

---

# Neural Non-Rigid Tracking

---

Aljaž Božič<sup>1\*</sup>

aljaz.bozic@tum.de

Pablo Palafox<sup>1\*</sup>

pablo.palafox@tum.de

Michael Zollhöfer<sup>2</sup>

Angela Dai<sup>1</sup>

Justus Thies<sup>1</sup>

Matthias Nießner<sup>1</sup>

<sup>1</sup>Technical University of Munich

<sup>2</sup>Stanford University

## Abstract

We introduce a novel, end-to-end learnable, differentiable non-rigid tracker that enables state-of-the-art non-rigid reconstruction by a learned robust optimization. Given two input RGB-D frames of a non-rigidly moving object, we employ a convolutional neural network to predict dense correspondences and their confidences. These correspondences are used as constraints in an as-rigid-as-possible (ARAP) optimization problem. By enabling gradient back-propagation through the weighted non-linear least squares solver, we are able to learn correspondences and confidences in an end-to-end manner such that they are optimal for the task of non-rigid tracking. Under this formulation, correspondence confidences can be learned via self-supervision, informing a learned robust optimization, where outliers and wrong correspondences are automatically down-weighted to enable effective tracking. Compared to state-of-the-art approaches, our algorithm shows improved reconstruction performance, while simultaneously achieving  $85\times$  faster correspondence prediction than comparable deep-learning based methods. We make our code available at <https://github.com/DeformableFriends/NeuralTracking>.

## 1 Introduction

The capture and reconstruction of real-world environments is a core problem in computer vision, enabling numerous VR/AR applications. While there has been significant progress in reconstructing static scenes, tracking and reconstruction of dynamic objects remains a challenge. Non-rigid reconstruction focuses on dynamic objects, without assuming any explicit shape priors, such as human or face parametric models. Commodity RGB-D sensors, such as Microsoft’s Kinect or Intel’s Realsense, provide a cost-effective way to acquire both color and depth video of dynamic motion. Using a large number of RGB-D sensors can lead to an accurate non-rigid reconstruction, as shown by Dou et al. [10]. Our work focuses on non-rigid reconstruction from a single RGB-D camera, thus eliminating the need for specialized multi-camera setups.

The seminal DynamicFusion by Newcombe et al. [25] introduced a non-rigid tracking and mapping pipeline that uses depth input for real-time non-rigid reconstruction from a single RGB-D camera. Various approaches have expanded upon this framework by incorporating sparse color correspondences [15] or dense photometric optimization [12]. DeepDeform [5] presented a learned correspondence prediction, enabling significantly more robust tracking of fast motion and re-localization. Unfortunately, the computational cost of the correspondence prediction network ( $\sim 2$  seconds per frame for a relatively small number of non-rigid correspondences) inhibits real-time performance.

---

\* Denotes equal contribution.

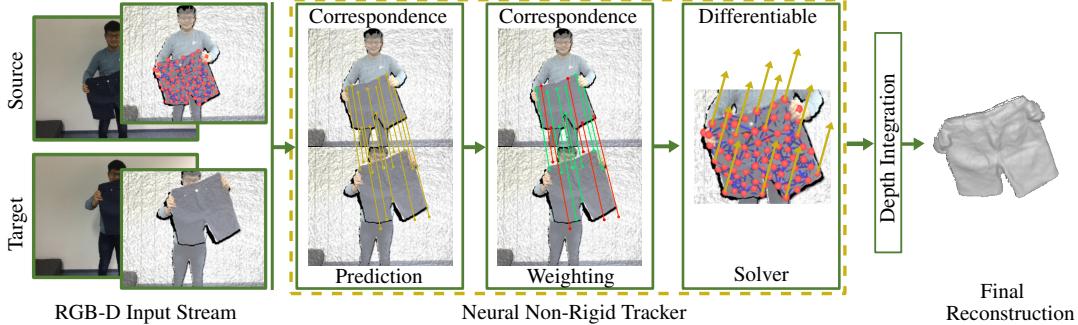


Figure 1: Neural Non-Rigid Tracking: based on RGB-D input data of a source and a target frame, our learned non-rigid tracker estimates the non-rigid deformations to align the source to the target frame. We propose an end-to-end approach, enabling correspondences and their importance weights to be informed by the non-rigid solver. Similar to robust optimization, this provides robust tracking, and the resulting deformation field can then be used to integrate the depth observations in a canonical volumetric 3D grid that implicitly represents the surface of the object (final reconstruction).

Simultaneously, work on learned optical flow has shown dense correspondence prediction at real-time rates [32]. However, directly replacing the non-rigid correspondence predictions from Božič et al. [5] with these optical flow predictions does not produce accurate enough correspondences for comparable non-rigid reconstruction performance. In our work, we propose a neural non-rigid tracker, i.e., an end-to-end differentiable non-rigid tracking pipeline which combines the advantages of classical deformation-graph-based reconstruction pipelines [25, 15] with novel learned components. Our end-to-end approach enables learning outlier rejection in a self-supervised manner, which guides a robust optimization to mitigate the effect of inaccurate correspondences or major occlusions present in single RGB-D camera scenarios.

Specifically, we cast the non-rigid tracking problem as an as-rigid-as-possible (ARAP) optimization problem, defined on correspondences between points in a source and a target frame. A differentiable Gauss-Newton solver allows us to obtain gradients that enable training a neural network to predict an importance weight for every correspondence in a completely self-supervised manner, similar to robust optimization. The end-to-end training significantly improves non-rigid tracking performance. Using our neural tracker in a non-rigid reconstruction framework results in  $85\times$  faster correspondence prediction and improved reconstruction performance compared to the state of the art.

In summary, we propose a novel neural non-rigid tracking approach with two key contributions:

- an end-to-end differentiable Gauss-Newton solver, which provides gradients to better inform a correspondence prediction network used for non-rigid tracking of two frames;
- a self-supervised approach for learned correspondence weighting, which is informed by our differentiable solver and enables efficient, robust outlier rejection, thus, improving non-rigid reconstruction performance compared to the state of the art.

## 2 Related Work

**Non-rigid Reconstruction.** Reconstruction of deformable objects using a single RGB-D camera is an important research area in computer vision. State-of-the-art methods rely on deformation graphs [31, 38] that enable robust and temporally consistent 3D motion estimation. While earlier approaches required an object template, such graph-based tracking has been extended to simultaneous tracking and reconstruction approaches [9, 25]. These works used depth fitting optimization objectives in the form of iterative closest points, or continuous depth fitting in [28, 29]. Rather than relying solely on depth information, recent works have incorporated SIFT features [15], dense photometric fitting [12], and sparse learned correspondence [5].

**Correspondence Prediction for Non-rigid Tracking.** In non-rigid tracking, correspondences must be established between the two frames we want to align. While methods such as DynamicFusion [25] rely on projective correspondences, recent methods leverage learned correspondences [5]. DeepDeform [5] relies on sparse predicted correspondences, trained on an annotated dataset of deforming objects. Since prediction is done independently for each correspondence, this results in a high compute cost, compared to dense predictions of state-of-the-art optical flow networks. Optical flow [8, 14, 32, 20] and scene flow [23, 3, 21, 35] methods achieve promising results in predicting dense correspondences between two frames, with some approaches not even requiring direct supervision [34, 18, 17]. In our proposed neural non-rigid tracking approach, we build upon PWC-Net [32] for dense correspondence prediction to inform our non-rigid deformation energy formulation. Since our approach allows for end-to-end training, our 2D correspondence prediction finds correspondences better suited for non-rigid tracking.

**Differentiable Optimization.** Differentiable optimizers have been explored for various tasks, including image alignment [6], rigid pose estimation [13, 22], multi-frame direct bundle-adjustment [33], and rigid scan-to-CAD alignment [1]. In addition to achieving higher accuracy, an end-to-end differentiable optimization approach also offers the possibility to optimize run-time, as demonstrated by learning efficient pre-conditioning methods in [11, 27, 19]. Unlike Li et al. [19], which employs an image-based tracker (with descriptors defined on nodes in a pixel-aligned graph), our approach works on general graphs and learns to robustify correspondence prediction for non-rigid tracking by learning self-supervised correspondence confidences.

### 3 Non-Rigid Reconstruction Notation

Non-rigid alignment is a crucial part of non-rigid reconstruction pipelines. In the single RGB-D camera setup, we are given a pair of source and target RGB-D frames  $\{(\mathcal{I}_s, \mathcal{P}_s), (\mathcal{I}_t, \mathcal{P}_t)\}$ , where  $\mathcal{I}_* \in \mathbb{R}^{H \times W \times 3}$  is an RGB image and  $\mathcal{P}_* \in \mathbb{R}^{H \times W \times 3}$  a 3D point image. The goal is to estimate a warp field  $\mathcal{Q} : \mathbb{R}^3 \mapsto \mathbb{R}^3$  that transforms  $\mathcal{P}_s$  into the target frame. Note that we define the 3D point image  $\mathcal{P}_s$  as the result of back-projecting every pixel  $\mathbf{u} \in \Pi_s \subset \mathbb{R}^2$  into the camera coordinate system with given camera intrinsic parameters. To this end, we define the inverse of the perspective projection to back-project a pixel  $\mathbf{u}$  given the pixel's depth  $d_{\mathbf{u}}$  and the intrinsic camera parameters  $\mathbf{c}$ :

$$\pi_{\mathbf{c}}^{-1} : \mathbb{R}^2 \times \mathbb{R} \rightarrow \mathbb{R}^3, \quad (\mathbf{u}, d_{\mathbf{u}}) \mapsto \pi_{\mathbf{c}}^{-1}(\mathbf{u}, d_{\mathbf{u}}) = \mathbf{p}. \quad (1)$$

To maintain robustness against noise in the depth maps, state-of-the-art approaches define an embedded deformation graph  $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$  over the *source* RGB-D frame, where  $\mathcal{V}$  is the set of graph nodes defined by their 3D coordinates  $\mathbf{v}_i \in \mathbb{R}^3$  and  $\mathcal{E}$  the set of edges between nodes, as described in [31] and illustrated in Fig. 1. Thus, for every node in  $\mathcal{G}$ , a global translation vector  $\mathbf{t}_{\mathbf{v}_i} \in \mathbb{R}^3$  and a rotation matrix  $\mathbf{R}_{\mathbf{v}_i} \in \mathbb{R}^{3 \times 3}$ , must be estimated in the alignment process. We parameterize rotations with a 3-dimensional axis-angle vector  $\omega \in \mathbb{R}^3$ . We use the exponential map  $\exp : \mathfrak{so}(3) \rightarrow \text{SO}(3)$ ,  $\hat{\omega} \mapsto e^{\hat{\omega}} = \mathbf{R}$  to convert from axis-angle to matrix rotation form, where the  $\hat{\cdot}$ -operator creates a  $3 \times 3$  skew-symmetric matrix from a 3-dimensional vector. The resulting graph motion is denoted by  $\mathcal{T} = (\omega_{\mathbf{v}_1}, \mathbf{t}_{\mathbf{v}_1}, \dots, \omega_{\mathbf{v}_N}, \mathbf{t}_{\mathbf{v}_N}) \in \mathbb{R}^{N \times 6}$  for a graph with  $N$  nodes.

