Sparse Localized Deformation Components
Supplementary Supporting Document

Thomas Neumann[1,2], Kiran Varanasi[3,4], Stephan Wenger[2], Markus Wacker[1], Marcus Magnor[2], Christian Theobalt[3]

[1]HTW Dresden, Germany, [2]Computer Graphics Lab, TU Braunschweig, Germany,
[3]Max-Planck-Institut Informatik, Saarbrücken, Germany, [4]Technicolor Research, France

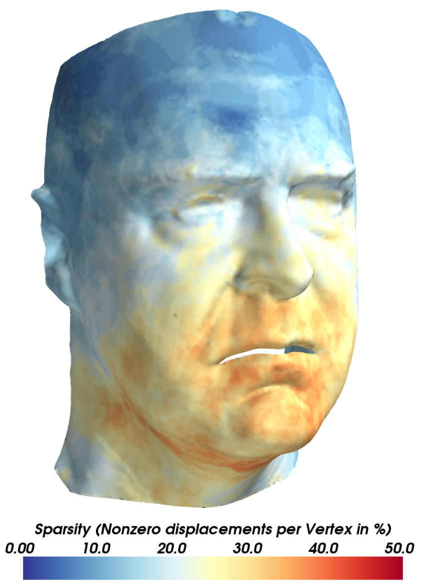

Figure 1: *Sparsity of our proposed deformation components. Each vertex is colored according to the number of components where it's displacement is non-zero. Compared to the fixed sparsity of previous decomposition methods such as [Kavan et al. 2010], our approach automatically finds a suitable sparsity level. Regions showing complex deformations, like the mouth, require more deformation components.*

Abstract

This supplementary document provides further visualizations, implementation details and analysis of convergence of our method.

1 Method - Implementation Details

Visualizing spatially varying sparsity Our method produces a set of sparse deformation components by analyzing an input mesh animation. Compared to methods like [Tena et al. 2011] and [Kavan et al. 2010], which in principle can also be viewed as sparse decompositions, our method finds the required sparsity and region segmentation automatically using a sparsity imposing regularization term. The other methods have a fixed sparsity per vertex. Fig. 1 visualizes the number of components that affect each vertex and thus gives a measure of sparsity, that is spatially varying on the mesh surface and indicates how complex the motions of different vertices are. The sparsity is low in highly deformable regions around the mouth (the subject is mostly speaking in the input animation) where many components are required to reconstruct the input.

Choice of ℓ_1/ℓ_2 minimization algorithm Updating the sparse components $\mathbf{C}$ involves optimizing a sum of a smooth continuously differentiable data term plus the non-smooth convex regularizer Ω

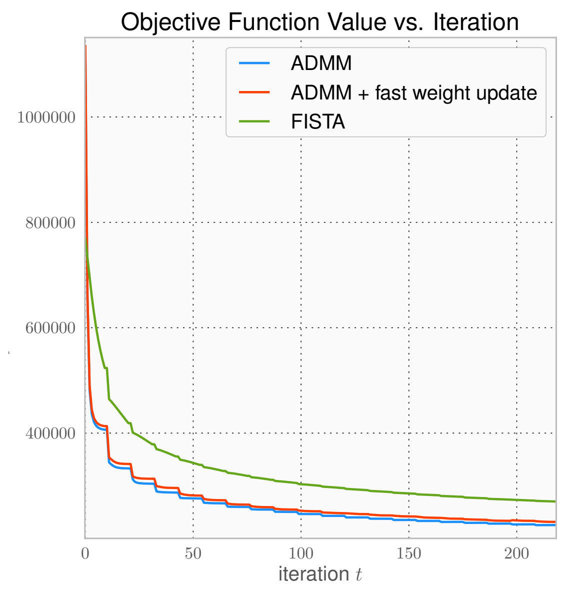

Figure 2: *Convergence: value of Objective function, Eq. (2), vs. Iteration (Dataset: [Zhang et al. 2004]). Different ℓ_1 minimizers (ADMM - blue, FISTA - green) for optimizing the components are compared. The minimization of the components is interweaved with the optimization of the weights using block-coordinate-descent every 10 iterations, which results in the step-like convergence curve. Notice that normally, the objective function is not computed during ADMM/FISTA iterations. When using only a random selection of 1% of the vertices for block-coordinate descent for fast weight update (red), the convergence is almost as good but significantly reduces computation time.*

based on the ℓ_1/ℓ_2 norm. We compare the convergence behavior of FISTA [Beck and Teboulle 2009] and ADMM [Boyd et al. 2011] in Fig. 2 which shows the same number of 10 interleaved component optimization steps between updating support maps and weights. Per iteration, both methods solve the proximal mapping of the ℓ_1/ℓ_2 norm; FISTA requires computing the gradient at each iteration (a simple matrix multiplication in our case) while ADMM requires inverting a linear system which can be pre-factorized using Cholesky decomposition. As preprocessing, FISTA requires estimation of the Lipschitz constant of the gradient of the data term (usually, few iterations of power iteration suffice to obtain it), and ADMM requires pre-factorizing the system to be solved at each iteration using Cholesky decomposition. In practice, we observed that the running time of ADMM iterations and preprocessing is a bit slower but the convergence is much better, so we choose ADMM for this implementation. FISTA requires computing the gradient of the data term while ADMM requires solving the proximal mapping for the data term. This is easy in our case but in the future, for more advanced dataterms, we suggest researchers to switch to

