– Supplemental Document –
Neural Deformation Graphs for Globally-consistent Non-rigid Reconstruction

Aljaž Božič[1] Pablo Palafox[1] Michael Zollhöfer[2] Justus Thies[1] Angela Dai[1] Matthias Niessner[1]
[1]Technical University of Munich [2]Facebook Reality Labs Research

In this supplemental document, we provide further details about our proposed neural deformation graphs. Specifically, we describe the network architectures in detail (Sec. 1), give additional training information (Sec. 2), provide details on training duration and runtime of test-time deformation and geometry computation (Sec. 3), include more ablations of our approach (Sec. 4), and present our capture setup used for recording real non-rigid motion sequences using Kinect Azure sensors (Sec. 5).

1. Network Architecture

Our method is composed of two learned components; the neural deformation graph based on a 3D CNN (shown in Fig. 1) and a set of multi-layer perceptrons (MLPs) which are used to implicitly represent the surface (see Fig. 4).

Neural Deformation Graph. The neural deformation graph implicitly stores the deformation graphs for each input frame k. The underlying 3D CNN encoder takes a dense 64^3 SDF grid $\mathbf{S}_k$ as input and outputs the $(3+3+1)N$ dimensional vector, representing position, rotation (in axis-angle format) and importance weight for each graph node (N being the number of graph nodes). The input grid is encoded to a spatial dimension of 4^3 and a feature dimension of 64 using 4 blocks of convolutional downsampling with stride of 2, followed by 2 residual units (with ReLU activation unit and batch norm). The downscale operation is detailed in Fig. 2. We use a linear layer to convert the $4^3 \cdot 64$ features to dimension 2048. This feature vector is input to two independent heads, each composed of 3 linear layers with feature dimension of 2048 and a leaky ReLU as activation unit with negative slope of 0.01 (no activation function after the last linear layer). One head predicts graph node rotations $\mathbf{R}_k$ of dimension $3N$, while the other head predicts graph node positions $\mathbf{V}_k$ and weights $\mathbf{W}_k$ (total dimension of $(3+1)N$). We concatenate both predictions, which results in a graph embedding of dimension $7N$. For all our experiments, we use $N = 100$.

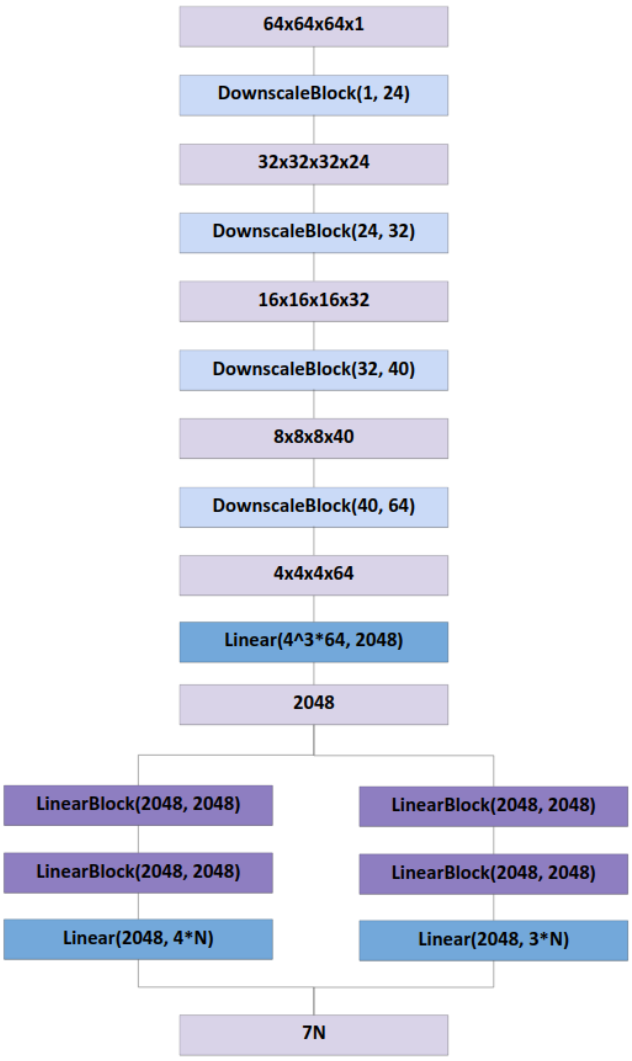

Figure 1: Our neural deformation graph is represented as a 3D CNN which takes as input a 64^3 SDF grid and outputs $7N$ graph node parameters. Node parameters are predicted via two independent heads, one for graph node positions and weights, and the other for graph node rotations. The architectures of the single computation blocks are detailed in Fig. 2 and Fig. 3.

