Supplements: Template-free Articulated Neural Point Clouds for Reposable View Synthesis

Lukas Uzolas **Elmar Eisemann** **Petr Kellnhofer**
Delft University of Technology
The Netherlands
`{l.uzolas, e.eisemann, p.kellnhofer}@tudelft.nl`

1 Extra Results *Blender* dataset

Table 1, and Fig. 1 show the quantitative and qualitative per-scene results respectively in the *Blender* dataset [1]. While the non-reposable methods often achieve excellent results, our method is a clear improvement with respect to the other reposable method, WIM [2], while simultaneously reducing training time[1].

	Jumping Jacks			**Mutant**			**Hook**			**T-Rex**		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
D-Nerf [1]	32.80	0.98	0.03	31.29	0.97	0.02	29.25	0.96	0.11	31.75	0.97	0.03
TiNeuVox [3]	34.23	0.98	0.03	**33.61**	0.98	0.03	**31.45**	0.97	0.05	32.70	0.98	0.03
HexPlane [4]	31.65	0.97	0.04	33.79	0.98	0.03	28.71	0.96	0.05	30.67	0.98	0.03
Tensor4D [5]	34.43	0.98	0.03	×	×	×	×	×	×	×	×	×
WIM [2]	29.77	0.97	0.04	25.80	0.95	0.06	25.33	0.94	0.06	26.19	0.94	0.08
Ours	**34.50**	0.98	0.03	28.56	0.96	0.03	30.24	0.97	0.05	**32.85**	0.98	0.02

	Stand Up			**Hell Warrior**			**Lego**			**Bouncing Ball**		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
D-Nerf[1]	32.79	0.98	0.02	25.02	0.95	0.06	21.64	0.83	0.16	38.93	0.98	0.10
TiNeuVox [3]	**35.43**	0.99	0.02	**28.17**	0.97	0.07	**25.02**	0.92	0.07	**40.73**	0.99	0.04
HexPlane [4]	34.36	0.98	0.02	24.24	0.94	0.07	25.22	0.94	0.04	39.69	0.99	0.03
Tensor4D [5]	36.32	0.98	0.02	×	×	×	26.71	0.95	0.003	×	×	×
WIM [2]	27.46	0.96	0.04	16.71	0.87	0.14	15.41	0.73	0.25	×	×	×
Ours	31.93	0.97	0.02	27.53	0.96	0.06	17.91	0.76	0.14	×	×	×

Table 1: Quantitative Results Per-Scene on *Blender* dataset. We only highlight best and second-best values for PSNR as the precision reported of SSIM and LPIPS in [3] is not enough for fair comparison in many cases.

2 Extra Results *Robots* dataset

Table 2, and Fig. 2 show the quantitative and qualitative per-scene results respectively for the *Robots* dataset [2]. Consistent with the findings in the main paper, we see that WIM produces overly smooth results while our method can capture more details in a shorter amount of training time. However, it struggles to accurately recover the true poses for long kinematic chains (Table 2, *Iiwa* and *Pandas*).

---

[1] Training times are shown in the main paper.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

