– Supplemental Document –
DeepDeform: Learning Non-rigid RGB-D Reconstruction with Semi-supervised Data

In this supplemental document, we provide further details about our approach. Online benchmark is presented in Sec. 1, network architecture and training is described in Sec. 2, the details of our least-squares GPU solver are given in Sec. 3, dataset statistics are provided in Sec. 4, the dense alignment that is used for training data generation is detailed in Sec. 5, and more qualitative matching and reconstruction results are given in Sec. 6, including a comparison to KillingFusion [3].

1. Online Benchmark

We made dataset publicly available and can be downloaded from https://github.com/AljazBozic/DeepDeform. We provide an online benchmark for two tasks: optical flow estimation and non-rigid reconstruction. Benchmark is available at http://kaldir.vc.in.tum.de/deepdeform_benchmark. The evaluation metrics follow the metrics described in the paper. For optical flow we evaluated average pixel error and pixel accuracy (i.e., ratio of pixels with less than 20 pixel error). For non-rigid reconstruction we evaluate average deformation and geometry error in cm.

2. Network Details

In this section, both the network architecture and the training process are described in detail.

2.1. Architecture Details

The network takes as input an RGB-D frame of dimension $224 \times 224 \times 6$, with 3 channels for RGB and 3 channels for backprojected depth points. The network outputs are a heatmap of size $224 \times 224 \times 1$, another heatmap of size $224 \times 224 \times 1$ and a visibility score of size $1 \times 1 \times 1$. The core unit used in our network is a residual block (as visualized in Fig. 1), the other two building blocks are the downscale (see Fig. 2) and the upscale (see Fig. 4) blocks. Our Siamese network is based on two towers that share the encoder. The encoder consists of residual blocks to extract a feature tensor of dimension $7 \times 7 \times 96$, with all layers detailed in Fig. 3. Encoded features of the source and target patch are concatenated along the dimension of the channels. Afterwards, a bottleneck layer (dimension of $7 \times 7 \times 96$) is applied to reduce the feature dimension back to 96, see Fig. 7. The network has one decoder that maps the features after the bottleneck layer to a probability heatmap $\mathcal{H}$ and a dense depth prediction $\mathcal{D}$, its layer dimensions are provided in Fig. 5. Our network architecture has skip connections between the encoder and decoder, similar to a U-Net [2]. In addition to the decoder, there is also a small convolutional network converting bottleneck features of dimension $7 \times 7 \times 96$ to a visibility score $\mathcal{O}$ of dimension $1 \times 1 \times 1$, presented in Fig. 6.

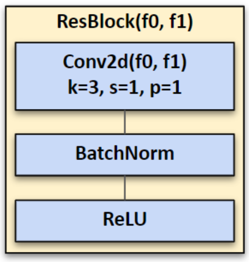

Figure 1: **Residual block:** the core unit in our network.

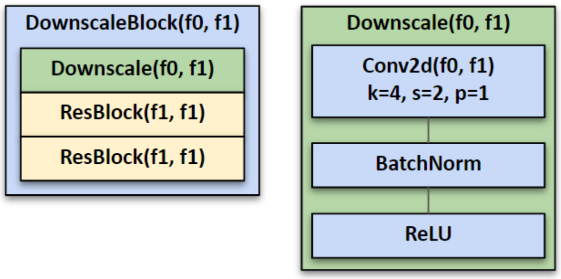

Figure 2: **Downscale block:** building block of the encoder network; it reduces the input dimension and increases the feature size.

2.2. Training Details

We implemented our non-rigid matching approach in PyTorch [1] and trained it using stochastic gradient descent with momentum ($m = 0.9$) and learning rate 0.01. For regularization, we use a weight decay of 0.0005. We use a batch size of 32. We divide the learning rate by 10 every

30k iteration steps. We first train the network for heatmap and depth prediction for $100,000$ iterations. Afterwards, we train only the visibility detection layers for another $100,000$ iterations, keeping the weights in the encoder and bottleneck layers fixed. We use different data augmentation techniques, such as random 2D rotation, translation, and horizontal flip. Every training sample is augmented on-the-fly.

