Supplementary – Learning monocular 3D reconstruction of articulated categories from motion

1. Overview

We provide additional results including visualizations of the learned deformations, comparisons to CMR [3] and ACSM [4], qualitative results on a collection of different objects and technical details about the training procedure. Please note that the supplementary material contains a collection of video results.

2. Training Procedure

2.1. Per-sample optimization training framework

We build on the camera multiplex training procedure proposed by Goel *et al*. [2] to train the proposed method. We provide a small review of the method for coherency, however we refer the reader to [2] for a thorough explanation and technical details.

The authors propose using a learnable set of possible camera hypotheses for each training instance that is learned simultaneously with the rest of the 3D reconstruction CNN. In detail, each training instance i has $C_i = \{\pi_1, \ldots, \pi_{N_c}\}$ associated camera hypotheses, which are modelled as weak perspective cameras ($s \in \mathbf{R}, \mathbf{t} \in \mathbf{R}^2, \mathbf{q} \in \mathbf{R}^4$), that are retrieved from a 'camera database' during training using unique indices for each training sample. Each camera π_i is optimized to minimize the reconstruction losses for the silhouette $L_{sil,i}$ and the texture $L_{tex,i}$ like those used in our proposed framework. For updating the parameters of the method, the resulting losses $L_i = L_{sil,i} + L_{tex,i}$ of the camera set are used as a distribution over the most likely camera pose. This is encoded as a probability $p_i = \frac{e^{-L_i}}{\sum_j e^{-L_j}}$ for π_i to be the most likely camera. Using the computed distribution the final loss is formulated as

$$L_{\text{total}} = \sum_i p_i \left(L_{\text{sil},i} + L_{\text{tex},i}\right).$$

The final step of this training procedure is to train a camera predictor using the most probable camera of each training image conditioned on the image features extracted from the backbone CNN that is driving the whole reconstruction process.

Building on this optimization driven paradigm, we extend the training protocol with three distinct modifications. First of all, alongside the silhouette and textures losses we incorporate also the motion re-projection loss that is described in detail in the main paper. Furthermore, we extend the camera set with per image deformations D which is only one per image unlike the multiple cameras. In our pipeline, each training image i has a camera set C_i and a single handle deformation vector $D_i \in \mathbf{R}^{K\times 3}$ (with K being the number of handles) that are both used to express a multi-hypotheses distribution similar to [2]. Thirdly, unlike the aforementioned work, we simultaneously train our deformation and camera prediction branches using the most probable explanation for both the camera and deformation in accordance to the resulting silhouette and texture losses. This is achieved by adding two extra losses in the total loss function that minimize the ℓ_2 norm of the difference between the predicted quantity and the optimized one retrieved from the 'database' for each of the cameras and deformations.

In the main paper we presented experiments that relied on mask, motion, texture losses and in some cases also semantic keypoint re-projection loss. In the case of keypoint trained networks, we set $N_c = 1$ and initialize the camera with a rigid SfM camera in accordance to CMR [3]. For all other experiments, we use $N_c = 8$ and initialize the camera set C for every image in the training set with camera hypotheses whose azimuth is uniformly spaced on the viewing sphere. The handle deformations D are initialized with zeros which corresponds to the template shape. As a first step, each camera is optimized using the silhouette loss L_{sil} and motion loss L_{motion} using the template shape before training the rest of the method. We implement a drop hypotheses procedure [2] to reduce the computational complexity where the most improbable hypotheses are discarded from the camera set. In detail, after 20 epochs we keep the four most probable cameras and after 100 epochs we keep only the 2 most probable cameras. Any training augmentations that scale and translate the training image i are directly encoded as affine transformations on the respective camera set C_i while the deformation D_i remains unchanged since the depicted deformation of the object remains identical.

