Watch It Move: Unsupervised Discovery of 3D Joints for Re-Posing of Articulated Objects
Supplemental Material

Atsuhiro Noguchi[†, ‡] Umar Iqbal[†] Jonathan Tremblay[†] Tatsuya Harada[‡,§] Orazio Gallo[†]
[†]NVIDIA [‡]The University of Tokyo [§]RIKEN

A. Implementation details

A.1. Pose Estimation (Section 3.2)

MLP $\mathcal{T}_\Theta$ takes the positionally encoded frame id as input and outputs a vector of $9P$ dimensions. The $3P$ dimensions correspond to each element of $\mathbf{t}_i$, and the remaining $6P$ dimensions correspond $\mathbf{R}_i$. For the rotation matrix calculation, please refer to Zhou *et al*. [13], Section B. $\mathcal{T}_\Theta$ is a 4-layer MLP with a hidden dimension of 256.

A.2. SDF Computation of an Ellipsoid (Section 3.3)

The merit of using ellipsoids as the representation of the parts is that their SDFs are continuous functions and can be computed cheaply. In this subsection, we explain how to calculate the SDF of an ellipsoid. First, the surface of an ellipsoid of radii $\mathbf{r}$ is given by

$$f(\mathbf{x}, \mathbf{r}) = \frac{x_1^2}{r_1^2} + \frac{x_2^2}{r_2^2} + \frac{x_3^2}{r_3^2} = 1, \tag{1}$$

where position $\mathbf{x} = (x_1, x_2, x_3)$ and radii $\mathbf{r} = (r_1, r_2, r_3)$. We calculate the SDF of an ellipsoid as follows. First, from the query point $\mathbf{x}$, we find the nearest ellipsoid surface point $\mathbf{x}_e$. Since this cannot be solved analytically, we use Lagrange's multiplier method and Newton's method to find the point. The cost function $\mathcal{L}$ is defined as

$$\mathcal{L} = |\mathbf{x} - \mathbf{x}_e|_2^2 - \lambda(f(\mathbf{x}_e, \mathbf{r}) - 1), \tag{2}$$

and we solve $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_e} = \frac{\partial \mathcal{L}}{\partial \lambda} = 0$. This can be transformed into the following equations

$$\begin{cases} \mathbf{x}_e = \mathbf{r}^2 \oslash (\mathbf{r}^2 + \lambda) \odot \mathbf{x} & (3) \\ |\mathbf{r} \oslash (\mathbf{r}^2 + \lambda) \odot \mathbf{x}|_2^2 = 1, & (4) \end{cases}$$

where $\oslash$ is element-wise division, $\odot$ is element-wise product, and $\mathbf{a}^2 = \mathbf{a} \odot \mathbf{a}$. We use Newton's method to find the largest solution for λ in Equation 4 and substitute it into Equation 3 to get $\mathbf{x}_e$.

The distance between the searched point and the input point $\mathbf{x}$ is the absolute value of the SDF, which has a negative sign when $\mathbf{x}$ is inside the ellipsoid and a positive sign when it is outside:

$$\mathrm{SDF}(\mathbf{x}, \mathbf{r}) = \mathrm{sign}(f(\mathbf{x}, \mathbf{r}) - 1)|\mathbf{x} - \mathbf{x}_e|_2. \tag{5}$$

However, since $\mathbf{x}_e$ is computed numerically, the gradient is not propagated to $\mathbf{r}$. Therefore, we re-parametrize $\mathbf{x}_e$ using $\mathbf{r}$ by projecting $\mathbf{x}_e$ onto the surface of the unit sphere and back onto the surface of the ellipsoid.

$$\tilde{\mathbf{x}}_e = (\mathbf{x}_e \oslash \mathbf{r}).\mathrm{detach}() \odot \mathbf{r}, \tag{6}$$

where $\oslash$ is element-wise division, $\odot$ is element-wise product, and detach() is a stop-gradient operation.

Finally, the differentiable SDF of an ellipsoid is computed as,

$$\mathrm{SDF}(\mathbf{x}, \mathbf{r}) = \mathrm{sign}(f(\mathbf{x}, \mathbf{r}) - 1)|\mathbf{x} - \tilde{\mathbf{x}}_e|_2. \tag{7}$$

A.3. Shape and Appearance Decoder (Section 3.3)

MLP $\mathcal{S}_\Theta$ takes a feature vector $\mathbf{f}$ at a query 3D location $\mathbf{x}^g$ and outputs the color $\mathbf{c}$ and a residual SDFs $\widetilde{\Delta d}$. The color on the surface of real objects changes in complex ways according to the time, pose, and view direction. We simplify the problem by assuming the color depends only on $\mathbf{f}$, meaning the color is constant across view and time. $\mathcal{S}_\Theta$ is a 8-layer MLP with a hidden dimension of 256.

