Memory-Scalable and Simplified Functional Map Learning

Supplementary Material

7. Implementation Details

In this section we provide more detailed information on the implementation of our model described in Figure 2 and Sec. 4.3.

Our model takes as input shapes using 128 WKS descriptors computed from 128 eigenfunctions of the Laplace-Betlrami operator. Similarly to [9, 23], each descriptor function is normalized on the shape with respect to the standard L^2 inner product on a mesh. These descriptors are then fed to a DiffusionBlock with 4 Diffusion blocks of width of 256, in a standard manner [2, 9, 43, 46]. The main difference with these implementations is that we only output 32 feature functions instead of the 128 or 256 usually used.

Features produces by DiffusionNet are used in our Differentiable ZoomOut block which first normalizes the pointwise features, and then computes a scalable dense map equivalent to standard two-branch networks, as shown in Section 3 and Eq. (3). Using this map, an initial functional map $\mathbf{C}_{\text{init}}$ of size $K_{\text{init}} = 30$ is computed, and is fed into a ZoomOut algorithm [30] for 10 iterations with a spectral upsampling step of 10, where the pointwise maps are replaced by our scalable dense maps. This eventually produces our refined map $\mathbf{C}_{\text{refined}}$ of size $K_{\text{refined}} = 130$.

Our loss consists in 3 terms, a orthogonality loss $L_{\text{orth}}(\mathbf{C}_{\text{init}}) = \|\mathbf{C}_{\text{init}}^\top \mathbf{C}_{\text{init}} - I\|_2^2$, a consistency loss $L_{\text{consist}}(\mathbf{C}_{\text{init}}, \mathbf{C}_{\text{refined}}) = \|\mathbf{C}_{\text{init}} - \mathbf{C}_{\text{refined}}\|_2^2$, and a Laplacian bijectivity loss $L_{\text{lap}}(\mathbf{C}_{\text{init}}) = \|\Delta \odot \mathbf{C}_{\text{init}}\|_2^2$, where $\odot$ denotes element-wise product and Δ is obtained from [8, 37]. More precisely, if $\lambda^{(1)}, \lambda^{(2)} \in \mathbb{R}^{K_{\text{init}}}$ denote the vector of eigenvalues of S_1 and S_2, then Δ is defined element-wise as

$$\Delta_{ij}^2 = \left(\frac{\sqrt{\lambda_i^{(2)}}}{1+\lambda_i^{(2)}} - \frac{\sqrt{\lambda_j^{(1)}}}{1+\lambda_j^{(1)}}\right)^2 + \left(\frac{1}{1+\lambda_i^{(2)}} - \frac{1}{1+\lambda_j^{(1)}}\right)^2 \tag{7}$$

This is an extension of the standard Laplacian commutativity loss, which has been used in most existing implementations since GeoFMaps [12].

We do not enforce orthogonality of $\mathbf{C}_{\text{refined}}$, since ZoomOut is proven to promote orthogonal functional maps [30]. We notice that given a sound initialization, ZoomOut produces great results, which inspires us mostly to penalize $\mathbf{C}_{\text{init}}$. Furthermore, during the first iterations, initial functional maps produced by the network have no guarantee to be sound, and we therefore tune down the consistency loss L_{consist} initially until the network converges towards good initialization. The consistency loss then provides meaningful guidance to the network. In practice, we increase the weight of this loss from 10^{-4} to 10^{-1} in 5 epochs using a multiplicative schedule.

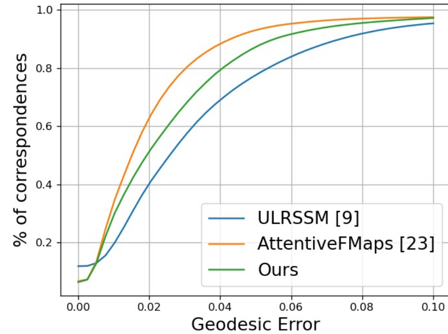

Figure 6. PCK curves on the SMAL dataset

The complete implementation is available at https://github.com/RobinMagnet/SimplifiedFmapsLearning.