2.2. Architecture Details

We use the same encoder-decoder architecture that is presented in [3, 2]. Every image is encoded using an ImageNet pre-trained ResNet18 to a latent feature map $z \in \mathbf{R}^{4\times4\times256}$. A flattened version of z is processed with two MLP linear layers with output channels equal to 200 and the final result is given to the handle deformation predictor and camera predictor branches. The handle deformation branch provides the handle offsets $\Delta_H \in \mathbf{R}^{K\times3}$ and the camera predictor predicts the scale, translation and rotation, which is encoded as quartenions, through 2 fully connected layers each with 200 channels. Finally, we use the same texture predictor architecture as [2] which takes as input the encoded features z and outputs the UV texture map $I^{uv} \in \mathbf{R}^{128\times256\times3}$.

3. Learnable Laplacian Solver

Our algorithm builds on Laplacian surface editing techniques [5] which allow us to control a template mesh through handles while minimally distorting the template's shape. We represent the 3D shape of a category as a triangular mesh $M = (V, F)$ with vertices $\mathbf{V} \in \mathbb{R}^{N\times3}$ and fixed edges $F \in \mathbb{Z}^{N_f\times3}$. Our deformation approach relies on the cotangent-based discretization $\mathbf{L} \in \mathbb{R}^{N\times N}$ of the continuous Laplace-Beltrami operator used to calculate the curvature at each vertex of a mesh [6].

We obtain our K handles $H_{1,\dots,K}$ through a learnable dependency matrix $\mathbf{A} \in \mathbb{R}_+^{K\times N}$ that is right-stochastic, i.e. $\sum_v \mathbf{A}_{k,v} = 1$, effectively forcing every handle to lie in the convex hull of the mesh vertices by $\mathbf{H} = \mathbf{AV}$. The network's task is phrased as regressing the handle positions, denoted as $\mathbf{\Delta}_H$. Based on those handles, we obtain the deformed mesh $\mathbf{V}^*$ as the minimum of the following quadratic loss:

$$\mathbf{V}^* = \arg\min_{\mathbf{V}} \frac{1}{2}\left\|\mathbf{LV} - \mathbf{LT}\right\|^2 + \frac{1}{2}\left\|\mathbf{AV} - \tilde{\mathbf{H}}\right\|^2, \quad (1)$$

where as in [5] the first term enforces the solution to respect the curvature of the template mesh, $\mathbf{LT}$, while the second one penalizes the difference between the location of the handles according to $\mathbf{V}$ and the target location, $\tilde{\mathbf{H}} = \mathbf{HT} + \mathbf{\Delta}_H$. The stationary point of (1) can be found by solving the following linear system:

$$(\mathbf{L}^\mathsf{T}\mathbf{L} + \mathbf{A}^\mathsf{T}\mathbf{A})\mathbf{V} = \mathbf{L}^\mathsf{T}\mathbf{LT} + \mathbf{A}^\mathsf{T}\tilde{\mathbf{H}} \quad (2)$$

The solution $\mathbf{V}^*$ of Eq. (2) can be very efficiently computed with conjugate gradients or sparse solvers. In the forward case we obtain the solution using a sparse least square solver by concatenating the two matrices $\mathbf{L}$ and $\mathbf{A}$. As it will be presented the backward operation requires a different treatment and as such in the backward operation we make use of a linear solver for PSD matrices.

We want to compute the gradients with respect to the learnable dependency matrix $\mathbf{A}$ and the handle offset $\tilde{\mathbf{H}}$. We rewrite equation (2) as $\mathbf{WV} = \mathbf{b}$ where $\mathbf{W} = \mathbf{L}^\mathsf{T}\mathbf{L} + \mathbf{A}^\mathsf{T}\mathbf{A}$ and $\mathbf{b} = \mathbf{L}^\mathsf{T}\mathbf{LT} + \mathbf{A}^\mathsf{T}\mathbf{H}$. The direct solution is $\mathbf{V} = \mathbf{W}^{-1}\mathbf{b}$ and $\mathbf{W}$ is a symmetric PSD matrix as the addition of two likewise matrices. Instead of resorting to matrix inversion, we compute V using a linear solver for symmetric PSD matrices. To compute the necessary gradients for backpropagation of gradients through Equation (1), we rely on matrix calculus and use of three Kronecker product $\otimes$ properties [1]: 1) $\text{vec}(\mathbf{QWE}) = (\mathbf{E}^\mathsf{T} \otimes \mathbf{Q})\text{vec}(\mathbf{W})$, 2) $T_{m,n}\text{vec}(Q) = \text{vec}(\mathbf{Q}^\mathsf{T})$ and 3) $(\mathbf{Q} \otimes \mathbf{W})(\mathbf{E} \otimes \mathbf{R}) = (\mathbf{QE} \otimes \mathbf{WR})$.