A.4. Joint Candidates (Section 3.5.1)

In this subsection, we explain the details of the joint candidates defined in the Section 3.5.1 in the main paper. When a point inside one ellipsoid e_i is close to a point inside another ellipsoid e_j throughout the entire video, the point is considered to be a joint between parts i and j. To find these points, we create several joint candidate points $\{\boldsymbol{\xi}_i^n\}_{\{n=1:N\}}$ inside the ellipsoids in advance, and minimize the distance between them. We define six candidates inside each ellipsoid in the local coordinate, as follows

$$\hat{\boldsymbol{\xi}}_i^n = \mathbf{r}_i \odot \boldsymbol{\xi}^n, \tag{8}$$

where $\boldsymbol{\xi}^n \in \{(\pm\frac{3}{4}, 0, 0), (0, \pm\frac{3}{4}, 0), (0, 0, \pm\frac{3}{4})\}$. This is then transformed into the global coordinate system using the rotation and translation of each part

$$\boldsymbol{\xi}_i^n = \mathbf{R}_i \hat{\boldsymbol{\xi}}_i^n + \mathbf{t}_i. \tag{9}$$

A.5. Frame scheduling (Section 3.6)

In order to stabilize the training, we progressively increase the number of frames used for training. First, we use the first T_0 frames of data to train for τ_0 iterations. Then, the number of frames used for training is increased linearly so that all frames T of the video are used at τ_1 iteration. After that, all frames are used for training until τ_{final}. In the experiment using human data, we set $T_0 = 10$, $\tau_0 = 10\text{k}$ $\tau_1 = 80\text{k}$.

A.6. Loss (Section 3.6)

The loss function is a weighted sum of $\mathcal{L}_{\text{SDF}}$, $\mathcal{L}_{\text{photo}}$, $\mathcal{L}_{\Gamma}$, $\mathcal{L}_{\text{merge}}$, $\mathcal{L}_{\mathcal{E}}$, $\mathcal{L}_{\mathbf{t}}$, and $\mathcal{L}_{\text{separation}}$:

$$\begin{aligned}\mathcal{L}_{\text{total}} =& \lambda_{\text{SDF}}\mathcal{L}_{\text{SDF}} + \lambda_{\text{photo}}\mathcal{L}_{\text{photo}} + \lambda_{\Gamma}\mathcal{L}_{\Gamma} + \\ & \lambda_{\text{merge}}\mathcal{L}_{\text{merge}} + \lambda_{\mathcal{E}}\mathcal{L}_{\mathcal{E}} + \lambda_{\mathbf{t}}\mathcal{L}_{\mathbf{t}} + \lambda_{\text{separation}}\mathcal{L}_{\text{separation}}.\end{aligned} \tag{10}$$

We used $\lambda_{\text{SDF}} = 0.2$, $\lambda_{\text{photo}} = 1$, $\lambda_{\text{merge}} = 0$, $\lambda_{\mathcal{E}} = 600$, $\lambda_{\mathbf{t}} = 1000$, and $\lambda_{\text{separation}} = 1$. We gradually increase $\mathcal{L}_{\Gamma}$ from 2 to 50 until iteration τ_0 for training stability. From iteration τ_2, we set $\lambda_{\text{merge}} = 5$ and train the model until τ_{final}. We set $\tau_2 = 150\text{k}$.

A.7. Other Training Details

The other hyper-parameters were set as $d_{\max} = 0.02$, $\lambda_l = 0.02$, $\lambda_{\text{motion}} = 3$, $\bar{D} = 0.1$, $\epsilon = 0.01$. For the weighted positional encoding in Equation 4 in the main paper, we apply positional encoding [8] to spatial locations $\mathbf{x}_i$ with 6 frequencies following the training setting of NeuS [12].

We used AdamW optimizer [7] with learning rate 0.0003, and $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\lambda = 0.005$. All training images are resized to 512×512. We train the model up to $\tau_{\text{final}} = 200\text{k}$ iterations with a batch size of 16. We randomly sample 384 rays from each image for human data, where P is set to 20. Training takes about 48 hours on a single NVIDIA A100 GPU.

