Supplemental Material
Transfer4D: A framework for frugal motion capture and deformation transfer

Contents

Introduction

To keep the overall manuscript self-contained, we include additional details in the supplementary material. Video explaining our work and showcasing our results can be found at `5738.mp4`. The source code for our pipeline can be found in directory `Code`. Details of our setup and implementation of the baselines can be found at: `Code/README.md`

1. Description of benchmark datasets

For hyperparameter-tuning and evaluating NRR and skeletonization, we use DeformingThings4D [24] dataset. It contains 31 categories of animals and humans. Overall, it contains 59 distinct animals (e.g. bear, dino, elk) and 67 humanoids (e.g. prisoner, mannequin) performing. We sample 80 sequences from the test split provided by Lepard [9]. Each animation contains between 12 and 150 frames (mean=47 frames). Depth videos are created using Blender's Eevee engine from distinct camera positions [24]. Unfortunately, the dataset doesn't provide texture or color information and only depth can be estimated. All depth maps are rendered using the intrinsic parameters of Azure Kinect. Meshes for the characters, `gorilla, minotaur, centaur, duck` were sourced from TurboSquid.[1]

2. Non-Rigid Registration (NRR)

NRR extracts the temporal motion information, establishing correspondences among subsequent frames.

2.1. NRR setup

Consider a single view depth camera setup that provides a set of depth images, $\mathcal{D} = \{D^t \in \mathbb{R}^{H \times W}\}$, where t, H, W represents timestep, height, and width of frames respectively.[2] We segment the object of interest based on the depth values to obtain a binary image mask. We assume the camera is stationary during the recording and camera intrinsic parameters $K \in \mathbb{R}^{3X3}$ are known. Using the camera parameters, for every pixel $u \in \mathbb{R}^2$ in-depth image D^t is back-projected to create the point cloud $P = \{p_u\}$.

$$p_u = D^t(u_x, u_y) \cdot K^{-1} \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix} \tag{1}$$

For a depth video of T frames, we choose one frame in the video as the source S. An incomplete mesh $\mathcal{M}_S = \{V_S, F_S\}$ is obtained from the source depth image where its vertices $V_S = P^S$ and the faces F_S are obtained by connecting adjacent pixels if the distance between their associated vertices is less than a fixed threshold (0.05 in our experiments).

2.2. Surface Deformation Model

To enforce spatial coherence, we represent the motion field using an embedded deformation graph [18] $G = \{V_G, E_G, \mathcal{R}_G^T, \mathcal{T}_G^T\}$. Here each node $g_j \in V_G$ is equipped with a time-varying rigid transformation, i.e. a rotation matrix $R_j^t \in \mathbb{R}^{3\times3}$ and translation vector T_j^t for each timestep

[1] www.turbosquid.com

[2] We experiment using the synthetic datasets processed from DeformingThings4D [24].

t. This provides a generalized solution to represent the deforming scene without using domain-specific methods like SMPL [12], SMAL [26], or MANO [16] (designed for humans, quadrupeds, and hands respectively).

The incomplete mesh at source frame $\mathcal{M}_S$ is used to create the embedded graph. Erosion is performed to remove outliers. Similar to [3], the nodes $V_\mathcal{G}$ of the graph are generated by uniformly sampling from the mesh $\mathcal{M}_\mathcal{S}$, while ensuring σ-node coverage (each vertex in the point cloud is at most σ distance from the nearest node in $V_\mathcal{G}$). The graph edges $E_\mathcal{G}$ connect nodes with overlapping influence. The edge weights are computed using the geodesic distance between the graph nodes. Each node can have a maximum of 8 neighbours. The skinning weights $W^\mathcal{G}(i,j)$ determines the influence of graph node g_j on vertex v_i. It is defined via

$$W^\mathcal{G}(i,j) = \alpha e^{-(\|v_i - g_j\|_2^2)/(2\sigma_{nc}^2)} \tag{2}$$

where α is the normalization constant so that weights sum to one, and the node coverage parameter σ_{nc} controls the weightage of multiple graph nodes on the vertices. Inspired by linear blend skinning, to enforce sparseness and faster GPU computation each vertex is influenced by at most by 4 nodes, i.e. $\|W^\mathcal{G}(i,:)\|_0 \leq 4$.