The gradients of any linear solver for symmetric matrices are the following

$$\begin{aligned}\frac{\partial g(\mathbf{V})}{\partial \mathbf{b}} &= \frac{\partial \mathbf{V}}{\partial \mathbf{b}}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} = \frac{\partial \mathbf{W}^{-1}\mathbf{b}}{\partial \mathbf{b}}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} \\ &= \mathbf{W}^{-\mathsf{T}}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} = \mathbf{W}^{-1}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}}\end{aligned} \quad (3)$$

$$\begin{aligned}\frac{\partial g(\mathbf{V})}{\partial \,\text{vec}(\mathbf{W})} &= \frac{\partial \mathbf{V}}{\partial \,\text{vec}(\mathbf{W})}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} = \frac{\mathbf{W}^{-1}\mathbf{b}}{\partial \,\text{vec}(\mathbf{W})}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} \\ &= \frac{\partial \,\text{vec}(\mathbf{W}^{-1})}{\partial \,\text{vec}(\mathbf{W})}\frac{\partial \mathbf{W}^{-1}\mathbf{b}}{\partial \,\text{vec}(\mathbf{W}^{-1})}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} \\ &= \frac{\partial \,\text{vec}(\mathbf{W}^{-1})}{\partial \,\text{vec}(\mathbf{W})}\frac{\partial \mathbf{W}^{-1}\mathbf{b}}{\partial \,\text{vec}(\mathbf{W}^{-1})}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} \\ &= \frac{\partial \,\text{vec}(\mathbf{W}^{-1})}{\partial \,\text{vec}(\mathbf{W})}\frac{\partial (\mathbf{b}^\mathsf{T} \otimes \mathbf{I})\,\text{vec}(\mathbf{W}^{-1})}{\partial \,\text{vec}(\mathbf{W}^{-1})}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} \\ &= (-\mathbf{W}^{-1} \otimes \mathbf{W}^{-1})^\mathsf{T}(\mathbf{b}^\mathsf{T} \otimes \mathbf{I})^\mathsf{T}\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} \\ &= (-\mathbf{W}^{-1} \otimes \mathbf{W}^{-1})(\mathbf{b} \otimes \mathbf{I})\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} \\ &= -(\mathbf{V} \otimes \mathbf{W}^{-1})\frac{\partial g(\mathbf{V})}{\partial \mathbf{V}} = -\mathbf{V} \otimes \frac{\partial g(\mathbf{V})}{\partial \mathbf{b}}\end{aligned} \quad (4)$$

As such,

$$\frac{\partial g(\mathbf{V})}{\partial \mathbf{W}} = -\frac{\partial g(\mathbf{V})}{\partial \mathbf{b}}\mathbf{V}^\mathsf{T} \quad (5)$$

After calculating the gradients of the linear solver, the final step is the computation of the gradients with respect to the handle position regression $\tilde{\mathbf{H}}$ and learnable dependency matrix $\mathbf{A}$. The gradient of the first quantity is straight forward to calculate using Equation (3).

$$\begin{aligned}\frac{\partial g(\mathbf{V})}{\partial \tilde{\mathbf{H}}} &= \frac{\partial \mathbf{b}}{\partial \tilde{\mathbf{H}}}\frac{\partial g(\mathbf{V})}{\partial \mathbf{b}} = \frac{\partial (\mathbf{L}^\mathsf{T}\mathbf{LT} + \mathbf{A}^\mathsf{T}\tilde{\mathbf{H}})}{\partial \tilde{\mathbf{H}}}\frac{\partial g(\mathbf{V})}{\partial \mathbf{b}} \\ &= \mathbf{A}\frac{\partial g(\mathbf{V})}{\partial \mathbf{b}}\end{aligned} \quad (6)$$