Dense motion can then be computed by interpolating the nodes' motion  $\mathcal{T}$  by means of a warping function  $\mathcal{Q}$ . When applied to a 3D point  $\mathbf{p} \in \mathbb{R}^3$ , it produces the point's deformed position

$$\mathcal{Q}(\mathbf{p}, \mathcal{T}) = \sum_{\mathbf{v}_i \in \mathcal{V}} \alpha_{\mathbf{v}_i} (e^{\hat{\omega}_{\mathbf{v}_i}} (\mathbf{p} - \mathbf{v}_i) + \mathbf{v}_i + \mathbf{t}_{\mathbf{v}_i}). \quad (2)$$

The weights  $\alpha_{\mathbf{v}_i} \in \mathbb{R}$ , also known as *skinning* weights, measure the influence of each node on the current point  $\mathbf{p}$  and are computed as in [37]. Please see the supplemental material for further detail.

### 4 Neural Non-rigid Tracking

Given a pair of source and target RGB-D frames  $(\mathcal{Z}_s, \mathcal{Z}_t)$ , where  $\mathcal{Z}_* = (\mathcal{I}_* | \mathcal{P}_*) \in \mathbb{R}^{H \times W \times 6}$  is the concatenation of an RGB and a 3D point image as defined in Section 3, we aim to find a function  $\Theta$  that estimates the motion  $\mathcal{T}$  of a deformation graph  $\mathcal{G}$  with  $N$  nodes (given by their 3D coordinates  $\mathcal{V}$ ) defined over the source RGB-D frame. This implicitly defines source-to-target dense 3D motion (see Figure 2). Formally, we have:

$$\Theta : \mathbb{R}^{H \times W \times 6} \times \mathbb{R}^{H \times W \times 6} \times \mathbb{R}^{N \times 3} \rightarrow \mathbb{R}^{N \times 6}, \quad (\mathcal{Z}_s, \mathcal{Z}_t, \mathcal{V}) \mapsto \Theta(\mathcal{Z}_s, \mathcal{Z}_t, \mathcal{V}) = \mathcal{T}. \quad (3)$$

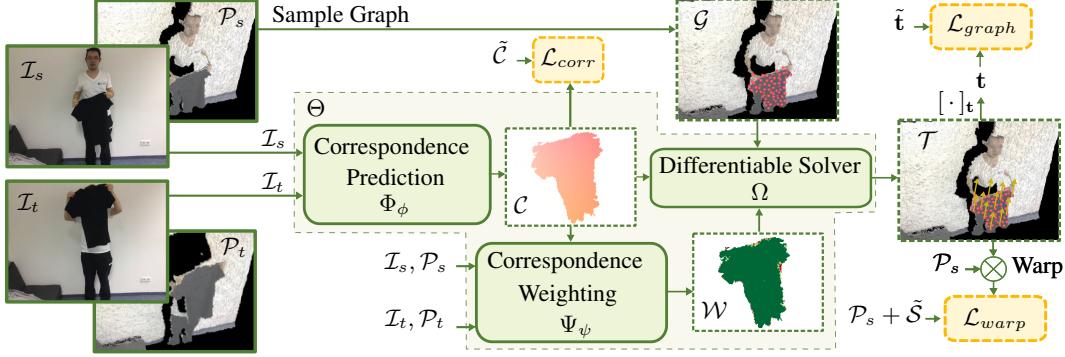


Figure 2: Overview of our neural non-rigid tracker. Given a pair of source and target images,  $\mathcal{I}_s$  and  $\mathcal{I}_t$ , a dense correspondence map  $\mathcal{C}$  between the frames is estimated via a convolutional neural network  $\Phi$ . Importance weights  $\mathcal{W}$  for these correspondences are computed through a function  $\Psi$ . Together with a graph  $\mathcal{G}$  defined over the source RGB-D frame  $\mathcal{P}_s$ , both  $\mathcal{C}$  and  $\mathcal{W}$  are input to a differentiable solver  $\Omega$ . The solver outputs the graph motion  $\mathcal{T}$ , i.e., the non-rigid alignment between source and target frames. Our approach is optimized end-to-end, with losses on the final alignment using  $\mathcal{L}_{graph}$  and  $\mathcal{L}_{warp}$ , and an intermediate loss on the correspondence map  $\mathcal{L}_{corr}$ .

To estimate  $\mathcal{T}$ , we first establish dense 2D correspondences between the source and target frame using a deep neural network  $\Phi$ . These correspondences, denoted as  $\mathcal{C}$ , are used to construct the data term in our non-rigid alignment optimization. Since the presence of outlier correspondence predictions has a strong negative impact on the performance of non-rigid tracking, we introduce a weighting function  $\Psi$ , inspired by robust optimization, to down-weight inaccurate predictions. Function  $\Psi$  outputs importance weights  $\mathcal{W}$  and is learned in a self-supervised manner. Finally, both correspondence predictions  $\mathcal{C}$  and importance weights  $\mathcal{W}$  are input to a differentiable, non-rigid alignment optimization module  $\Omega$ . By optimizing the non-rigid alignment energy (see Section 4.3), the differentiable optimizer  $\Omega$  estimates the deformation graph parameters  $\mathcal{T}$  that define the motion from source to target frame:

$$\mathcal{T} = \Theta(\mathcal{Z}_s, \mathcal{Z}_t, \mathcal{V}) = \Omega(\Phi(\cdot), \Psi(\cdot), \mathcal{V}) = \Omega(\mathcal{C}, \mathcal{W}, \mathcal{V}). \quad (4)$$

In the following, we define the dense correspondence predictor  $\Phi$ , the importance weighting  $\Psi$  and the optimizer  $\Omega$ , and describe a fully differentiable approach for optimizing  $\Phi$  and  $\Psi$  such that we can estimate dense correspondences with importance weights best suited for non-rigid tracking.

#### 4.1 Dense Correspondence Prediction

The dense correspondence prediction function  $\Phi$  takes as input a pair of source and target RGB images  $(\mathcal{I}_s, \mathcal{I}_t)$ , and for each source pixel location  $\mathbf{u} \in \Pi_s \subset \mathbb{R}^2$  it outputs a corresponding pixel location in the target image  $\mathcal{I}_t$ , which we denote by  $\mathbf{c}_u \in \Pi_t \subset \mathbb{R}^2$ . Formally,  $\Phi$  is defined as

$$\Phi : \mathbb{R}^{H \times W \times 3} \times \mathbb{R}^{H \times W \times 3} \rightarrow \mathbb{R}^{H \times W \times 2}, \quad (\mathcal{I}_s, \mathcal{I}_t) \mapsto \Phi(\mathcal{I}_s, \mathcal{I}_t) = \mathcal{C}, \quad (5)$$

where  $\mathcal{C}$  is the resulting dense correspondence map. The function  $\Phi$  is represented by a deep neural network that leverages the architecture of a state-of-the-art optical flow estimator [32].

#### 4.2 Correspondence Importance Weights

For each source pixel  $\mathbf{u} \in \Pi_s \subset \mathbb{R}^2$  and its correspondence  $\mathbf{c}_u \in \Pi_t \subset \mathbb{R}^2$ , we additionally predict an importance weight  $w_u \in (0, 1)$  by means of the weighting function  $\Psi$ . The latter takes as input the source RGB-D image  $\mathcal{Z}_s$ , the corresponding sampled target frame values  $\mathcal{Z}'_t$ , and intermediate features from the correspondence network  $\Phi$ , and outputs weights for the correspondences between source and target. Note that  $\mathcal{Z}'_t$  is the result of bilinearly sampling [16] the target image  $\mathcal{Z}_t$  at the predicted correspondence locations  $\mathcal{C}$ . The last layer of features  $\mathcal{H}$  of the correspondence network  $\Phi$ , with dimension  $D = 565$ , are used to inform  $\Psi$ . The weighting function is thus defined as

$$\Psi : \mathbb{R}^{H \times W \times 6} \times \mathbb{R}^{H \times W \times 6} \times \mathbb{R}^{H \times W \times D} \rightarrow \mathbb{R}^{H \times W \times 1}, \quad (\mathcal{Z}_s, \mathcal{Z}'_t, \mathcal{H}) \mapsto \Psi(\mathcal{Z}_s, \mathcal{Z}'_t, \mathcal{H}) = \mathcal{W}. \quad (6)$$

### 4.3 Differentiable Optimizer

We introduce a differentiable optimizer  $\Omega$  to estimate the deformation graph parameters  $\mathcal{T}$ , given the correspondence map  $\mathcal{C}$ , importance weights  $\mathcal{W}$ , and  $N$  graph nodes  $\mathcal{V}$ :

$$\Omega : \mathbb{R}^{H \times W \times 2} \times \mathbb{R}^{H \times W \times 1} \times \mathbb{R}^{N \times 3} \rightarrow \mathbb{R}^{N \times 6}, \quad (\mathcal{C}, \mathcal{W}, \mathcal{V}) \mapsto \Omega(\mathcal{C}, \mathcal{W}, \mathcal{V}) = \mathcal{T}, \quad (7)$$

with  $\mathcal{C}$  and  $\mathcal{W}$  estimated by functions  $\Phi$  (Eq. 5) and  $\Psi$  (Eq. 6), respectively. Using the predicted dense correspondence map  $\mathcal{C}$ , we establish the data term for the non-rigid tracking optimization. Specifically, we use a 2D data term that operates in image space and a depth data term that leverages the depth information of the input frames. In addition to the data terms, we employ an As-Rigid-As-Possible regularizer [30] to encourage node deformations to be locally rigid, enabling robust deformation estimates even in the presence of noisy input cues. Note that the resulting optimizer module  $\Omega$  is fully differentiable, but contains no learnable parameters. In summary, we formulate non-rigid tracking as the following nonlinear optimization problem:

$$\arg \min_{\mathcal{T}} (\lambda_{2D} E_{2D}(\mathcal{T}) + \lambda_{depth} E_{depth}(\mathcal{T}) + \lambda_{reg} E_{reg}(\mathcal{T})). \quad (8)$$

**2D reprojection term.** Given the outputs of the dense correspondence predictor and weighting function,  $\Phi(\mathcal{I}_s, \mathcal{I}_t)$  and  $\Psi(\mathcal{Z}_s, \mathcal{Z}'_t, \mathcal{H})$ , respectively, we query for every pixel  $\mathbf{u}$  in the source frame its correspondence  $\mathbf{c}_u$  and weight  $w_u$  to build the following energy term:

$$E_{2D}(\mathcal{T}) = \sum_{\mathbf{u} \in \Pi_s} w_{\mathbf{u}}^2 \|\pi_{\mathbf{c}}(\mathbf{Q}(\mathbf{p}_{\mathbf{u}}, \mathcal{T})) - \mathbf{c}_{\mathbf{u}}\|_2^2, \quad (9)$$

where  $\pi_{\mathbf{c}} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ ,  $\mathbf{p} \mapsto \pi_{\mathbf{c}}(\mathbf{p})$  is a perspective projection with intrinsic parameters  $\mathbf{c}$  and  $\mathbf{p}_{\mathbf{u}} = \pi_{\mathbf{c}}^{-1}(\mathbf{u}, d_{\mathbf{u}})$  as defined in Eq. 1. Each pixel is back-projected to 3D, deformed using the current graph motion estimate as described in Eq. 2 and projected onto the target image plane. The projected deformed location is compared to the predicted correspondence  $\mathbf{c}_{\mathbf{u}}$ .