2.3. Surface Registration

Our first step is to obtain the trajectory of the vertices of the incomplete mesh $\mathcal{M}_\mathcal{S}$ by aligning it to all the future frames using non-rigid registration. The trajectory for each vertex $v_i \in V_S$ at timestep $t \in T$ is computed using the deformation graph as

$$Traj_i^t = \sum_{j=0}^{N_\mathcal{G}} W^\mathcal{G}(i,j)(\mathcal{R}_j^t(v_i - g_j) + g_j + \mathcal{T}_j^t), \tag{3}$$

The registration is performed using RANSAC and N-ICP [7]. For each timestep t, the graph deformation parameters $\{\mathcal{R}_G^t, \mathcal{T}_G^t\}$, are estimated by optimising the following energy function $E(\mathcal{G})$:

$$\begin{aligned} E(\mathcal{G}) &= \lambda_{corresp} E_{corresp}(\mathcal{G}) + \lambda_{smooth} E_{smooth}(\mathcal{G}), \\ E_{corresp}(\mathcal{G}) &= \sum_{v_i \in \mathcal{V}_\mathcal{S}} \| Traj_i^t - P_x^t \|_2^2, \\ E_{smooth}(\mathcal{G}) &= \sum_{(i,j)\in\mathcal{E}_\mathcal{G}} \|R_i^t(g_j - g_i) + g_i + T_i^t - (g_j + T_j^t)\|_2^2, \end{aligned} \tag{4}$$

where P_x^t is the closest point to $Traj_i^t$ in the new point cloud P^t. In each iteration, K points are sampled from $V_\mathcal{S}$ and transformed using Eq. 3. The data term $E_{corresp}$ seeks to align the incomplete mesh M_S to P^t. The as-rigid-as-possible [17] constraint E_{smooth} enforces nearby graph nodes to have similar transformations.

Method	DIC $\times 10^{-3} \downarrow$	IOU $\uparrow$
N-ICP [7]	$5.90^{\pm 4.56}$	$0.90^{\pm 0.06}$
Lepard [9]	$9.53^{\pm 8.80}$	$0.61^{\pm 0.23}$
N-ICP + Lepard	$6.36^{\pm 4.92}$	$0.85^{\pm 0.13}$
N-ICP + Lepard + Confidence	$6.38^{\pm 6.34}$	$0.84^{\pm 0.14}$
N-ICP + Lepard + OcclusionFusion [11]	$6.41^{\pm 6.42}$	$0.84^{\pm 0.14}$

Table 1. Performance scores across Non-Rigid-Registration variants

2.4. Optimization

Eq. (4) is optimised by stochastic gradient descent (SGD) [5] for the non-convex optimisation using pytorchv1.11 [14]. To represent rotations we use lietorch [21]. It represents rotation as an element of the SE(3) Lie group and performs backpropagation in the tangent space of the group's manifold. Node coverage σ_{nc} is set to 0.05. The learning rate is initialized to $6e^{-2}$ and exponentially decayed by 0.999 after each iteration. We run 100 iterations using chamfer distance. $K = 5000$ points are sampled at each iteration of SGD. $(\lambda_{corresp}, \lambda_{arap})$ are set to $(1000, 10)$. We use the transformations estimated at the previous timestep $t-1$ to initialize registration at the current timestep t.

2.5. NRR Experiments

To evaluate our setup for non-rigid registration, we compare our setup and incorporate other methods during NRR. Table. 1 summarises our observations. We observe that N-ICP results in the lowest depth inverse cost (DIC) and the highest IOU. These two metrics are defined below.

Metrics: For each timestep we generate the depth image $\hat{D}^t \in \mathbb{R}^{H\times W}$ using the deformed source mesh $\mathcal{M}_S^t = \{Traj^t, F_S\}$. We use the MeshRasterizer from pytorch3Dv0.6 [15] to create the depth images. The depth value of each pixel is calculated from the z-buffer during rasterization. We report 2 metrics to evaluate our setup for NRR.

- **Depth Inverse Cost:** To measure discrepancy in deformation, we evaluate the difference in the predicted ($\hat{D}^t$) and input depth values (D^t). The background represents infinite depth. Silhouette is defined as pixels with valid depth values.

$$DIC = \sum_t^T \sum_i^H \sum_j^W \left\| \frac{1}{D^t[i,j]} - \frac{1}{\hat{D}^t[i,j]} \right\|^2 \tag{5}$$

- **IOU:** To measure surface misalignment, we measure the IOU (intersection over union) between the predicted and input silhouette images, where:

$$\hat{Silh}^t[i,j] = \begin{cases} 1 & \text{if } \frac{1}{\hat{D}^t[i,j]} > 0 \\ 0 & \text{otherwise} \end{cases}$$
$$IOU = \sum_t^T \frac{\hat{Silh}^t \cap Silh^t}{\hat{Silh}^t \cup Silh^t} \quad (6)$$

Other methods:

- **Lepard [9]:** Lepard is used to in estimating scene flow from the source and to the target point cloud, Lepard uses a KP-FCN [22] backbone for feature extraction and down-sampling. KP-FCN uses spherical convolution to aggregate information across the point cloud. After down-sampling the source and target point cloud, Lepard uses 2 self-attention and 2 cross-attention blocks between the down-sampled source and target point cloud to obtain a confidence map between the points. Correspondences with confidence less than 0.1 are considered invalid. These correspondences are then interpolated using k-nearest neighbours to get the target position of each source point. We run for 200 iterations using correspondence predicted by Lepard, followed by 100 iterations with RANSAC and N-ICP similar to our setup.

