Supplementary Material

Remeshing Algorithm

At first, a coarse approximation of the input mesh is built. To coarsen the surface meshes we employ the Garland-Heckbert-algorithm for surface simplification using quadric error metrics [13]. It simplifies the mesh by collapsing edges until the target number of faces is reached, always contracting the pair of edges with the lowest cost. The cost measures the shape changes. As in [46], we regularize the edge lengths to have regularly distributed vertices. Additionally, we prevent the algorithm from contracting edges, that lead to non-manifold edges in the mesh.

The coarse approximation of the input mesh is subdivided to the desired level of subdivision.

Now, the resulting semi-regular mesh $\mathcal{M}_{SR}$ has to be fit to the original irregular mesh $\mathcal{M}_{IR}$ in order to describe the surface well.

We chose stochastic gradient descent to optimize a loss function that describes how well the semi-regular mesh fits to the irregular mesh. The loss function is optimized with respect to a deformation vector, that contains a 3D offset for each vertex of the semi-regular mesh $\mathcal{M}_{SR}$. The loss function employs the average chamfer distance between sampled points S_{IR} from the surfaces described by the original mesh $\mathcal{M}_{IR}$ and sampled points S_{SR} from the iteratively deformed semi-regular mesh M_{SR} respectively. We use the following definition of the chamfer distance [1] which measures the average squared distance between each point in set S_{IR} to its nearest neighbor in the other set S_{SR}.

$$d_{avgCD}(S_{IR}, S_{SR}) = \frac{1}{|S_{IR}|} \sum_{x \in S_{IR}} \min_{y \in S_{SR}} \|x - y\|_2^2 + \frac{1}{|S_{SR}|} \sum_{y \in S_{SR}} \min_{x \in S_{IR}} \|x - y\|_2^2$$

Additionally, we regularize the lengths of the edges of $\mathcal{M}_{SR}$, smooth the Laplacian of $\mathcal{M}_{SR}$ and enforce consistency across the normals of neighboring faces of $\mathcal{M}_{SR}$. The regularization terms are weighted. To fit the semi-regular mesh to the original irregular mesh we utilize an implementation in Pytorch3D [37] that is based on [37][2].

Note that if for a deforming shape the mesh topology stays constant over time, one can just remesh one undeformed template mesh. The semi-regular remeshing result is parameterized and transferred to the meshes at the different timesteps, which describe the same shape. For that, after projecting the vertices of the semi-regular mesh to the closest face of the irregular template mesh, we can calculate the barycentric coordinates and obtain a parametrization. This parametrization of the remeshing result can be applied to the other deformed meshes and the complete sequence of the deforming shape is discretized by semi-regular meshes. Note that this is for simplification of the overall workflow, and for ease of visualization of the galloping sequences. In principle, a parametrization can be calculated for every timestep between the irregular mesh and the semi-regular mesh.

[2] https://pytorch3d.org/tutorials/deform_source_mesh_to_target_mesh

Tables and Figures

As an addition to the architecture's description in section 5 and visualization in Figure 3 we give a detailed distribution of parameters over the hexagonal convolutional, fully connected, and pooling layers in Table 5.

Layer	Output Shape	KS	Param.
Input	$(\bullet, 3, 111)$		0
HexConv	$(\bullet, 16, 111)$	2	912
Pooling	$(\bullet, 16, 33)$		0
HexConv	$(\bullet, 32, 33)$	1	3584
Pooling	$(\bullet, 32, 6)$		0
Fully Connected	$(\bullet, 8)$		2312
Hidden Representation for each patch of size 8			
Fully Connected	$(\bullet, 32, 6)$		2592
Unpooling	$(\bullet, 32, 33)$		0
HexConv	$(\bullet, 16, 33)$	1	3584
Unpooling	$(\bullet, 16, 111)$		0
HexConv	$(\bullet, 16, 111)$	2	4864
HexConv	$(\bullet, 3, 111)$	1	336

Table 5. Structure of the autoencoder. The bullets $\bullet$ reference the corresponding batch size. The data's last dimension is the number of vertices considered for each padded patch. For hexagonal convolutional layers the kernel size (KS) is given.