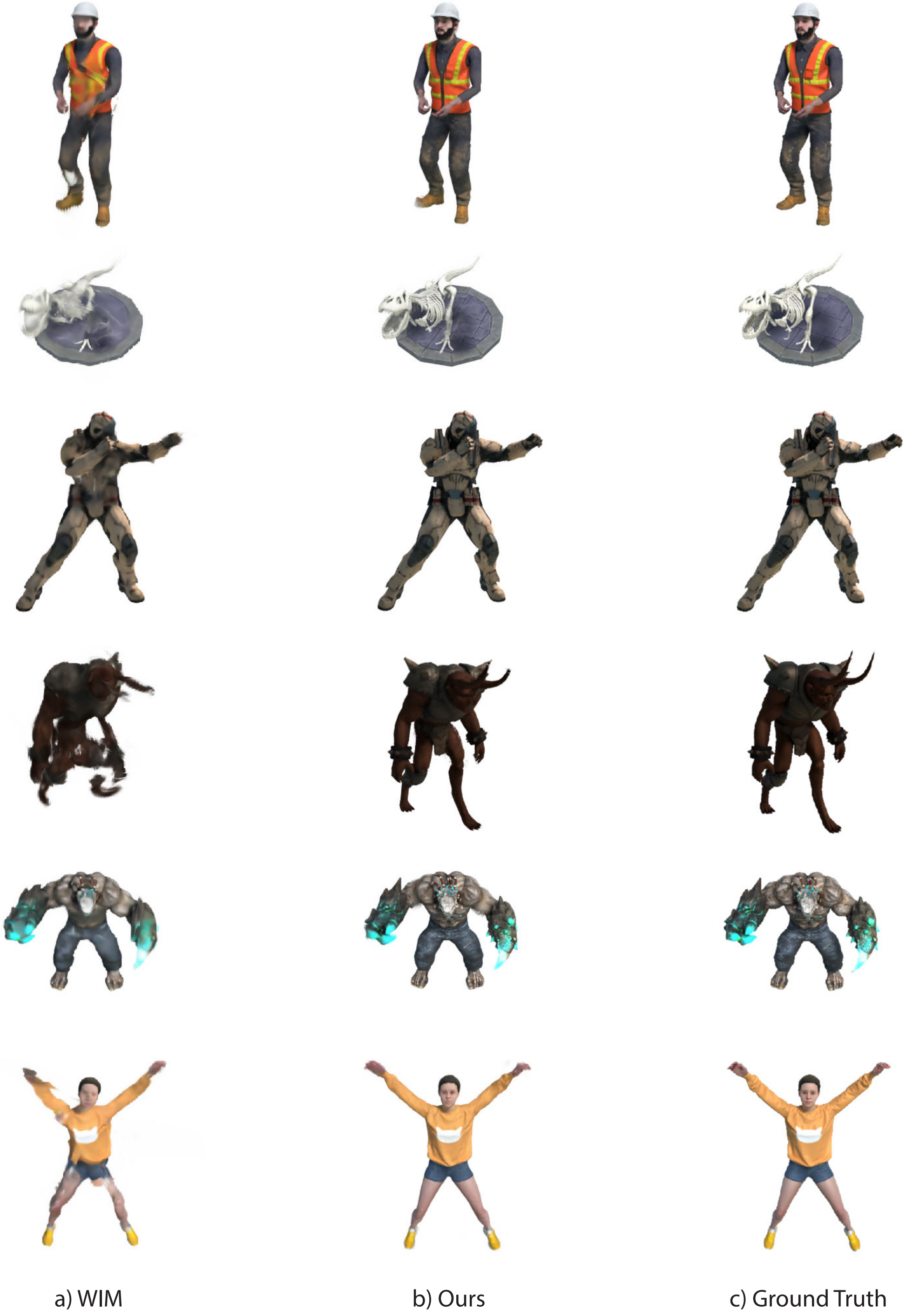

a) WIM b) Ours c) Ground Truth

Figure 1: Additional qualitative per-scene results of our method for the *Blender* dataset. We compare WIM [6], with our method and the ground truth. The displayed results correspond to the final state of the training procedures as described in the paper.