A.8. Pose Manipulation (Section 4.1)

Since our method estimates explicit joint relationships between parts, we can freely manipulate the pose of the object. First, the position of the final joints $\boldsymbol{\xi}_{i,j}$ is defined as the midpoint of the candidate points $\boldsymbol{\xi}_i^m$ and $\boldsymbol{\xi}_j^n$ connected in Equations 16 and 17 of the main paper:

$$\boldsymbol{\xi}_{i,j} = \frac{1}{2}(\boldsymbol{\xi}_i^m + \boldsymbol{\xi}_j^n) \quad \text{s.t. } (m,n) = \arg\min_{m,n} \bar{l}_{i,j}^{\,m,n}. \tag{11}$$

By manually rotating the part i or j and its children parts around this joint $\boldsymbol{\xi}_{i,j}$, the pose of the object can be freely changed, and novel rotations and translations for each part can be obtained $\{\mathbf{R}_i, \mathbf{t}_i\}_{\{i=1:P\}}^{\text{novel}}$. To render the novel pose image, we directly input $\{\mathbf{R}_i, \mathbf{t}_i\}_{\{i=1:P\}}^{\text{novel}}$ to the second network $\mathcal{S}_{\Theta}$.

A.9. Baselines (Section 4.3)

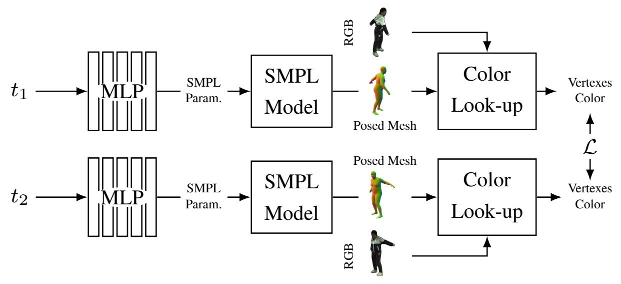
Kundu* *et al.* [6] Their original model estimates SMPL parameters in an unsupervised manner as follows. First, a CNN-based encoder receives an image and estimates the parameters of the SMPL model. Based on the estimated parameters, the SMPL mesh is deformed and a pixel value of the image is assigned to each vertex according to its position in the image. The model is trained to match the colors of the estimated mesh vertices from different images of the same person at different times. By using CNN-based encoder, they can reconstruct the mesh from unseen monocular images. However, our method uses videos of specific scenes from multiple viewpoints, which gives us an unfair advantage. In order to allow for a fair comparison between our method and theirs, we replaced their CNN-based encoder with an MLP $\mathcal{T}_{\Theta}^{\text{Kundu*}}$ that takes frame id as input and estimates SMPL parameters for each frame.

$$\mathcal{T}_{\Theta}^{\text{Kundu*}} : \gamma(t) \rightarrow \mathbf{R}(t), \mathbf{t}(t), \boldsymbol{\theta}(t), \boldsymbol{\beta}, \tag{12}$$

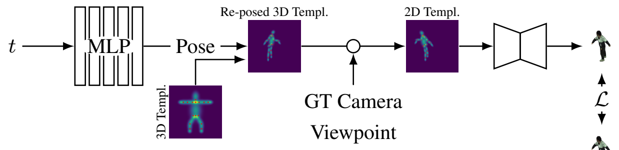
where $\mathbf{R}(t)$ is a global orientation, $\mathbf{t}(t)$ is a global translation, $\boldsymbol{\theta}(t)$ is joint poses, and $\boldsymbol{\beta}$ is a shape parameters. An overview of the method is visualized in Figure A (a). This MLP allows Kundu* *et al.* to overfit to the specific video sequence, like our method does. Please note that $\boldsymbol{\beta}$ is not time dependent. The structure of the MLP $\mathcal{T}_{\Theta}^{\text{Kundu*}}$ is the same as that of $\mathcal{T}_{\Theta}$ used in the proposed method except for the output dimension. Since the authors do not publish their training implementation, all modules and loss functions are our replicated implementation. For the human pose prior, instead of training the adversarial auto-encoder, we used the pre-trained human pose prior from Bogo *et al.* [1]. Also, since our experimental setup uses multi-view videos and overfits to a single subject, we do not use reflectional symmetry or shape-consistency loss. Please refer to Kundu *et al.* [6] for more details. In addition, since human foreground masks are available in our experimental setup, we use a differentiable renderer [10] to render the mask of the mesh and train it so that the L2 norm with the GT foreground mask is small. We apply the same frame scheduling as in Section A.5 for training stabilization.