**Depth term.** The depth term leverages the depth cues of the source and target images. Specifically, it compares the  $z$  components of a warped source point, i.e.,  $[\mathbf{Q}(\mathbf{p}_{\mathbf{u}}, \mathcal{T})]_z$ , and a target point sampled at the corresponding location  $\mathbf{c}_{\mathbf{u}}$  using bilinear interpolation:

$$E_{depth}(\mathcal{T}) = \sum_{\mathbf{u} \in \Pi_s} w_{\mathbf{u}}^2 ([\mathbf{Q}(\mathbf{p}_{\mathbf{u}}, \mathcal{T})]_z - [\mathbf{P}_t(\mathbf{c}_{\mathbf{u}})]_z)^2. \quad (10)$$

**Regularization term.** We encourage the deformation of neighboring nodes in the deformation graph to be locally rigid. Each node  $\mathbf{v}_i \in \mathcal{V}$  has at most  $K = 8$  neighbors in the set of edges  $\mathcal{E}$ , computed as nearest nodes using geodesic distances. The regularization term follows [30]:

$$E_{reg}(\mathcal{T}) = \sum_{(\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{E}} \left\| e^{\tilde{\omega}_{\mathbf{v}_i}} (\mathbf{v}_j - \mathbf{v}_i) + \mathbf{v}_i + \mathbf{t}_{\mathbf{v}_i} - (\mathbf{v}_j + \mathbf{t}_{\mathbf{v}_j}) \right\|_2^2. \quad (11)$$

Equation 8 is minimized using the Gauss-Newton algorithm, as described in Algorithm 1. In the following, we denote the number of correspondences by  $|\mathcal{C}|$  and the number of graph edges by  $|\mathcal{E}|$ . Moreover, we transform all energy terms into a residual vector  $\mathbf{r} \in \mathbb{R}^{3|\mathcal{C}|+3|\mathcal{E}|}$ . For every graph node, we compute partial derivatives with respect to translation and rotation parameters, constructing a Jacobian matrix  $\mathbf{J} \in \mathbb{R}^{(3|\mathcal{C}|+3|\mathcal{E}|) \times 6N}$ , where  $N$  is the number of nodes in the set of vertices  $\mathcal{V}$ . Analytic formulas for partial derivatives are described in the supplemental material.

Initially, the deformation parameters are initialized to  $\mathcal{T}_0 = \mathbf{0}$ , corresponding to zero translation and identity rotations. In each iteration  $n$ , the residual vector  $\mathbf{r}_n$  and the Jacobian matrix  $\mathbf{J}_n$  are computed using the current estimate  $\mathcal{T}_n$ , and the following linear system is solved (using LU decomposition) to compute an increment  $\Delta \mathcal{T}$ :

$$\mathbf{J}_n^T \mathbf{J}_n \Delta \mathcal{T} = -\mathbf{J}_n^T \mathbf{r}_n. \quad (12)$$

At the end of every iteration, the motion estimate  $\mathcal{T}$  is updated as  $\mathcal{T}_{n+1} = \mathcal{T}_n + \Delta \mathcal{T}$ . Most operations are matrix-matrix or matrix-vector multiplications, which are trivially differentiable. Derivatives of the linear system solve operation are computed analytically, as described in [2] and detailed in the supplement. We use  $max\_iter = 3$  Gauss-Newton iterations, which encourages the correspondence prediction and weight functions,  $\Phi$  and  $\Psi$ , respectively, to make predictions such that convergence in 3 iterations is possible. In our experiments we use  $(\lambda_{2D}, \lambda_{depth}, \lambda_{reg}) = (0.001, 1, 1)$ .

---

**Algorithm 1** Gauss-Newton Optimization

---

```

1:  $\mathcal{C} \leftarrow \Phi(\mathcal{I}_s, \mathcal{I}_t)$                                  $\triangleright$  Estimate correspondences
2:  $\mathcal{W} \leftarrow \Psi(\mathcal{Z}_s, \mathcal{Z}'_t, \mathcal{H})$                        $\triangleright$  Estimate importance weights
3: function SOLVER( $\mathcal{C}, \mathcal{W}, \mathcal{V}$ )
4:    $\mathcal{T} \leftarrow \mathbf{0}$ 
5:   for  $n \leftarrow 0$  to  $max\_iter$  do
6:      $\mathbf{J}, \mathbf{r} \leftarrow \text{ComputeJacobianAndResidual}(\mathcal{V}, \mathcal{T}, \mathcal{Z}_s, \mathcal{Z}'_t, \mathcal{C}, \mathcal{W})$ 
7:      $\Delta\mathcal{T} \leftarrow \text{LUDecomposition}(\mathbf{J}^T \mathbf{J} \Delta\mathcal{T} = -\mathbf{J}^T \mathbf{r})$            $\triangleright$  Solve linear system
8:      $\mathcal{T} \leftarrow \mathcal{T} + \Delta\mathcal{T}$                                       $\triangleright$  Apply increment
9:   return  $\mathcal{T}$ 

```

---

#### 4.4 End-to-end Optimization

Given a dataset of samples  $\mathcal{X}_{s,t} = \{[\mathcal{I}_s|\mathcal{P}_s], [\mathcal{I}_t|\mathcal{P}_t], \mathcal{V}\}$ , our goal is to find the parameters  $\phi$  and  $\psi$  of  $\Phi_\phi$  and  $\Psi_\psi$ , respectively, so as to estimate the motion  $\mathcal{T}$  of a deformation graph  $\mathcal{G}$  defined over the source RGB-D frame. This can be formulated as a differentiable optimization problem (allowing for back-propagation) with the following objective:

$$\arg \min_{\phi, \psi} \sum_{\mathcal{X}_{s,t}} \lambda_{\text{corr}} \mathcal{L}_{\text{corr}}(\phi) + \lambda_{\text{graph}} \mathcal{L}_{\text{graph}}(\phi, \psi) + \lambda_{\text{warp}} \mathcal{L}_{\text{warp}}(\phi, \psi) \quad (13)$$

**Correspondence loss.** We use a robust  $q$ -norm as in [32] to enforce closeness of correspondence predictions to ground-truth:

$$\mathcal{L}_{\text{corr}}(\phi) = \tilde{M}^{\mathcal{C}}(|\Phi_\phi(\mathcal{I}_s, \mathcal{I}_t) - \tilde{\mathcal{C}}| + \epsilon)^q. \quad (14)$$

Operator  $|\cdot|$  denotes the  $\ell_1$  norm,  $q < 1$  (we set it to  $q = 0.4$ ) and  $\epsilon$  is a small constant. Ground-truth correspondences are denoted by  $\tilde{\mathcal{C}}$ . Since valid ground truth for all pixels is not available, we employ a ground-truth mask  $\tilde{M}^{\mathcal{C}}$  to avoid propagating gradients through invalid pixels.

**Graph loss.** We impose an  $l_2$ -loss on node translations  $\mathbf{t}$  (ground-truth rotations are not available):

$$\mathcal{L}_{\text{graph}}(\phi, \psi) = \tilde{M}^{\mathcal{V}} \left\| \underbrace{[\Omega(\Phi_\phi(\mathcal{I}_s, \mathcal{I}_t), \Psi_\psi(\mathcal{Z}_s, \mathcal{Z}'_t, \mathcal{H}), \mathcal{V})]}_{\mathcal{T}} \right\|_2 - \tilde{\mathbf{t}} \right\|_2^2, \quad (15)$$

where  $[\cdot]_{\mathbf{t}} : \mathbb{R}^{N \times 6} \rightarrow \mathbb{R}^{N \times 3}$ ,  $\mathcal{T} \mapsto [\mathcal{T}]_{\mathbf{t}} = \mathbf{t}$  extracts the translation part from the graph motion  $\mathcal{T}$ . Node translation ground-truth is denoted by  $\tilde{\mathbf{t}}$  and  $\tilde{M}^{\mathcal{V}}$  masks out invalid nodes. Please see the supplement for further details on how  $\tilde{M}^{\mathcal{V}}$  is computed.

**Warp loss.** We have found that it is beneficial to use the estimated graph deformation  $\mathcal{T}$  to deform the dense source point cloud  $\mathcal{P}_s$  and enforce the result to be close to the source point cloud when deformed with the ground-truth scene flow  $\tilde{\mathcal{S}}$ :

$$\mathcal{L}_{\text{warp}}(\phi, \psi) = \tilde{M}^{\mathcal{S}} \left\| \mathbf{Q} \left( \mathcal{P}_s, \underbrace{\Omega(\Phi_\phi(\mathcal{I}_s, \mathcal{I}_t), \Psi_\psi(\mathcal{Z}_s, \mathcal{Z}'_t, \mathcal{H}), \mathcal{V})}_{\mathcal{T}} \right) - (\mathcal{P}_s + \tilde{\mathcal{S}}) \right\|_2^2. \quad (16)$$

Here, we extend the warping operation  $\mathbf{Q}$  (Eq. 2) to operate on the dense point cloud  $\mathcal{P}_s$  element-wise, and define  $\tilde{M}^{\mathcal{S}}$  to mask out invalid points.

Note that We found it to be a more general notation to disentangle them (e.g., for scenarios where graph nodes are not sampled on the RGB-D frame).

#### 4.5 Neural Non-rigid Tracking for 3D Reconstruction

We introduce our differentiable tracking module into the non-rigid reconstruction framework of Newcombe et al. [25]. In addition to the dense depth ICP correspondences employed in the original method, which help towards local deformation refinement, we employ a keyframe-based tracking objective. Without loss of generality, every 50th frame of the sequence is chosen as a keyframe, to

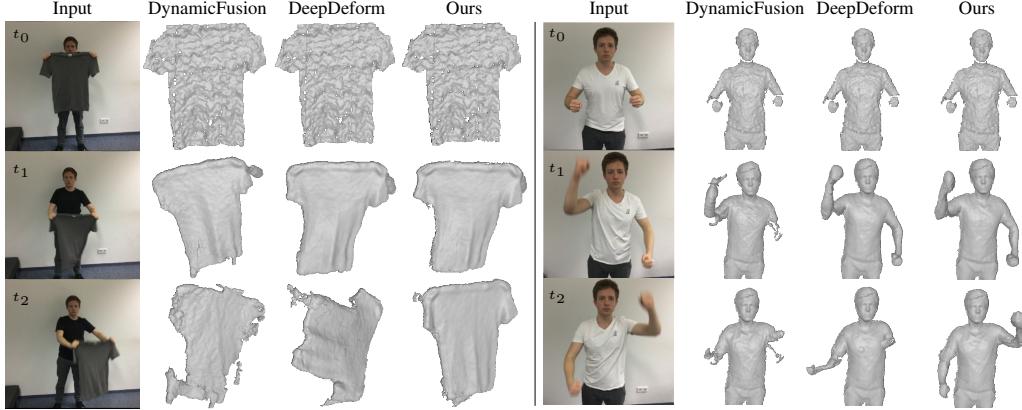


Figure 3: Qualitative comparison of our method with DynamicFusion [25] and DeepDeform [5] on test sequences from [5]. The rows show different time steps of the sequence.

which we establish dense correspondences including the respective weights. We apply a conservative filtering of the predicted correspondences based on the predicted correspondence weights using a fixed threshold  $\delta = 0.35$  and re-weight the correspondences based on bi-directional consistency, i.e., keyframe-to-frame and frame-to-keyframe. Using the correspondence predictions and correspondence weights of valid keyframes ( $> 50\%$  valid correspondences), the non-rigid tracking optimization problem is solved. The resulting deformation field is used to integrate the depth frame into the canonical volume of the object. We refer to the original reconstruction paper [25] for details regarding the fusion process.

## 5 Experiments

In the following, we evaluate our method quantitatively and qualitatively on both non-rigid tracking and non-rigid reconstruction. To this end, we use the DeepDeform dataset [5] for training, with the given 340-30-30 train-val-test split of RGB-D sequences. Both non-rigid tracking and reconstruction are evaluated on the hidden test set of the DeepDeform benchmark.