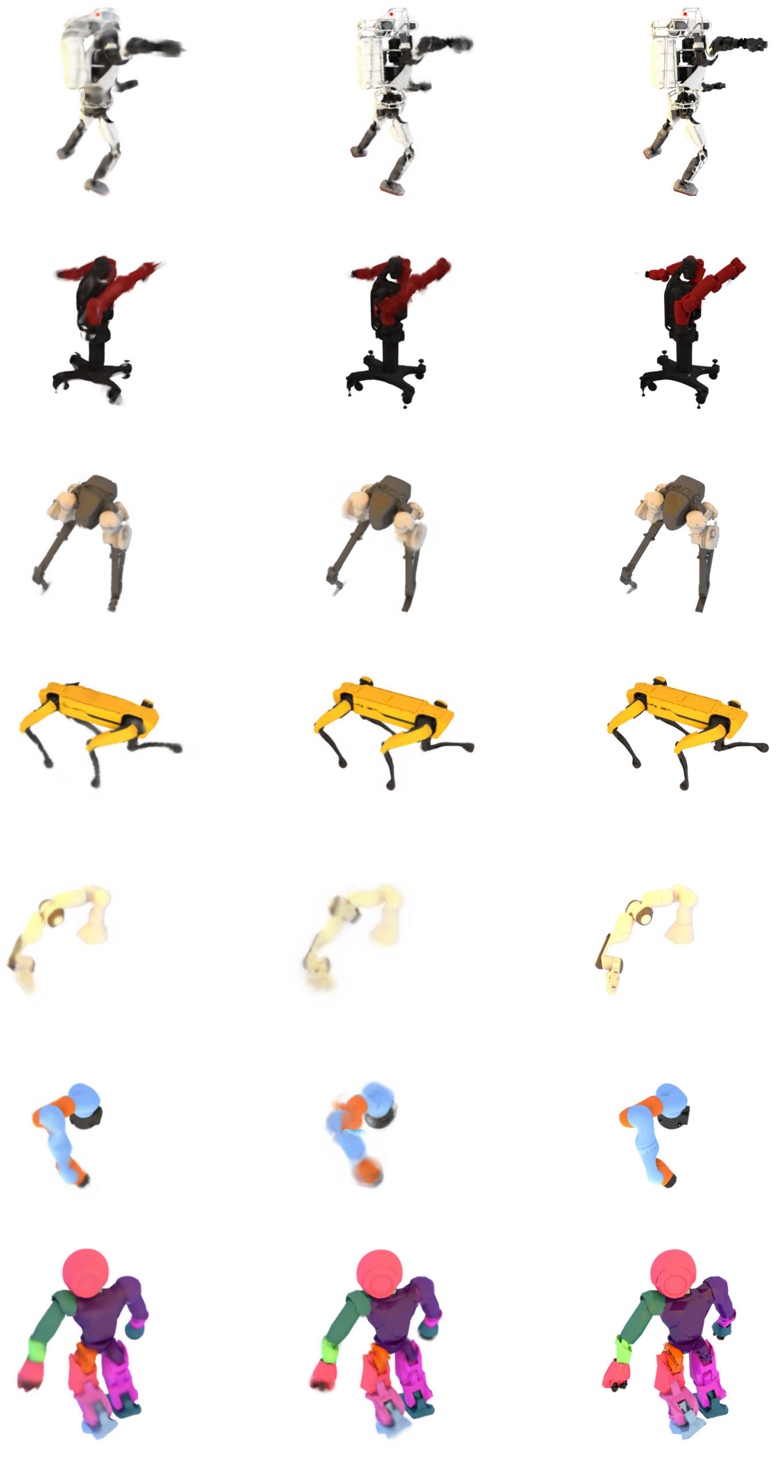


Figure 2: Additional qualitative per-scene results of our method in the *Robots* dataset. We compare WIM [6], with our method and the ground truth. The displayed results correspond to the final state of the training procedures as described in the paper.

	Atlas			Baxter			Cassie			Iiwa		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
WIM[2]	23.99	0.94	0.07	22.70	0.95	0.06	30.20	0.97	0.04	**31.58**	**0.98**	**0.03**
Ours	**28.71**	**0.97**	**0.04**	**28.60**	**0.97**	**0.04**	**31.84**	**0.98**	**0.04**	29.74	0.98	0.04

	Nao			Pandas			Spot		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR ↑	SSIM ↑	LPIPS↓	PSNR ↑	SSIM↑	LPIPS↓
WIM[2]	26.85	0.95	0.06	**32.31**	**0.98**	**0.03**	26.30	0.97	0.04
Ours	**29.50**	**0.96**	**0.05**	30.56	0.97	0.05	**32.40**	**0.98**	**0.02**

Table 2: Quantitative Results Per-Scene in the *Robots* dataset.