Lastly we provide the gradient with respect to the learnable dependency matrix $\mathbf{A}$

$$\frac{\partial g(\mathbf{V})}{\partial \operatorname{vec}(\mathbf{A})} = \frac{\partial \mathbf{b}}{\partial \operatorname{vec}(\mathbf{A})}\frac{\partial g(\mathbf{V})}{\partial \mathbf{b}} + \frac{\partial \operatorname{vec}(\mathbf{W})}{\partial \operatorname{vec}(\mathbf{A})}\frac{\partial g(\mathbf{V})}{\partial \operatorname{vec}(\mathbf{W})} \tag{7}$$

$$\begin{aligned}\frac{\partial \mathbf{b}}{\partial \operatorname{vec}(\mathbf{A})} &= \frac{\partial(\mathbf{L}^\mathsf{T}\mathbf{L}\mathbf{T} + \mathbf{A}^\mathsf{T}\tilde{\mathbf{H}})}{\partial \operatorname{vec}(\mathbf{A})} = \frac{\partial \operatorname{vec}(\mathbf{A}^\mathsf{T}\tilde{\mathbf{H}})}{\partial \operatorname{vec}(\mathbf{A})} \\ &= \frac{\partial \operatorname{vec}(\mathbf{A}^\mathsf{T})}{\partial \operatorname{vec}(\mathbf{A})}\frac{\partial(\tilde{\mathbf{H}} \otimes \mathbf{I})\operatorname{vec}(\mathbf{A}^\mathsf{T})}{\partial \operatorname{vec}(\mathbf{A}^\mathsf{T})} \\ &= \frac{\partial \mathbf{T}_{K,N}\operatorname{vec}(\mathbf{A})}{\partial \operatorname{vec}(\mathbf{A})}(\tilde{\mathbf{H}}^\mathsf{T} \otimes \mathbf{I})^\mathsf{T} = \mathbf{T}_{N,K}(\tilde{\mathbf{H}} \otimes \mathbf{I})\end{aligned} \tag{8}$$

$$\begin{aligned}\frac{\partial \operatorname{vec}(\mathbf{W})}{\partial \operatorname{vec}(\mathbf{A})} &= \frac{\partial(\mathbf{L}^\mathsf{T}\mathbf{L} + \mathbf{A}^\mathsf{T}\mathbf{A})}{\partial \operatorname{vec}(\mathbf{A})} = \frac{\partial \operatorname{vec}(\mathbf{A}^\mathsf{T}\mathbf{A})}{\partial \operatorname{vec}(\mathbf{A})} \\ &= \mathbf{I}_N \otimes \mathbf{A}^\mathsf{T} + (\mathbf{A}^T \otimes \mathbf{I}_N)\mathbf{T}_{K,N}\end{aligned} \tag{9}$$

In Figure 1 we provide a PyTorch implementation of the differentiable Laplacian deformation module which has been thoroughly gradient checked. We omit some PyTorch related boilerplate code for clarity.

4. PCA on deformations

In Figure 2 we visualize the learned deformations for a wide range of articulated objects. The visualization is created by running PCA for each object on all 3D reconstructions obtained on training and test dataset. The mean shape is depicted in the center while the first three PCA axis are visualized alongside the mean shape. The visualizations shows some interesting deformations across objects. In all cases the deformations capture clearly movements of legs, the tail and head.

5. More Results

5.1. Comparisons on Horses

In Figure 3 we provide comparisons against prior articulated reconstruction work [4]. Note that both methods used identical template mesh. The proposed method is capable of achieving realisting deformations without the requirement of manual part segmentation as a result of the proposed differentiable deformation module. We also include several videos of horses for extra comparisons. The videos demonstrate that our method provides camera predictions and deformations that are robust across frames and match closely the movements of the depicted object.