### 5.1 Training Scheme

The non-rigid tracking module has been implemented using the PyTorch library [26] and trained using stochastic gradient descent with momentum 0.9 and learning rate  $10^{-5}$ . We use an Intel Xeon 6240 Processor and an Nvidia RTX 2080Ti GPU. The parameters of the dense correspondence prediction network  $\phi$  are initialized with a PWC-Net model pre-trained on FlyingChairs [8] and FlyingThings3D [24]. We use a 10-factor learning rate decay every 10k iterations, requiring

Table 1: We evaluate non-rigid tracking on the DeepDeform dataset [5], showing the benefit of end-to-end differentiable optimizer losses and self-supervised correspondence weighting. We denote correspondence prediction as  $\Phi_c$ ,  $\Phi_{c+g}$  and  $\Phi_{c+g+w}$ , depending on which losses  $\mathcal{L}_{corr}$ ,  $\mathcal{L}_{graph}$ ,  $\mathcal{L}_{warp}$  are used, and correspondence weighting as  $\Psi_{supervised}$  and  $\Psi_{self-supervised}$ , either using an additional supervised loss or not.

Model	EPE 3D (mm)	Graph Error 3D (mm)
$\Phi_c$	44.05	67.25
$\Phi_{c+g}$	39.12	57.34
$\Phi_{c+g+w}$	36.96	54.24
$\Phi_c + \Psi_{supervised}$	28.95	36.77
$\Phi_{c+g+w} + \Psi_{supervised}$	27.42	34.68
$\Phi_{c+g+w} + \Psi_{self-supervised}$	<b>26.29</b>	<b>31.00</b>

about 30k iterations in total for convergence, with a batch size of 4. For optimal performance, we first optimize the correspondence predictor  $\Phi_\phi$  with  $(\lambda_{\text{corr}}, \lambda_{\text{graph}}, \lambda_{\text{warp}}) = (5, 5, 5)$ , without the weighting function  $\Psi_\psi$ . Afterwards, we optimize the weighting function parameters  $\psi$  with  $(\lambda_{\text{corr}}, \lambda_{\text{graph}}, \lambda_{\text{warp}}) = (0, 1000, 1000)$ , while keeping  $\phi$  fixed. Finally, we fine-tune both  $\phi$  and  $\psi$  together, with  $(\lambda_{\text{corr}}, \lambda_{\text{graph}}, \lambda_{\text{warp}}) = (5, 5, 5)$ .

## 5.2 Non-rigid Tracking Evaluation

For any frame pair  $\mathcal{X}_{s,t}$  in the DeepDeform data [5], we define a deformation graph  $\mathcal{G}$  by uniformly sampling graph nodes  $\mathcal{V}$  over the source object in the RGB-D frame, given a segmentation mask of the former. Graph node connectivity  $\mathcal{E}$  is computed using geodesic distances on a triangular mesh defined over the source depth map. As a pre-processing step, we filter out any frame pairs where more than 30% of the source object is occluded in the target frame. In Table 1 non-rigid tracking performance is evaluated by the mean translation error over node translations  $\mathbf{t}$  (Graph Error 3D), where the latter are compared to ground-truth with an  $l_2$  metric. In addition, we evaluate the dense end-point-error (EPE 3D) between the source point cloud deformed with the estimated graph motion,  $Q(\mathcal{P}_s, \mathcal{T})$ , and the source point cloud deformed with the ground-truth scene flow,  $\mathcal{P}_s + \tilde{\mathcal{S}}$ . To support reproducibility, we report the mean error metrics of multiple experiments, running every setting 3 times. We visualize the standard deviation with an error plot in the supplement.

We show that using graph and warp losses,  $\mathcal{L}_{\text{graph}}$  and  $\mathcal{L}_{\text{warp}}$ , and differentiating through the non-rigid optimizer considerably improves both EPE 3D and Graph Error 3D compared to only using the correspondence loss  $\mathcal{L}_{\text{corr}}$ . Adding self-supervised correspondence weighting further decreases the errors by a large margin. Supervised outlier rejection with binary cross-entropy loss does bring an improvement compared to models that do not optimize for the weighting function  $\Psi_\psi$  (please see supplemental material for details on this supervised training of  $\Psi_\psi$ ). However, optimizing  $\Psi_\psi$  in a *self-supervised* manner clearly outperforms the former supervised setup. This is due to the fact that, in the self-supervised scenario, gradients that flow from  $\mathcal{L}_{\text{graph}}$  and  $\mathcal{L}_{\text{warp}}$  through the differentiable solver  $\Omega$  can better inform the optimization of  $\Psi_\psi$  by minimizing the end-to-end alignment losses.

## 5.3 Non-rigid Reconstruction Evaluation

We evaluate the performance of our non-rigid reconstruction approach on the DeepDeform benchmark [5] (see Table 2). The evaluation metrics measure *deformation error*, a 3D end-point-error between tracked and annotated correspondences, and *geometry error*, which compares reconstructed shapes with annotated foreground object masks. Our approach performs about 8.9% better than the state-of-the-art non-rigid reconstruction approach of Božič et al. [5] on the deformation metric. While our approach consistently shows better performance on both metrics, we also significantly lower the per-frame runtime to 27 ms per keyframe, in contrast to [5], which requires 2299 ms. Thus, our approach can also be used with multiple keyframes at interactive frames rates, e.g., 90 ms for 5 keyframes and 199 ms for 10 keyframes.

Table 2: Our method achieves state-of-the-art non-rigid reconstruction results on the DeepDeform benchmark [5]. Both our end-to-end differentiable optimizer and the self-supervised correspondence weighting are necessary for optimal performance. Not only does our approach achieve lower deformation and geometry error compared to state of the art, our correspondence prediction is about 85 $\times$  faster.

Method	Deformation error (mm)	Geometry error (mm)
DynamicFusion [25]	61.79	10.78
VolumeDeform [15]	208.41	74.85
DeepDeform [5]	31.52	4.16
Ours ( $\Phi_c$ )	54.85	5.92
Ours ( $\Phi_{c+g+w}$ )	53.27	5.84
Ours ( $\Phi_c + \Psi_{\text{supervised}}$ )	40.21	5.39
Ours ( $\Phi_{c+g+w} + \Psi_{\text{self-supervised}}$ )	<b>28.72</b>	<b>4.03</b>

To show the influence of the different learned components of our method, we perform an ablation study by disabling either of our two main components: the end-to-end differentiable optimizer or the self-supervised correspondence weighting. As can be seen, our end-to-end trained method with self-supervised correspondence weighting demonstrates the best performance. Qualitatively, we show this in Figure 3. In contrast to DynamicFusion [25] and DeepDeform [5], our method is notably more robust in fast motion scenarios. Additional qualitative results and comparisons to the methods of Guo et al. [12] and Slavcheva et al. [28] are shown in the supplemental material.

## 6 Conclusion

We propose Neural Non-Rigid Tracking, a differentiable non-rigid tracking approach that allows learning the correspondence prediction and weighting of traditional tracking pipelines in an end-to-end manner. The differentiable formulation of the entire tracking pipeline enables back-propagation to the learnable components, guided by a loss on the tracking performance. This not only achieves notably improved tracking error in comparison to state-of-the-art tracking approaches, but also leads to better reconstructions, when integrated into a reconstruction framework like DynamicFusion [25]. We hope that this work inspires further research in the direction of neural non-rigid tracking and believe that it is a stepping stone towards fully differentiable non-rigid reconstruction.

## Broader Impact

Our paper presents learned non-rigid tracking. It is establishing the basis for the important research field of non-rigid tracking and reconstruction, which is needed for a variety of applications where man-machine and machine-environment interaction is required. These applications range from the field of augmented and virtual reality to autonomous driving and robot control. In the former, a precise understanding of dynamic and deformable objects is of major importance in order to provide an immersive experience to the user. Applications such as holographic calls would greatly benefit from research like ours. This, in turn, could provide society with the next generation of 3D communication tools. On the other hand, as a low-level building block, our work has no direct negative outcome, other than what could arise from the aforementioned applications.

## Acknowledgments and Disclosure of Funding

This work was supported by the ZD.B (Zentrum Digitalisierung.Bayern), the Max Planck Center for Visual Computing and Communications (MPC-VCC), a TUM-IAS Rudolf Mößbauer Fellowship, the ERC Starting Grant Scan2CAD (804724), and the German Research Foundation (DFG) Grant Making Machine Learning on Static and Dynamic 3D Data Practical.

## References

- [1] A. Avetisyan, A. Dai, and M. Nießner. End-to-end cad model retrieval and 9dof alignment in 3d scans. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2551–2560, 2019.
- [2] J. T. Barron and B. Poole. The fast bilateral solver. In *European Conference on Computer Vision (ECCV)*, pages 617–632. Springer, 2016.
- [3] A. Behl, D. Paschalidou, S. Donné, and A. Geiger. Pointflownet: Learning representations for rigid motion estimation from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7962–7971, 2019.
- [4] J.-L. Blanco. A tutorial on se (3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, 3, 2010.
- [5] A. Božić, M. Zollhöfer, C. Theobalt, and M. Nießner. Deepdeform: Learning non-rigid rgb-d reconstruction with semi-supervised data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [6] C.-H. Chang, C.-N. Chou, and E. Y. Chang. Clkn: Cascaded lucas-kanade networks for image alignment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2213–2221, 2017.
- [7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM Symposium on Theory of Computing*, pages 1–6, 1987.

- [8] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015.
- [9] M. Dou, J. Taylor, H. Fuchs, A. Fitzgibbon, and S. Izadi. 3d scanning deformable objects with a single rgbd sensor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 493–501, 2015.
- [10] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. O. Escalano, C. Rhemann, D. Kim, J. Taylor, et al. Fusion4d: Real-time performance capture of challenging scenes. *ACM Transactions on Graphics (TOG)*, 35(4):114, 2016.
- [11] M. Götz and H. Anzt. Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation. In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*, pages 49–56, 2018.
- [12] K. Guo, F. Xu, T. Yu, X. Liu, Q. Dai, and Y. Liu. Real-time geometry, albedo, and motion reconstruction using a single rgbd camera. *ACM Transactions on Graphics (TOG)*, 36(3):32, 2017.
- [13] L. Han, M. Ji, L. Fang, and M. Nießner. Regnet: Learning the optimization of direct image-to-image pose registration. *arXiv preprint arXiv:1812.10212*, 2018.
- [14] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2462–2470, 2017.
- [15] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger. Volumedeform: Real-time volumetric non-rigid reconstruction. In *European Conference on Computer Vision*, pages 362–379. Springer, 2016.
- [16] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems (NeurIPS) 28*, pages 2017–2025, 2015.
- [17] Z. Lai, E. Lu, and W. Xie. Mast: A memory-augmented self-supervised tracker. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6479–6488, 2020.
- [18] X. Li, S. Liu, S. De Mello, X. Wang, J. Kautz, and M.-H. Yang. Joint-task self-supervised learning for temporal correspondence. In *Advances in Neural Information Processing Systems*, pages 318–328, 2019.
- [19] Y. Li, A. Božič, T. Zhang, Y. Ji, T. Harada, and M. Nießner. Learning to optimize non-rigid tracking. *arXiv preprint arXiv:2003.12230*, 2020.
- [20] P. Liu, M. Lyu, I. King, and J. Xu. Selfflow: Self-supervised learning of optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4571–4580, 2019.
- [21] X. Liu, C. R. Qi, and L. J. Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 529–537, 2019.
- [22] Z. Lv, F. Dellaert, J. M. Rehg, and A. Geiger. Taking a deeper look at the inverse compositional algorithm. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4581–4590, 2019.
- [23] W.-C. Ma, S. Wang, R. Hu, Y. Xiong, and R. Urtasun. Deep rigid instance scene flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3614–3622, 2019.
- [24] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 2016.
- [25] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, 2015.
- [26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS) 32*, pages 8024–8035, 2019.
- [27] J. Sappl, L. Seiler, M. Harders, and W. Rauch. Deep learning of preconditioners for conjugate gradient solvers in urban water related problems. *arXiv preprint arXiv:1906.06925*, 2019.
- [28] M. Slavcheva, M. Baust, D. Cremers, and S. Ilic. Killingfusion: Non-rigid 3d reconstruction without correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1386–1395, 2017.
- [29] M. Slavcheva, M. Baust, and S. Ilic. Sobolevfusion: 3d reconstruction of scenes undergoing free non-rigid motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2646–2655, 2018.