Figure 9 shows more reconstruction results of our architecture and the baseline CoMA [36] and Neural3DMM [6] autoencoder on test samples from the GALLOP and FAUST dataset.

Figure 10 shows the embedding in the low-dimensional space for the TRUCK's left front beam. The beam deforms in two different branches, which manifests in the embedding. The results are similar to [4, 17].

For the YARIS and TRUCK dataset we visualize in Figure 11 and Figure 12 respectively the selected car components, whose deformation over time we analyze with the mesh autoencoder for semi-regular meshes. All the components have different mesh representations, which we handle with only one autoencoder.

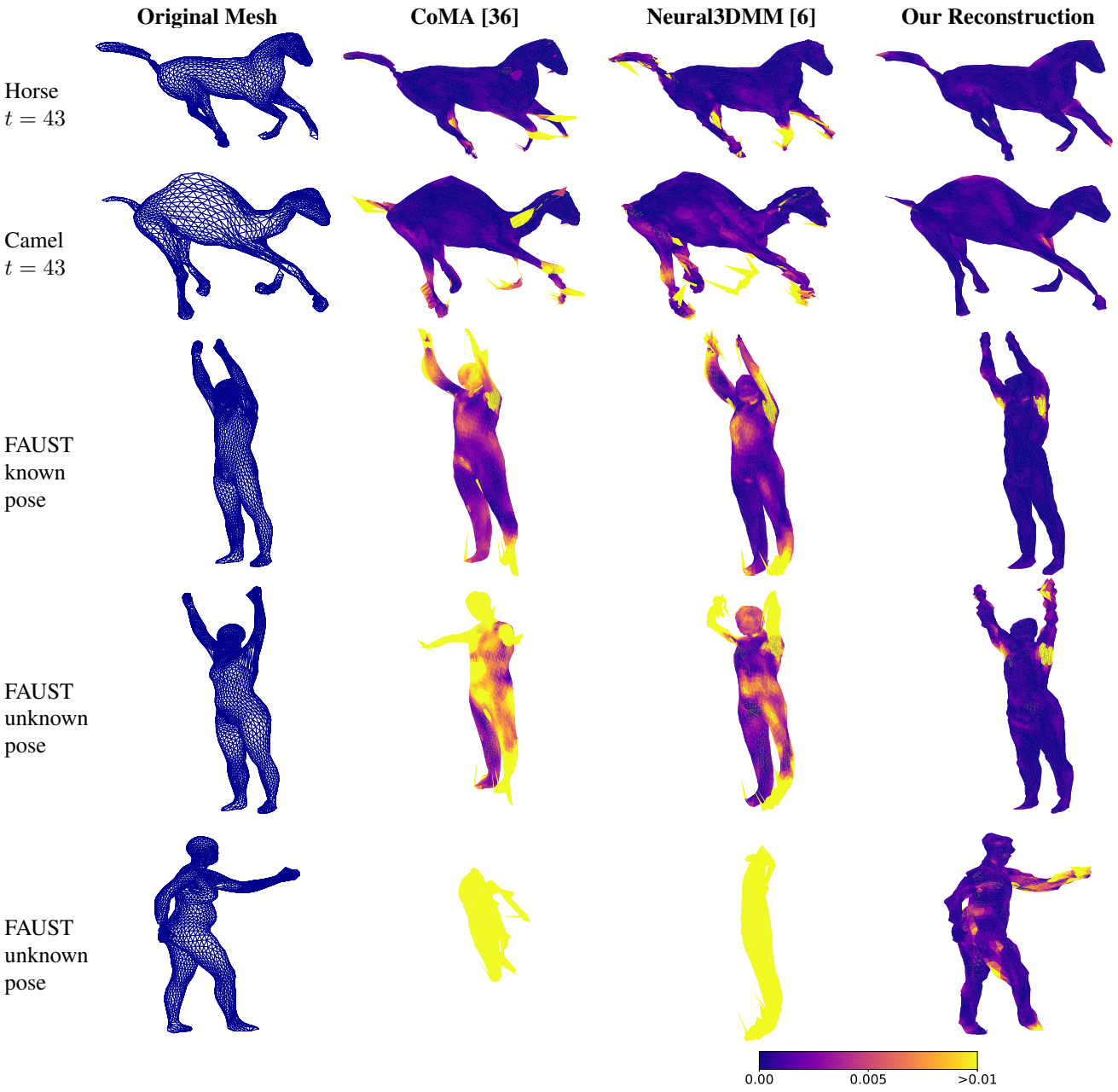

Figure 9. Additional reconstructed GALLOP and FAUST test samples by CoMA [36], Neural3DMM [6], and our network. The mean squared error of the reconstructed faces is highlighted.

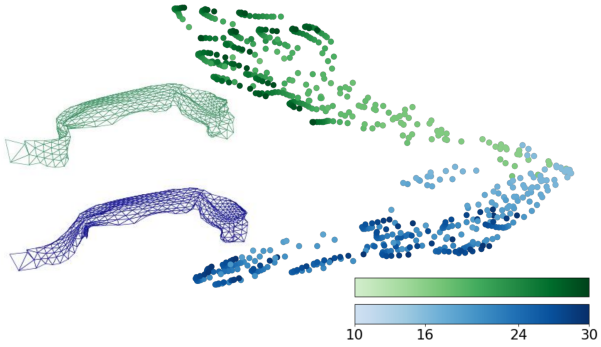

Figure 10. Embedding of the TRUCK's left front beam for $t = 10, \ldots, 30$. 32 simulations deform in two branches. Color encodes timestep and branch.

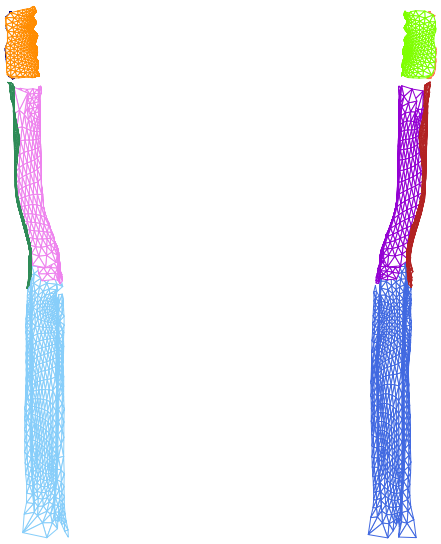

Figure 11. Selected car components in the YARIS dataset.

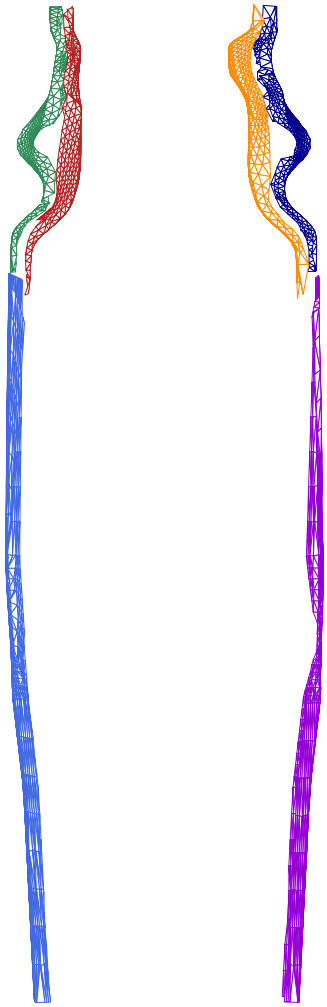

Figure 12. Selected car components in the TRUCK dataset.