5.2. Comparisons on CUB

In Figure 4 we provide comparisons against prior work on common training supervision. It is apparent that our method is capable of correctly deforming the template mesh to produce highly flexible wings or bending and turning the body and head of the birds. In nearly all results, ACSM lacks the necessary 3D shape deformation which is caused by the segmentation of the template shape in 3 parts (head, body and tail). CSM is providing results with better deformations to ACSM, however as it can be seen almost all open wing results are obtained with the offset of a small set of points which causes uneven surfaces (for example Row 4 left). Our method due to the Laplacian based Deformation produces always meshes with well allocated vertices. Furthermore, our results exhibit interesting features like rotation of the head (Row 2-left, Row 1-right), open wing formations and bending of the beak (Row 5-left), i.e. pecking.

5.3. More Results

In Figures 5-9 we provide a wide collection of 3D reconstructions for several highly articulated objects. We also include a collection of videos as part of the supplementary material with reconstructions of several classes. Our common visualization setup is input image, 3D reconstruction from the predicted viewpoint and a different one and finally the textured mesh reconstruction.

5.4. Failure Cases

We visualize some failure cases of the proposed method in Figure 10. Common failure cases are related to the inability to predict a good camera pose and the inference of simplistic textures.

References

[1] Paul L Fackler. Notes on matrix calculus. *Privately Published*, 2005. 2

[2] Shubham Goel, Angjoo Kanazawa, , and Jitendra Malik. Shape and viewpoints without keypoints. In *ECCV*, 2020. 1, 2

[3] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018. 1, 2, 7

[4] Nilesh Kulkarni, Abhinav Gupta, David F Fouhey, and Shubham Tulsiani. Articulation-aware canonical surface mapping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 452–461, 2020. 1, 3, 6, 7

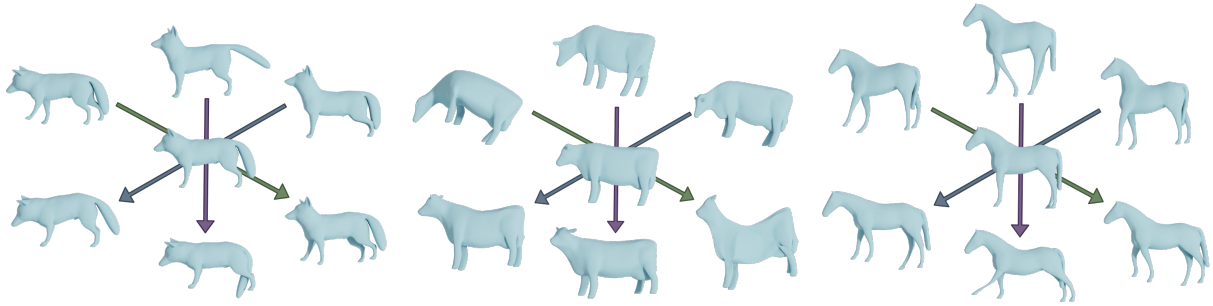
[5] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and H-P Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184, 2004. 2