- [30] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, pages 109–116, 2007.
- [31] R. W. Sumner, J. Schmid, and M. Pauly. Embedded deformation for shape manipulation. *ACM Transactions on Graphics (TOG)*, 26(3):80–es, 2007.
- [32] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8934–8943, 2018.
- [33] C. Tang and P. Tan. Ba-net: Dense bundle adjustment network. *arXiv preprint arXiv:1806.04807*, 2018.
- [34] X. Wang, A. Jabri, and A. A. Efros. Learning correspondence from the cycle-consistency of time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2566–2576, 2019.
- [35] Z. Wang, S. Li, H. Howard-Jenkins, V. Prisacariu, and M. Chen. Flownet3d++: Geometric losses for deep scene flow estimation. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 91–98, 2020.
- [36] T. Yu, K. Guo, F. Xu, Y. Dong, Z. Su, J. Zhao, J. Li, Q. Dai, and Y. Liu. Bodyfusion: Real-time capture of human motion and surface geometry using a single depth camera. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 910–919, 2017.
- [37] T. Yu, Z. Zheng, K. Guo, J. Zhao, Q. Dai, H. Li, G. Pons-Moll, and Y. Liu. Doublefusion: Real-time capture of human performances with inner body shapes from a single depth sensor. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition (CVPR)*, pages 7287–7296, 2018.
- [38] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt, et al. Real-time non-rigid reconstruction using an rgb-d camera. *ACM Transactions on Graphics (TOG)*, 33(4):1–12, 2014.

## A Non-rigid Deformation Model

To represent the dense motion from a source to a target RGB-D frame, we adapt the embedded deformation model of Sumner et al. [31]. We uniformly sample graph nodes  $\mathcal{V}$  over the source RGB-D frame (see Fig. 4), ensuring  $\sigma$ -coverage of the foreground object, i.e., the distance of every foreground point to the nearest graph node is at most  $\sigma > 0$  (we set  $\sigma = 0.05$  m).

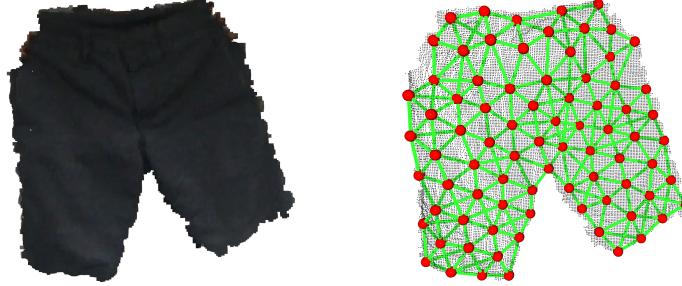


Figure 4: Given a an object in the source RGB-D frame, we define a deformation graph  $\mathcal{G}$  over the former. Nodes  $\mathcal{V}$  (red spheres) are uniformly subsampled over the source RGB-D frame. Edges  $\mathcal{E}$  (green lines) are computed between nodes based on geodesic connectivity among the latter.

For every node  $v_i \in \mathcal{V}$ , we estimate its global translation vector  $\mathbf{t}_{v_i} \in \mathbb{R}^3$  and local rotation matrix  $\mathbf{R}_{v_i} \in \mathbb{R}^{3 \times 3}$ , represented in axis-angle notation as  $\omega_{v_i} \in \mathbb{R}^3$ . Using the deformation parameters  $\mathcal{T} = (\omega_{v_1}, \mathbf{t}_{v_1}, \dots, \omega_{v_N}, \mathbf{t}_{v_N})$  a 3D point  $\mathbf{p} \in \mathbb{R}^3$  is deformed by interpolating the nodes' motion:

$$Q(\mathbf{p}, \mathcal{T}) = \sum_{v_i \in \mathcal{V}} \alpha_{v_i}^{\mathbf{p}} (e^{\hat{\omega}_{v_i}} (\mathbf{p} - \mathbf{v}_i) + \mathbf{v}_i + \mathbf{t}_{v_i}). \quad (17)$$

The weights  $\alpha_{v_i}^{\mathbf{p}} \in \mathbb{R}$  are called *skinning* weights and measure the influence of each node on the current point  $\mathbf{p}$ . They are computed as in DoubleFusion [37]:

$$\alpha_{v_i}^{\mathbf{p}} = C e^{\frac{1}{2\sigma^2} \|\mathbf{v}_i - \mathbf{p}\|_2^2}.$$

Here,  $C$  denotes the normalization constant, ensuring that skinning weights add up to one for point  $\mathbf{p}$ :

$$\sum_{v_i \in \mathcal{V}} \alpha_{v_i}^{\mathbf{p}} = 1.$$

For each node  $v_i \in \mathcal{V}$ , we represent its rotation in axis-angle notation as  $\omega_{v_i} \in \mathbb{R}^3$ . This representation has singularities for larger angles, i.e., two different vectors  $\omega$  and  $\omega'$  can represent the same rotation (for example keeping the same axis and increasing the angle by  $2\pi$  results in identical rotation). To avoid singularities, we decompose the rotation matrix into  $e^{\hat{\omega}_{v_i}} = e^{\hat{\epsilon}_{v_i}} \mathbf{R}_{v_i}$  with  $\epsilon_{v_i} = 0$ , therefore optimizing only for delta rotations that have rather small rotation angles.

## B Differentiable Non-rigid Optimization

Our non-rigid optimization is based on the Gauss-Newton algorithm and minimizes an energy formulation that is based on three types of residual components: the 2D reprojection term, the depth term and the regularization term of the non-rigid deformation.

For a pixel  $\mathbf{u} \in \Pi_s \subset \mathbb{R}^2$  and graph edge  $(v_i, v_j) \in \mathcal{E}$ , we define such terms as:

$$\begin{aligned} r_{2D}^{\mathbf{u}}(\mathcal{T}) &= w_{\mathbf{u}}(\pi_c(Q(\mathbf{p}_{\mathbf{u}}, \mathcal{T})) - \mathbf{c}_{\mathbf{u}}) \\ r_{\text{depth}}^{\mathbf{u}}(\mathcal{T}) &= w_{\mathbf{u}}([Q(\mathbf{p}_{\mathbf{u}}, \mathcal{T})]_z - [P_t(\mathbf{c}_{\mathbf{u}})]_z) \\ r_{\text{reg}}^{v_i, v_j}(\mathcal{T}) &= e^{\hat{\omega}_{v_i}} (\mathbf{v}_j - \mathbf{v}_i) + \mathbf{v}_i + \mathbf{t}_{v_i} - (\mathbf{v}_j + \mathbf{t}_{v_j}), \end{aligned}$$

where  $\mathbf{c}_\mathbf{u} \in \mathbb{R}^2$  and  $w_\mathbf{u} \in \mathbb{R}$  represent the predicted correspondence and the importance weight, respectively; and  $\mathbf{p}_\mathbf{u} = \pi_c^{-1}(\mathbf{u}, d_\mathbf{u})$  is a 3D point corresponding to the pixel  $\mathbf{u}$  with depth value  $d_\mathbf{u}$ .

The Gauss-Newton method is an iterative scheme. In every iteration  $n$ , we compute the Jacobian matrix  $\mathbf{J}_n$  and the residual vector  $\mathbf{r}_n$ , and get a solution increment  $\Delta \mathcal{T}$  by solving the normal equations:

$$\mathbf{J}_n^T \mathbf{J}_n \Delta \mathcal{T} = -\mathbf{J}_n^T \mathbf{r}_n.$$

The construction of the Jacobian matrix  $\mathbf{J} \in \mathbb{R}^{(3|\mathcal{C}|+3|\mathcal{E}|) \times 6N}$ , consisting of partial derivatives of the residual vector  $\mathbf{r}^{3|\mathcal{C}|+3|\mathcal{E}|}$  with respect to deformation parameters  $\mathcal{T} = (\omega_{\mathbf{v}_1}, \mathbf{t}_{\mathbf{v}_1}, \dots, \omega_{\mathbf{v}_N}, \mathbf{t}_{\mathbf{v}_N}) \in \mathbb{R}^{6N}$  is detailed in Section B.1. The linear system is solved using LU decomposition. To enable differentiation through the entire Gauss-Newton solver, we have to ensure that the linear solve is differentiable. We detail the differentiable linear solve operation in Section B.2.

## B.1 Partial Derivatives

In the following, we derive the partial derivatives of the residual vector  $\mathbf{r}$  with respect to  $\epsilon_{\mathbf{v}_i}$  and  $\mathbf{t}_{\mathbf{v}_i}$  of every node  $\mathbf{v}_i$ , to construct the Jacobian matrix  $\mathbf{J}$ .

To simplify notation, we define the rotation operator that takes as input an angular velocity vector  $\epsilon \in \mathbb{R}^3$ , rotation matrix  $\mathbf{R} \in \mathbb{R}^{3 \times 3}$  and point  $\mathbf{p} \in \mathbb{R}^3$  and outputs the rotated point:

$$\mathbf{R}(\epsilon, \mathbf{R}, \mathbf{p}) = e^{\hat{\epsilon}} \mathbf{R} \mathbf{p}.$$

To compute the partial derivative with respect to  $\epsilon$ , we follow the derivation from Blanco [4]:

$$\frac{\partial \mathbf{R}(\epsilon, \mathbf{R}, \mathbf{p})}{\partial \epsilon} \Big|_{\epsilon=0} = -\widehat{\mathbf{R}} \mathbf{p}$$

Here, the  $\widehat{\cdot}$ -operator creates a  $3 \times 3$  skew-symmetric matrix from a 3-dimensional vector.