- **Correspondence confidence:** Not all surface correspondences from Lepard can be used as outliers have to be reduced. Self-occlusion can also affect correspondence. Although confidence values are predicted by Lepard, they do not accurately represent outliers. We obtain the initial correspondence using Lepard and inspired from [8], we also calculate the confidence score based on cyclic consistency. For a point $v_i \in V_S$, the confidence $Conf_p$ is defined as:

$$SF_i = Lepard(V', T), SF'_i = Lepard(V' + SF, V')$$
$$Conf_i = \exp^{-\|SF - SF'\|^2/\sigma_{conf}} \quad (7)$$

  where SF, SF', $Conf$ denote scene flow, scene flow inverse (from target space to the source), and predictive confidence respectively. σ_{conf} is set to 0.01. We replace $E_{corresp}$ in Eq. (4) with:

$$E_{corresp}(\mathcal{G}) = \sum_{v_i \in \mathcal{V}_S} Conf_i \| Traj_i^t - P_x^t \|_2^2, \quad (8)$$

  Similarly to Lepard, we run for 200 iterations using confidence calibrated correspondence followed by 100 iterations with RANSAC and N-ICP similar to our setup.

- **OcclusionFusion [11]:** As occlusion adversely affects motion tracking, we even attempt to incorporate OcclusionFusion into NRR. It uses a LSTM-involved graph neural network to predict the motion of the occluded region. Visible nodes are calculated from the confidence score described above. We incorporate an additional loss term E_{Occ} defined as:

$$E_{Occ} = \sum_{j=0}^{N_\mathcal{G}} w_j \| \mathcal{T}_j^t - (\mathcal{T}_j^{t-1} + \mu_i) \|_2^2 \quad (9)$$

  where w_j and μ_j are predicted by OcclusionFusion [11]. $\mathcal{T}_j^t$ is the translation of the graph node j and timestep t. λ_{Occ} is set to 1.0.

2.6. Limitations of NRR

We use the traditional setup of RANSAC and N-ICP [7] for NRR and use its estimated trajectory for further steps of our pipeline. This makes our setup susceptible to failure in case of large deformation between the source and target frame. There exist other methods such as Lepard (for improving correspondence prediction), cyclic-consistency based predictive confidence, OcclusionFusion (adding additional constraints on the optimization) to handle large deformations. However, we observe that, an auto-regressive setup using standard N-ICP where the deformation parameters are initialized using values estimated at previous timestep results received lower in better scores on the DIC and IOU metrics (see Tab. 1). This supports our assumption that the extent of deformation between consecutive frames of the source object to be low.

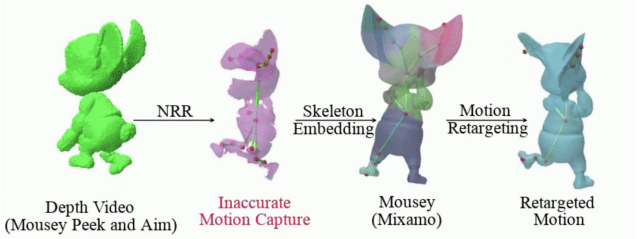


Figure 1. Failure case from non-rigid registration step.

A better strategy for reconstruction is employed by DynamicFusion [13] where depth images are registered and fused to simultaneously reconstruct the source object geometry and estimate its warpfield. Unfortunately, we were not able to reproduce the results provided by DynamicFusion. Therefore, we do not use DynamicFusion during NRR. We do encounter some cases where NRR fails, particularly motions with $180°$ rotations like spinning or turning such as Fig. 1. We believe that incorporating dynamic volumetric integration methods like DynamicFusion as future work would fix these artifacts.

3. Skeletonization

Algorithm 1 describes our skeletonization algorithm. The input to our algorithm is the incomplete mesh $\mathcal{M}_S$ =

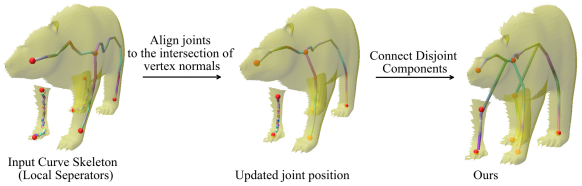


Figure 2. **Curve skeleton extraction:** Notice that the extracted joint position from Local separators [1] lies on the surface of the incomplete mesh. Our optimization aligns the joint position to the medial axis of the object.