3 Implementation Details

3.1 Our method

For training, we utilize PyTorch and the Adam optimizer. We pre-train TiNeuVox for 20k iterations on the synthetic *Blender* data set, and 40k iterations on the WIM data set, and add distortion loss regularization [7] as implemented in [8]. All hyperparameters are adopted from [3].

The neural point clouds are trained for 160k iterations. In each iteration, we sample 8192 rays randomly from multiple views at time step t. On each ray, we sample multiple points p_i as a function of the voxel size of the pre-trained NeRF volume, as specified in [3]. For each sampling point, we query a maximum of 8 neighborhood points $N(p_i)$ within a radius of 0.01 using the KeOps framework [9].

For the kinematic skeleton, we use a bone length of $B_{length} = 10$ for all experiments and a point cloud density threshold of 0.05. We apply the MAT on the binary volume that we retrieve after thresholding the density volume. Assuming a single object, we remove small holes from the volume and extract the medial axis on the biggest blob via the method proposed in [10] and use the scikit-image library [11].

For the mask loss, we project the warped point cloud to five views for the *Robots* dataset and to a single view for *Blender*, as only a single view per timestamp is available. Furthermore, we subsample the point cloud and ground truth mask to $3\,000$ points.

We fine-tune neural point features $\mathbf{f_i}$, skinning weights $\hat{\mathbf{w}}_i$, joints J, density and color regressor Φ_d and Φ_c and train the pose regressor Φ_r and feature point decoder Φ_p from scratch. The MLP of Φ_r has 4 layers of size 128, except for the first layer which has a size of $128 + dim(\gamma(\mathbf{x})) = 191$, where γ is the positional encoding. For all MLPs and the weights, we use a learning rate of 0.0001, except for Φ_r which uses a learning rate of 0.001. We further set the learning rate of α and J to 0.00001. We further utilize learning decay, as in [3, 12] which decays the learning rate by 0.1 after $80\,000$ iterations. We train on a single Nvidia GPU RTX 3090.

3.2 Validation

We used the author's code[2] to reproduce the results of Watch-It-Move [2].

For the *Robots* datasets we adapted the author's configuration files. First, we modified the selection of training and testing views to match the split used in our experiments. That way we held out images from the 10th and 20th camera for testing and provided the remaining 18 cameras for training. Second, to enable early pose fitting and a meaningful computation of progressive learning metrics (see Fig. 4 in the paper), we disabled the gradual scheduler of the training frames. We kept the author's initialization phase with only the first 10 timestamps accessible for the first 10 000 iterations but we provided the entire training set afterwards. This corresponds to the moment past the 1-hour mark in the paper Fig. 4 where the metrics start to improve as the model gets a chance to fit poses of the entire sequence. Without this modification, the network would need another 70 000 steps to access this information. Note that we have not observed a meaningful difference in the quality of the

---

[2] https://github.com/NVlabs/watch-it-move

final trained model as a result of this modification. Finally, we reduced the batch size from 16 to 8 views to fit into the 24 GB VRAM of our Nvidia RTX 3090 GPU.

We applied the same training strategy and settings when training with the *Blender* dataset and used the author's pre-trained snapshots for the *ZJU-MoCap* dataset.

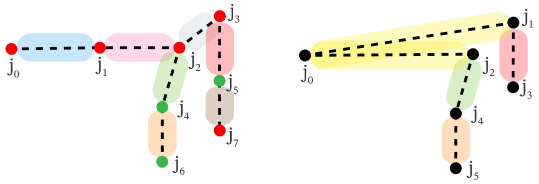
4 Details of Skeleton Simplification

The skeleton simplification is an optional post-processing step that allows to simplify our kinematic model (see Sec. 4.2 in the paper). While our model supports reposing without this step, we suggest that a smaller number of skeletal parameters makes the process easier for the user/animator.