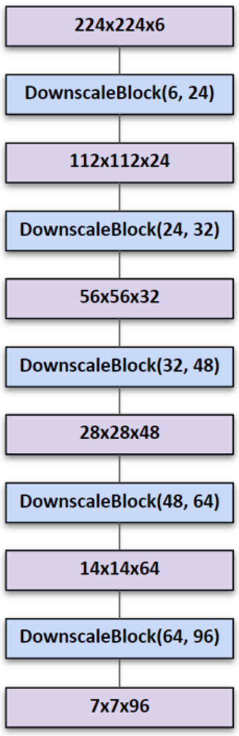

Figure 3: **Network encoder:** our encoder network takes an RGB-D frame (3 channels for RGB and 3 channels for the backprojected depth points) as input and outputs a $7 \times 7 \times 96$ feature tensor.

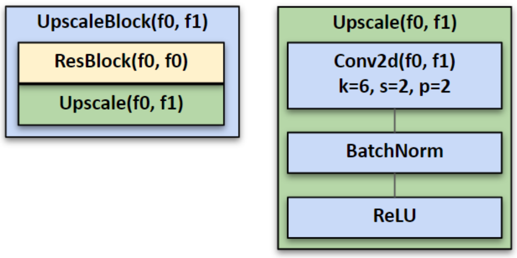

Figure 4: **Upscale block:** building block of the decoder network; it increases the input dimension and decreases the feature size.

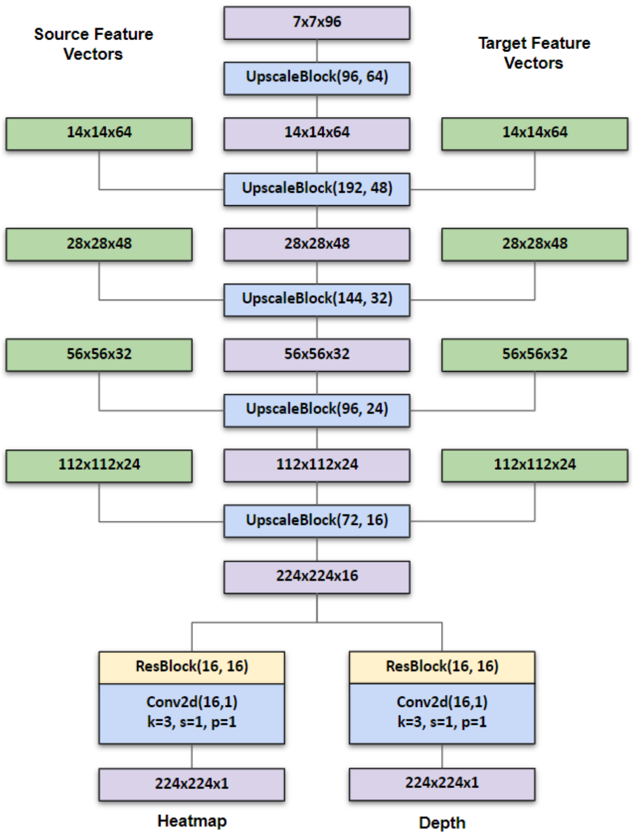

Figure 5: **Network decoder:** the network decoder takes a feature tensor of dimension $7 \times 7 \times 96$ as input and outputs a heatmap of dimension $224 \times 224 \times 1$ and a depth prediction of dimension $224 \times 224 \times 1$. We employ skip connections that directly provide encoded features of the source and target RGB-D frames to the upscale blocks.

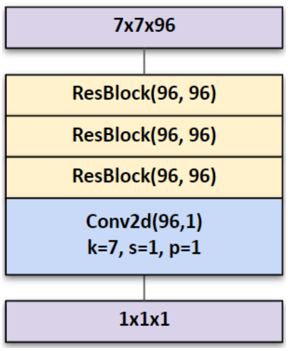

Figure 6: **Visiblity block:** converts bottleneck features into a visibility score $\in [0, 1]$ that measures whether the source point is visible (high value) or occluded (low value) in the target frame.