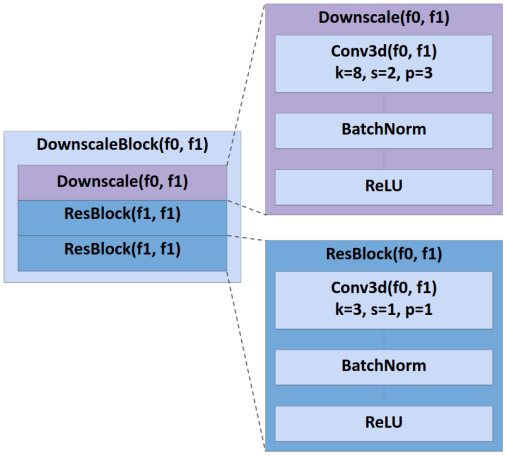

Figure 2: The downscale block used in our 3D CNN architecture (see Fig. 1) reduces the spatial dimension by 2.

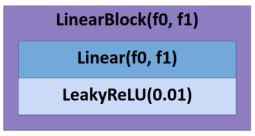

Figure 3: Each linear block of the 3D CNN architecture (see Fig. 1) applies a linear layer, followed by a leaky ReLU with negative slope of 0.01.

Implicit Surface Representation For each graph node i, we train a separate MLP f_i that predicts the signed distance field around the node, as presented in Fig. 4. As explained in the main paper, the complete shape geometry is reconstructed by warping each node's local SDF using estimated node deformations and interpolating the local SDFs using the estimated node interpolation weights (computed from node radii and importance weights). Each MLP takes a 3D position as input, and outputs an SDF value for the given position. The position is represented with positional encoding [5] of dimension $F = 30$. To reconstruct pose-dependent details, the MLP is conditioned on a pose code. This D-dimensional pose code ($D = 32$ in our experiments) is computed by inputting the current deformation graph predictions through a linear projection layer Π_i shown in Fig. 5. Thus, in total, the MLP takes an input of dimension $D + F$, and using 8 linear layers with feature dimension of 32, it outputs the SDF value of the corresponding node. There is a skip connection between the input and the sixth linear layer, and a leaky ReLU with negative slope of 0.01 is applied after each linear layer, as shown in Fig. 3.

In Tab. 1, we evaluate the influence of the pose codes with respect to the reconstruction quality measured using a Chamfer distance. The pose conditioning improves the Chamfer distance by a large margin, which is also visible in the qualitative comparison in Fig. 7.

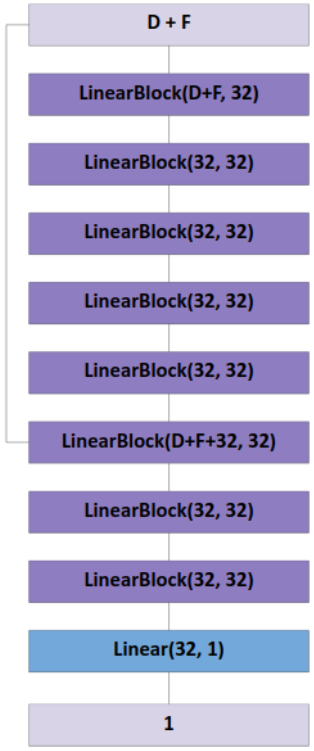

Figure 4: We represent the surface of an object using an implicit function (SDF) which is based on multi-layer perceptrons (MLPs). Each graph node MLP takes as input a $(D + F)$-dimensional vector (consisting of the pose code and the sample position, represented with positional encoding [5] in local coordinates) and outputs the SDF value.

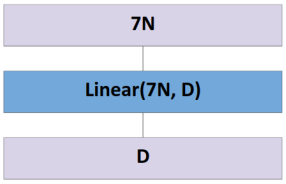

Figure 5: To compute the sparse pose code of the graph, we use a single linear projection layer that converts the graph parameters of dimension $7N$ to a code of dimension D (we use $N = 100$ and $D = 32$). This pose code is input to the local MLP that represents the pose-dependent local surface.

2. Training Details

Our method's training process is described in Sec. 3.4 of the main paper. In this section, we specify the sampling strategy during training of both the neural deformation graph and the implicit surface representations (see visualization of samples in Fig. 6).