A key part of this procedure is selection of skeleton joints that are redundant and can be removed. To this goal, we identify *static joints* as those that do not exhibit a rotational deviation with respect to the rest pose and relative to its parent joint above a specified threshold in more than 5% of the observed timestamps. Given the low computational cost of this procedure, a user can quickly experiment with ideal selection of this threshold after the training procedure is completed.

After the static joints are identified, they can be removed and the skinning weights corresponding to their children and parent bones can be merged. Here, the parent bone refers to the skeletal tree edge starting in the given joint and pointing towards the root while the children bones are the edges in the leaf direction. The merge only happens for two specific configurations: First, for a static joint, we merge the skinning weights of the children with their parent. Second, we merge the weights of sibling bones if their motion does not differ based on the 5%-heuristic.

To simplify the kinematic skeleton, we further prune bones where possible. A kinematic chain that has multiple consecutive static joints connected by bones will be reduced to the longest possible bone, removing unnecessary joints. However, a joint that functions as a center of rotation for non-static children joints is never pruned. Lastly, static end effector joints are removed. An example of the procedure can be seen Fig. 3.



a) Kinematic Model b) Simplified Kinematic Model

Figure 3: Visualization of kinematic model simplification. a) The trained kinematic mode: The nodes visualize whether the joint is static (red) or not (green). Each bone is associated with a blend-skinning weight (rounded colored rectangles), and j_0 is the root node. b) The simplified kinematic model: Based on the static joints, bones have been pruned and weights have been merged where possible. Weights of bones (j_0, j_1), (j_1, j_2), and (j_2, j_3) have been merged into the root weight (yellow), as all of the joints were marked as static. Furthermore, joints j_1 and j_7 could be removed without harming the underlying kinematic model because they do not have an effect on point cloud deformation. The root node is never pruned.

5 Extra Results Ablations

Extra ablation results for the *Jumping Jacks* scene from the *Blender* dataset can be seen in Fig. 4. We observe that without $\mathcal{L}_{smooth}$ (Fig. 4b), the skinning weights are less consolidated and wrong skinning weights are produced (compared to the full model in Fig. 4a). This limits the image quality represented by the PSNR score. Next, without $\mathcal{L}_{tranf}$ (Fig. 4c) joint rotations are less sparse and, therefore, we detect fewer static joints for removal during the skeleton simplification (see Sec. 4). The presented static joint counts were measured for a simplification threshold of 20 degrees). Next, $\mathcal{L}_{skel}$ (Fig. 4d) does not have a major effect in this scene. However, for scenes with thin structures, omission of $\mathcal{L}_{skel}$ allows the skeleton to drift outside of the geometry. This results in bad reconstruction (see Fig. 8c in the paper). Next, utility of $\mathcal{L}_{ARAP}$ depends on the object class. For