3. Least Squares GPU Solver

We solve the following non-linear energy minimization problem based on a data-parallel Gauss-Newton solver:

$$\mathcal{G}^* = \underset{\mathcal{G}}{\operatorname{argmin}}\, E_{\text{total}}(\mathcal{G}) \ . \tag{1}$$

To this end, we reformulate the energy function E_{total} in terms of a vector field $\mathbf{F}(\mathcal{G})$ by stacking all residuals:

$$E_{\text{total}}(\mathcal{G}) = \left|\left|\mathbf{F}(\mathcal{G})\right|\right|_2^2 \ .$$

In the following, we will drop the dependence on the parameters $\mathcal{G}$ to simplify the notation. We perform 10 non-linear Gauss-Newton optimization steps. In each non-linear optimization step, the vector field $\mathbf{F}$ is first linearized using a first order Taylor expansion. The resulting linear system of normal equations is then solved based on a data-parallel preconditioned conjugate gradient (PCG) solver. The normal equations are defined as follows:

$$\mathbf{J}^T\mathbf{J}\boldsymbol{\delta} = -\mathbf{J}^T\mathbf{F} \ .$$

Here, $\mathbf{J}$ is the Jacobian matrix of $\mathbf{F}$. After solving the normal equations the unknowns are updated based on $\boldsymbol{\delta}$:

$$\mathcal{G}_k = \mathcal{G}_{k-1} + \boldsymbol{\delta} \ .$$

Each normal equation is solved based on 20 iteration steps. In each PCG step, we first materialize $\mathbf{J}$ in global GPU memory. The central operation in the PCG solver is to apply the system matrix $\mathbf{J}^T\mathbf{J}$ to the current decent direction. In order to run in a data-parallel manner, we launch one thread per residual term in $\mathbf{F}$. Each thread reads the required entries of $\mathbf{J}$ (and therefore also of $\mathbf{J}^T$) and computes its contribution to the partial derivatives of the unknowns. All contributions are summed up for each unknown using atomic operations.

4. Dataset Statistics

We provide a train-val-test split using the following distribution of sequences: 340 sequences are in the training set, 30 in the test set, and 30 in the validation set. We made sure that there is no overlap between captured environments between training and validation/test scenes. We crowd-sourced a large number of annotations for the recorded RGB-D sequences, which makes our dataset suitable for supervised learning of correspondence matching, see Tab. 1. Our novel dataset for learning non-rigid matching covers a diverse range of non-rigid object classes, as shown in Fig. 8, and also includes challenging deformations and camera motion, see Tab. 2.

Data type	Count
Object masks	$4,479$
Sparse matches	$149,228$
Point occlusions	$63,512$
Frame pairs	$5,533$

Table 1: Dataset annotation statistics, presenting the total number of annotations.

Motion type	Average motion
2D change (pixel)	65.4
3D point motion (m)	0.22

Table 2: Motion and deformation statistics, computed from correspondence annotations. Rigid camera motion is also contained in the motion statistics.

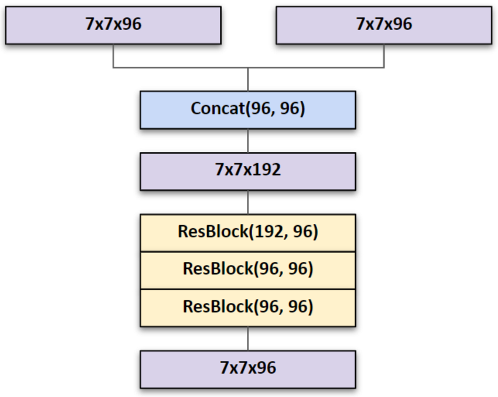

Figure 7: **Bottleneck:** it combines the feature tensors that correspond to the source and target RGB-D frames.