After training the model, we assign colors to the vertices of the SMPL mesh for novel view and pose synthesis, where we average the estimated vertex color for various frame ids and viewpoints using the learned SMPL poses.

Schmidtke* *et al.* [11] Their original model trains the deformation of a 2D template of a person's structure in an unsupervised manner using image reconstruction. They use a CNN-based encoder to estimate the 2D deformation parameters. To extend the method to 3D, we replaced the 2D templates with 3D templates, where the shape is approximated with 3D gaussians. For a fair comparison, we also



(a) Kundu* *et al.* [6]



(b) Schmidtke* *et al.* [11]

Figure A: Baselines used for comparison. The * indicates that these are adaptations of the original methods.

replaced the CNN-based encoder with an MLP $\mathcal{T}_\Theta^{\text{Schmidtke*}}$ that takes a frame id as input and outputs the deformation parameters of the template in the global coordinate, as in Kundu* *et al.*

$$\mathcal{T}_\Theta^{\text{Schmidtke*}} : \gamma(t) \rightarrow \{\mathbf{R}_i(t), \mathbf{t}_i(t), \mathbf{s}_i\}_{\{i=1:18\}}, \tag{13}$$

where $\mathbf{R}_i(t)$ and $\mathbf{t}_i(t)$ are rotation and translation for each part, and $\mathbf{s}_i$ is 3D scale parameters for each part. An overview of the method is visualized in Figure A (b). We replace the affine transformation $\mathbf{\Theta}_i$ defined in Equation 3 in Schmidtke *et al.* [11] with a physically meaningful transformation

$$\mathbf{\Theta} = \begin{bmatrix} s_1R_{1,1} & s_1R_{1,2} & s_1R_{1,3} & t_1 \\ s_2R_{2,1} & s_2R_{2,2} & s_2R_{2,3} & t_2 \\ s_3R_{3,1} & s_3R_{3,2} & s_3R_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{14}$$

Please note that $\mathbf{s}_i$ is not time dependent. The deformed 3D template is projected onto a 2D heatmap using the viewpoint of the training image, and then transformed into an RGB image using a second CNN-based network. Since the model is overfitted on a single scene, we do not input the reference frame to the second network, but only the 2D heatmap. We modified the implementation based on their public training code[1], and the loss function used for training is exactly the same as the original. Please refer to Schmidtke *et al.* [11] for more details.

We apply the same frame scheduling as in Section A.5 for training stabilization.

A.10. Regression of GT SMPL Pose (Section 4.2, 4.4)

For human re-posing we learn a linear mapping from the GT SMPL poses to estimated joints and for joint evaluation we learn a mapping from the estimated joints to the GT SMPL poses.

[1]https://github.com/lschmidtke/shape_templates

Mapping from SMPL For re-posing, we perform a regression from the GT SMPL mesh to our part pose $\mathbf{R}_i$ and $\mathbf{t}_i$. First, for all frames of the training data, we optimize the linear transformation $\mathbf{X}$ from the mesh vertices $\mathbf{V}(t)$ to the concatenation of the learned part centers and the candidate points $\mathbf{P}(t) = \text{CAT}\{\mathbf{t}_i, \boldsymbol{\xi}_i^1, \ldots, \boldsymbol{\xi}_i^N\}_{\{i=1:P\}}(t)$ using the least-squares method

$$\min_{\mathbf{X}} \left(\sum_t |\mathbf{X}\mathbf{V}(t) - \mathbf{P}(t)|_F + \frac{1}{2}\lambda|\mathbf{X}|_F \right). \tag{15}$$