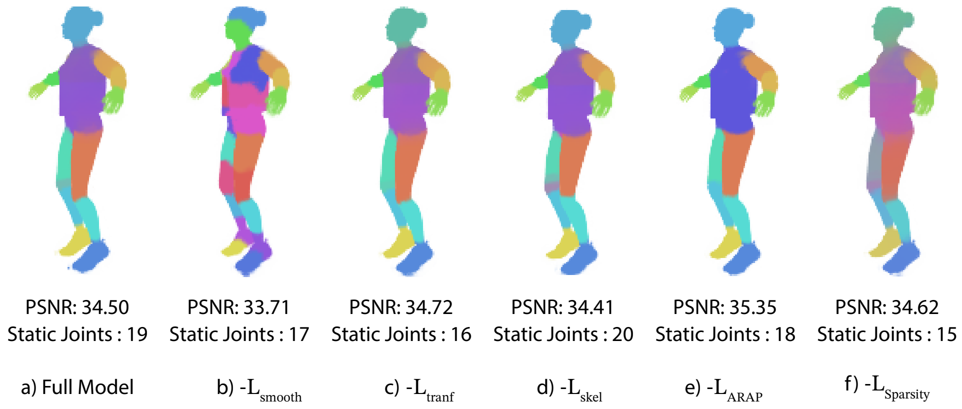


Figure 4: Additional ablation results on *Jumping Jacks* scene from the *Blender* dataset. The skinning weights and PSNR values are displayed prior to the simplification step. The static joint counts refer to the number of joints that can be removed in our simplification step to ease the reposing task. See Sec. 4. b) - f) show the show the results without the denoted regularization term.

Symbol	Description	Trainable	Initialization
$\mathbf{p}_i$	3D Point from canonical point cloud	Fixed	Pre-trained model
$\mathbf{f}_i$	Feature vector associated each $\mathbf{p}_i$	Fine-tuned	Pre-trained model
$\hat{\mathbf{w}}_i$	Blend skinning vector **before** softmax	Fine-tuned	Inverse bone-to-point distance
α	global scalar which scales $\hat{\mathbf{w}}_i$	Trained from scratch	0.1
Φ_c	Color regressor	Fine-tuned	Pre-trained model
Φ_d	Density regressor	Fine-tuned	Pre-trained model
Φ_p	Feature decoder	Trained from scratch	Random
Φ_r	Pose regressor	Trained from scratch	Random
Φ_{pe}	Pose embedding network	Trained from scratch	Random

Table 3: Overview of our notation.

strictly articulated shapes (see the robot in Fig. 8a in the paper), it enforces part rigidity and avoids unrealistic deformations. However, its contribution is less obvious for partially soft shapes such as humans (see Fig. 4e). Despite this, we used the same loss weights for all our results without notable issues. Finally, omission of $\mathcal{L}_{sparse}$ decreases separation of the part labels (note the reduced label color saturation in Fig. 4f). This suggests that this term reduces entanglement between points and joints which indirectly leads to a higher static joint count for skeleton simplification.

6 Notation

An overview of our notation and trainable parameters can be found in Table 3.

7 Training Schedule for the *ZJU-MoCap* dataset

We make two adjustments while training in the *ZJU-MoCap* dataset that both aim to compensate for a bias towards observation of early timestamps due to our incremental training data scheduler. We deem this important since the real captured human subjects do not exhibit all motion modalities uniformly through the entire sequences. This is different from the synthetic *Robots* and *Blender* datasets where motion is simplistic, exaggerated and evenly distributed throughout the sequences.

First, we sample the training timestamps with importance sampling to increase probability of later timestamps and compensate for their shorter overall accessibility during the course of training. We define an importance of a timestamp as the inverse of the total sample count so far. Second, we only enable the sparsity regularization after 160K training iterations when the scheduler converges and all timestamps become accessible. This avoids loss of kinematic joints by early merging when not all motion modalities were observed yet. Neither change leads to a computational overhead.

8 Ours^{pose}: An extension for local deformations

Our full model is based on skeletal articulation with Linear Blend Skinning and as such it can well model large locally rigid deformations. While well suited for many synthetic objects, this alone does not accurately model local deformations of fabrics in the clothes of humans subjects in the *ZJU-MoCap* dataset. Despite this, our full method can perform meaningful articulation of the human subjects as shown in the main paper.