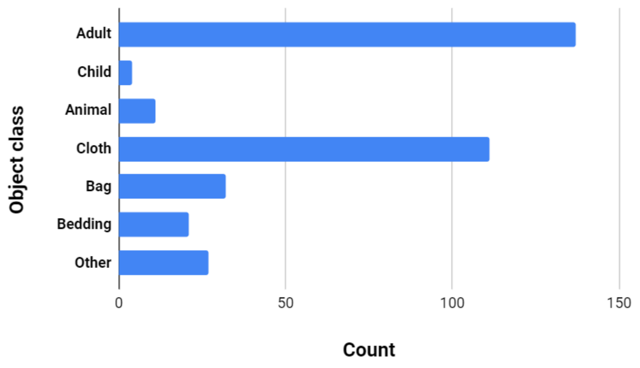

Figure 8: Object class variety: we include many different sequences of dynamic objects, such as cloths, bags, etc.

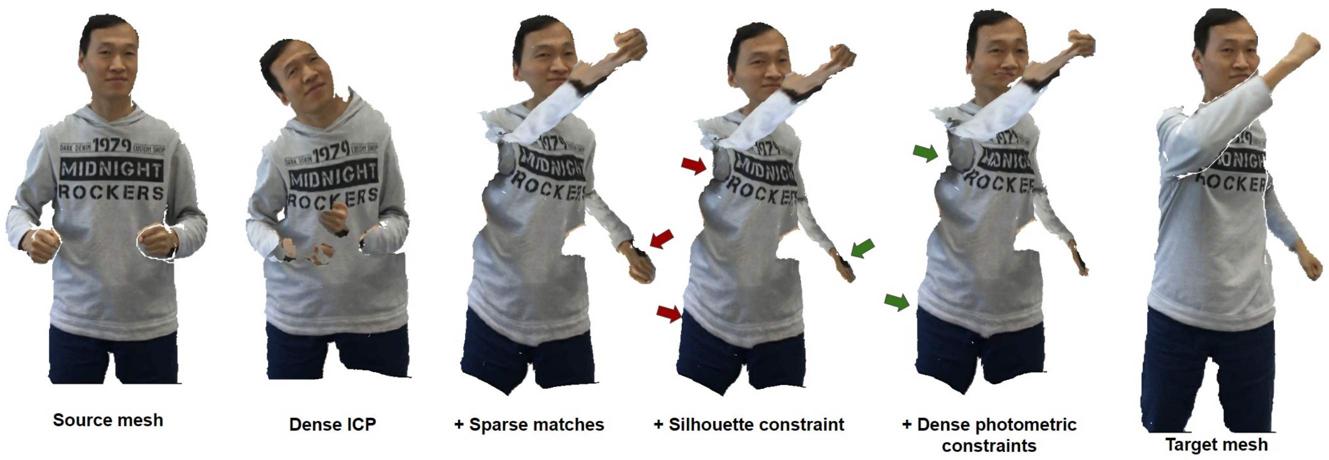

Figure 9: For dense alignment, different optimization constraints are used to align the source mesh to the target mesh. Here we qualitatively show the effect of different constraints on the alignment of the given RGB-D frame pair.

5. Dense Alignment Details

Given two RGB-D frames with dynamic object segmentation masks and sparse correspondences, dense alignment computes dense matches between the source and target RGB-D frames. Initially, a mesh is extracted from the source frame by back-projecting the pixels into 3D and using pixel-wise triangle connectivity. All vertices that are outside the source object mask are filtered out. Afterwards, a deformation graph is sampled uniformly over the source object mesh, and the deformations $\mathcal{T}$ of all nodes that deform the source mesh onto the target mesh are estimated by minimizing the following optimization energy:

$$E_{\text{total}}(\mathcal{T}) = E_{\text{data}}(\mathcal{T}) + \lambda_{\text{photo}} E_{\text{photo}}(\mathcal{T}) + \lambda_{\text{silh}} E_{\text{silh}}(\mathcal{T}) + \lambda_{\text{sparse}} E_{\text{sparse}}(\mathcal{T}) + \lambda_{\text{reg}} E_{\text{reg}}(\mathcal{T}) \ .$$