[6] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358, 1995. 2

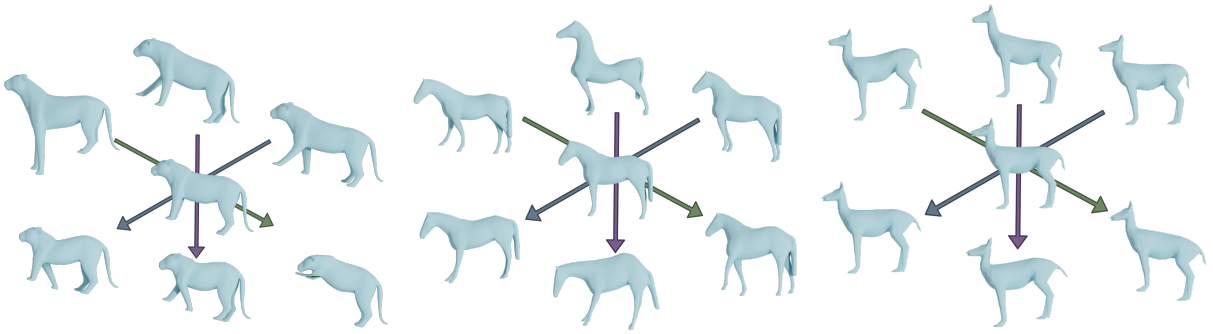
```
class Diff_LPL(Function):
    @staticmethod
    def forward(ctx, L, A, T, tildeH):
        delta = L @ T
        A_lstsq = np.concatenate([L, A],axis=0) # sparse matrix
        b_lstsq = np.concatenate([delta, tildeH],axis=0)
        V = scipy.sparse.linalg.lsqr(A_lstsq, b_lstsq) # or any other␣
↪appropriate sparse solver
        # save required quantities for backward
        return V

    @staticmethod
    def backward(ctx, grad_output):
        W = ctx.L.T @ ctx.L + ctx.A.T @ ctx.A # dense matrix
        grad_over_b = scipy.linalg.solve(W, grad_output, sym_pos=True) #␣
↪gradient of solver w.r.t b
        grad_over_W = - grad_over_b @ ctx.V.T # gradient of solver w.r.t. W
        grad_over_tildeH = ctx.A @ grad_over_b # gradient w.r.t. \tilde{H}
        dg_b_dA = ctx.tildeH @ grad_over_b.T
        dg_W_dA = ctx.A @ grad_over_W + ctx.A @ grad_over_W.T
        grad_over_A = dg_b_dA + dg_W_dA  # gradient w.r.t. A
        return None, grad_over_A, None, grad_over_tildeH
```

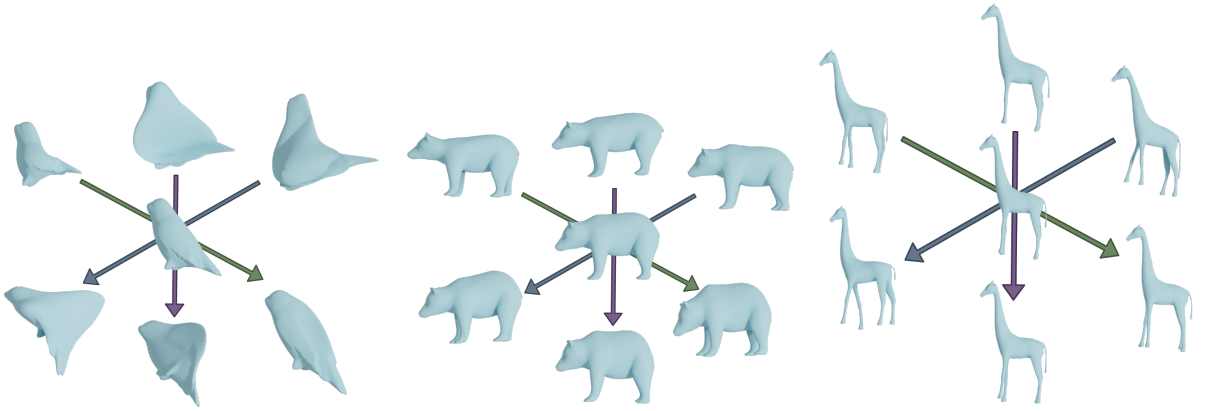
Figure 1: PyTorch implementation of the proposed Differentiable Laplacian Solver.



(a) Fox (b) Cow (c) Horse



(d) Tiger (e) Zebra (f) Deer



(g) Birds (h) Bear (i) Giraffe

Figure 2: **Bird Reconstructions** Visualization of the predicted deformations for several objects by depicting the mean shape in the center and the first 3 modes obtained by PCA on the handle estimates obtained across the dataset.

