SUPPLEMENTARY
LIMP: Learning Latent Shape Representations with Metric Preservation Priors

Luca Cosmo[1,2], Antonio Norelli[1], Oshri Halimi[3], Ron Kimmel[3], Emanuele Rodolà[1]

[1] Sapienza University of Rome, Italy
[2] University of Lugano, Switzerland
[3] Technion - Israel Institute of Technology, Israel

Abstract. This supplementary material includes technical details and additional results that were excluded from the main submission due to lack of space. All the results are obtained with exactly the same methodology as described in the main manuscript. This document is also accompanied by a video, showing animated examples of disentanglement and interpolation in the latent space. We strongly encourage visualization of the accompanying video[4].

1 Implementation details

1.1 Architecture & optimization

Our architecture takes the form of a variational autoencoder (VAE) [5] operating on point clouds. The dimension of the bottleneck changes depending on the dataset, as described below, the rationale being that different shape classes have different kinds of variability; for example, human facial expressions tend to be less articulated than animals changing their pose.

The Encoder is based on the PointNet [8] architecture. The initial x, y, z coordinates are given as input to a Spatial Transform (ST) module, followed by few layers of PointNet Convolution. Max-pooling throughout point features is then applied to obtain a global shape feature which is projected onto the latent space via a MLP. The output of the Encoder is actually twice the size of the latent space since it predicts both mean and variance of the multivariate Gaussian distribution from which the latent code is sampled.

The Decoder is a simple MLP, transforming the input latent space vector to the corresponding triangular mesh embedding in $\mathbb{R}^3$. We report in Table 1 the layer dimensions for each dataset.

Optimization. Our loss is composed additively by three terms:

$$\ell(\mathcal{S}) = \ell_{\text{recon}}(\mathcal{S}) + \ell_{\text{interp}}(\mathcal{S}) + \ell_{\text{disent}}(\mathcal{S})\,. \tag{1}$$

[4] https://youtu.be/jSszaaMTg9Q

Table 1. Detailed description of layers size used for each dataset.

	FAUST, DFAUST	COMA	TOSCA, HANDS
ENCODER			
ST (Conv)	64,128,512	-	-
ST (MLP)	512,256	-	-
PointNet (Conv)	32,128,256,512	64, 128	64,64,128
PointNet (MLP)	512,258,128,256	128, 64, 64	128,64,64,64
LATENT SPACE ($\mathbf{z}^{\text{int}}$|$\mathbf{z}^{\text{ext}}$)	128 (96|32)	32 (24|8)	32
DECODER	128,1024,2048,$N \times 3$	32,256,$N \times 3$	32,64,128,$N \times 3$

The interpolation and disentanglement terms involve the computation of geodesic distances, which might get unstable if the underlying surface is self-intersecting or degenerate. For this reason, we disable ℓ_{interp} and ℓ_{disent} for the first 10^4 training epochs. The action of the reconstruction loss alone provides an initial guess for the decoded shape; once this is in place, we re-activate the two missing losses and complete the training process.

Training times. One crucial point of our method is the backpropagation through all pairwise geodesic distances during training. Standard fast-marching algorithms are usually not differentiable. To overcome this limitation we use a loss based on intrinsic distances, leveraging a very efficient (and differentiable) way of calculating the geodesics, i.e. the heat method [4]. Its computational complexity is guaranteed to be sub-quadratic in the total number of vertices , and an efficient implementation using sparse matrices has near-linear cost. In practice, our distances are computed one order of magnitude faster than a state-of-the-art fast marching implementation. Once computed, we have to keep distance values in memory. In our system, with an Intel Xeon E5-2620v4 and one Nvidia Titan Xp, it takes around 4 hours to train on FAUST dataset and around 2.67e-3 seconds for inference (2.44e-3 for encoding and 2.19e-4 for decoding). The runtime cost for the computation of the loss is dominated by the calculation of geodesic distances (80%), which are involved in the interpolation and disentanglement terms of the loss.

