60, 61, 70, 71, 79] are highly prominent, with RealNVP [16] applied to 3D volume deformations, e.g., long-range optical flow [81] or for learning 3D deformations [43, 65]. They work by defining an injective transformation of a subset of the spatial coordinates at each block, which is ideal for a high-dimensional settings but loses expressivity when the subsets must lie in 1D/2D. Continuous flows through Neural solutions to Ordinary Differential Equation (NeuralODE) [10] have also been successfully applied to 3D deformations, e.g., for shape autoencoding [27], dynamic mesh reconstruction [59], for point cloud generation [90] and asset deformation [30, 36]. Finally, i-ResNet [6] achieves invertibility of a ResNet by enforcing Lipschitz bounds, with [91] using this formulation for 3D deformations. We empirically evaluate and compare to these methods in Section 4.2. In Table 1, we compare desirable properties for 3D invertible deformations. TutteNet can be considered as a variant of a normalizing flow, however is an explicit representation through composition of geometrically-expressive 2D mesh deformations, designed specifically for geometric 3D deformations, without the inclusion of a neural network in the deformation process.

Injective deformations of meshes in geometry processing. 3D injective deformations have been extensively researched in geometry processing, mainly for piecewise linear maps on triangle meshes. Local and global injectivity can be achieved through energies that encourage or enforce it [17, 22, 69, 72, 73] but cannot guarantee injectivity when additional objectives are added, or in learning settings. Injectivity can be achieved via convex constraints [2, 41], tailor-made solvers [44], or discrete modifications of the triangulation to preserve or recover injectivity [20, 38, 57] that cannot be applied in a learning setting or without a well-defined triangle mesh. Other methods use non-discrete representations [7] that cannot be predicted or optimized. We use layers of 2D injective deformations define via Tutte's embedding [19, 80], and control each layer by the method proposed in [1], of optimizing the mesh Laplacian and boundary conditions.

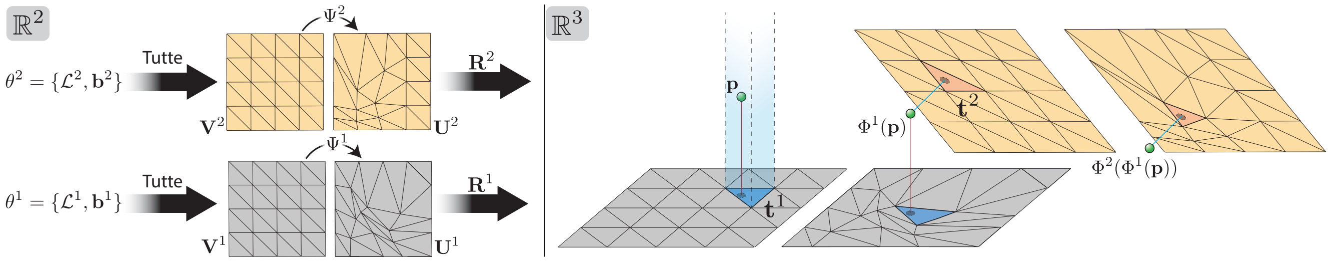

Figure 2. **Our representation of injective 3D deformations**, visualized for the process of mapping a given point $\mathbf{p}$ inside the volume, for two-layer TutteNet, $i \in \{1, 2\}$. Left: the (learnable) parameters θ^i, consisting of the mesh-Laplacian L^i and the boundary conditions $\mathbf{b}^i$, define a 2D deformation Ψ^i of the square mesh $\mathbf{M}$, through Tutte's embedding [80]. Ψ^i is embedded in 3D to the local coordinates $\mathbf{R}^i$ to define a 3D deformation, Φ^i. Right: given a point $\mathbf{p}$, it is projected to the local coordinates of Φ^1, landing on triangle $\mathbf{t}^1$. Φ^1 defines an affine map over the infinite prism of $\mathbf{t}^1$ (represented in blue with dotted lines), mapping $\mathbf{p}$ to $\Phi^1(\mathbf{p})$. The resulting $\Phi^1(\mathbf{p})$ is projected onto the local coordinates of Ψ^2, landing on triangle $\mathbf{t}^2$, from which it is mapped by the affine map Φ^2 defined over the infinite prism of $\mathbf{t}^2$.

NeRF Deformation. Several works use 3D volume deformations defined by MLPs that input/output spatial coordinates as a means to achieve various applications for NeRF, e.g., dynamic scenes [21, 63, 64, 78, 83], stylization [85], and controlling trajectories [50]. Other works focus on providing NeRF deformation tools for end users, usually via a proxy geometry that controls the volume deformation, e.g., using a mesh scaffold and transfers a user-defined mesh deformation to a volume deformation Nerf-Editing [94], or through enveloping cages [34, 46, 68, 86]. Others *bake* the NeRFs into a more deformation-friendly representation, such as meshes [12] or point clouds [9, 47]. None of these approaches is injective nor can be applied to a NeRF without a preprocessing step of fitting the proxy to the NeRF, making learning and optimization less straightforward. We compare with [47, 86, 94] and demonstrate the importance of injectivity. Many other learning techniques exist for deforming shapes that are not NeRFs, e.g., by predicting per-points offsets via coordinate-based MLPs [11, 14, 18, 25, 26, 54, 62, 76, 92], Jacobians [4], rigs [28, 45, 48, 88, 89, 93], or point handles [33, 49]. To avoid self-intersection, they often rely on regularizing the Jacobian [5, 31, 74] or the Laplacian [39].

3. Method

We now describe our expressive representation of 3D injective functions through composition of 2D injective mesh deformations, see Figure 2 for visualization of the full pipeline. We begin by setting some necessary preliminaries regarding piecewise-linear maps and Tutte embeddings (Section 3.1), then describe our representation (3.2) and conclude with analyzing its core properties (3.3).

3.1. Preliminaries

Piecewise-linear maps. We assume to have a 2D triangular mesh $\mathbf{M}$ with triangles $\mathbf{T}$ and vertices $\mathbf{V}$ embedded in $\mathbb{R}^2$. $\mathbf{M}$ can be any disk-topology mesh - in all experiments, we use the unit square, $\Omega = [-1,1]^2$, and triangulate it with a regular triangulation of same-size isosceles triangles. We consider 2D piecewise-linear maps $\Psi : \mathbf{M} \to \mathbb{R}^2$ of this mesh, meaning the map is affine over each triangle $\mathbf{t} \in \mathbf{T}$,

$$\Psi|_{\mathbf{t}}(\mathbf{p}) \equiv A_{\mathbf{t}}\mathbf{p} + \delta_{\mathbf{t}}, \tag{1}$$

for some $A_{\mathbf{t}} \in \mathbb{R}^{2\times 2}, \delta_{\mathbf{t}} \in \mathbb{R}^2$. (Note that this map can map *any* point $\mathbf{p} \in \Omega$ and not just the vertices of a mesh). The gradient of a map at point $\mathbf{p}$, denoted $D_{\mathbf{p}}\Psi$, is called the *Jacobian*. For piecewise-linear maps, the Jacobian is constant over each triangle $\mathbf{t}$ and is exactly the linear transformation $D_{\mathbf{t}}\Psi = A_{\mathbf{t}}$. To define a continuous piecewise linear map Ψ, it suffices to define deformed vertex positions $\mathbf{U} = \{\mathbf{u}_i\}_{i=0}^{|\mathbf{V}|}$, assigning position $\mathbf{u}_i \in \mathbb{R}^2$ to each vertex $\mathbf{v}_i \in \mathbf{V}$, and define the map via $\Psi(\mathbf{v}_i) = \mathbf{u}_i$. Given $\mathbf{u}_i$ the Jacobian can be obtained by solving the linear equation

$$A_{\mathbf{t}}\mathbf{v}_i + \delta_{\mathbf{t}} = \mathbf{u}_i, \ i \in \mathbf{t} \tag{2}$$

w.r.t $A_{\mathbf{t}}$ - the resulting small 6×6 linear equations can be inverted *once* at initialization of training/optimization.