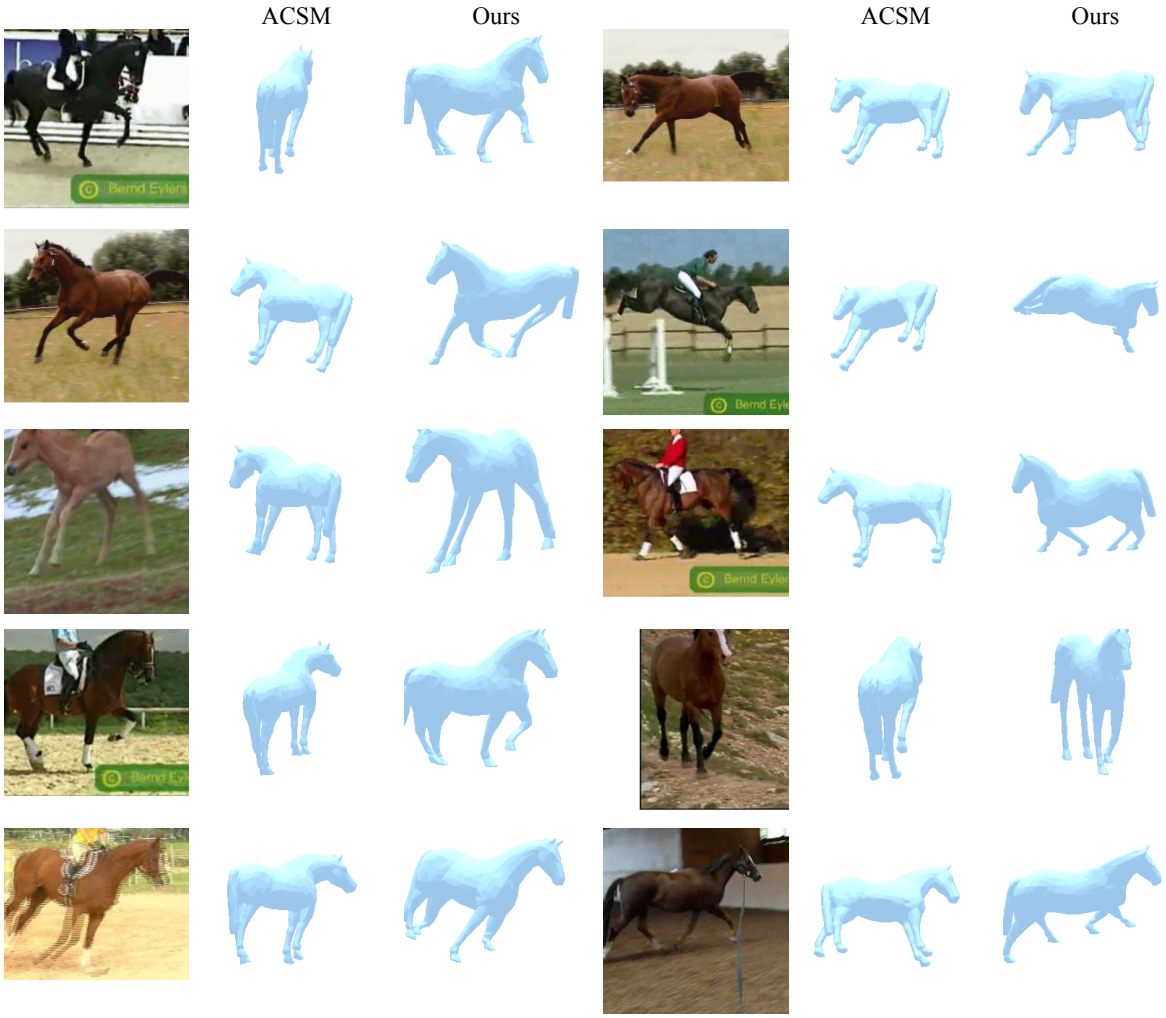


Figure 3: Reconstruction comparisons of our method and ACSM [4] for the object horse.

Figure 4: **Bird Reconstructions:** Qualitative comparisons between our method, CMR [3] and ACSM [4] with images from the CUB test set.

Figure 5: **3D reconstructions** We show the input image, the reconstructed shape from the predicted and a novel view and finally the predicted texture.

Figure 6: **3D reconstructions** We show the input image, the reconstructed shape from the predicted and a novel view and finally the predicted texture.

Figure 7: **3D reconstructions** We show the input image, the reconstructed shape from the predicted and a novel view and finally the predicted texture.