When re-posing with SMPL meshes, we compute the part centers and candidate points $\mathbf{P}^{\text{new}} = \text{CAT}\{\mathbf{t}_i, \boldsymbol{\xi}_i^1, \ldots, \boldsymbol{\xi}_i^N\}^{\text{new}}_{\{i=1:P\}}$ corresponding to the novel pose SMPL mesh $\mathbf{V}^{\text{new}}$ using the learned linear transformation $\mathbf{X}$,

$$\mathbf{P}^{\text{new}} = \mathbf{X}\mathbf{V}^{\text{new}}. \tag{16}$$

The part centers $\mathbf{t}_i^{\text{new}}$ of the new pose are obtained by extracting the corresponding elements of $\mathbf{P}_i^{\text{new}}$. The rotation matrix $\mathbf{R}_i^{\text{new}}$ is obtained by solving the following optimization:

$$\min_{\mathbf{R}_i^{\text{new}}} |\mathbf{R}_i^{\text{new}}\hat{\Xi}_i - \Xi_i|_F \quad s.t.\ (\mathbf{R}_i^{\text{new}})^\top \mathbf{R}_i^{\text{new}} = \mathbf{I}, \tag{17}$$

where $\hat{\Xi}_i = \left[\hat{\boldsymbol{\xi}}_i^1, \ldots, \hat{\boldsymbol{\xi}}_i^N\right]$, $\Xi_i = \left[\boldsymbol{\xi}_i^1 - \mathbf{t}_i, \ldots, \boldsymbol{\xi}_i^N - \mathbf{t}_i\right]$. By using the resulting $\mathbf{t}_i^{\text{new}}$ and $\mathbf{R}_i^{\text{new}}$ into the second network $\mathcal{S}_\Theta$ directly, we can re-pose the object.

Similarly, for the re-posing of the baseline Schmidtke* *et al.* [11], the part center $\mathbf{t}_i$ of the 3D template and the points around it

$$\boldsymbol{\xi}_i^n \in \{\mathbf{t}_i \pm r\mathbf{R}_i^1, \mathbf{t}_i \pm r\mathbf{R}_i^2, \mathbf{t}_i \pm r\mathbf{R}_i^3\} \tag{18}$$

are used to learn the same linear mapping, where $\mathbf{R}_i = [\mathbf{R}_i^1, \mathbf{R}_i^2, \mathbf{R}_i^3]$ and $r = 0.1$.

Mapping to SMPL To evaluate the joint, we regress the translation $\mathbf{J}(t) = \text{CAT}\{\mathbf{j}_i\}_{\{i=1:23\}}(t)$ of the GT SMPL joints at frame t from the learned object poses at t, where

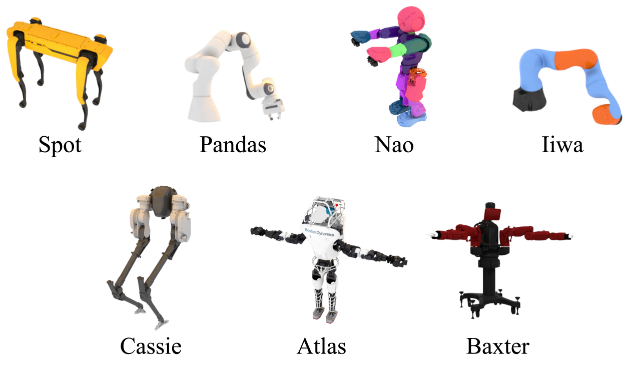


Figure B: The robots in our datasets.

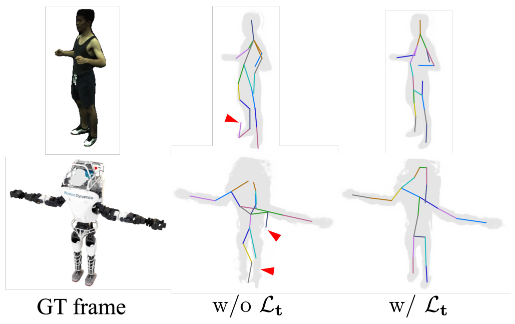


Figure C: Learned structures with and without $\mathcal{L}_{\mathbf{t}}$. Each line connects the center of each part. Red marks show the parts that should be noted.

CAT is the concatenation operator. We obtain a linear mapping $\mathbf{X}$ from the learned poses to the SMPL joints by solving the following optimization problem:

$$\min_{\mathbf{X}} \left(\sum_{t\in T_{\text{train}}} |\mathbf{X}\mathbf{P}(t) - \mathbf{J}(t)|_F + \frac{1}{2}\lambda|\mathbf{X}|_F \right), \qquad (19)$$

where T_{train} is the set of frames used for this optimization, which are uniform sample of 10% of the available frames.

For joint evaluation, a learned linear transformation $\mathbf{X}$ was applied to the remaining frames to compute the mean per joint position error (MPJPE) [4] between the regressed joint position and the GT joint position.