Tutte's embedding is a method for computing injective 2D mesh mappings [19, 80], for meshes with disk topology (i.e., having one loop of boundary vertices). Given a mesh-Laplacian matrix L, defined by assigning some positive scalar $L_{ij} \in \mathbb{R}^+$ to each edge (i, j) of the mesh $\mathbf{M}$, along with a sequence of 2D points $\mathbf{b}_1, ..., \mathbf{b}_k \in \mathbb{R}^2$ that lie on a convex polygon, Tutte's embedding computes deformed vertex positions $\mathbf{U} = \{\mathbf{u}_i\}_{i=0}^{|\mathbf{V}|}$ by solving the sparse linear system defined via:

$$\begin{gathered}\sum_j L_{ij}(\mathbf{u}_j - \mathbf{u}_i) = 0 \text{ for each interior vertex } \mathbf{v}_i \\ \mathbf{u}_i = \mathbf{b}_i \text{ for each boundary vertex } \mathbf{v}_i.\end{gathered} \tag{3}$$

While Tutte's embedding is guaranteed to be injective in 2D, it is unfortunately well-known to not hold in 3D (see, e.g., [7]) hence extensions to 3D do not exist.

3.2. 3D injections through 2D mesh deformations

We wish to devise an optimizable family of injective deformations f_θ of 3D volumetric space, which leverages mesh deformations. Since no simple parameterization of injective 3D mesh deformations is known, the key idea of TutteNet is instead to define the 3D deformation through the composition of 2D injective mesh deformations (see Figure 2).

Prismatic layers. Postponing the discussion on how to compute 2D injective mesh deformations, assume for now that we have one such injective 2D mesh deformation, $\Psi : \mathbf{M} \leftrightarrow \mathbb{R}^2$. Our basic building block constituting one "layer" in our architecture, is a type of map we dub a *prismatic* map, meaning it is a lifting of the 2D mesh deformation Ψ into a 3D piecewise-linear map that operates over some plane and preserves the normal direction. Specifically, let $\mathbf{p} \in \mathbb{R}^3$ be a 3D point, and define

$$\tilde{\Psi}\left(\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z\right) \triangleq \Psi\left(\mathbf{p}_x, \mathbf{p}_y\right), \mathbf{p}_z, \tag{4}$$

i.e., $\tilde{\Psi}$ acts on the xy coordinates of each point and preserves the z coordinate. By rotating the coordinate system by a 3D rotation $\mathbf{R} \in SO\left(3\right)$ we can apply the deformation on any desired plane instead of on the main axes:

$$\Phi\left(\mathbf{p}\right) \triangleq \mathbf{R}\tilde{\Psi}\left(\mathbf{R}^T\mathbf{p}\right). \tag{5}$$

The process of mapping through Φ is summarized in Algorithm 1. Finally, composing multiple Φ^i (defined over different planes $\mathbf{R}^i$ and with different Ψ^i) yields the final 3D deformation f,

$$f = \Phi^k \circ \Phi^{k-1} ... \circ \Phi^0. \tag{6}$$

Lastly, we need to parameterize the injective 2D mesh deformation space $\Psi : \mathbf{M} \rightarrow \mathbb{R}^2$. Our reduction of the 3D injective problem to 2D sub-problems enables us to take advantage of recent advances in 2D injective mesh deformations [1]. Originally designed for generative 2D techniques, [1] provides a parameterization of all 2D injective deformations of a given mesh, via Tutte's embedding [3, 19, 80].

Following [1], we use the entries of the Laplacian L_{ij} and the boundary conditions $\mathbf{b}_i$ (defined in Section 3.1) as optimizable/learnable parameters, which in turn produce the deformed vertices $\mathbf{U}$ of the 2D mesh, through Tutte's embedding, by solving the linear system of Eq. (3) w.r.t. $L, \mathbf{b}$. Following the proof in [1], this covers *all* possible piecewise-linear maps of $\mathbf{M}$ into the convex polygon $\mathbf{b}$.

To ensure that $\{\mathbf{b}_i\}$ form a convex polygon, we parameterize them via positive angle increments $\alpha_i > 0, \sum \alpha_i = 2\pi$, and define the angle $\beta_j = \sum_{i=1}^{j} \alpha_i$. $\mathbf{b}_i$ is then the intersection of the line at angle β_i with the unit square. Hence, $\mathbf{b}$ is a function of α. To keep all parameters positive and bounded, we add a sigmoid and scaling function on those parameters before feeding them to the Tutte layer, $x' = \text{sigmoid}\left(x\right)\left(1 - 2\epsilon\right) + \epsilon$, with $\epsilon = 0.2$ for L and $\epsilon = 0.1$ for $\mathbf{b}$.

The parameters of a prismatic map Φ^i are thus $\theta^i = \left(L^i, \mathbf{b}^i\right)$, and the local coordinate system $\left\{\mathbf{R}^i\right\}$. The final 3D injective piecewise-linear map f_θ is thus parameterized by $\theta = \left\{\theta^i, \mathbf{R}^i\right\}_{i=0}^{n}$. We summarize the computation of f_θ in Algorithm 2. θ (and possibly $\left\{\mathbf{R}^i\right\}$) can be directly optimized with respect to an objective (Section 4.1), or otherwise predicted by a neural network (Section 4.2) - in both cases, we run Algorithm 2 at each iteration, compute the loss and back-propagate gradients back to θ, as all steps in the algorithm are differentiable.

Layer regularization. To better-condition our architecture, we can regularize its layers, ensuring each layer's Jacobian's distortion is low. We define an elastic energy measuring the Cauchy-Green strain tensor's deviation from the identity matrix I at a given point $\mathbf{p}$ for a given map g,

$$E_g\left(\mathbf{p}\right) = \left\|D_{\mathbf{p}}g^T D_{\mathbf{p}}g - I\right\|^2, \tag{7}$$

where $D_{\mathbf{p}}g$ is the map's Jacobian (defined in Section 3.1).

As opposed to functional representations [16, 30], the layer's Jacobians are enumerable (one per triangle), enabling us to compute the *exact* integral of E (as opposed to an approximation via sampling), by summing the energy over all Jacobians of the layer:

$$\mathcal{L}_{\text{Reg}} \triangleq \int_{\mathbf{p}\in\Omega} E_{\Psi^i}\left(\mathbf{p}\right) \equiv \sum_{\mathbf{t}\in\mathbf{T}} |\mathbf{t}|\, E_{\Psi^i}\left(\mathbf{t}\right), \tag{8}$$

where $|\mathbf{t}|$ is the area of triangle $\mathbf{t}$. This technique could be extended in the future to, e.g., provide absolute bounds on the distortion of each layer [41].

Algorithm 1: Prismatic map $\Phi\left(\mathbf{p}\right)$

```
1 Rotate point to local coordinate frame: $\mathbf{q} = \mathbf{R}^T\mathbf{p}$
2 Keep only the $xy$ coordinates: $\tilde{\mathbf{q}} = \left(\mathbf{q}_x, \mathbf{q}_y\right)$
3 Find triangle $\mathbf{t}$ that contains $\tilde{\mathbf{q}}$
4 Map through $\Psi$: $\tilde{\mathbf{r}} = A_{\mathbf{t}}\tilde{\mathbf{q}} + \delta_{\mathbf{t}}$
5 Concatenate the $z$ coordinate back: $\mathbf{r} = \left(\tilde{\mathbf{r}}_x, \tilde{\mathbf{r}}_y, \mathbf{q}_z\right)$
6 Rotate back to global coordinates: $\Phi\left(\mathbf{p}\right) = \mathbf{R}\mathbf{r}$
```

Algorithm 2: Computation of f_θ from θ

```
1 for each deformation layer $i$ do
2     Compute $\mathbf{U}^i$ via Tutte's embedding, by solving
        the linear system (3) defined by $L^i, \mathbf{b}^i$
3     Compute $\Psi^i$ from $\mathbf{U}^i$ using Eq. (1), and store
        $A^i_{\mathbf{t}}, \delta^i_{\mathbf{t}}$ for each triangle $\mathbf{t}$
4     Define $\Phi^i$ via $\Psi^i$ and $\mathbf{R}^i$
5 end
6 Define $f_\theta$ using all $\left\{\Phi^i\right\}_{i=0}^{n}$ via Eq. (6)
```

3.3. Discussion: properties of the deformation f_θ

Table 1 compares different injective approaches for 3D deformations. Constructing f through composition of mesh deformations provides it with the following desirable properties:

• **Learnable and optimizable.** The representation of the deformation f_θ through the unconstrained parameters θ in turn allows simple gradient-based learning/optimization.

• **Injective, with an immediate, explicit inverse.** f is guaranteed to be an injective piecewise-linear map, and we can swap the roles of $\mathbf{V}^i$ and $\mathbf{U}^i$ to immediately get the inverse map as well as the inverse's Jacobian.

• **Easy and fast Jacobian computation.** Computing the Jacobian (see supplemental) requires $2n$ multiplications of small 3×3 matrices, where n is the number of layers. In comparison, methods such as RealNVP [16] are designed to have efficient access to the *determinant* of the Jacobian, but require n multiplications of large, dense matrices to get a full Jacobian. This is even worse when second-order optimization is needed, e.g., when the Jacobians are involved in a loss (Section 4.1).