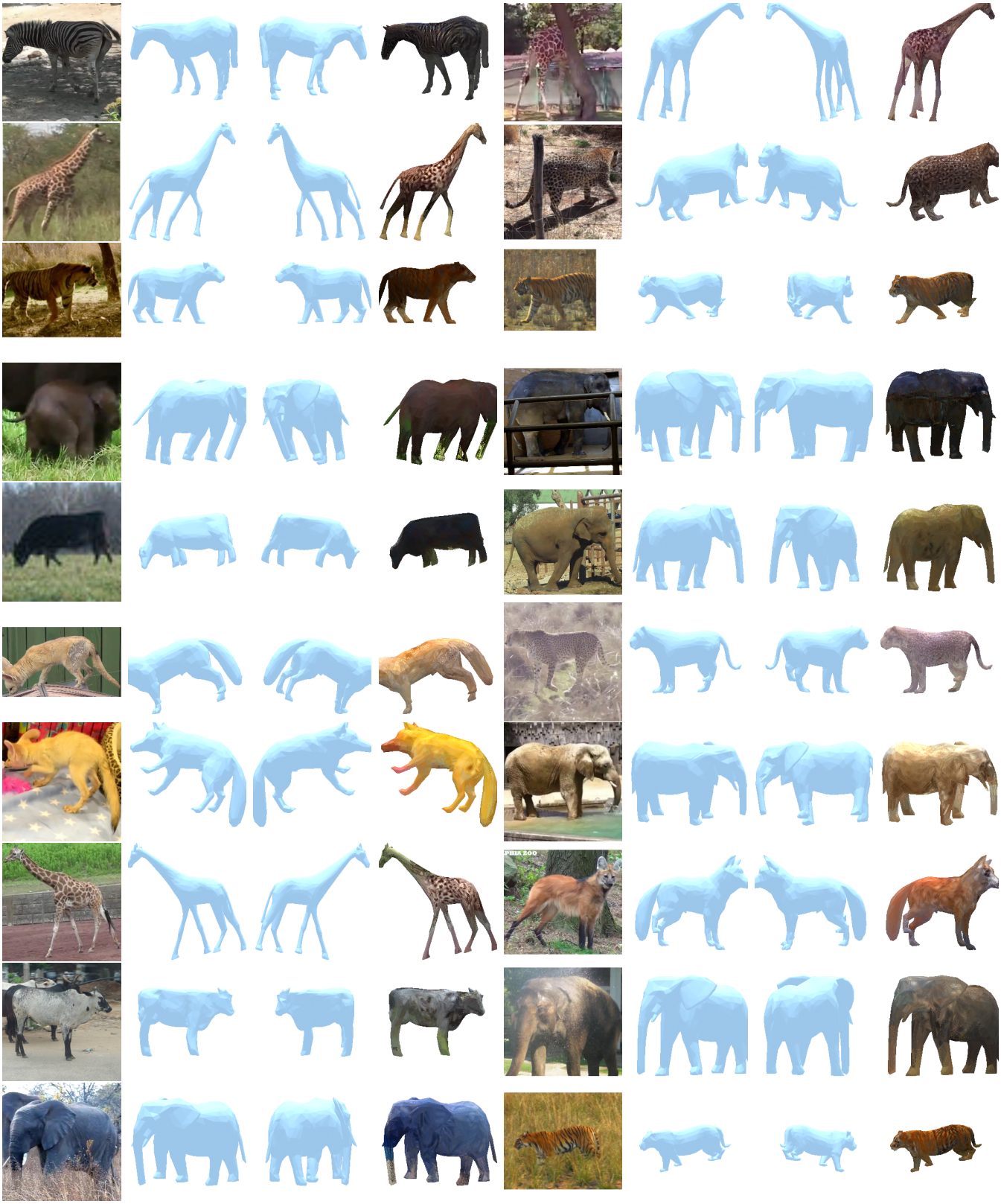


Figure 8: **3D reconstructions** We show the input image, the reconstructed shape from the predicted and a novel view and finally the predicted texture.

Figure 9: **3D reconstructions** We show the input image, the reconstructed shape from the predicted and a novel view and finally the predicted texture.

Figure 10: **Failure Cases:** We visualize some failure modes of our method. The columns present the input image, 3D reconstruction from the predicted viewpoint and a different one and the predicted texture .