The rotation operator  $\mathbf{R}(\epsilon, \mathbf{R}, \mathbf{p})$  is a core part of the warping operator  $\mathbf{Q}(\mathbf{p}, \mathcal{T})$ . It follows that partial derivatives of a warping operator  $\mathbf{Q}(\mathbf{p}, \mathcal{T})$  with respect to  $\epsilon_{\mathbf{v}_i}$  and  $\mathbf{t}_{\mathbf{v}_i}$  for every node  $\mathbf{v}_i$  can be computed as

$$\begin{aligned} \frac{\partial \mathbf{Q}(\mathbf{p}, \mathcal{T})}{\partial \epsilon_{\mathbf{v}_i}} &= -\alpha_{\mathbf{v}_i}^{\mathbf{p}} \widehat{\mathbf{R}}_{\mathbf{v}_i}(\mathbf{p} - \mathbf{v}_i), \\ \frac{\partial \mathbf{Q}(\mathbf{p}, \mathcal{T})}{\partial \mathbf{t}_{\mathbf{v}_i}} &= \alpha_{\mathbf{v}_i}^{\mathbf{p}} \mathbf{I}. \end{aligned}$$

Another building block of our optimization terms is the perspective projection  $\pi_c$  with intrinsic parameters  $\mathbf{c} = (f_x, f_y, c_x, c_y)$ :

$$\begin{aligned} \pi_c : \mathbb{R}^3 &\rightarrow \mathbb{R}^2, \\ \pi_c \left( \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right) &= \begin{bmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \end{bmatrix}, \end{aligned}$$

whose partial derivatives with respect to the point  $\mathbf{p} = (x, y, z)^T$  are derived as

$$\frac{\partial \pi_c(\mathbf{p})}{\partial \mathbf{p}} = \begin{bmatrix} \frac{f_x}{z} & 0 & -\frac{f_x x}{z^2} \\ 0 & \frac{f_y}{z} & -\frac{f_y y}{z^2} \end{bmatrix}.$$

By applying the chain rule, derivatives of all three optimization terms are computed.

**Derivative of 2D reprojection term.** For a pixel  $\mathbf{u} \in \Pi_s \subset \mathbb{R}^2$  and its corresponding 3D point  $\mathbf{p}_\mathbf{u}$ , we derive partial derivatives of  $r_{2D}^\mathbf{u}(\mathcal{T})$  as follows:

$$\begin{aligned}\frac{\partial r_{2D}^\mathbf{u}(\mathcal{T})}{\partial \epsilon_{\mathbf{v}_i}} &= -w_\mathbf{u} \alpha_{\mathbf{v}_i}^{\mathbf{p}_\mathbf{u}} \begin{bmatrix} \frac{f_x}{\mathbf{p}_\mathbf{u}^z} & 0 & -\frac{f_x \mathbf{p}_\mathbf{u}^x}{(\mathbf{p}_\mathbf{u}^z)^2} \\ 0 & \frac{f_y}{\mathbf{p}_\mathbf{u}^z} & -\frac{f_y \mathbf{p}_\mathbf{u}^y}{(\mathbf{p}_\mathbf{u}^z)^2} \end{bmatrix} \widehat{\mathbf{R}_{\mathbf{v}_i}(\mathbf{p}_\mathbf{u} - \mathbf{v}_i)}, \\ \frac{\partial r_{2D}^\mathbf{u}(\mathcal{T})}{\partial \mathbf{t}_{\mathbf{v}_i}} &= w_\mathbf{u} \alpha_{\mathbf{v}_i}^{\mathbf{p}_\mathbf{u}} \begin{bmatrix} \frac{f_x}{\mathbf{p}_\mathbf{u}^z} & 0 & -\frac{f_x \mathbf{p}_\mathbf{u}^x}{(\mathbf{p}_\mathbf{u}^z)^2} \\ 0 & \frac{f_y}{\mathbf{p}_\mathbf{u}^z} & -\frac{f_y \mathbf{p}_\mathbf{u}^y}{(\mathbf{p}_\mathbf{u}^z)^2} \end{bmatrix}.\end{aligned}$$

**Derivative of depth term.** When computing the partial derivatives of the depth term  $r_{\text{depth}}^\mathbf{u}(\mathcal{T})$ , we need to additionally apply the projection to the  $z$ -component in the chain rule:

$$\begin{aligned}\frac{\partial r_{\text{depth}}^\mathbf{u}(\mathcal{T})}{\partial \epsilon_{\mathbf{v}_i}} &= -w_\mathbf{u} \alpha_{\mathbf{v}_i}^{\mathbf{p}_\mathbf{u}} [0 \quad 0 \quad 1] \widehat{\mathbf{R}_{\mathbf{v}_i}(\mathbf{p}_\mathbf{u} - \mathbf{v}_i)}, \\ \frac{\partial r_{\text{depth}}^\mathbf{u}(\mathcal{T})}{\partial \mathbf{t}_{\mathbf{v}_i}} &= w_\mathbf{u} \alpha_{\mathbf{v}_i}^{\mathbf{p}_\mathbf{u}} [0 \quad 0 \quad 1].\end{aligned}$$

**Derivative of regularization term.** For a graph edge  $(\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{E}$ , the partial derivatives of  $r_{\text{reg}}^{\mathbf{v}_i, \mathbf{v}_j}(\mathcal{T})$  with respect to  $\epsilon_{\mathbf{v}_i}, \mathbf{t}_{\mathbf{v}_i}, \epsilon_{\mathbf{v}_j}, \mathbf{t}_{\mathbf{v}_j}$  are computed as:

$$\begin{aligned}\frac{\partial r_{\text{reg}}^{\mathbf{v}_i, \mathbf{v}_j}(\mathcal{T})}{\partial \epsilon_{\mathbf{v}_i}} &= -\widehat{\mathbf{R}_{\mathbf{v}_i}(\mathbf{v}_j - \mathbf{v}_i)}, & \frac{\partial r_{\text{reg}}^{\mathbf{v}_i, \mathbf{v}_j}(\mathcal{T})}{\partial \epsilon_{\mathbf{v}_j}} &= \mathbf{0}, \\ \frac{\partial r_{\text{reg}}^{\mathbf{v}_i, \mathbf{v}_j}(\mathcal{T})}{\partial \mathbf{t}_{\mathbf{v}_i}} &= \mathbf{I}, & \frac{\partial r_{\text{reg}}^{\mathbf{v}_i, \mathbf{v}_j}(\mathcal{T})}{\partial \mathbf{t}_{\mathbf{v}_j}} &= -\mathbf{I}.\end{aligned}$$

## B.2 Differentiable Linear Solve Operation

To simplify the notation, in the following we use  $\mathbf{A} = \mathbf{J}_n^T \mathbf{J}_n$ ,  $\mathbf{b} = -\mathbf{J}_n^T \mathbf{r}_n$  and  $\mathbf{x} = \Delta \mathcal{T}$ , which results in the linear system of the form

$$\mathbf{Ax} = \mathbf{b}. \quad (18)$$

For matrix  $\mathbf{A} \in \mathbb{R}^{6N \times 6N}$  and vectors  $\mathbf{b} \in \mathbb{R}^{6N}$  and  $\mathbf{x} \in \mathbb{R}^{6N}$  we define the linear solve operation as

$$\Lambda : \mathbb{R}^{6N \times 6N} \times \mathbb{R}^{6N} \rightarrow \mathbb{R}^{6N}, \quad (\mathbf{A}, \mathbf{b}) \mapsto \mathbf{A}^{-1} \mathbf{b} = \mathbf{x}. \quad (19)$$

In order to compute the derivative of the linear solve operation, we follow the analytic derivative formulation of Barron and Poole [2]. If we denote the partial derivative of the loss  $\mathcal{L}$  with respect to linear system solution  $\mathbf{x}$  as  $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$ , we can compute the partial derivatives with respect to matrix  $\mathbf{A}$  and vector  $\mathbf{b}$  as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \mathbf{A}^{-1} \frac{\partial \mathcal{L}}{\partial \mathbf{x}}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \left( \mathbf{A}^{-1} \frac{\partial \mathcal{L}}{\partial \mathbf{x}} \right) \mathbf{x}^T = -\frac{\partial \mathcal{L}}{\partial \mathbf{b}} \mathbf{x}^T. \quad (20)$$

Thus, the computation of  $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$  requires solving a linear system with matrix  $A$ . To solve this system, we re-use the LU decomposition from the forward pass.

## B.3 Ablations

We experimented with different design choices for our solver.

**ARAP edge re-weighting.** In non-rigid tracking, it is possible to weight ARAP terms for every graph edge differently, depending on the distance between the nodes. In our method, we sample nodes uniformly on the surface, thus, all edges have similar length ( $7.13 \pm 1.38$  cm). Hence, edge re-weighting changes EPE 3D only marginally: 0.8% lower EPE 3D and 1.7% lower Graph Error 3D.

**Nearest-neighbor vs. bilinear depth sampling.** When querying depth after predicting 2D correspondences, we found bilinear sampling to perform better, with 5.8% lower EPE 3D and 6.29% lower Graph Error 3D compared to nearest-neighbor sampling.

**Influence of graph density.** We sample graph nodes with 5 cm node coverage, which fits well with our setup of 11 GiB for training. Using coarser graphs, with 10 cm and 15 cm node coverage resulted in poorer performance: 5.49% and 8.33% higher EPE 3D, as well as 7.34% and 28.46% higher Graph Error 3D, respectively. In turn, the memory footprint on the GPU during training (with batch size 4) decreases with node coverage: 10 513 MiB, 6153 MiB and 5931 MiB for 5 cm, 10 cm and 15 cm node coverage, respectively.

**Number of optimization steps.** We empirically found 3 solver iterations to be the best compromise between performance and computational cost. Unrolling 3 solver iterations instead of 1 / 2, results in 24.9% / 0.16% lower EPE 3D and 22.7% / 0.23% lower Graph Error 3D. Using 4 iterations only improves slightly with respect to 3 (0.04% lower EPE 3D, 0.03% lower Graph Error 3D). More than 4 does not change performance notably.

**Warp loss as a superset of graph loss.** Note that since we sample graph nodes on depth maps, the graph loss (Eq. 15 in the paper) is in practice a subset of the warp loss (Eq. 16 in the paper). However, we found it to be a more general notation to disentangle them. For instance, this notation is helpful for scenarios where graph nodes are not sampled on the RGB-D frame.

## C Losses

In this section, we provide implementation details related to the training losses used for end-to-end optimization. Following the architecture of Sun et al. [32], our correspondence prediction function  $\Phi$  computes a hierarchy of correspondence predictions, instead of only the correspondences at the highest resolution. These predictions are used to compute the correspondence loss in a coarse-to-fine fashion (see Section C.1). To achieve numerically stable optimization (Gauss-Newton solver) during training, a ground-truth mask is needed for the graph loss. In Section C.2, we detail how to ensure stable optimization by filtering invalid graph nodes.

In the ablation studies, we also included a comparison to a weight function  $\Psi$  that is trained in a supervised manner (see Table 1 in the main paper). Section C.3 details the training of this baseline using a supervised binary cross-entropy loss.

### C.1 Coarse-to-fine Correspondence Loss

The design of our correspondence prediction function  $\Phi$  follows the PWC-Net [32] architecture that predicts the correspondences in a coarse-to-fine manner. Initially the correspondences are predicted at a coarse resolution of  $10 \times 7$  px, and then refined to a resolution of  $20 \times 14$  px, etc. In total, there are  $L = 5$  levels in the correspondence hierarchy, and the finest level predictions are used in the differentiable non-rigid optimization. The correspondence loss is applied on every level  $l$ , by bilinear downsampling of the groundtruth correspondences  $\tilde{\mathcal{C}}$  to a coarser resolution, resulting in  $\tilde{\mathcal{C}}^l$ . Similarly, the ground-truth mask matrix  $\tilde{M}^C$  is downsampled to a coarser version  $\tilde{M}^{C^l}$ , to avoid propagating gradients through invalid pixels. For each training sample  $(\mathcal{I}_s, \mathcal{I}_t)$  and every level  $l$ , we therefore compute ground-truth correspondences  $\tilde{\mathcal{C}}^l$  and the ground-truth mask matrix  $\tilde{M}^{C^l}$ . At every level  $l$  the correspondence loss has the following form:

$$\mathcal{L}_{\text{corr}}^l(\phi) = \tilde{M}^{C^l} (|\Phi_{\phi}^l(\mathcal{I}_s, \mathcal{I}_t) - \tilde{\mathcal{C}}^l| + \epsilon)^q. \quad (21)$$

With  $q < 1$  (in our case  $q = 0.4$ ) and  $\epsilon$  being a small constant.

