LART: Neural Correspondence Learning with Latent Regularization Transformer for 3D Motion Transfer -Supplementary Materials-

Haoyu Chen^{1} **Hao Tang**^{2,3} **Radu Timofte**^{2,4} **Luc Van Gool**^{2,5,6} **Guoying Zhao**^{1*}
^{1}CMVS, University of Oulu ^{2}Computer Vision Laboratory, ETH Zurich
^{3}Carnegie Mellon University ^{4}University of Wurzburg ^{5}ESAT-PSI, KU Leuven
^{6}INSAIT, Sofia Un. St. Kliment Ohridski
`chen.haoyu@oulu.fi, hao.tang@vision.ee.ethz.ch, radu.timofte@uni-wuerzburg.de`
`vangool@vision.ee.ethz.ch, guoying.zhao@oulu.fi`

The Supplementary Materials include additional technical details and extra experimental results that were not included in the main submission due to the lack of space. All results were obtained with exactly the same methodology as the one described in the main manuscript. We first provide more implementation details of our LART networks (Sec. 1).

Next, we provide more details of processing datasets (Sec. 2), followed by detailed training settings in Sec. 3. Finally, we present more experimental results in Sec. 4.

1 LART Network Architecture

Our LART framework consists of two main parts: a 3D mesh feature extractor, and a LART Decoder for 3D generation. We first introduce the network structures of each component and then give the architectural parameters of the full model.

3D Mesh Feature Extractor. The architecture of the feature extractor is presented in Table 1. The feature extractors are used to extract a latent embedding of motions from the given mesh sequences for further mesh generation with the following decoders. Note that in order to fit our model to a flexible vertex number of mesh models (not only 6,890 of SMPL [2] templates, but also MG-cloth [1] of more than 27,000 vertices), we use the geometry-adaptive feature extractor. It can flexibly process meshes with different sizes into the desired one. Basically, it works as a normal 3D feature extractor that encodes the input meshes with vertex number N_1 into a latent vector with C size. The difference is that the latent vector will be expanded up based on the target meshes along the topology dimension (vertex number N_2), resulting in a latent vector with $C \times N_2$. In this way, we align the sizes between the source mesh and target mesh as the same and make the learning of the correspondence possible. Note that, although our LART encoder can handle large-size meshes with different sizes than driving motions, using a Geometry Adaptive 3D Feature Encoder might lead to degenerated results if the size gap between target and source meshes is too big.

LART Decoder. The network architecture of a LART decoder is presented in Table 2. As many previous works discussed [10, 5, 11], the MLP blocks in the Vanilla Transformer are not suitable for preserving the detailed geometry of the meshes, thus we customize the MLP blocks into an instance normalization (InsNorm) block inspired by [15] presented in Table 3. The LART decoder is used to generate the animated sequence of the target mesh with given motions with full geometric details preserved.

LART with VAE. To better gain the latent space learning ability of LART, we couple the LART with VAE [7] for the task of temporal interpolation and unseen motion transfer. As VAE provides a partial remedy by modeling a distributional prior on the data via a parametrized density on the latent space.

---

[*] Corresponding author.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

This induces additional regularization to construct the linear latent space. Similar to LIMP [6], we extend the LART to the VAE version by revising the output of the feature extractor. Specifically, we split the latent code into half/half to predict both the mean and variance of the multivariate Gaussian distribution from which the latent code is sampled. An extra Kullback Leibler Divergence loss is then added to the training with its weight of 1.0. For the accurate motion transfer via the seen motions, VAE is not applied.

Table 1: Detailed architectural parameters for the 3D mesh feature extractor. “B” stands for batch size and “N” stands for vertex number. The first parameter of Conv1D is the kernel size, the second is the stride size. “T” stands for the frame number. The same as below.