To further improve reconstruction of small surface details, we propose to extend our full method and additionally condition the feature regressor Φ_p by a pose embedding $pe^t \in \mathbb{R}^{64}$ such that $\mathbf{f}^t_{i,x} = \Phi_p(\mathbf{f}_i, \underline{pe^t}, x^t_{\mathbf{p}_i})$ (see Eq. 5 from the main paper). Here, pe^t is regressed by another MLP (4 layers, $0.5 \times |\gamma(\mathbf{J}^c)|$ hidden dimensions) from the canonical joint positions $\mathbf{J^c}$ and the joint positions $\mathbf{J^t}$ observed at time t as $pe^t = \Phi_{pe}(\gamma(\mathbf{J}^c - \mathbf{J}^t))$. This allows the method to learn additional local geometry deformations and color changes specific to the current pose (see Fig. 5). Note, that the poses are not an input to our method as they are already jointly learned in our original full model. We observe, that the learned deformations are locally constrained by the fixed neighborhood search radius in Eq. 6. Furthermore, we detach the gradient flow from $\mathbf{J^t}$ such that the skeletal poses are not optimized to fit the local deformations. Please refer to the main paper for a comparison to our unmodified full method and WIM [2].

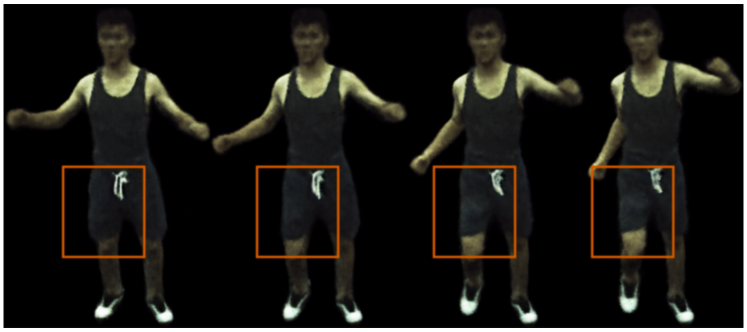


Figure 5: Example of clothes deformations when conditioning Φ_p on the pose embedding in Ours^{pose}.

9 Additional Animations

We demonstrate animation capabilities of our method by creating two animation sequences based on manually defined joint rotations. First, we create a walking sequence of the *Spot* robot from the *Robots* dataset in Fig. 6. Second, we create an over-extended jaw opening animation for the *Trex* from the *Blender* dataset in Fig. 6. Lastly, we animate a scene from *ZJU-MoCap* from two different views Fig. 8.