Data term $E_{\text{data}}(\mathcal{T})$ and regularization term $E_{\text{reg}}(\mathcal{T})$ are defined the same as in the traditional non-rigid reconstruction pipeline, minimizing point-to-point and point-to-plane distances, and using as-rigid-as-possible (ARAP) regularization. As can be seen in Fig. 9, dense ICP constraints alone can result in poor frame-to-frame alignment. A big improvement is achieved by using sparse match constraints, defined as:

$$E_{\text{sparse}}(\mathcal{T}) = \sum_{(\mathbf{s}_i, \mathbf{t}_i) \in \mathcal{M}} \left(\mathcal{W}_{\mathcal{T}}(\mathbf{s}_i) - \mathbf{t}_i\right)^2 \ .$$

Here $\mathbf{s}_i$ and $\mathbf{t}_i$ are annotated match points, and $\mathcal{W}_{\mathcal{T}}(\bullet)$ is the deformation operator that for a given point takes the nearest deformation nodes and executes linear blending of their deformations. An additional silhouette constraint encourages all source mesh vertices to be projected inside the target object's mask:

$$E_{\text{silh}}(\mathcal{T}) = \sum_{\mathbf{v} \in \mathcal{S}} \left(\text{PixDist}(\Pi(\mathcal{W}_{\mathcal{T}}(\mathbf{v})))\right)^2 \ .$$

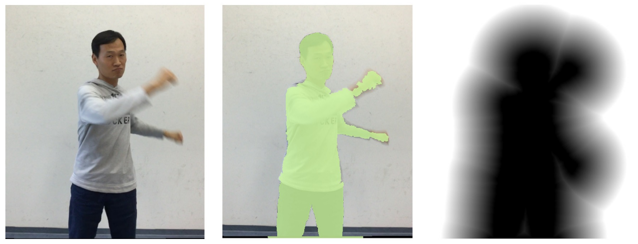

Figure 10: For a target object (left), given its mask (middle), we can compute the pixel distance map (right), with zero distances inside the object mask and increasing distances outside the object mask.

The constraint is computed for every vertex $\mathbf{v}$ in the source mesh, where $\Pi(\bullet)$ is a projection from 3D to 2D image space, and PixDist is the pixel distance map, computed as shown in Fig. 10. Lastly, dense color constraints enable alignment correction in cases where we have useful texture information:

$$E_{\text{photo}}(\mathcal{T}) = \sum_{\mathbf{v} \in \mathcal{S}} \left(\nabla \mathrm{I}_t(\Pi(\mathcal{W}_{\mathcal{T}}(\mathbf{v}))) - \nabla \mathrm{I}_s(\Pi(\mathbf{v}))\right)^2 \ .$$

Here I_s and I_t are source and target color images. We use color gradients instead of raw colors to be invariant to constant intensity change.

6. Qualitative Results

In the following figures we present more qualitative results. More specifically, in Fig. 11, our network heatmap predictions are presented for a set of chosen points, in Fig. 12 we provide a qualitative reconstruction comparison to the approach of [3], and in Fig. 13 reconstruction results of our method are shown, including the shape in the canonical space.