Index	Inputs	Operation	Output Shape
(1)	-	Input mesh	B×T×3×N
(2)	(1)	Conv1D (1 × 1, 1)	B×T×64×N
(3)	(2)	Instance Norm, Relu	B×T×64×N
(4)	(3)	Conv1D (1 × 1, 1)	B×T×128×N
(5)	(4)	Instance Norm, Relu	B×T×128×N
(6)	(5)	Conv1D (1 × 1, 1)	B×T×1024×N
(7)	(6)	Instance Norm, Relu	B×T×1024×N
(8)	(7)	Max pooling and tiling (for targets with different sizes)	B×T×1024×N’
(9)	(8)	Positional Embedding	B×T×1024×N

Finally, we present the full model architecture in Table 4. The embedded motion features will be fed into LART decoders together with the target mesh and sequential meshes will be generated.

2 Dataset Settings

Training Sets. We use the DFAUST dataset [4] to prepare the training dataset for quantitative evaluation. It has 129 motions from ten subjects, and each motion lasts for hundreds of frames. One motion (driving sequence) and one appearance (target mesh) will be randomly combined as a pair for training. When training, each time we sample 3 continuous frames (with a random interval between 1-20 frames) of a motion (60 frames) as a driving sequence inputs and feed them to the networks with a randomly paired target mesh. The driving sequences are split into two settings, i.e., the seen driving sequences (80 motions from the first 6 subjects of DFAUST) and the unseen driving sequences (20 motions from the rest 4 subjects of DFAUST). Those 20 motions are only available for evaluation.

Table 2: Detailed architectural parameters for LART decoder.

Index	Inputs	Operation	Output Shape
(1)	-	Identity Embedding	B×T×C×N
(2)	-	Pose Embedding	B×T×C×N
(3)	(1)	conv1d (1 × 1, 1)	B×T×C×N
(4)	(2)	conv1d (1 × 1, 1)	B×T×C×N
(5)	(3)	Reshape	B×T×N×C
(6)	(5)(4)	Batch Matrix Product	B×T×N×N
(7)	(6)	Softmax	B×T×N×N
(8)	(7)	Reshape	B×T×N×N
(9)	(2)	conv1d (1 × 1, 1)	B×T×C×N
(10)	(2)(8)	Batch Matrix Product	B×T×C×N
(11)	(10)	Parameter gamma	B×T×C×N
(12)	(11)(2)	Add	B×T×C×N
(13)	-	Pose Mesh	B×T×3×N
(14)	(12)(13)	SPAdaIN	B×T×C×N
(15)	(14)	conv1d(1 × 1, 1), Relu	B×T×C×N
(16)	(14)(15)	SPAdaIN	B×T×C×N
(17)	(16)	conv1d(1 × 1, 1), Relu	B×T×C×N
(18)	(12)(13)	SPAdaIN	B×T×C×N
(19)	(18)	conv1d(1 × 1, 1), Relu	B×T×C×N
(20)	(17)(19)	Add	B×T×C×N

Table 3: Detailed architectural parameters for SPAdaIN block.

Index	Inputs	Operation	Output Shape
(1)	-	Driving Motion Embedding	B×T×C×N
(2)	(1)	Instance Norm	B×T×C×N
(3)	-	Target Mesh	B×3×N
(4)	(3)	Conv1D (1 × 1, 1)	B×T×C×N
(5)	(3)	Conv1D (1 × 1, 1)	B×T×C×N
(6)	(4)(2)	Multiply	B×T×C×N
(7)	(6)(5)	Add	B×T×C×N

Table 4: Detailed architectural parameters for the full model.

Index	Inputs	Operation	Output Shape
(1)	-	Target Mesh	B×3×N
(2)	-	Driving Motion Mesh	B×T×3×N
(3)	(2)	Feature Extractor	B×T×1024×N
(4)	(3)	Conv1D (1 × 1, 1)	B×T×1024×N
(5)	(4)(1)	LART decoder 1	B×T×1024×N
(6)	(5)	Conv1D (1 × 1, 1)	B×T×512×N
(7)	(6)(1)	LART decoder 2	B×T×512×N
(8)	(7)	Conv1D (1 × 1, 1)	B×T×512×N
(9)	(8)(1)	LART decoder 3	B×T×512×N
(10)	(9)	Conv1D (1 × 1, 1)	B×T×256×N
(11)	(10)(1)	LART decoder 4	B×T×256×N
(13)	(12)	Conv1D (1 × 1, 1)	B×T×3×N
(14)	(13)	Tanh	B×T×3×N

For the DFAUST dataset, we use those 20 motions mentioned above for testing. Since there are more than $3e11$ potential training pairs (target meshes: $80\times$ 60×16 with driving sequential meshes: $80\times$ $60\times 16\times$ 57), which is way larger than our computational capacity to cover the whole space, we randomly select 8,000 training pairs at each epoch during the training.