### C.2 Numerically Stable Optimization

The non-rigid tracking optimization objective includes data (correspondence and depth) terms and a regularization (ARAP) term. If we only use regularization term, the optimization problem becomes ill-posed, since any rigid transformation of all graph nodes has no effect on the regularization term.

In order to satisfy memory limits, we do not use all pixel correspondences  $\mathcal{C}$  at training time, but instead randomly sample 10k correspondences.

The edge set  $\mathcal{E}$  of the deformation graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is computed by connecting each graph node with  $K = 8$  nearest nodes, using geodesic distances on the depth map mesh as a metric. This can lead to multiple disconnected graph components, i.e., different node clusters. To ensure the optimization problem is well-defined, we ensure that we have a minimum number of correspondences in each node cluster. In our experiments we filter out all node clusters with less than 2000 correspondences. This filtering has to be reflected in the loss computation. Thus, we define the mask matrix  $\tilde{M}^{\mathcal{V}}$  to have zeros for nodes from filtered clusters, which prevents gradient back-propagation through invalidated graph nodes.

### C.3 Supervised Weight Network Baseline

As a baseline, we introduce a model where we supervise the optimization of the weighting function  $\Psi$ . The ground-truth correspondence weighting  $\tilde{W} \in \mathbb{R}^{H \times W \times 1}$  for this supervision is generated by comparing current correspondence predictions  $\mathcal{C}$  against the ground-truth correspondences  $\tilde{\mathcal{C}}$ . We compare 3D distances between correspondences, using the target depth map  $\mathcal{D}_t$  to query corresponding depth values. A pixel in the ground-truth weighting  $\tilde{W}$  is assigned a 1 or 0 depending on the correspondence error. Optimal performance was achieved by assigning 1 to correspondences that are at most 0.1 m away from groundtruth, and 0 to correspondences that are at least 0.3 m away from groundtruth, without propagating any gradient through remaining correspondence weights. Binary cross-entropy loss is used to optimize  $\Psi$  in this supervised setting.

## D Reproducibility

### D.1 Time and Memory Complexity

**Correspondence Prediction  $\Phi$ .** Function  $\Phi$  scales as a standard convolutional neural network linearly with the number of pixels in the input image (both in time and memory complexity). The number of layers and kernel sizes are independent of the input an, thus, constant. Our correspondence prediction network  $\Phi$  consists of 55 layers with a total number of 9.374M parameters. The processing of an image of resolution  $640 \times 480$  takes 21.6 ms.

**Correspondence Weighting  $\Psi$ .** Function  $\Psi$  is a convolutional neural network with a fixed number of layers and kernels, and, thus, has a complexity that is linear in the number of pixels of the input image. In total the network consists of 316K learnable parameters that are distributed among 7 layers. For a forward pass with an image of resolution  $640 \times 480$  the network takes 5.5 ms.

**Differentiable Optimizer.** Time and memory complexity of our differentiable Gauss-Newton solver is dominated by two operations: matrix-matrix multiplication of  $\mathbf{J}^T$  and  $\mathbf{J}$  and LU decomposition of  $\mathbf{J}^T \mathbf{J}$  matrix. For a matrix  $\mathbf{J} \in \mathbb{R}^{(3|\mathcal{C}|+3|\mathcal{E}|) \times 6N}$  the matrix-matrix multiplication  $\mathbf{J}^T \mathbf{J}$  has a time complexity of  $O(N^2 \cdot (|\mathcal{C}| + |\mathcal{E}|))$  and memory complexity of  $O(N \cdot (|\mathcal{C}| + |\mathcal{E}|))$ . We denoted the number of correspondences with  $|\mathcal{C}|$ , the number of graph edges with  $|\mathcal{E}|$  and the number of graph nodes with  $N$ . On the other hand, the time and memory complexity of LU decomposition of a matrix  $\mathbf{J}^T \mathbf{J} \in \mathbb{R}^{6N \times 6N}$  is  $O(N^3)$  and  $O(N^2)$ , respectively. Note that LU is dominated by matrix-matrix multiplication; in theory there exist algorithms better then  $n^3$ , like  $n^{2.376}$  based on the Coppersmith–Winograd algorithm [7]. The total time complexity is, therefore,  $O(N^2 \cdot (|\mathcal{C}| + |\mathcal{E}|) + N^3)$  and memory complexity is  $O(N \cdot (|\mathcal{C}| + |\mathcal{E}|) + N^2)$ .

### D.2 Training Details

For reproducibility, the analysis of the achieved performance of the network with different training runs is important. In the main paper (Table 1), we show an ablation study of our method and report average test errors of 3 training runs. In Figure 5 the corresponding standard deviations are plotted. As can be seen, the training of our network is stable and results in small variations in performance. For all experiments we used an Intel Xeon 6240 Processor with 18 cores and an Nvidia GeForce RTX 2080Ti GPU. A typical power consumption of Nvidia 2080Ti GPU is around 280 Watts. Network

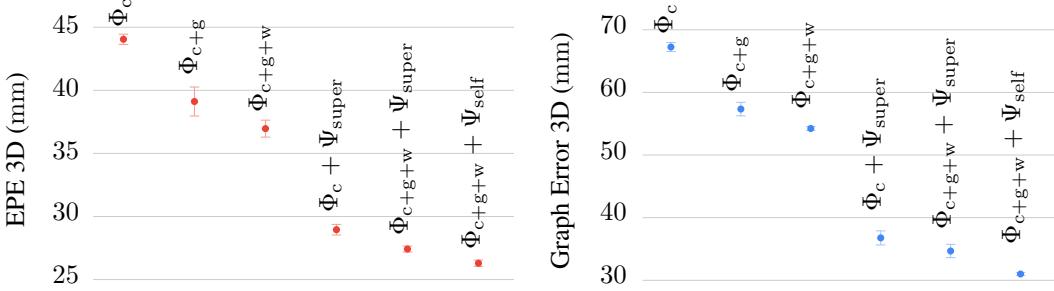


Figure 5: Plots of non-rigid tracking EPE 3D (left) and Graph Error 3D (right) values, together with standard deviation bars. Corresponds to Table 1 in the main paper.

experiments were run for 30k iterations with batch size 4, requiring in total about 14 hours till convergence.

### D.3 Keyframe-based Non-rigid Reconstruction

To achieve robust non-rigid reconstruction, we propose the usage of a keyframe-based strategy. We explored different keyframe sampling, as shown in Table 3, and opted for sampling a keyframe every 50 frames in our setup. For every keyframe, the correspondences to the latest frame in the video are predicted. Our neural non-rigid tracker provides us with correspondence  $c_u$  and an importance weight  $w_u \in [0, 1]$  for every pixel  $u \in \Pi_s \subset \mathbb{R}^2$ . We invalidate all correspondences of a keyframe with  $w_u < \delta$  (we set  $\delta = 0.35$  in all experiments). In case of large occlusions between the current frame and the keyframe, many correspondences are invalid. If 50% of the correspondences are invalid, we completely ignore the keyframe, which leads to less outliers and faster runtime.

Table 3: We evaluate how the deformation error (mm) varies with the keyframe sampling. More frames means lower keyframe sampling rate, i.e., larger frame-to-frame motion.

Keyframe density	Deformation error (mm)
DeepDeform [5] (filtering w/ neighboring frames)	31.52
Ours: keyframe every 100 frames	30.70
Ours: keyframe every 75 frames	29.68
Ours: keyframe every 50 frames	<b>28.72</b>

In addition, we apply correspondence reweighting based on cycle consistencies. Specifically, we enforce bi-directional consistency and multi-keyframe consistency. *Bi-directional consistency* enables us to detect self-occlusions between a keyframe and current frame. Correspondences are predicted in both directions keyframe-to-frame and frame-to-keyframe. If following the correspondence in forward keyframe-to-frame and afterwards in backward frame-to-keyframe direction results in a 3D error larger than 0.20 m, we reject the correspondence. For *multi-keyframe consistency*, multiple keyframe-to-frame predictions are estimated that correspond to the same 3D point in the canonical volume and the mean prediction value is computed. If any of the predictions is more than 0.15 m away from the mean value, we reject all correspondences for a given 3D canonical point.

## E Benchmark Results

In Figure 6 we provide a screenshot of the currently best-performing, non-rigid reconstruction methods on the DeepDeform [5] benchmark. Please visit [http://kaldir.vc.in.tum.de/deepdeform\\_benchmark/benchmark\\_reconstruction](http://kaldir.vc.in.tum.de/deepdeform_benchmark/benchmark_reconstruction).



## Non-rigid Reconstruction Benchmark

This table lists the benchmark results for the Non-rigid Reconstruction scenario.

Method	Info	Geometry error (cm)	Deformation error (cm)
Neural Non-rigid Tracking		0.403	2.872
DeepDeform	Aljaž Božič, Michael Zollhöfer, Christian Theobalt, Matthias Nießner: DeepDeform: Learning Non-rigid RGB-D Reconstruction with Semi-supervised Data. CVPR 2020	0.416	3.152
DynamicFusion	Richard Newcombe, Dieter Fox, Steve Seitz: DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time. CVPR 2015	1.078	6.179
VolumeDeform	Matthias Inmann, Michael Zollhöfer, Matthias Nießner, Christian Theobalt, Marc Stamminger: VolumeDeform: Real-time Volumetric Non-rigid Reconstruction. ECCV 2016	7.485	20.841

Figure 6: Screenshot of non-rigid reconstruction results on DeepDeform [5] benchmark (taken on 11th June 2020).

## F Additional Results

In the following, we present additional qualitative results of our method in comparison to state-of-the-art methods. Figure 7 shows a comparison to Guo et al. [12]. As can be seen, our method better handles non-rigid movements with fast motions (t-shirt) and occlusions (arm).

In Figure 8, we show the results of applying our method on test sequences of the DeepDeform dataset [5], and compare to the results of Slavcheva et al. [28]. The reconstruction of our method leads to more complete and smooth meshes. Note that the results of both methods [12] and [28] were kindly provided by the authors.

We show qualitative reconstruction results of our method on VolumeDeform [15] sequences in Figure 9. Our method can robustly reconstruct these RGB-D sequences, despite the fact that a Kinect sensor was used to record them, whereas our training data was obtained using a Structure IO sensor. This shows that our network predictions can generalize to a different structured-light sensor input.

We also compared to DoubleFusion [37] and BodyFusion [36], which focus on human body reconstruction by assuming human body prior. We were able to compare on a sequence provided by [36], and even without assuming any explicit shape or motion priors, we achieve competitive performance. In particular, our method achieves an average tracking error of 0.0317 m, while BodyFusion and DoubleFusion achieve 0.0227 m and 0.0221 m, respectively. Note that these methods cannot reconstruct non-human sequences, unlike our approach.

Additionally, in Figure 10 we show texturing results, computed by aggregating color images over 100 frames of motion into a voxel grid.