When training the neural deformation graph, we use $|P_{\text{un}}| = 3000$ uniform point samples, $|P_{\text{ns}}| = 3000$ near

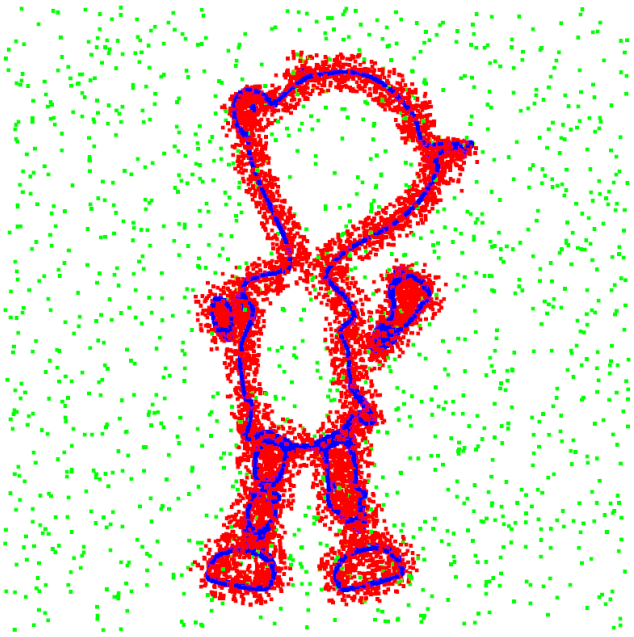

Figure 6: We visualize the uniform samples P_{un} (green), near surface samples P_{ns} (red) and the surface samples P_{s} (blue) for a slice of samples with $z \in [-0.01, 0.01]$ at one frame of the character sequence shown in the Fig. **??** of the main paper.

surface point samples and $|P_{\text{s}}| = 3000$ surface point samples, sampled randomly for each batch from 100k preprocessed point samples. For the graph coverage loss, we apply an additional weight of 10.0 for interior point samples (determined by the SDF sign).

To train our multi-MLP network that implicitly represents the surface, we use $|P_{\text{un}}| = 1500$ uniform point samples and $|P_{\text{ns}}| = 1500$ near surface point samples. Note that this reduced set of samples is applied to satisfy memory limits of our used GPU (Nvidia Geforce 2080Ti). We use a truncation of 0.1 (in normalized units of the object in the unit cube) for the signed distance field used for the reconstruction loss.

Note, when interpolating the SDF grid for the node interior loss, in the case that a graph node's position is predicted outside the grid, we define the out-of-grid loss by an ℓ_2-distance to the nearest bounding box corner. This encourages graph node positions to be always predicted inside the shape's bounding box.

3. Runtime Analysis

Our approach is trained independently (from scratch) for every character sequence, optimizing our neural deformation graph to best fit the observed sequence. For our unoptimized implementation, the optimization takes about 2 days with a single Nvidia GeForce 2080Ti. The dense deformation field between any two frames in the sequence is computed for (near) surface points by estimating skinning weights w.r.t. all nodes and interpolating the frame-to-frame motion of the nodes, which takes 0.02s for about $30k$ points. The mesh is extracted for every frame by sampling 128^3 grid sample points, predicting their SDF values and executing Marching Cubes on the resulting SDF grid, which takes about 3.70s per-frame.

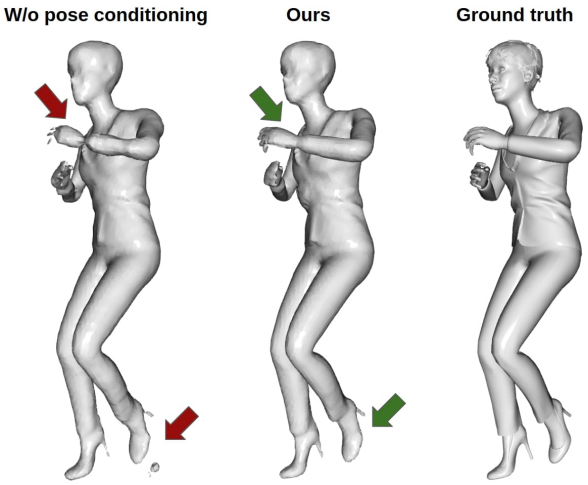

Figure 7: We compare our implicit reconstruction with pose conditioning (middle) to without pose conditioning (left). Pose conditioning clearly improves reconstruction performance in regions of very strong deformation.