• **Robust and expressive.** Our framework inherits the virtues of mesh-based deformation, e.g., numerical stability, and ability to represent elaborate deformations, with this expressivity boosted by the ability to create deep compositions of these deformations. Each Tutte layer relies on a single well behaved linear system, solved with a constant memory footprint and speed. In contrast, NeuralODE [10] poses a hard-to-tune tradeoff between accuracy, speed, and memory consumption.

4. Experiments

We evaluate the capabilities of our injective representation both on learning a space of deformations, as well as within an optimization setting. We additionally compare to several state of the art methods for deformations - both ones that are injective as well as ones that are not. In the supplementary material, we ablate the main design choices of our method *i.e.* number of layers, resolution of each layer, as well as the orientation of the projection planes.

4.1. Deformation of Neural Radiance Fields

Neural Radiance Fields (NeRFs) [55] are quickly becoming one of the most popular representations for 3D scenes. Applications that use NeRFs thus require methods to manipulate and deform them, and significant research has been dedicated to NeRF deformation methods [34, 68, 94]. We represent a NeRF using Instant-NGP [58], and render it using NeRFStudio [77], by interfacing with their code and modifying the sampling function to go through our deformation, as we explain next.

Rendering the deformed NeRF. As discussed in previous works [86], for correct rendering of a deformed NeRF, one requires both the inverse deformation as well as its Jacobian: a NeRF $N(\mathbf{p}, \boldsymbol{r}) \rightarrow c, \sigma$ maps a point $\mathbf{p}$ and a view direction $\boldsymbol{r}$ to color c and density σ, thus renderable via a ray-tracing process. Given a deformation f, the point $\mathbf{p}$ and ray $\boldsymbol{r}$ in the deformed space correspond to the point $\mathbf{p}' \triangleq f^{-1}(p)$ ray $\boldsymbol{r}' \triangleq D_{\mathbf{p}} f^{-1} \cdot \boldsymbol{r}$ in undeformed NeRF space. Hence, we require efficient computation of $f^{-1}, D_{\mathbf{p}} f^{-1}$ in order to compute $N(\mathbf{p}', \boldsymbol{r}')$.

The only injective method, aside from ours, that supports an efficient computation of the inverse *and* its Jacobian is NeuralODE [10, 30, 36] - we compare to this method and show our robustness and higher accuracy. We additionally compare to non-injective methods and show the criticality of injectivity.

Elastically deforming the NeRF. In order to deform the NeRF, we optimize the map f to satisfy the user-specified constraints in a variational as-rigid-as-possible [74] manner, minimizing the elastic energy, Equation 7. Previous methods use constructions such as proxy "rigs", e.g., cages [68] or point clouds [47], leading to inaccuracies (e.g., when recovering the rig's geometry from an inaccurate NeRF, or when mapping between the NeRF and the rig). Our guaranteed injectivity enables deforming NeRFs directly without the tedious, brittle, proxy construction process, and we define the elastic energy of f_θ over the density field itself:

$$\mathcal{L}_{\text{elastic}} = \int_{\Omega} E_{f_\theta}(\mathbf{p})\, \sigma(\mathbf{p}), \tag{9}$$

where the integral is over the volumetric unit cube Ω, $E_{f_\theta}(\mathbf{p})$ is defined in Equation (7), and σ is the NeRF's density function. The constraints are set by a user through a simple GUI, selecting a region $\mathcal{P}$ as a "handle" and shifting it by a rigid motion R to a new position $R(\mathcal{P})$. We enforce these constraints through an additional loss term,

$$\mathcal{L}_{\text{handle}} = \int_{\mathcal{P}} \| f(\mathbf{p}) - R(\mathbf{p}) \|^2 . \tag{10}$$

We estimate these integrals by rejection sampling on the density function σ, using a threshold of 1 on its value. Finally, we optimize the parameters θ of f_θ w.r.t. the loss,

$$\mathcal{L} = \lambda_{\text{elastic}} \mathcal{L}_{\text{elastic}} + \lambda_{\text{handle}} \mathcal{L}_{\text{handle}} + \lambda_{\text{Reg}} \mathcal{L}_{\text{Reg}} . \tag{11}$$

We show rendered NeRFs deformed by our method in Figure 3, with the original NeRF shown from two views, and the selected constraints illustrated with green arrows on the left column. See more results in the supp. material.

Comparison to NeRF-deformation approaches. We compare our method with three state-of-the-art NeRF deformation techniques [47, 86, 94] in Figure 3. For each

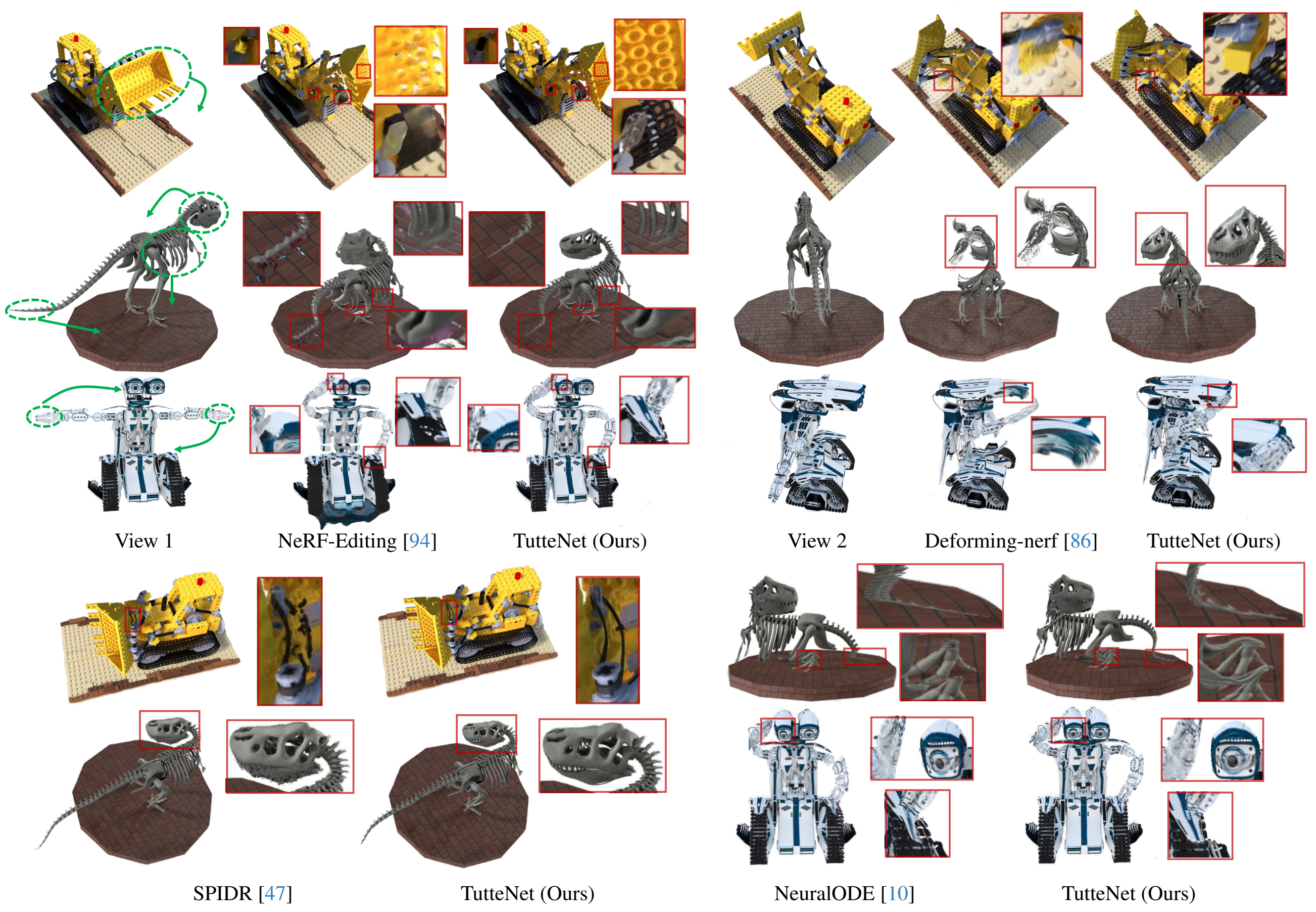

Figure 3. **Comparing NeRF deformation methods.** We minimize the elastic deformation energy of NeRFs under user-specified constraints (left, in green) and compare the visual quality of our results with other techniques. Non-injective methods such as NeRF-Editing [94] and Deforming-NeRF [86] lead to non-injective deformations due to internal inversions and intersections, in turn leading to visible artifacts. SPIDR [47] relies on a hybrid SDF/point cloud representation, leading to degradation in detail (T-Rex teeth) as well non-injective artifacts (tractor). We additionally compare to the only other injective method that is applicable for this experiment, NeuralODE [10] whose injectivity avoids visual artifacts, but causes *geometric* artifacts such as squashing the T-Rex's tail and the robot's eye.

method, we optimize its deformation g with respect to its degrees of freedom, fitting it to the injective deformation f produced by our method. We sample points and minimize the L^2 distance between the images of corresponding points, $\sum_{\mathbf{p}} \|f(\mathbf{p}) - g(p)\|$, by optimizing the deformation's degrees of freedom with respect to this loss. We perform these tests using each method's own deformation and rendering code.