1.2 Geodesic distance calculation

Our loss involves the calculation of geodesic distances between all points of a given decoded shape $\mathbf{X}$. In particular, the distance calculation must be differentiable with respect to the vertex positions encoded in $\mathbf{X}$. To this end, we employ the geodesics in heat algorithm introduced in [4]. The calculation involves three steps:

1. Solve the linear system $(\mathbf{I} - t\mathbf{L}(\mathbf{X}))\mathbf{u} = \mathbf{u}_i$
2. Compute the normalized gradient field $\mathbf{U} = -\frac{\mathbf{G}(\mathbf{X})\mathbf{u}}{\|\mathbf{G}(\mathbf{X})\mathbf{u}\|}$
3. Solve the linear system $\mathbf{L}(\mathbf{X})\mathbf{d}_i = \mathbf{D}(\mathbf{X})\mathbf{U}$

In step (1) the time parameter is given and fixed to $t = 10^{-1}$, while $\mathbf{u}_i$ is an indicator vector at point i. The solution vector $\mathbf{d}_i$ in step (3) contains the geodesic distance from point i to all other points in $\mathbf{X}$.

The matrices $\mathbf{D}(\mathbf{X})$, $\mathbf{G}(\mathbf{X})$, and $\mathbf{L}(\mathbf{X})$ involved in these steps represent, respectively, the *Laplacian*, *gradient*, and *divergence* operators over a mesh with vertices $\mathbf{X}$; the coordinates in $\mathbf{X}$ change at each forward step during training.

Remark. This is the only point in our pipeline where a mesh structure is needed, although we remark that *any* mesh can be used (and in particular, different meshes for different shapes are allowed); these operators can also be defined directly on point clouds [4, Sec. 3.2.3], although under some regularity condition on the point density.

The standard formulas that define $\mathbf{D}(\mathbf{X})$, $\mathbf{G}(\mathbf{X})$, and $\mathbf{L}(\mathbf{X})$ in terms of $\mathbf{X}$ can be found, for instance, in [4, Sec. 3.2.1]. Apart from some degenerate cases (which we rule out by the incremental optimization of Sec. 1), all the operations involved in their definition are differentiable with respect to $\mathbf{X}$.

2 Additional results

We present here some additional results and comparisons that could not fit in the main manuscript due to space limitations.

2.1 Effect of the geodesic prior

We first illustrate how the metric preservation prior under the *geodesic* metric induces realistic deformations, and therefore helps in learning a well behaved generative model for deformable objects. To this end, we designed the following simple experiment. We modify the loss of Eq. (1) by just keeping the geometric reconstruction term; then, in this reconstruction loss, we use the *geodesic* distance between all points instead of the Euclidean distance. We then run the following optimization for the example shown in the inset figure:

1. Compute the geodesic distances $\mathbf{D}_{\text{tar}}$ for the leftmost shape;
2. Initialize the optimization with the middle shape;
3. Optimize for a shape having geodesic distances $\mathbf{D}_{\text{opt}} \approx \mathbf{D}_{\text{tar}}$.

The result is a shape having approximately the same metric as the target ball, but with a different embedding in $\mathbb{R}^3$; hence the deflated effect. Therefore, by prescribing the geodesic metric one gets a type of regularization that avoids excessive surface stretching and compression, while promoting limited distortion in the reconstruction (decoding). In the accompanying **video**, we show a comparison between the adoption of metric priors vs. no priors, demonstrating that the former lead to better synthesis of novel shapes.

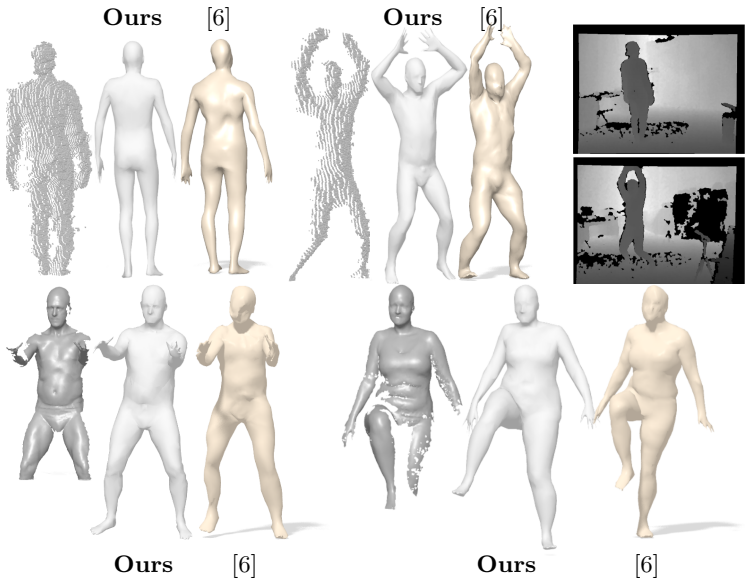

Fig. 1. Examples of deformable shape completion obtained with our method and with the method of Litany et al. [6]. *Top row*: Completion of real Kinect scans from the MHAD dataset [7]; we show the original 2.5D frames on the right. *Bottom row*: Completion of scans from the DFAUST dataset [3].