Approach	Chamfer
Without pose conditioning	0.46
With pose conditioning	**0.40**

Table 1: Evaluation of the pose conditioning effect on reconstructions on synthetic data using Chamfer distance ($\times 10^{-4}$).

4. Additional Ablations

In this section, we provide further ablations of our method that evaluate the effect of pose conditioning, different graph sizes, and 3D CNN encoder variations.

Pose Conditioning. Our approach uses a pose-conditioned multi-MLP formulation for the prediction of an implicit surface at every frame, which helps to refine local geometry details that are not captured properly by the graph deformation. The benefits of pose conditioning over not using any pose codes can be observed both qualitatively (in Fig. 7) and quantitatively (in Tab. 1).

Approach	EPE3D
Without Encoder (Direct Optimization)	25.07
Without Batch Norm	1.36
Graph with 30 nodes	1.71
Graph with 200 nodes	1.19
Ours: Encoder with Batch Norm + 100 nodes	**1.16**

Table 2: Evaluation of encoder and graph size variations on synthetic data using 3D end-point-error ($\times 10^{-2}$).

Graph Size. We use $N = 100$ graph nodes in our experiments. This is just an upper limit though, since by optimizing over graph node weights the method can eliminate redundant nodes. In Tab. 2 we show that using only $N = 20$ graph nodes does not provide enough degrees of freedom to fully represent deformations in our sequences and leads to worse performance. On the other hand, using $N = 200$ nodes seems to be redundant compared to $N = 100$ nodes, and does not lead to any performance improvement.

3D Encoder. We compare our 3D CNN encoder formulation with direct parameter optimization, where instead of optimizing over 3D CNN parameters we directly optimize rotation, translation and weight node parameters independently for every frame, using random initialization. As can be seen in Tab. 2, the use of the encoder is very important for the overall performance. One possible explanation is that the encoder enables us to additionally enforce viewpoint consistent predictions, as described in Section 3.2 in the main paper. Furthermore, the inductive bias of the encoder considerably improves optimization convergence. We also find that using batch normalization in the encoder architecture leads to better performance.

Variance along Sequence Frames. We report the mean and standard deviation of reconstruction and deformation errors over different frames of our synthetic evaluation sequences in Tab. 3. Our method outperforms all baselines w.r.t. both mean frame errors as well standard deviation of the errors along the frames, which suggests that the geometry and deformation estimation results of our approach are consistent over the length of the sequences.

5. Real Data Capture Setup

In the main paper as well as in the supplemental video, we demonstrate that our method performs well on real data. To capture real-world non-rigid motion sequences, we record with four Kinect Azure sensors as shown in Fig. 8. All four sensors are connected to the same computer via USB-C cables and are hardware-synced using a daisy-chain configuration (connecting the trigger of the master camera with the other 3 cameras in a chain). To avoid interference between depth sensors, we set a delay of 160 microseconds between different depth camera captures.

Method	Chamfer	EPE3D
SIF [2]	1.12 ± 0.31	8.56 ± 5.09
LDIF [1]	2.41 ± 2.07	10.40 ± 6.73
OccupancyFlow [7]	53.83 ± 28.44	16.29 ± 5.83
MV DynamicFusion [6]	2.19 ± 1.73	3.06 ± 1.18
MV DF [6] + FlNet3D [4]	1.93 ± 1.54	2.55 ± 0.86
Robust L0 Tracking [3]	2.31 ± 1.60	2.50 ± 0.92
Ours	$\mathbf{0.40 \pm 0.07}$	$\mathbf{1.16 \pm 0.78}$

Table 3: We show quantitative comparisons with state-of-the-art approaches, evaluating geometry using chamfer distance ($\times 10^{-4}$), and deformation using EPE3D ($\times 10^{-2}$), with corresponding standard deviations across different sequence frames.

References

[1] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4857–4866, 2020. 4

[2] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 4

[3] Kaiwen Guo, Feng Xu, Yangang Wang, Yebin Liu, and Qionghai Dai. Robust non-rigid motion tracking and surface reconstruction using l0 regularization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3083–3091, 2015. 4

[4] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 529–537, 2019. 4

[5] 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. *arXiv preprint arXiv:2003.08934*, 2020. 2

[6] Richard A Newcombe, Dieter Fox, and Steven 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. 4

[7] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5379–5389, 2019. 4