We use the AMASS dataset [8] to prepare the training dataset for qualitative evaluation (to show more diversity of the motions). AMASS contains 15 different sub-datasets (including SMPL-registered DFAUST), spanning over 300 subjects with more than 1,1000 motions. Here we select some representative motions from AMASS (in total, 188 motions) to train the model. The subsets include ACCAD, DFAUST, MoSh, and PosePrior.

Target Meshes. We randomly generate 16 mesh shapes for training and another 8 meshes for testing. We use the SMPL model to generate the shape by randomly sampling from the shape parameter spaces. The ground truth is obtained by using the SMPL model [2] to synthesize the target animated sequence with the shape and pose parameters provided by the dataset. The mesh vertices are shuffled randomly and the generated faces are correspondingly shuffled to construct the meshes.

The Driving Sequences for Quantitative Evaluation. We split motions into two settings, i.e., the seen driving sequences (80 motions selected from the first 6 subjects of DFAUST) and the unseen driving sequences (20 motions selected from the rest 4 subjects of DFAUST). Those 20 motions are only available for evaluation.

Inference. We employ the model trained from DFAUST and AMASS directly to drive the target meshes from other datasets, e.g., FAUST [3] and MG-dataset [1]. As mentioned in the network architecture section, to process the large vertex number (more than 27,000) from the MG-cloth dataset, we use geometry-adaptive positional embeddings. For the model trained to evaluate on the DFAUST dataset (with a typical SMPL template), similar to NPT and 3D-CoreNet, we use the original features without adaptive embedding.

Other Domains. At last, we extend the LART to other domains with domain-specific learning. We verify the animal domain on the Animal dataset [14] and the hand domain on the MANO dataset [12]. The Animal dataset provides frame-level correspondences of the motions of two animals. Thus,

Table 5: Licenses of the assets used in the paper.

Data	License websites
SMPL [2]	`https://smpl.is.tue.mpg.de/modellicense.html`
MANO [12]	`https://mano.is.tue.mpg.de/license.html`
MG-Dataset [1]	`https://github.com/bharat-b7/MultiGarmentNetwork`
AMASS [8]	`https://amass.is.tue.mpg.de/license.html`
DFAUST [4]	`https://dfaust.is.tue.mpg.de/license.html`
FAUST [3]	`http://faust.is.tue.mpg.de/data_license`
Animal [14]	`https://people.csail.mit.edu/sumner/research/deftransfer/`

we prepare the dataset by pairing the animal templates directly as ground truth. Since the camel mesh has a different vertex number 21,887 than the elephant mesh 42,321, we use geometry-adaptive embeddings. The inputs are meshes with 21,887 vertices, and the outputs are meshes with 42,321 vertices. For hand meshes from the MANO dataset, the input and output meshes are all with 778 vertices.

Driving Sequence Canonicalization. To unify the sequence length and world coordinates, we canonicalize DFAUST and AMASS datasets, as follows: First, we downsample the long sequences of original motions from 120fps to 12fps and then trim them into 60-frame (2.5-second) subsequences. Second, we unify the world coordinates as in [15]. For each subsequence, we reset the world coordinates by shifting all the input meshes to the center and conducting the motion transfer to the target mesh, then compensate for the global translation to each frame after the motion transfer.

Licenses of the Assets. The licenses of the assets used in this paper are shown in Table 5. Their licenses are given on the websites.