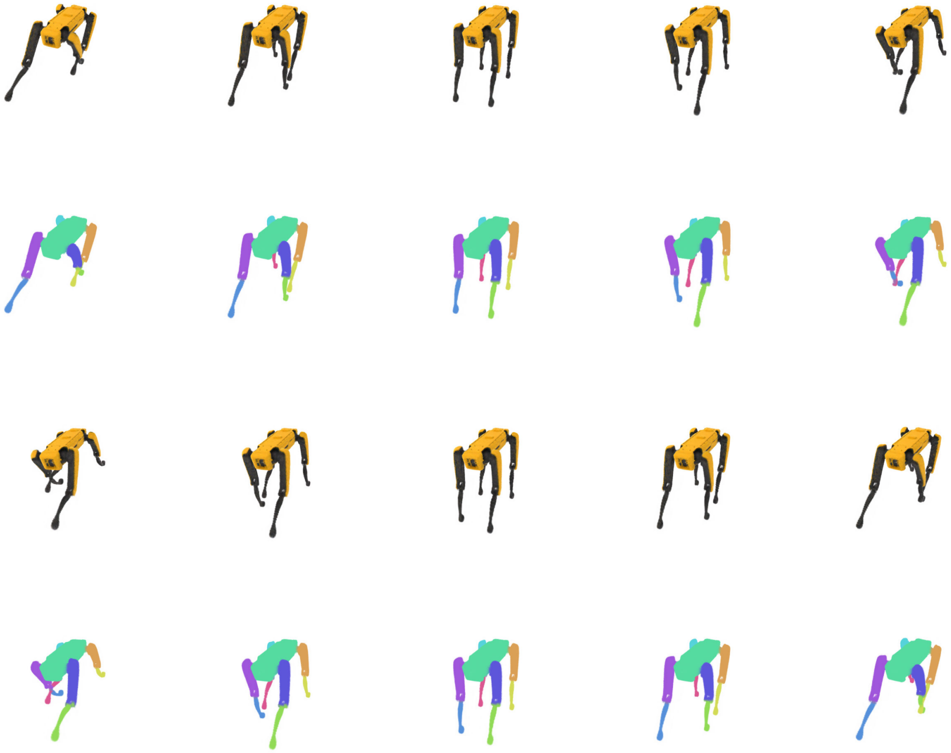


Figure 6: Manually defined walking animation for the *Spot* from the *Robots* dataset.

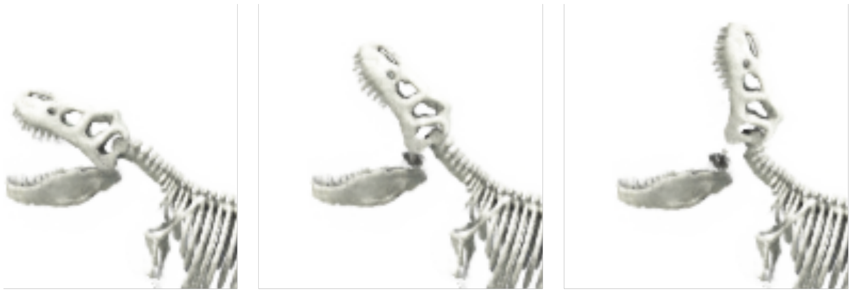


Figure 7: Manually defined jaw over-extension animation for the *T-Rex* from the *Blender* dataset.

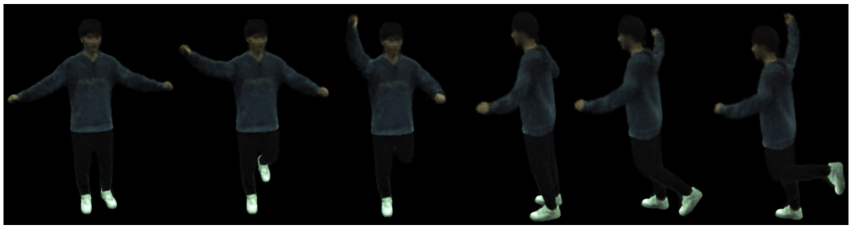


Figure 8: Manually defined animation for scene *384* from the *ZJU-MoCap* dataset from two different views.

References

[1] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. *arXiv preprint arXiv:2011.13961*, 2020.

[2] Atsuhiro Noguchi, Umar Iqbal, Jonathan Tremblay, Tatsuya Harada, and Orazio Gallo. Watch it move: Unsupervised discovery of 3d joints for re-posing of articulated objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3677–3687, 2022.

[3] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022.

[4] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 130–141, 2023.

[5] Ruizhi Shao, Zerong Zheng, Hanzhang Tu, Boning Liu, Hongwen Zhang, and Yebin Liu. Tensor4d: Efficient neural 4d decomposition for high-fidelity dynamic reconstruction and rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16632–16642, 2023.

[6] Atsuhiro Noguchi, Xiao Sun, Stephen Lin, and Tatsuya Harada. Unsupervised learning of efficient geometry-aware neural articulated representations. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVII*, pages 597–614. Springer, 2022.

[7] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

[8] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Improved direct voxel grid optimization for radiance fields reconstruction. *arXiv preprint arXiv:2206.05085*, 2022.

[9] Jean Feydy, Joan Glaunès, Benjamin Charlier, and Michael Bronstein. Fast geometric learning with symbolic matrices. *Advances in Neural Information Processing Systems*, 33, 2020.

[10] Ta-Chih Lee, Rangasami L Kashyap, and Chong-Nam Chu. Building skeleton models via 3-d medial surface axis thinning algorithms. *CVGIP: graphical models and image processing*, 56(6):462–478, 1994.

[11] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[12] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022.