8. More Baselines & Ablation

We here present additional results on the standard baselines presented in the manuscript. In particular, some works [23, 46] provided multiple versions of their algorithm. Furthermore, we display results from [9] using further test-time optimization. Note that this test-time optimization fine-tunes the network for each shape on the test set and should be applied to all other methods for fairness. All these additional baselines can be found on Table 4.

We additionally provide results on the SMAL dataset [52], where we additionally show the result of our pipeline without using the consistency loss ("w/o Consistency"), which serves as an ablation study similar to the one presented in [46]. However, in this ablation, we still use the ZoomOut algorithm at test-time, only the consistency loss was removed. PCK curves for ULRSSM [9] and AttentiveFMaps [23] are also provided on Figure 6.

9. ZoomOut Algorithm [30]

The ZoomOut algorithm [30] is a simple functional map refinement algorithm, which uses iterative conversions between functional and pointwise maps.

Train	F			S			F+S		
Test	F	S	S19	F	S	S19	F	S	S19
BCICP [38]	6.1	-	-	-	11.	-	-	-	-
ZoomOut [30]	6.1	-	-	-	7.5	-	-	-	-
SmoothShells [16]	2.5	-	-	-	4.7	-	-	-	-
DiscreteOp [39]	5.6	-	-	-	13.1	-	-	-	-
GeomFmaps [12]	3.5	4.8	8.5	4.0	4.3	11.2	3.5	4.4	7.1
Deep Shells [17]	1.7	5.4	27.4	2.7	2.5	23.4	1.6	2.4	21.1
NeuroMorph [18]	8.5	28.5	26.3	18.2	29.9	27.6	9.1	27.3	25.3
DUO-FMNet [14]	2.5	4.2	6.4	2.7	2.6	8.4	2.5	4.3	6.4
UDMSM [8]	**1.5**	7.3	21.5	8.6	2.0	30.7	1.7	3.2	17.8
ULRSSM [9]	1.6	6.4	14.5	4.5	**1.8**	18.5	**1.5**	**2.0**	7.9
ULRSSM (w/ fine-tune) [9]	1.6	2.2	5.7	1.6	1.9	6.7	1.6	2.1	4.6
AttentiveFMaps Fast [23]	1.9	2.6	5.8	1.9	2.1	8.1	1.9	2.3	6.3
AttentiveFMaps [23]	1.9	2.6	6.4	2.2	2.2	9.9	1.9	2.3	5.8
ConsistentFMaps [46]	2.3	2.6	**3.8**	2.4	2.5	**4.5**	2.2	2.3	4.3
ConsistentFMaps (dim 80) [46]	1.7	2.6	5.5	2.2	2.0	5.8	1.7	2.2	5.6
Ours	1.9	**2.4**	4.2	**1.9**	2.4	6.9	1.9	2.3	**3.6**

Table 4. Mean geodesic errors (×100) when training and testing on the Faust, Scape and Shrec19 datasets. Due to the fine-tuning strategy on ULRSSM (w/ fine-tune), we do not hightlight its results. See text for details.

Method	SMAL
ULRSSM [9]	6.9
ULRSSM (w/ fine-tune) [9]	**3.5**
AttentiveFMaps [23]	5.4
ConsistentFMaps [46]	5.4
Ours (w/o Consistency)	6.7
Ours	5.9

Table 5. Mean geodesic errors (×100) when training and testing on the SMAL dataset.

Algorithm 1 The ZoomOut algorithm

Require: Initial pointwise map $\Pi_{21} \in \{0,1\}^{n_2 \times n_1}$ from S_2 to S_1, eigenvectors $\Phi_1 \in \mathbb{R}^{n \times k_1}$ and $\Phi_2^{n \times k_2}$ on each shape.