These methods focus on interactivity and speed, often losing injectivity of 3D volumetric space when fitted to strong deformations, thus resulting in artifacts. Deforming-nerf [86] deforms the NeRF by building a cage around it and moves points by linear dependencies with the cage's vertices. This low dimensional space cannot capture the desired deformation, and without careful attention easily leads to noninjective and tangled configurations which squash the head of the T-rex and lead to rendering artifacts that blur the texture of the robot's head. Nerf-Editing [94]'s deformation of the tail of the T-rex creates an entanglement of rendering rays with the brick floor, creating "bleeding" artifacts in the rendering (see zoom-in). SPIDR [47] bakes the NeRF into a point cloud, and we used their dataset. When deformed, it can lead to a "discrete" version of non-injectivity, mixing points, resulting in merged teeth for the T-Rex and incorrect rendering of the Lego model. In contrast, our deformations remain plausible and crisp, for large displacements and for diverse shapes.

Comparison to NeuralODE [10, 30, 36]. As discussed above, NeuralODE is the only other method that provides both inverse and Jacobians in a computationally-feasible manner (e.g., not resorting to second-order derivation of an MLP when optimizing the Jacobian-dependent energy, Equation (9)). We replaced our representation with theirs and ran the experiment with exactly the same setup. Results are shown in Figure 3. This comparison highlights

	Fitting		Learning		Timing (sec.)	
	Vert. ↓	Grad. ↓	Vert. ↓	Grad. ↓	Forward	Jacobian
RealNVP [16]	1.7	12.5	4.21	35.2	0.006	136
i-Resnet [6]	17.9	40.2	13.4	92.3	0.005	39
NeuralODE [10]	0.19	2.6	1.24	25.4	0.11	0.19
TutteNet (ours)	0.15	4.4	0.16	7.7	0.09	0.02

Table 2. **Quantitative comparison of injective deformation methods.** We compare the ability of our TutteNet, i-ResNet [6], RealNVP [16], and NeuralODE [10] on the human deformation fitting and learning experiments, Section 4.2. We report the vertex and mesh gradient terms from Equation 12, both multiplied by 10^3. We report average timings on the right.

the importance of injectivity for NeRF deformation, as neither of the two methods exhibits rendering artifacts in any scenario. However, the zoom-ins reveal geometric issues: NeuralODE completely collapses part of the T-Rex's leg, and squashes the eye and hand of the robot, due to their proximity to one another. We additionally note that NeuralODE's numerical integration sometimes leads to running out of memory or significant stalling, when run on a large set of sample points, reducing its applicability to the NeRF rendering setting.

4.2. Learning Injective Deformations

We evaluate the applicability of TutteNet in a learning setting. Here, the parameters θ, which define the deformation f_θ, are predicted by a neural network - note that as opposed to a standard "hypernetwork" which predicts parameters of another neural network, here the neural network predicts geometrically meaningful degrees of freedom and hence we do not expect significant degradation in accuracy. To quantify and evaluate our representation's ability to accurately capture injective deformations, we require a dataset with ground truths, and hence we choose to use the highly popular SMPL [51] model, which can generate a dataset of human meshes with groundtruth 1-to-1 correspondences between their vertices.

SMPL is parameterized by two sets of parameters: P and B, dictating the human pose and body shape, resp. We generate a dataset of different body-shaped humans, each in a pair of source and target poses, S_{B,P_s}, S_{B,P_t}. We randomly sample poses, discarding results with self-intersections. See the supp. material for full details.

Our training scheme trains the neural network to receive the source human S_{B,P_s}, and the target pose parameters P_t, and based on them predict the deformation f_θ that deforms the source to the target $f_\theta\left(S_{B,P_s}\right) = S_{B,P_t}$. To avoid inference that relies on specific geometric structure, we encode each source human S_{B,P_s} by rendering it from several viewpoints and using a visual encoder. We concatenate the output of the encoder along with the pose parameters P_t into a code $\boldsymbol{z}$, which is fed into an MLP architecture that predicts the final deformation parameters θ. We compute the map f_θ via Algorithm 2 and use it to compute the same loss used

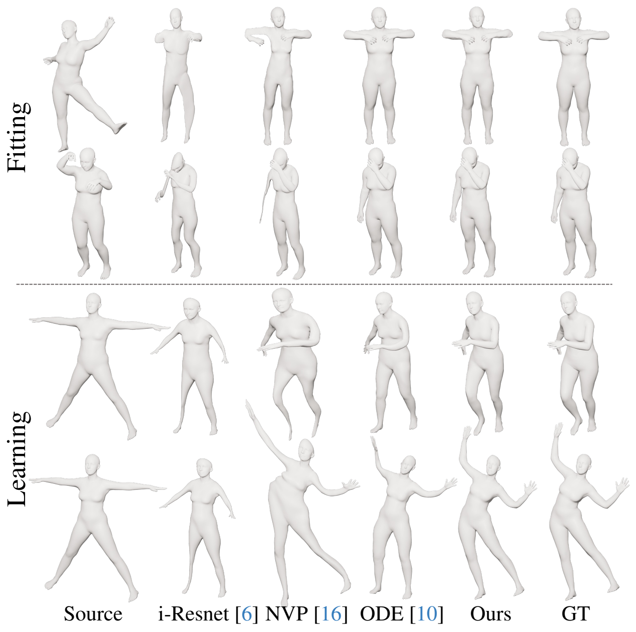

Figure 4. **Visual comparison of accuracy of injective deformation methods.** We compare the ability of our TutteNet, i-ResNet [6], RealNVP [16], and NeuralODE [10] on fitting (top) and learning (bottom) human deformations, Section 4.2. Our method produces highly-accurate results in the learning experiment while all others show visible artifacts. For the fitting experiment, only NeuralODE [10] achieves similar accuracy to ours.

by [4] for learning mesh deformations:

$$\mathcal{L} = \left\| f_\theta\left(S_{B,P_s}\right) - S_{B,P_t} \right\|^2 + \\ 0.1 \left\| J f_\theta\left(S_{B,P_s}\right) - J S_{B,P_t} \right\|^2, \tag{12}$$

where the first term is the L_2 distance between the deformed human mesh's vertices and the ground-truth target vertices, and the second term is the L_2 distance between the deformed mesh's and the ground truth mesh's intrinsic deformation gradient (note: *not* the map's Jacobians), obtained by using the source mesh's gradient operator. See supplementary for full details on training. Figure 4, bottom shows the predicted deformations. See more results in the supp. material.

Since the network (trained solely on meshes) produces volumetric injective deformations, it can be readily applied to other neural fields such as NeRFs [55] and SDFs [62] - Figure 5 shows results of applying the same network, without retraining, on: 1) synthetic NeRF created from a rendered model; 2) real in-the-wild NeRF captured by a smartphone; 3) SDF, showing the SDF isolines as well as the marching cube reconstruction (note, though, that any deformation of an SDF violates the Eikonal equation. Hence, while the deformed field represents a valid shape, it is no longer an SDF). Although the network was only trained with respect to points *on the surface* of the mesh, its injectivity ensures it produces meaningful deformations on the volume of the models. We additionally note that there are many methods that focus specifically on deforming NeRFs of humans (*e.g*. SHERF [29]), while we use humans as a benchmark for comparing and measuring the accuracy of our method, as well as showing its generality: unlike these other techniques [8, 35, 37, 42, 66, 67, 75, 82, 84, 95], we did not use any human-specific priors in the design of the representation, and the same exact method could be applied as-is to any other deformation dataset.