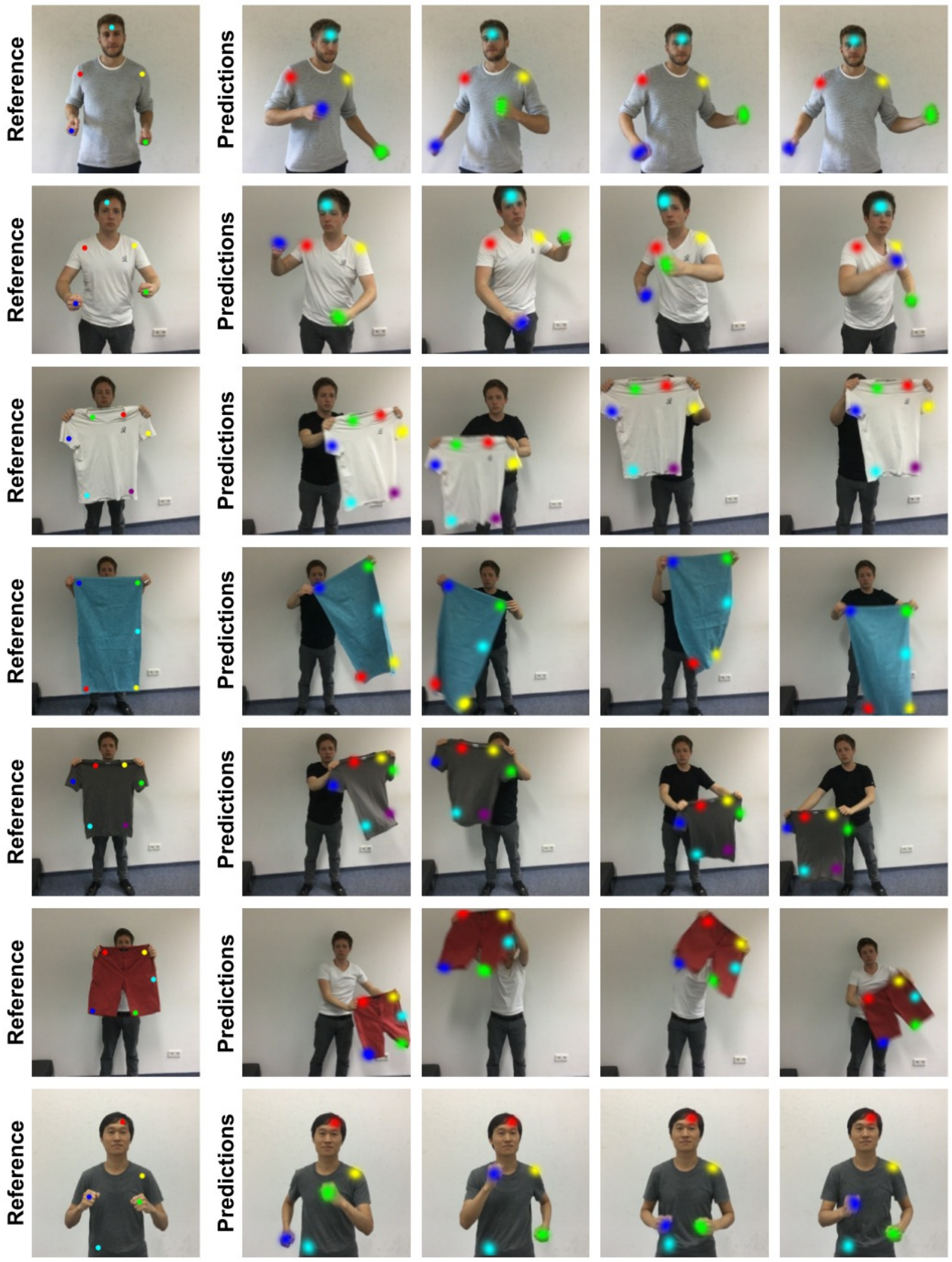

Figure 11: Qualitative heatmap prediction results. Our approach works well even for highly non-rigid motions.