1: **for** $k = k_{\text{init}}$ to k_{final} **do**
2: Compute $C_{12} = [\Phi_2]^{\dagger}_{[:,:k]} \Pi_{21} [\Phi_1]_{[:,:k]}$
3: Compute $\Pi_{21} = \text{NN}\big([\Phi_1]_{[:,:k]} C_{12}^{\top}, [\Phi_2]_{[:,:k]}\big)$
4: **end for**
5: **Return** C_{12}, Π_{21}

The algorithm is presented on Algorithm 1, where NN denotes the nearest neighbor query between the rows of the two arguments.

10. Adapting Scalable ZoomOut [26]

In [26], the authors present an approximation of the functional map for dense shapes using only sparse samples.

This approximation allows running the ZoomOut algorithm on a sparse subset of the vertices of both shapes, only using a complete high dimensional nearest neighbor query at the last step of the algorithm. This last step appears as the heaviest speed bottleneck of the algorithm as presented in [26].

When porting [26] to GPU, this query makes the GPU run out of memory on very dense meshes, which we solve by using our scalable dense maps.

However, this algorithm adds a layer of approximation, which can potentially hinder the results. Furthermore, since it only uses values at sparse samples, the gradient can only propagates through these samples and not to the entire vertex-wise embeddings. In particular, it is not possible to use different samples each time the shape is used in training, as the preprocessing time is not negligible. This refrains us from using this adapted version within our learning framework.

11. Dense Meshes

In this section, we provide more information on dense mesh processing using our pipeline, using meshes from the original version of the SHREC19 dataset [29].

While DiffusionNet needs to store the eigenvectors of each shape of size $N \times K$ in memory, it is still able

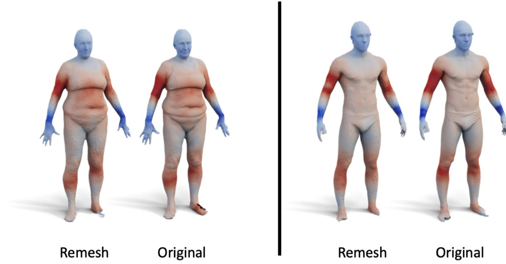

Figure 7. We leverage on the capacity of DiffusionNet [43] to perform on various discretization of the same shape. Left and right are two shapes from the SHREC19 dataset [29]. We show on each shape features obtained on the remeshed and original version of the dataset.

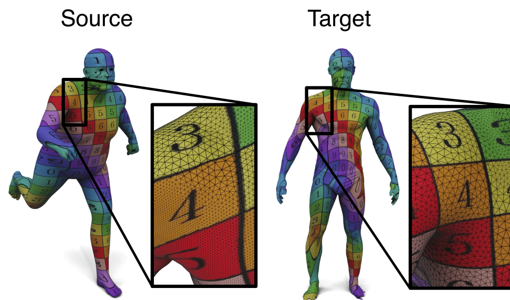

Figure 8. Example of texture transfer of our method on the SHREC19 [29] dataset using our pipeline.

to compute features quickly for each shape. Due to its discretization-agnostic architecture, the features obtained on the dense and remeshed version are similar, as noted on Figure 7, where each mesh contains $N = 2 \cdot 10^5$ vertices. However, fitting a dense pointwise map would for this mesh require 10^7 MiB of GPU memory, without even storing the gradient, which is infeasible in most cases.

In contrast, our scalable dense map can easily compute these maps. In particular, at test time when no gradient information is stored, our DifferentiableZoomOut has a negligible memory cost since intermediate maps don't need to be stored.

We show an example of texture transfer on another pair of this dataset in Figure 8. Here, we used our network, trained on the standard remeshed [38] versions of the Faust [5] and Scape [1] datasets, and evaluate at test time on shapes with around 10^5 vertices. We transform the output functional map into a precise map [19]. This demonstrates our pipeline can be trained on simple remeshed versions of datasets, but then used at test time on denser shapes without issues.