Comparison to other methods for learning injective deformations. We use the learning experiment to compare our method with other key representatives of families of invertible neural representations: RealNVP [16] (normalizing flows), i-ResNet [6] (Lipschitz-bounded networks) and NeuralODE [10] (continuous flow/ODE solutions) - all of which have been successfully applied to 3D tasks [27, 36, 43, 59, 65, 81, 90]. We trained these methods exactly as we trained ours, with their provided code and with same number of model parameters, and performed hyperparameter sweeps to find the best-performing choices for each - refer to the supplementary. Quantitative results are shown in Table 2, and representative example deformations are visualized in Figure 4, bottom. Our method accurately learns the deformation space, achieving near-identical deformations and the lowest fitting error, while the other methods achieve higher errors, leading to visible artifacts.

As an additional experiment, we measure the ability of each technique to represent *one, single deformation*, by "overfitting" the network (without a conditional) on a pair of source/target humans. For this evaluation, we randomly generated 200 different-bodied humans, and sampled pose source/target pairs from the AMASS dataset [53] for each of them. We show quantitative results in Table 2 and qualitative results in Figure 4.

As is evident from both experiments, RealNVP [16] and i-ResNet [6] produce inaccurate results compared to us, both in the learning experiment and in the fitting experiment. Indeed, they are designed to perform extremely well on high-dimensional tasks, but are less successful in 3D tasks which require very high accuracy (note the shrunk parts in Figure 4). RealNVP [16] has low-dimensional injective coupling layers similar to us, however, each of their layers is less expressive than a Tutte layer. i-ResNet [6] achieves invertibility by regularizing ResNet blocks to have a Lipschitz constant < 1, however, this family of functions leads to reduced expressivity to fit 3D deformations.

While NeuralODE [10] produces significantly less accurate deformations in the learning experiment, for the fitting experiment, they achieve results comparable to ours, with results visually close to indistinguishable in Figure 4, and in fact, for the fitting experiment, attain a slightly lower average error than us on the mesh-gradient fitting term ("Grad.") while we achieve a lower vertex fitting term, see Table 2.

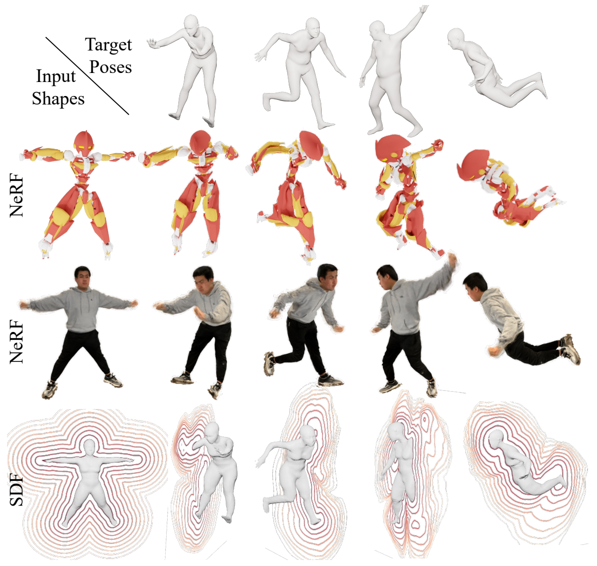

Figure 5. **Applying the trained network to deform neural fields.** The neural network from the learning experiment (Section 4.2), trained to predict deformations on a dataset of human SMPL [51] meshes (top row, demonstrating the desired target pose), is seamlessly applied to deform synthetic and real *NeRFs* [55] (middle two rows), and SDFs [62] (bottom row).

The degradation in NeuralODE's performance when scaling up for the learning experiment is expected: to achieve injectivity, they leverage uniqueness of ODE solutions and plot reversible trajectories of points in 3D space - this requires numerical integration, which becomes increasingly difficult as the learned functional space represented by the neural network grows more convoluted (refer to [10] and their discussion on their Figure 3(d)). For the fitting experiment we used the default hyperparameters from [10], and for the learning experiment used the ones from [36].

5. Conclusion

We have presented an expressive, numerically robust, and computationally efficient representation of 3D injective deformations that can be plugged without modification into other applications requiring injective deformation modules.

Our method has two main limitations. First, the map evaluation cannot be done at interactive times, preventing real-time rendering and interaction for the moment. Second, deforming one part of the space may have an effect on another part, and it is non-trivial to completely localize deformations to one part of a shape. This is true for all the other injective deformation techniques as well.

We are excited by the possible uses of our framework, e.g., for long-range optical flow [81], or to regularize non-rigid 3D registration [13]. Additionally, use cases for other types of low-dimensional injective maps are highly attractive, e.g., for surface-to-surface mappings through common domains, which require an injective 2D map [56].

References

[1] Noam Aigerman and Thibault Groueix. Generative escher meshes. *arXiv preprint arXiv:2309.14564*, 2023. 2, 4

[2] Noam Aigerman and Yaron Lipman. Injective and bounded distortion mappings in 3d. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 32(4):106:1–106:14, 2013. 2

[3] Noam Aigerman and Yaron Lipman. Orbifold tutte embeddings. *ACM Trans. Graph.*, 34(6):190–1, 2015. 4

[4] Noam Aigerman, Kunal Gupta, Vladimir G Kim, Siddhartha Chaudhuri, Jun Saito, and Thibault Groueix. Neural jacobian fields: Learning intrinsic mappings of arbitrary meshes. *SIGGRAPH*, 2022. 3, 7

[5] Jan Bednarik, Shaifali Parashar, Erhan Gundogdu, Mathieu Salzmann, and Pascal Fua. Shape reconstruction by learning differentiable surface representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4716–4725, 2020. 3

[6] 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. 2, 7, 8

[7] Marcel Campen, Cláudio T Silva, and Denis Zorin. Bijective maps from simplicial foliations. *ACM Transactions on Graphics (TOG)*, 35(4):1–15, 2016. 2, 3

[8] Jianchuan Chen, Ying Zhang, Di Kang, Xuefei Zhe, Linchao Bao, Xu Jia, and Huchuan Lu. Animatable neural radiance fields from monocular rgb videos. *arXiv preprint arXiv:2106.13629*, 2021. 8

[9] Jun-Kun Chen, Jipeng Lyu, and Yu-Xiong Wang. NeuralEditor: Editing neural radiance fields via manipulating point clouds. In *CVPR*, 2023. 3

[10] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018. 2, 5, 6, 7, 8

[11] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019. 3

[12] Chong Bao and Bangbang Yang, Zeng Junyi, Bao Hujun, Zhang Yinda, Cui Zhaopeng, and Zhang Guofeng. Neumesh: Learning disentangled neural mesh-based implicit field for geometry and texture editing. In *European Conference on Computer Vision (ECCV)*, 2022. 3

[13] Bailin Deng, Yuxin Yao, Roberto M Dyke, and Juyong Zhang. A survey of non-rigid 3d registration. In *Computer Graphics Forum*, pages 559–589. Wiley Online Library, 2022. 1, 8

[14] Yu Deng, Jiaolong Yang, and Xin Tong. Deformed implicit field: Modeling 3d shapes with learned dense correspondence. In *IEEE Computer Vision and Pattern Recognition*, 2021. 3

[15] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014. 2

[16] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 2, 4, 5, 7, 8

[17] Xingyi Du, Danny M Kaufman, Qingnan Zhou, Shahar Z Kovalsky, Yajie Yan, Noam Aigerman, and Tao Ju. Optimizing global injectivity for constrained parameterization. *ACM Trans. Graph.*, 40(6):260–1, 2021. 2

[18] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017. 3

[19] Michael Floater. One-to-one piecewise linear mappings over triangulations. *Mathematics of Computation*, 72(242):685–696, 2003. 1, 2, 3, 4

[20] Xiao-Ming Fu and Yang Liu. Computing inversion-free mappings by simplex assembly. *ACM Trans. Graph.*, 35(6), 2016. 2

[21] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021. 3

[22] Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva, François Protais, Nicolas Ray, and Dmitry Sokolov. Foldover-free maps in 50 lines of code. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021. 2

[23] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pages 881–889. PMLR, 2015. 2

[24] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*, 2018. 2

[25] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[26] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. 3d-coded : 3d correspondences by deep deformation. In *European Conference on Computer Vision (ECCV)*, 2018. 3

[27] Kunal Gupta and Manmohan Chandraker. Neural mesh flow: 3d manifold mesh generation via diffeomorphic flows. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 2, 8

[28] Daniel Holden, Jun Saito, and Taku Komura. Learning an inverse rig mapping for character animation. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, page 165–173, New York, NY, USA, 2015. Association for Computing Machinery. 3