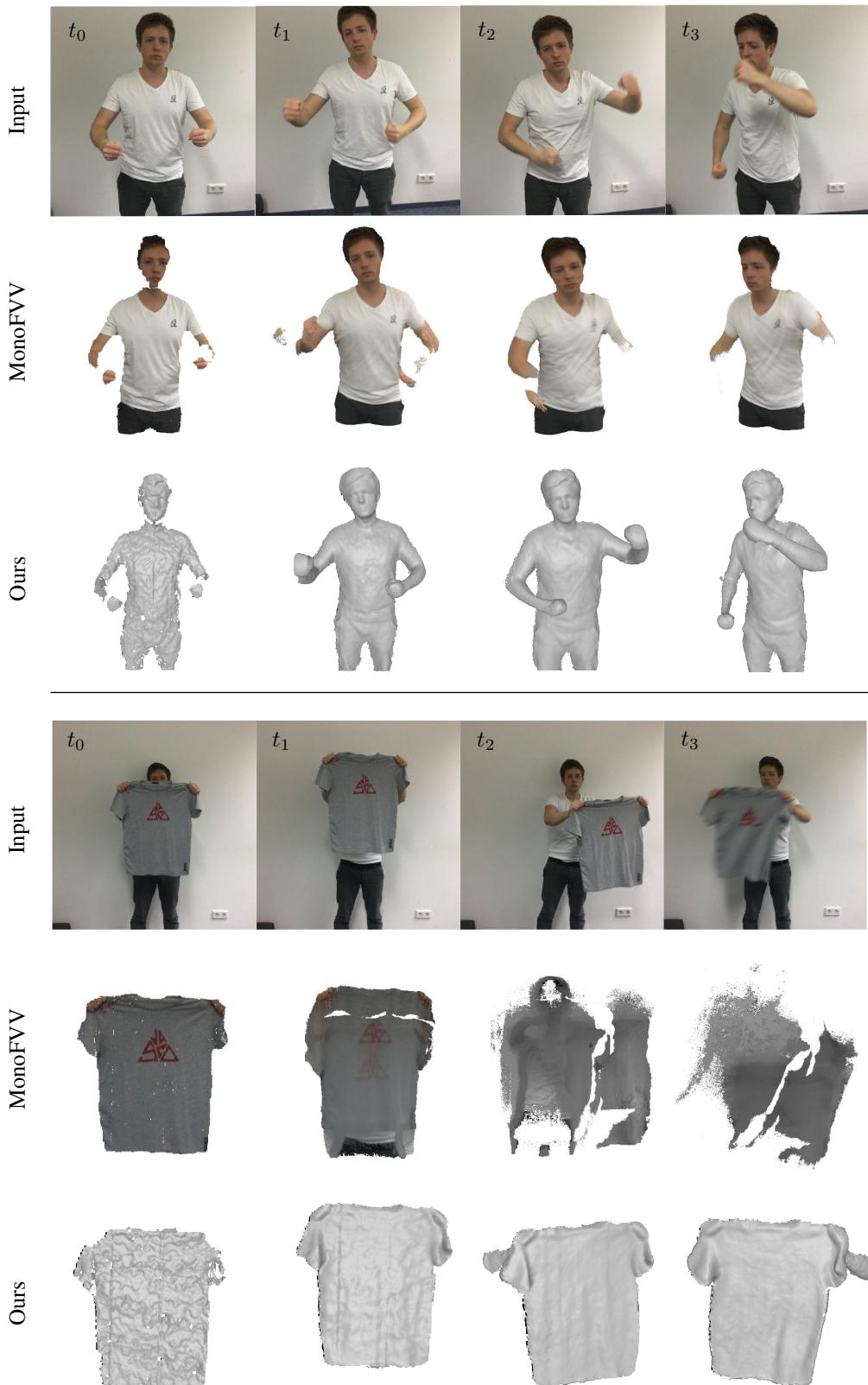


Figure 7: Qualitative comparison of our method to MonoFVV [12] (test sequences from [5]).

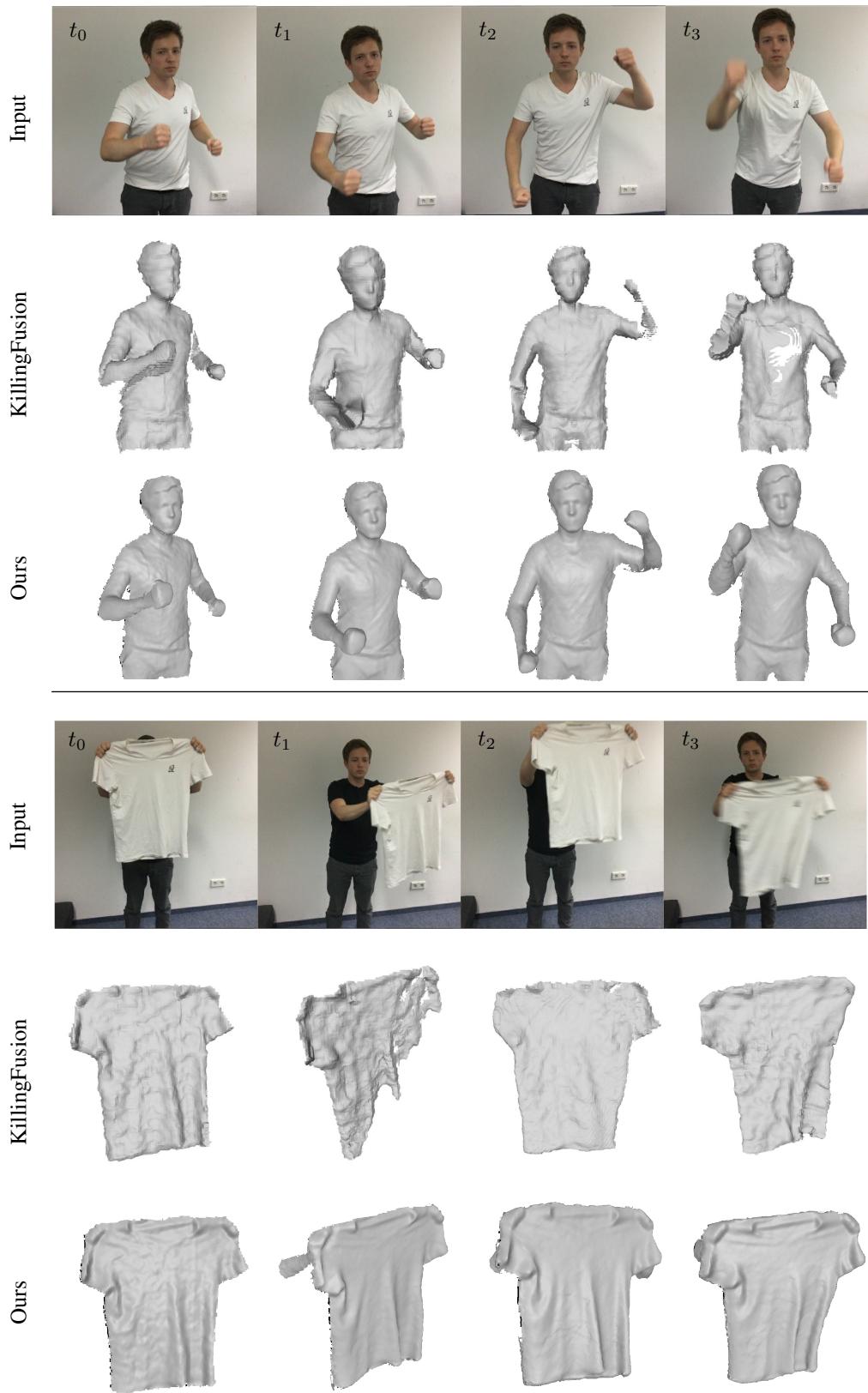


Figure 8: Qualitative comparison of our method to KillingFusion [28] (test sequences from [5]).

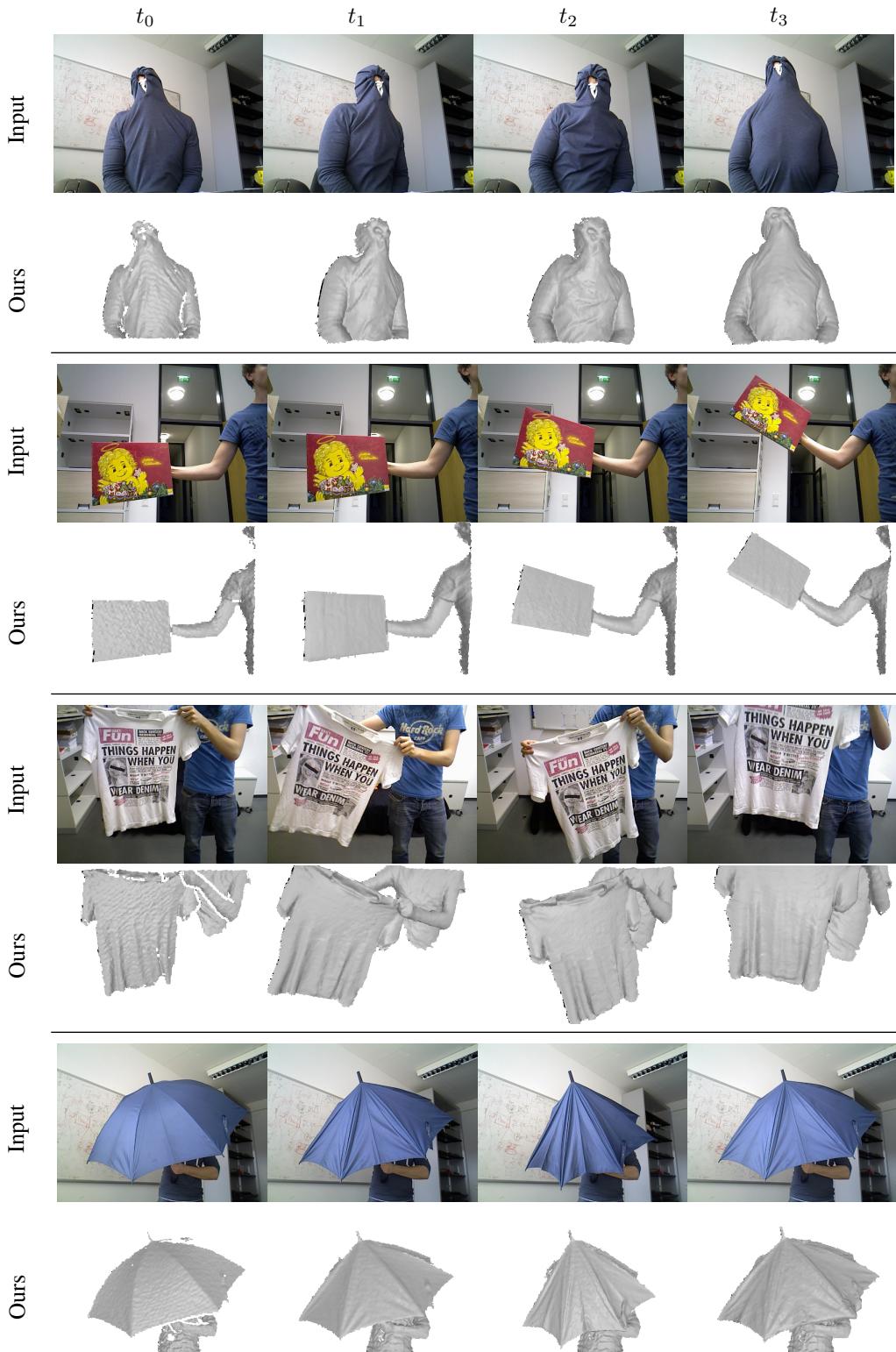


Figure 9: Qualitative reconstruction results on VolumeDeform [15] sequences.



Figure 10: Texturing results, computed by aggregating color images over 100 frames of motion into a voxel grid.