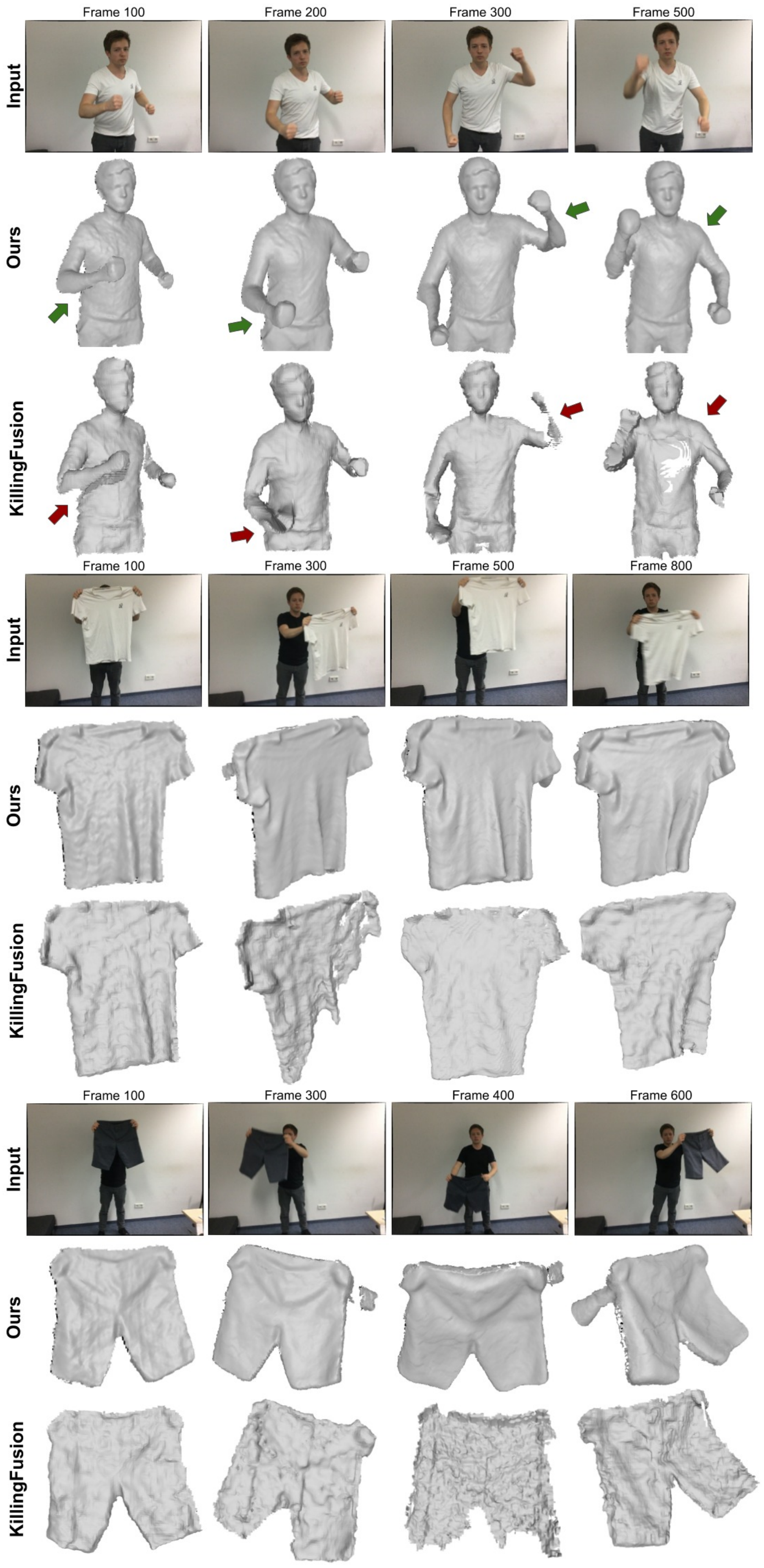

Figure 12: Qualitative reconstruction comparison of our approach to KillingFusion [3]. Reconstruction results were kindly provided by the authors. Our approach obtains more robust and higher quality results.