Similarly, for the baseline Schmidtke* *et al.* [11], we learn a linear mapping from the part center $\mathbf{t}_i$ of the 3D template and the points around it $\boldsymbol{\xi}_i^n$ defined in Equation 18 to the GT SMPL joints.

For the evaluation of Kundu* *et al.* [6], we learn a linear regression $\mathbf{X}$ from the learned SMPL mesh vertices $\mathbf{V}_{\text{Kundu}}$ to the GT SMPL joints

$$\min_{\mathbf{X}} \left(\sum_{t\in T_{\text{train}}} |\mathbf{X}\mathbf{V}_{\text{Kundu}}(t) - \mathbf{J}(t)|_F + \frac{1}{2}\lambda|\mathbf{X}|_F \right). \qquad (20)$$

We evaluated both models in the same way as ours.

A.11. Effect of $\mathcal{L}_{\mathbf{t}}$ (Section 4.5)

Qualitative results with and without $\mathcal{L}_{\mathbf{t}}$ are shown in Figure C. We can see that the parts sometimes do not cover the entire object or go outside of the object without $\mathcal{L}_{\mathbf{t}}$. This result demonstrates the effectiveness of $\mathcal{L}_{\mathbf{t}}$.

B. Training on the RGBD-Dog Dataset

We used RGBD-Dog dataset [5] for training the dog model. We apply off-the-shelf instance segmentation model [2] to obtain the ground truth masks. In this dataset, it is not possible to isolate the exact region of the dog due to occlusion by a person or overlapping background objects. Therefore, we modified the loss function to ignore occluded regions and regions where the estimated mask is unreliable. Specifically, when calculating $\mathcal{L}_{\text{photo}}$, $\mathcal{L}_{\mathcal{E}}$, and $\mathcal{L}_{\mathbf{t}}$, we replace the loss values with 0 for elements that use unreliable pixels in their calculations.

C. Robot dataset

In order to demonstrate the applicability of our method to objects with various structures, we created a dataset of robots with seven different structures, see Figure B. The dataset consists of 1000 frames of synchronized video with 20 viewpoints per robot. We sampled five of these views and trained on the first 300 frames of each video.

In order to generate the dataset we use a recent python-based renderer, NViSII [9]. We use robots that are freely available and have URDF associated with 3D meshes. In order to animate the robot, we use PyBullet [3]. The robot is given random joint goals, and once it reaches these goals, we repeat the process of giving it random joint goals. We place 20 fixed cameras on the hemisphere at a fixed distance from the robot, and we add a warm sunlight to add more light to the scene. Each frame is rendered with 2000 samples per pixel at 512×512 resolution. We use the OptiX denoiser to clean the final renders to provide noise free images. Both the datasets and the scripts to generate the multiview animated objects will be available at `https://github.com/NVlabs/watch-it-move`.

D. Additional Results for Parts Merging

Additional results of parts merging are shown in Figure D (a). The results confirm that meaningful parts and the structure are obtained by merging. To show the effect of $\mathcal{L}_{\text{merge}}$, we show the merging results when it is disabled in Figure D (b). It can be seen that by using $\mathcal{L}_{\text{merge}}$, we can appropriately pull parts together that have the same relative motion, and learn more meaningful decomposition of parts.