2.2 Shape completion of real scans

One straightforward application of our generative model is *deformable* shape completion. Given a partial view of a deformable shape, the task is to complete this partial view in a semantically plausible manner. For this task, we follow verbatim the pipeline of Litany et al. [6], but we replace their VAE with ours. The completion procedure consists in searching over the learned latent space for a latent vector which, once decoded, produces a shape that has a close overlap with the input part. We implemented this idea, where we measured the quality of the overlap by the asymmetric Chamfer distance between part and completion.

The results are shown in Figure 1, where we compare directly with the method of [6] over *real scans* from the MHAD dataset [7] (Kinect) and from the DFAUST dataset [3]. The training data for this experiment consists of **just 4 representative frames** for each subject/sequence pair, resulting in < 500 training shapes in total; this is opposed to Litany et al. [6], which is trained on a dataset of ~ 7000 shapes.

2.3 Sampling of the latent space

In Figure 2 we show random samples from our learned latent space, trained on 80 shapes of the FAUST dataset [2]. The sampling is done according to a normal distribution with standard deviation equal to **3 times** the one of the training set,

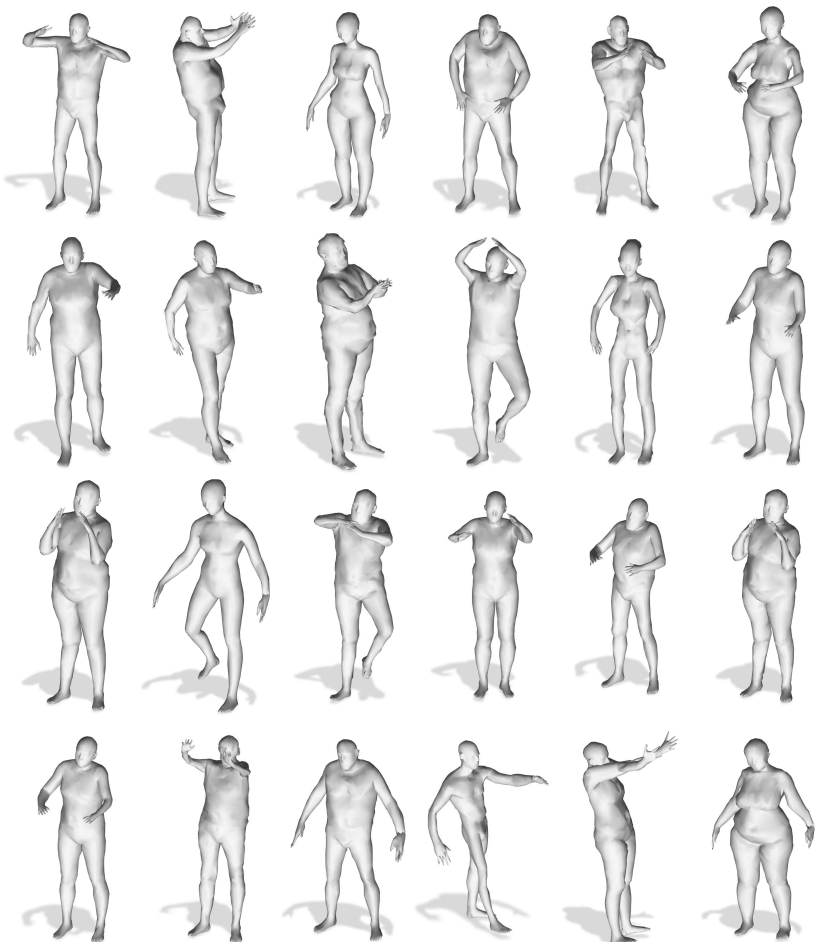

Fig. 2. Random samples from a latent space trained on FAUST. We sample a normal distribution with large standard deviation, thus going outside of the "known region" seen at training time. This generates individuals with new and sometimes extreme body features, although always within a bounded metric distortion as imposed by our priors.

thus going well beyond the span of the known shapes, allowing us to qualitatively evaluate how much our metric priors help to regularize the generative process.