[29] Shoukang Hu, Fangzhou Hong, Liang Pan, Haiyi Mei, Lei Yang, and Ziwei Liu. Sherf: Generalizable human nerf from a single image. *arXiv preprint arXiv:2303.12791*, 2023. 8

[30] Jingwei Huang, Chiyu Max Jiang, Baiqiang Leng, Bin Wang, and Leonidas Guibas. Meshode: A robust and scalable framework for mesh deformation. *arXiv preprint arXiv:2005.11617*, 2020. 2, 4, 5, 6

[31] Qixing Huang, Xiangru Huang, Bo Sun, Zaiwei Zhang, Junfeng Jiang, and Chandrajit Bajaj. Arapreg: An as-rigid-as possible regularization loss for learning deformable shape generators. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5815–5825, 2021. 3

[32] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*, 2014. 1

[33] Tomas Jakab, Richard Tucker, Ameesh Makadia, Jiajun Wu, Noah Snavely, and Angjoo Kanazawa. Keypointdeformer: Unsupervised 3d keypoint discovery for shape control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12783–12792, 2021. 3

[34] Clément Jambon, Bernhard Kerbl, Georgios Kopanas, Stavros Diolatzis, Thomas Leimkühler, and George" Drettakis. Nerfshop: Interactive editing of neural radiance fields". *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 6(1), 2023. 3, 5

[35] Boyi Jiang, Yang Hong, Hujun Bao, and Juyong Zhang. Selfrecon: Self reconstruction your digital avatar from monocular video. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 8

[36] 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. 2, 5, 6, 8

[37] Wei Jiang, Kwang Moo Yi, Golnoosh Samei, Oncel Tuzel, and Anurag Ranjan. Neuman: Neural human radiance field from a single video, 2022. 8

[38] Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. Simplicial complex augmentation framework for bijective maps. *ACM Transactions on Graphics*, 36(6), 2017. 2

[39] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 371–386, 2018. 3

[40] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016. 2

[41] Shahar Z Kovalsky, Noam Aigerman, Ronen Basri, and Yaron Lipman. Controlling singular values with semidefinite programming. *ACM Trans. Graph.*, 33(4):68–1, 2014. 2, 4

[42] Youngjoong Kwon, Dahun Kim, Duygu Ceylan, and Henry Fuchs. Neural human performer: Learning generalizable radiance fields for human performance rendering. *Advances in Neural Information Processing Systems*, 34, 2021. 8

[43] Jiahui Lei and Kostas Daniilidis. Cadex: Learning canonical deformation coordinate space for dynamic surface representation via neural homeomorphism. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 2, 8

[44] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4):49, 2020. 2

[45] Peizhuo Li, Kfir Aberman, Rana Hanocka, Libin Liu, Olga Sorkine-Hornung, and Baoquan Chen. Learning skeletal articulations with neural blend shapes. *ACM Transactions on Graphics (TOG)*, 40(4):1–15, 2021. 3

[46] Shaoxu Li and Ye Pan. Interactive geometry editing of neural radiance fields. *arXiv preprint arXiv:2303.11537*, 2023. 3

[47] 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. 3, 5, 6

[48] Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. Neuroskinning: Automatic skin binding for production characters with deep graph networks. *ACM Trans. Graph.*, 38(4), 2019. 3

[49] Minghua Liu, Minhyuk Sung, Radomir Mech, and Hao Su. Deepmetahandles: Learning deformation meta-handles of 3d meshes with biharmonic coordinates. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12–21, 2021. 3

[50] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu, and Bryan Russell. Editing conditional radiance fields. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 3

[51] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, 2015. 7, 8

[52] Jinxin Lv, Zhiwei Wang, Hongkuan Shi, Haobo Zhang, Sheng Wang, Yilang Wang, and Qiang Li. Joint progressive and coarse-to-fine registration of brain mri via deformation field integration and non-rigid feature fusion. *IEEE Transactions on Medical Imaging*, 41(10):2788–2802, 2022. 1

[53] 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. 8

[54] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019. 3

[55] 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. 1, 5, 7, 8

[56] Luca Morreale, Noam Aigerman, Vladimir G. Kim, and Niloy J. Mitra. Neural surface maps. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 4639–4648. Computer Vision Foundation / IEEE, 2021. 8

[57] Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. Air meshes for robust collision handling. *ACM Transactions on Graphics (TOG)*, 34(4):1–9, 2015. 2

[58] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 5

[59] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019. 2, 8

[60] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. 2

[61] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017. 2

[62] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 3, 7, 8

[63] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021. 3

[64] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021. 3

[65] Despoina Paschalidou, Angelos Katharopoulos, Andreas Geiger, and Sanja Fidler. Neural parts: Learning expressive 3d shape abstractions with invertible neural networks. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2, 8

[66] Sida Peng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In *CVPR*, 2021. 8

[67] Sida Peng, Chen Geng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Xiaowei Zhou, and Hujun Bao. Implicit neural representations with structured latent codes for human body modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. 8

[68] Yicong Peng, Yichao Yan, Shengqi Liu, Yuhao Cheng, Shanyan Guan, Bowen Pan, Guangtao Zhai, and Xiaokang Yang. Cagenerf: Cage-based neural radiance field for generalized 3d deformation and animation. In *Advances in Neural Information Processing Systems*, pages 31402–31415. Curran Associates, Inc., 2022. 3, 5

[69] Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. Scalable locally injective mappings. *ACM Trans. Graph.*, 36(2), 2017. 2

[70] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015. 2

[71] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International conference on machine learning*, pages 1218–1226. PMLR, 2015. 2

[72] Christian Schüller, Ladislav Kavan, Daniele Panozzo, and Olga Sorkine-Hornung. Locally injective mappings. In *Computer Graphics Forum*, pages 125–135. Wiley Online Library, 2013. 2

[73] Jason Smith and Scott Schaefer. Bijective parameterization with free boundaries. *ACM Transactions on Graphics (TOG)*, 34(4):70, 2015. 2

[74] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, page 109–116, Goslar, DEU, 2007. Eurographics Association. 3, 5

[75] Shih-Yang Su, Frank Yu, Michael Zollhöfer, and Helge Rhodin. A-nerf: Articulated neural radiance fields for learning human shape, appearance, and pose. *Advances in Neural Information Processing Systems*, 34:12278–12291, 2021. 8

[76] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2846–2855, 2021. 3

[77] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, Justin Kerr, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings, SIGGRAPH 2023, Los Angeles, CA, USA, August 6-10, 2023*, pages 72:1–72:12. ACM, 2023. 5

[78] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12959–12970, 2021. 3

[79] Brian L Trippe and Richard E Turner. Conditional density estimation with bayesian normalising flows. *arXiv preprint arXiv:1802.04908*, 2018. 2

[80] William Thomas Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963. 1, 2, 3, 4

[81] Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. Tracking everything everywhere all at once. *arXiv preprint arXiv:2306.05422*, 2023. 1, 2, 8

[82] Chung-Yi Weng, Brian Curless, Pratul P Srinivasan, Jonathan T Barron, and Ira Kemelmacher-Shlizerman. Humannerf: Free-viewpoint rendering of moving people from monocular video. In *Proceedings of the IEEE/CVF conference on computer vision and pattern Recognition*, pages 16210–16220, 2022. 8

[83] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Wang Xinggang. 4d gaussian splatting for real-time dynamic scene rendering. *arXiv preprint arXiv:2310.08528*, 2023. 3

[84] Hongyi Xu, Thiemo Alldieck, and Cristian Sminchisescu. H-nerf: Neural radiance fields for rendering and temporal reconstruction of humans in motion. In *Neural Information Processing Systems*, 2021. 8

[85] Shiyao Xu, Lingzhi Li, Li Shen, and Zhouhui Lian. Desrf: Deformable stylized radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 709–718, 2023. 3

[86] Tianhan Xu and Tatsuya Harada. Deforming radiance fields with cages. In *ECCV*, 2022. 3, 5, 6

[87] Wei-Wei Xu and Kun Zhou. Gradient domain mesh deformation—a survey. *Journal of computer science and technology*, 24:6–18, 2009. 1

[88] Zhan Xu, Yang Zhou, Evangelos Kalogerakis, and Karan Singh. Predicting animation skeletons for 3d articulated models via volumetric nets. In *2019 International Conference on 3D Vision (3DV)*, 2019. 3

[89] Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. Rignet: Neural rigging for articulated characters. *ACM Trans. on Graphics*, 39, 2020. 3

[90] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019. 2, 8

[91] Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. 2

[92] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 206–215, 2018. 3

[93] Wang Yifan, Noam Aigerman, Vladimir G Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. Neural cages for detail-preserving 3d deformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 75–83, 2020. 3