3 Experimental Implementation

Our algorithm is implemented in PyTorch [9]. All the experiments are carried out on a PC with a single NVIDIA Tesla V100, 32GB. We train our networks for 200 epochs with a learning rate of 0.00005 and the Adam optimizer. The weight settings in the paper are λ_{rec}=1, λ_{edge}=0.0005. The weight settings directly follow the previous work [15]. For the weight λ_{metric}, we follow the setting in LIMP [6] and set it as 1.0. The batch size is fixed as 2 for all the settings and the frame number is 3. Training time is around 80-90 hours. Note that a batch size of 2 is only available with 32GB memory GPUs to run the LART. For GPUs with 12 or 24GB memory, the batch size should be adjusted to 1.

Average Inference Times. In this section, we compare the average inference times for every pose transfer of different methods in the same experimental settings. As shown in Table 6, for [15], they do not learn the correspondence between meshes, so they have the shortest inference time, but the generation performance is degraded. 3D-CoreNet [13] achieves notable improvements in generating high-quality results. LART also learns the correspondence but in an implicit way, resulting in a faster processing speed.

4 Experiments Results

In this section, we perform an extra experimental evaluation of the proposed LART, including extra visualized results and an ablation study.

4.1 Visualized Results

We use LART to achieve motion transfer with driving motion sequences of different noisy levels, see Figure 1. The noisy sequences are obtained by adding Gaussian noise to every vertex (scale as 0.1, 0.01 and 0.005, mean as 0.0). We can see that when it comes to a high noisy level (0.1), the pose is not identical to the original one. But the output sequence still shows acceptable visual results.

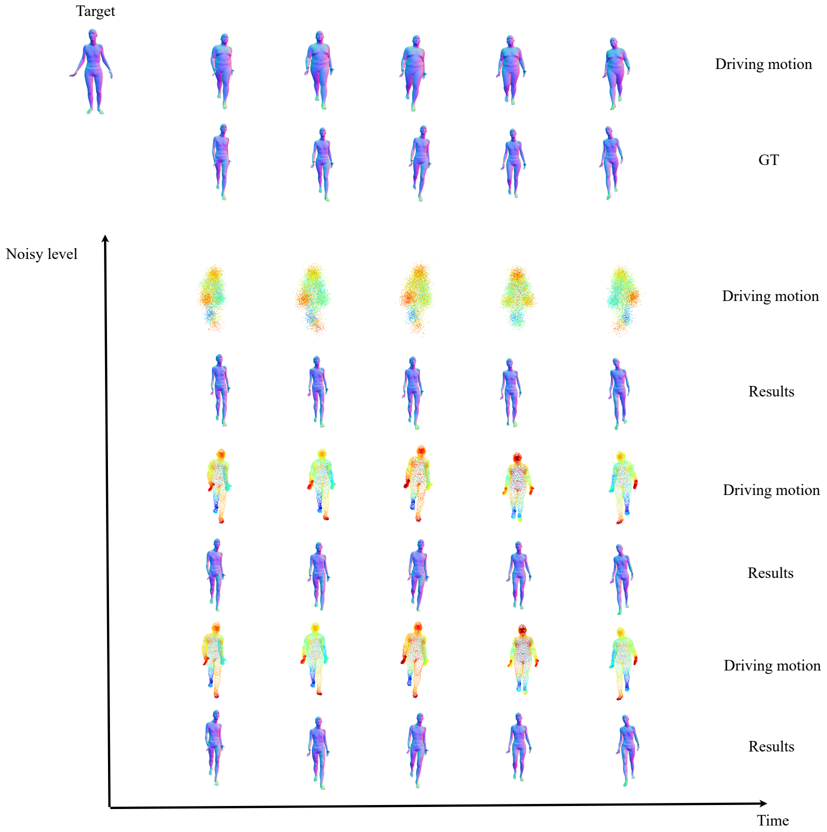

Figure 1: Qualitatively comparison of driving motions with different noises as inputs.

4.2 Ablation Study

We conducted the ablation study to verify the configuration of the designed architecture. The ablation study is evaluated on the part of the DFAUST dataset.

Inference speed. We first compare the inference speed of LART with other methods in Table 6. The time is obtained by taking the per-frame processing duration.