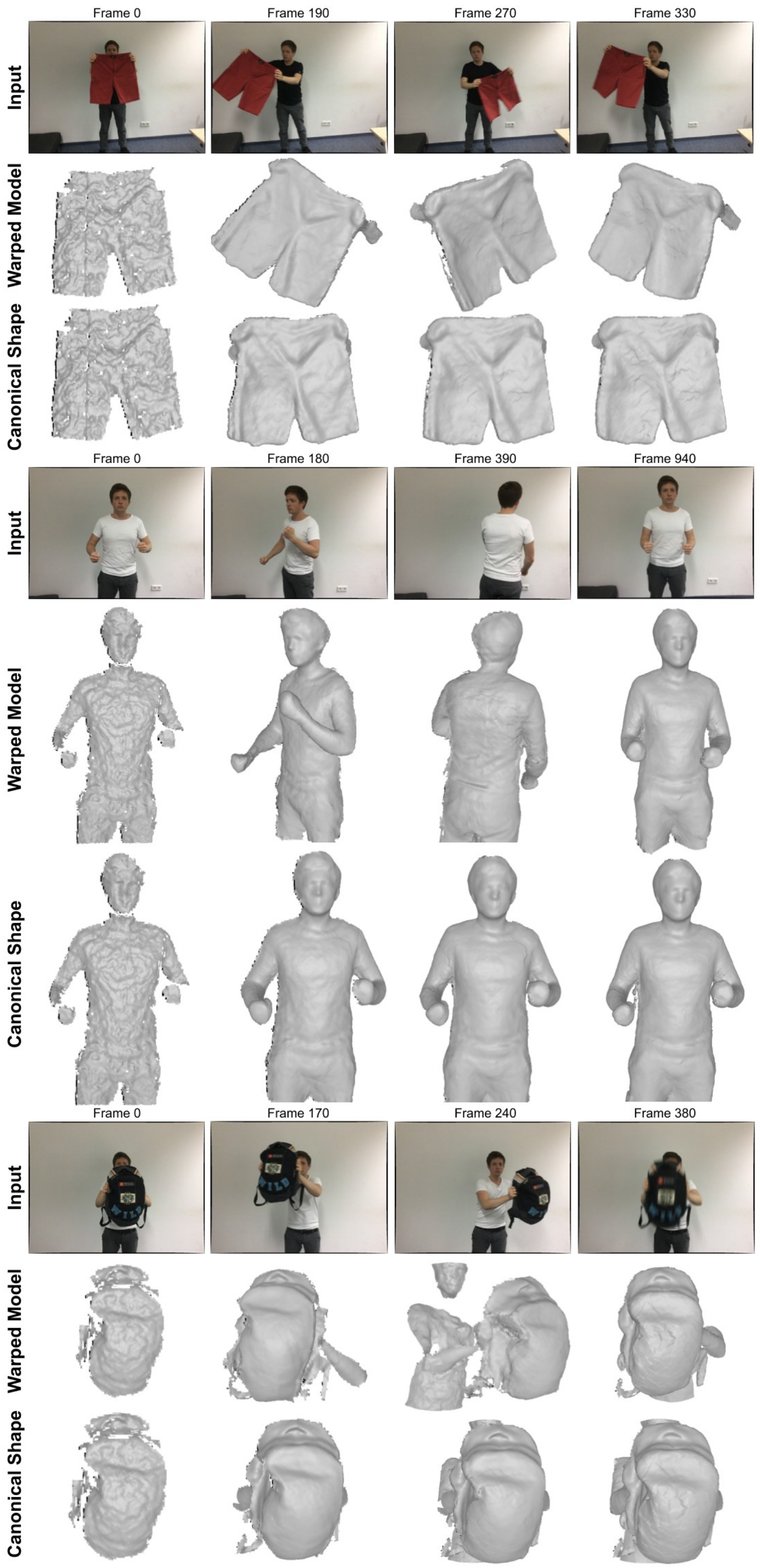

Figure 13: Qualitative reconstruction results, showing both the warped model (in the current frame) and the canonical shape (in the reference frame). Our approach obtains high quality reconstruction results.

References

[1] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 1

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 1

[3] Miroslava Slavcheva, Maximilian Baust, Daniel Cremers, and Slobodan Ilic. Killingfusion: Non-rigid 3d reconstruction without correspondences. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, page 7, 2017. 1, 4, 6