2.4 Comparison with the state of the art

We show additional comparisons with the recent state-of-the-art method of Aumentado-Armstrong et al. [1], in terms of shape interpolation and disentanglement. For this experiment, we trained both methods on the same 80 FAUST shapes. The two training shapes used as end-points for the interpolation are shown on the top-left and bottom-right of Figure 3.

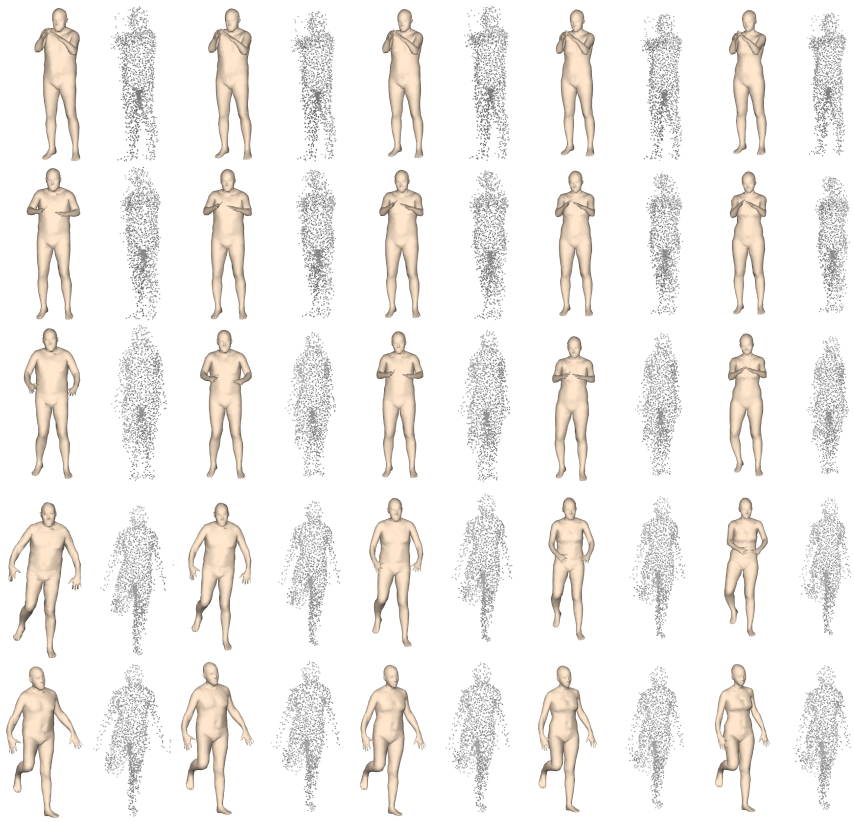

Fig. 3. Comparison with the disentanglement method of [1] on FAUST data. The yellow meshes are generated with our model, while the gray point clouds are obtained with [1]. Style changes horizontally, while pose changes vertically.

References

1. Aumentado-Armstrong, T., Tsogkas, S., Jepson, A., Dickinson, S.: Geometric disentanglement for generative latent shape models. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 8181–8190 (2019)
2. Bogo, F., Romero, J., Loper, M., Black, M.J.: FAUST: Dataset and evaluation for 3D mesh registration. In: Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). IEEE, Piscataway, NJ, USA (Jun 2014)
3. Bogo, F., Romero, J., Pons-Moll, G., Black, M.J.: Dynamic FAUST: Registering human bodies in motion. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (Jul 2017)
4. Crane, K., Weischedel, C., Wardetzky, M.: Geodesics in heat: A new approach to computing distance based on heat flow. ACM Trans. Graph. **32**(5) (Oct 2013)
5. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)

6. Litany, O., Bronstein, A., Bronstein, M., Makadia, A.: Deformable shape completion with graph convolutional autoencoders. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1886–1895 (2018)
7. Ofli, F., Chaudhry, R., Kurillo, G., Vidal, R., Bajcsy, R.: Berkeley mhad: A comprehensive multimodal human action database. In: 2013 IEEE Workshop on Applications of Computer Vision (WACV). pp. 53–60. IEEE (2013)
8. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)