[94] 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. 3, 5, 6

[95] Fuqiang Zhao, Wei Yang, Jiakai Zhang, Pei Lin, Yingliang Zhang, Jingyi Yu, and Lan Xu. Humannerf: Efficiently generated human radiance field from sparse inputs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7743–7753, 2022. 8

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-

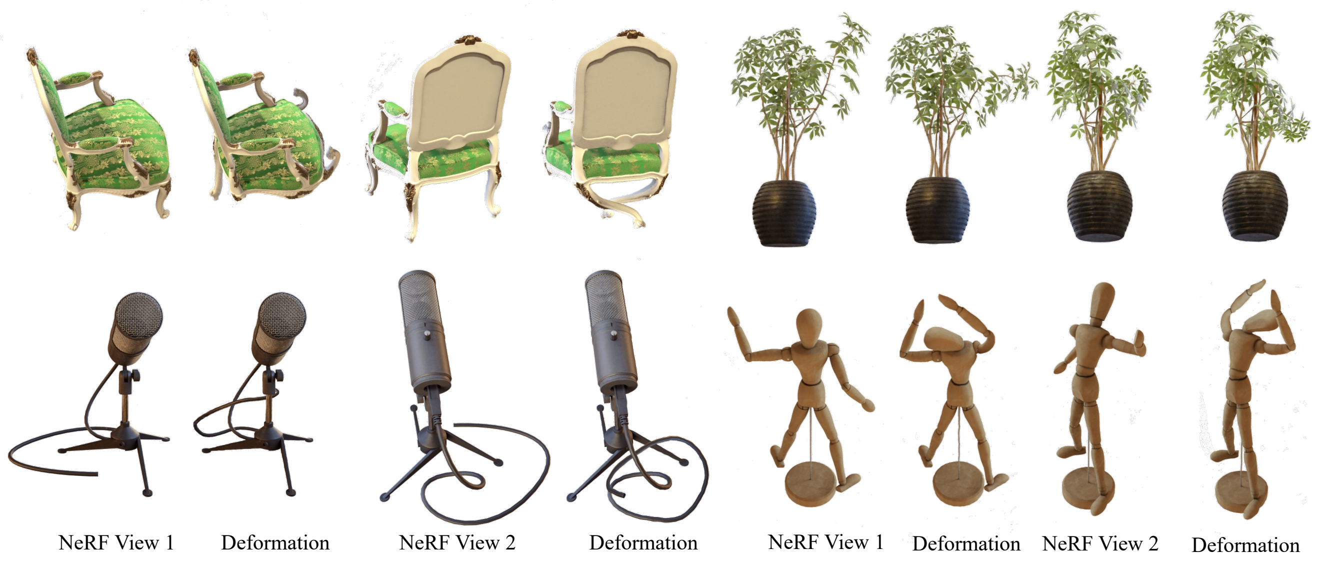

Figure 1. Additional elastic deformations of various NeRFs [7] via our representation, as described in Section 4.1 in the main paper.

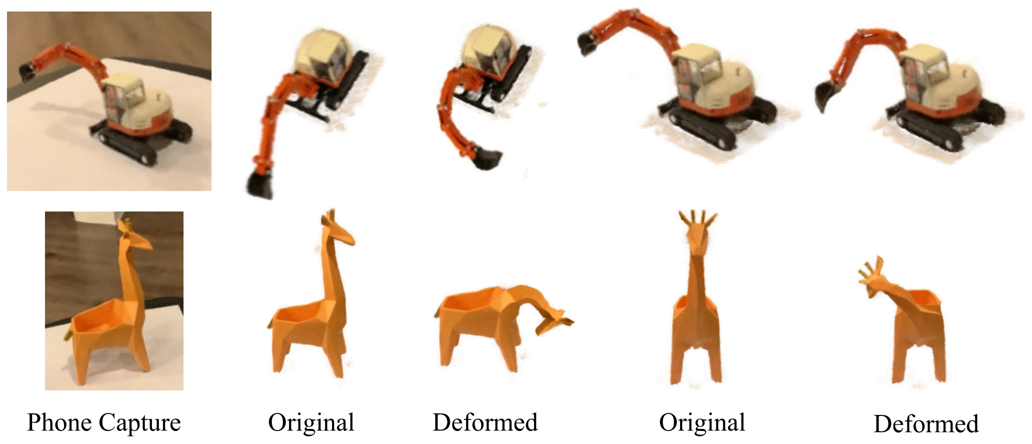

Figure 2. Additional elastic deformations of phone-captured NeRFs [7] via our representation.

	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-

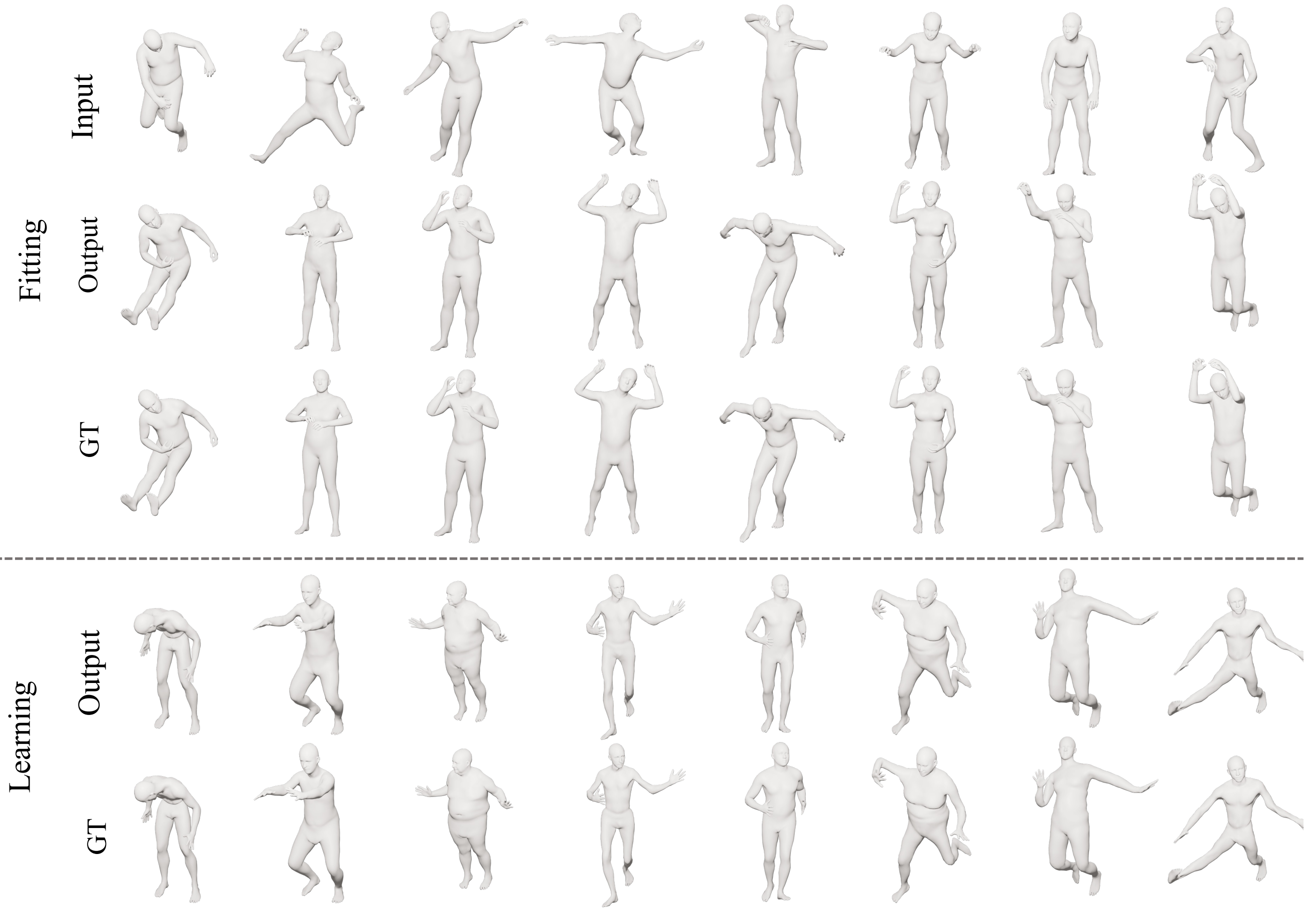

Figure 3. Additional results on the learning (bottom) and fitting (top) human poses experiment (section 4.2).