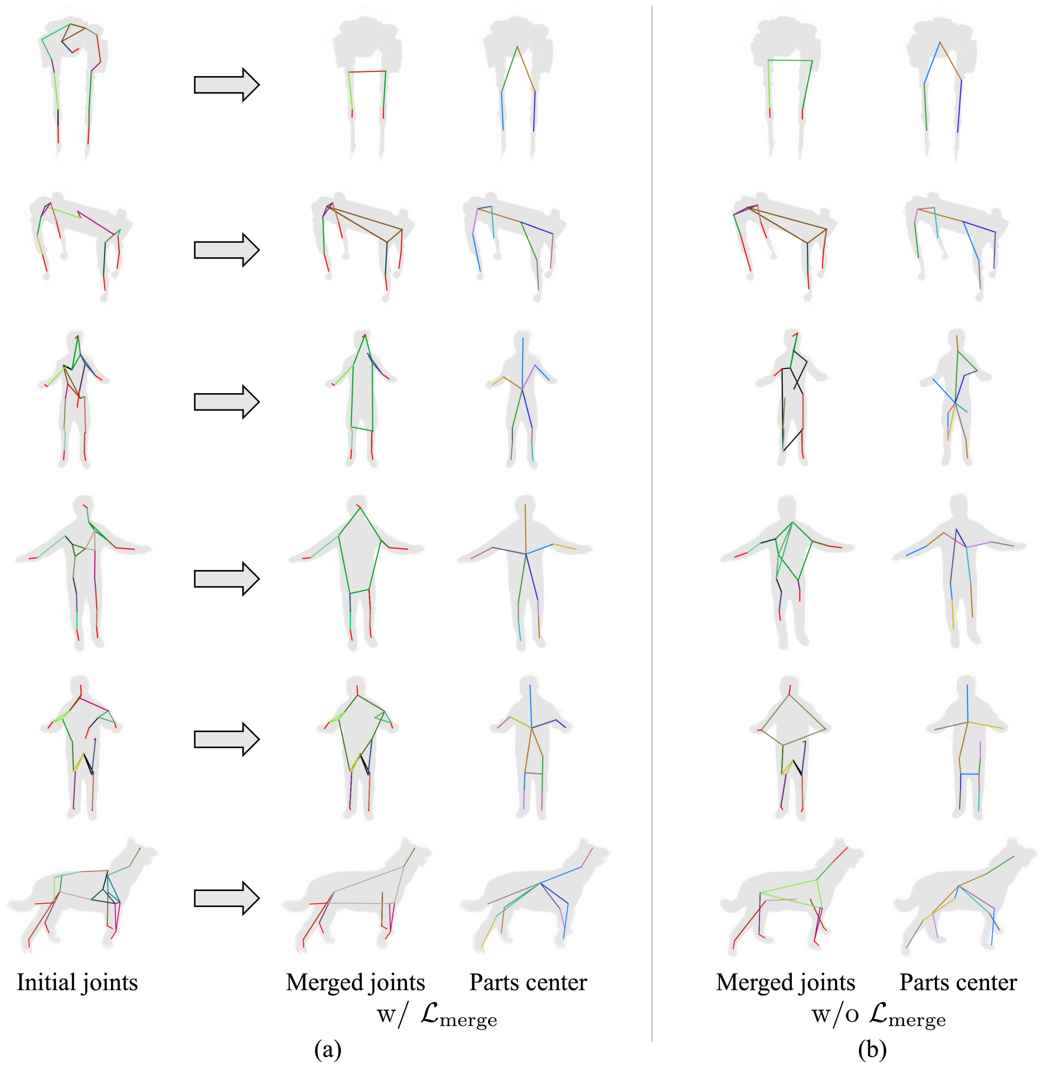


Figure D: (a) Additional results for part merging. From left to right, we show the initial joints and structure, the joints and structure after merging, and the connection between the centers of the parts. Since there is no joint at the endpoint, the center of the part is connected instead (red lines). A polygonal path indicates a part that is connected to multiple parts. The centers of the parts are shown for clarity. (b) Joints and part centers obtained when $\mathcal{L}_{\text{merge}}$ is disabled.

References

[1] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J Black. Keep it smpl: Automatic estimation of 3d human pose and shape from a single image. In *European Conference on Computer Vision (ECCV)*, 2016. 2

[2] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 4

[3] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019. 4

[4] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6M: Large scale datasets and predictive methods for 3D human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2014. 4

[5] Sinead Kearney, Wenbin Li, Martin Parsons, Kwang In Kim, and Darren Cosker. RGBD-Dog: Predicting canine pose from RGBD sensors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 4

[6] Jogendra Nath Kundu, Mugalodi Rakesh, Varun Jampani, Rahul Mysore Venkatesh, and R Venkatesh Babu. Appearance consensus driven self-supervised human mesh recovery. In *European Conference on Computer Vision (ECCV)*, 2020. 2, 3, 4

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. 2

[8] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, 2020. 2

[9] Nathan Morrical, Jonathan Tremblay, Yunzhi Lin, Stephen Tyree, Stan Birchfield, Valerio Pascucci, and Ingo Wald. NViSII: A scriptable tool for photorealistic image generation. *arXiv:2105.13962*, 2021. 4

[10] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D deep learning with PyTorch3D. *arXiv:2007.08501*, 2020. 2

[11] Luca Schmidtke, Athanasios Vlontzos, Simon Ellershaw, Anna Lukens, Tomoki Arichi, and Bernhard Kainz. Unsupervised human pose estimation through transforming shape templates. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2, 3, 4

[12] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 2

[13] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1