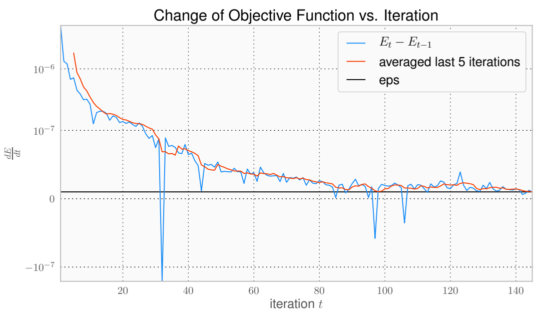

Figure 3: *Due to the non-convexity and the added objective of spatially varying regularization, the objective function does not decrease all the time, because the support maps are changing. This makes monitoring convergence tricky. Here we plot the change in objective function over iterations. The blue line is the change after each iteration, while the red line averages the changes over the last 5 iterations. We use this value (red line) to compare to a threshold (eps, black line) to check for convergence.*

FISTA should their data term become more complicated. Both algorithms require only a few lines of Python or Matlab code.

Accelerating weight optimization For datasets containing very densely sampled meshes featuring tens of thousands of vertices or more, the block coordinate descent step to update the weights $\mathbf{W}$ takes considerable amount of computation time, because the update of the residual is an notably big matrix multiplication. This can be acellerated by only selecting a subset of vertices for weight update. Since the components likely involve many vertices for such big meshes, the weights are over-determined even by a subset of the vertices. As can be seen in Fig. 2, even using only 1% of vertices for the weight update (blue line) achieves almost the same convergence as the full update (red line).

2 Deformation Representation for Rotations

As mentioned in the paper (Discussion) our algorithm cannot model articulated rotations perfectly, since the sparse components model vertex displacements. In this supplementary document, we want to give an illustrating example for this limitation. In Fig. 4, we show the sparse deformation components extracted automatically by our method from a galopping horse sequence [Sumner et al. 2007]. It can be seen that even with simple vertex displacement encoding, our method can extract and segment meaningful parts from this animation that correspond to the limbs of the horse. However, linear combination of these components is not suited for for modeling curvilinear paths of the limbs due to articulated rotation of joints. Since the path of the leg cannot be described by a linear path, multiple components are required to fit it, and some components show a shrinking of the leg on its path. This limitation can be adressed in future work by evaluating sparse localized deformation components on different rotation-invariant deformation encodings such as deformation gradients or intrinsic shape parametrizations. At present, our method requires a prefactoring step where such pose related deformations are separated out, and the residual deformations can be nicely parameterized by our method. We acknowledged this limitation in the paper.

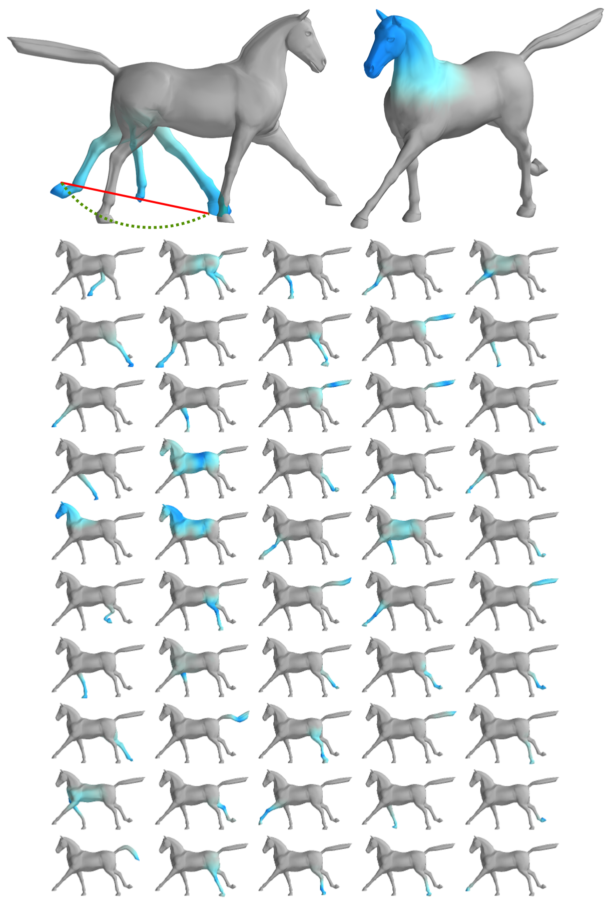

Figure 4: *Limitations: Our approach extracts linear components representing vertex displacements. This is not suitable for animations involving rotations (left). In this case, the linear path (red) is extracted for the leg motion instead of the green path. However, the motion of the teetering head (right) is captured well by our method, and the the segmentation into body parts is very convincing even using vertex displacements as shape encoding.*

References

BECK, A., AND TEBOULLE, M. 2009. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM J. Imaging Sci. 2*, 1.

BOYD, S., PARIKH, N., CHU, E., PELEATO, B., AND ECKSTEIN, J. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn. 3*, 1.

KAVAN, L., SLOAN, P.-P., AND O'SULLIVAN, C. 2010. Fast and efficient skinning of animated meshes. *Comp. Graph. Forum (Proc. EG) 29*, 2.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3.

TENA, J. R., DE LA TORRE, F., AND MATTHEWS, I. 2011. Interactive region-based linear 3D face models. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4.

ZHANG, L., SNAVELY, N., CURLESS, B., AND SEITZ, S. 2004. Spacetime faces: High-resolution capture for modeling and animation. *ACM Trans. Graph. (Proc. SIGGRAPH)*.