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

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

where $\mathbf{t}$ is the triangle the point lies in (per Algorithm 1), and $\tilde{A}_{\mathbf{t}} = \begin{pmatrix} A_{\mathbf{t}} & \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \\ 0 \; 0 & 1 \end{pmatrix}$ 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

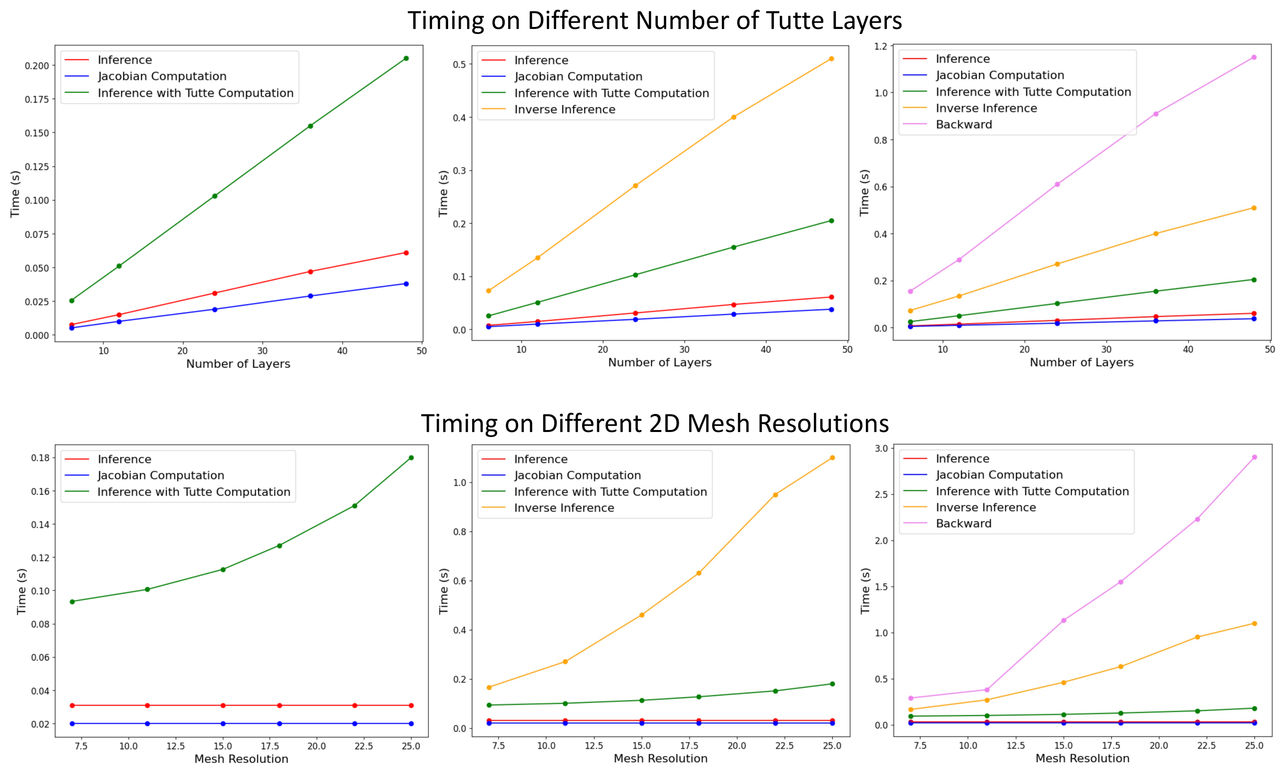

Figure 4. Timing analysis on number of Tutte layers and 2D mesh resolutions. Top: timing w.r.t. number of Tutte layers. Bottom timing w.r.t. the 2D mesh's resolutions, with the fixed number of layers as 24.

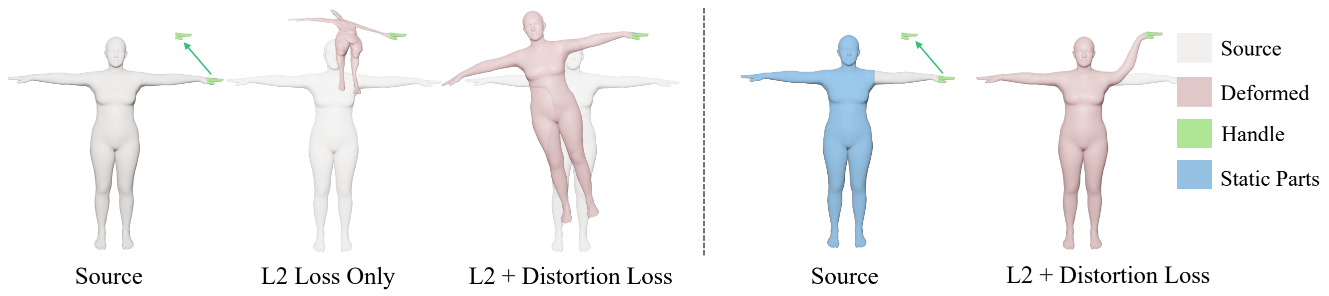

Figure 5. Our deformation is not localized: we show the *source* model and the constraint (green) dragging the hand to a new position. Second from left, we show the result of optimizing for the fitting of the constraint, without applying any regularization to other parts of the shape; the unconstrained 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.

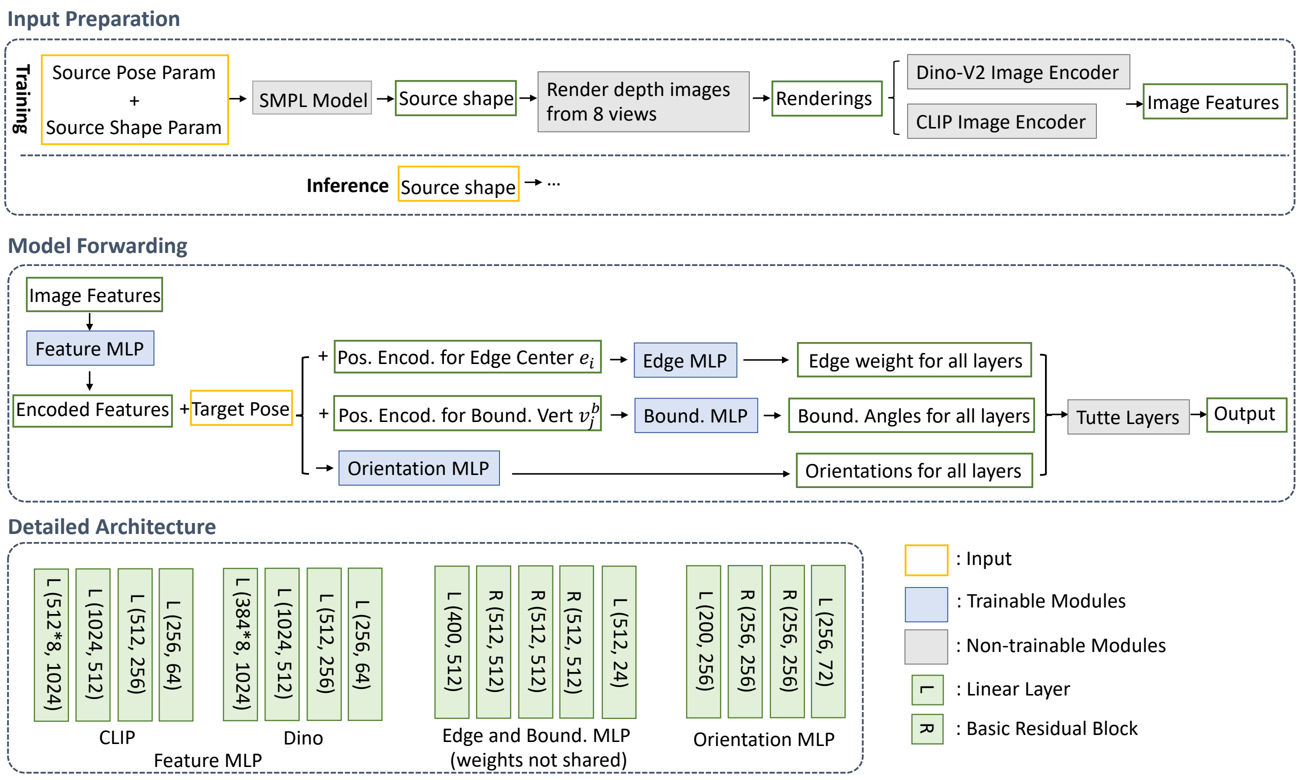

Figure 6. Detailed model architecture and data preparation for the learning experiment (Section 4.2 in the main paper).

- **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