Cross-Attention Scheme. We then verify different attention scheme in Table 7, by setting different latent codes into $\mathbf{q}, \mathbf{k}, \mathbf{v}$.

Positional Embedding Scheme. Besides, we verify different positional embedding schemes as below. As we see in Table 8, the embedding will steadily bring gains to the performance. With the fixed embedding method, the result is the best (0.00067) but it cannot be adapted to a flexible target mesh size. Meanwhile, adaptive embedding is a good trade-off between performance and adaption.

Correspondence learning. Lastly, we demonstrate the quantitative evaluation by comparing the reconstruction error of our work LART with other methods to showcase correspondence learning ability. To do so, we use one subject, 50027 from the DFAUST dataset, to conduct reconstruction and evaluate the correspondence learning ability via point-wise comparison with PMD. Results are shown in Table 9, demonstrating the correspondence learning ability of our LART.

Table 6: Average inference times of different methods.

Method	NPT [15]	3D-CoreNet [13]	LART (Ours)
Time	0.0068s	0.0131s	0.0118s

Table 7: Difference cross-attention scheme.

Method	q	k	v	Loss
w/o attention	-	-	-	0.00224
self-attention	Z_{pose}	Z_{pose}	Z_{pose}	0.00185
cross-attention	Z_{id}	Z_{pose}	Z_{pose}	0.00145
cross-attention	Z_{pose}	Z_{id}	Z_{id}	0.00084

Table 8: Difference embedding scheme.

Method	Target mesh size	Loss
w/o embedding	Flexible	0.00317
adaptive embedding	Flexible	0.00102
fixed embedding	Fixed	0.00067

References

[1] Bharat Lal Bhatnagar, Garvita Tiwari, Christian Theobalt, and Gerard Pons-Moll. Multi-garment net: Learning to dress 3d people from images. In *ICCV*, 2019.

[2] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J Black. Keep it smpl: Automatic estimation of 3d human pose and shape from a single image. In *ECCV*, 2016.

[3] Federica Bogo, Javier Romero, Matthew Loper, and Michael J Black. Faust: Dataset and evaluation for 3d mesh registration. In *CVPR*, 2014.

[4] Federica Bogo, Javier Romero, Gerard Pons-Moll, and Michael J Black. Dynamic faust: Registering human bodies in motion. In *CVPR*, 2017.

[5] Haoyu Chen, Hao Tang, Zitong Yu, Nicu Sebe, and Guoying Zhao. Geometry-contrastive transformer for generalized 3d pose transfer. *AAAI*, 2021.

[6] Luca Cosmo, Antonio Norelli, Oshri Halimi, Ron Kimmel, and Emanuele Rodolà. Limp: Learning latent shape representations with metric preservation priors. *ECCV*, 2020.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, 2013.

[8] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. Amass: Archive of motion capture as surface shapes. In *ICCV*, 2019.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[10] Sida Peng, Junting Dong, Qianqian Wang, Shangzhan Zhang, Qing Shuai, Xiaowei Zhou, and Hujun Bao. Animatable neural radiance fields for modeling dynamic human bodies. In *ICCV*, 2021.

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

[12] Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied hands: Modeling and capturing hands and bodies together. *TOG*, 36(6), 2017.

[13] Chaoyue Song, Jiacheng Wei, Ruibo Li, Fayao Liu, and Guosheng Lin. 3d pose transfer with correspondence learning and mesh refinement. *Advances in Neural Information Processing Systems*, 34, 2021.

[14] Robert W Sumner and Jovan Popović. Deformation transfer for triangle meshes. *TOG*, 23(3):399–405, 2004.

Table 9: Correspondence learning evaluation.

Method	NPT	NPT-MP	3D-CoreNeT	LART (Ours)
PMD↓ $(\times 10^{-4})$	1.14	1.99	0.80	**0.27**

[15] Jiashun Wang, Chao Wen, Yanwei Fu, Haitao Lin, Tianyun Zou, Xiangyang Xue, and Yinda Zhang. Neural pose transfer by spatially adaptive instance normalization. In *CVPR*, 2020.