$\{V_S, F_S\}$ of the source object and its trajectory $Traj \in \mathbb{R}^{T\times|V_S|\times 3}$, where T is the number of time steps. The output is a kinematic pose tree, i.e. a motion skeleton $\mathcal{MS} = \{J_{MS}, B_{MS}\}$ where J_{MS} is a set of joints in $\mathbb{R}^3$ and bones B_{MS} are the edges connecting the joints, along with its skeletal motion $SM \in \mathbb{R}^{T\times|J_{MS}|\times 3}$.

3.1. Curve Skeleton Extraction

We compute the curve skeleton from the incomplete source mesh $\mathcal{M}_S$ using the local separators method of Bærentzen et al. [1].

Bærentzen et al.'s method places each node of the curve skeleton at the centroid of its associated mesh vertices S_i. We replace this simple scheme for node placement with an approach inspired by ROSA [20] to better handle incomplete shapes, as follows. The optimal position of a node j_i^* is obtained as

$$j_i^* = \arg\min_{j_i\in\mathbb{R}^3} \sum_{v\in S_i} \lambda_{cpp}\|(j_i - v)\times n(v)\|_2^2 + \lambda_{eucl}\|j_i - v\|_2^2 + \lambda_{smooth}\sum_{k\in N(i)}\|j_i - j_k\|_2^2 \tag{10}$$

where $n(v)$ is the vertex normal for vertex v, $N(i)$ are the neighbours of node j_i.

The first term enforces the node to lie near the medial surface of the object by constraining the node to the intersection of the separator's vertex normals, also known as its closest projection point (CPP). If only a fraction of the surface is visible, the CCP cannot be accurately calculated. Hence we also align each joint to the centroid of the separator. Lastly, we spatially smooth the joint position by constraining nearby joints to be close to one another. $(\lambda_{cpp}, \lambda_{eucl}, \lambda_{smooth})$ are set to $0.8, 0.2, 200$.

Fig. 2 shows the result of the optimization on the bear drinking example. The extracted joint position from local separators [1] lies on the surface of the incomplete mesh. Our optimization aligns the joint position to the medial axis of the object.

3.2. Splitting Clusters

Fig. 3 shows the splitting procedure for the bear example. e_{split} is set to 0.2 for all samples. At most, each curve will be split thrice.

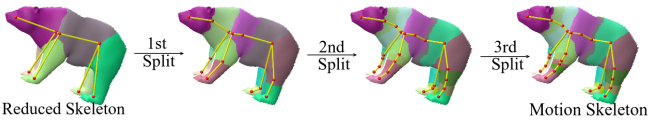


Figure 3. **Curve splitting procedure:** To incorporate the motion information, each curve of the extracted curve skeleton is split into multiple bones.

3.3. Skeletonization Experiments

Curve Skeleton Comparison: Fig. 4 compares our approach with different static skeleton extraction methods. Since these methods do not incorporate motion information, they produce temporally incoherent skeletons. We don't compare with Au et al [19] since it requires a complete watertight mesh for skeletonization. Although ROSA works on incomplete mesh, it cannot accurately capture the structure of the object. Notice, LS [1] struggles to find the accurate position of the joints. We overcome this drawback by aligning the joints to the closest projection point of each local separator. We observe that Rignet [23] and point2skeleton [10] are not able to generalize to incomplete mesh.

Skinning weights evaluation:

To evaluate the skeletonization algorithm, we cannot directly compare the computed skeleton with a ground truth skeleton as the dataset does not contain such data. Instead, we propose two geometric metrics to evaluate the quality of the resulting articulated motion AM of the incomplete source mesh.

- **Point-Plane Distance (p2p):** We compare AM with the motion of the complete mesh sequence GT. For this, we compute the average point-to-plane ($p2p$) distance as

$$p2p = \frac{1}{TN}\sum_{t=0}^{T}\sum_{v\in V_S}\|n(GT_x^t)(AM_v^t - GT_x^t)\|_2 \tag{11}$$

  where $x = \arg\min_{x\in GT^t}||AM_v^t - GT_x^t||_2$ is the closest point in the ground truth mesh, T is the number of time steps, and $n(GT_x^t)$ is the unit normal vector at the closest point.

- **Reconstruction Error (RE):** We also can compare AM with the trajectory $Traj$ obtained from NRR. We define the reconstruction error RE of the entire mesh as

$$RE = \sqrt{\frac{1}{TN}\sum_{t=0}^{T}\sum_{v\in V_S}\|Traj_v^t - AM_n^t\|_2^2} \tag{12}$$

  where T is the number of time steps in the input sequence, and N is the number of vertices at each time step.

Algorithm 1: Skeleton extraction of the source object.

Input: An incomplete mesh $\mathcal{M}_S = \{V_S, F_S\}$ of the source object with trajectory $Traj \in \mathbb{R}^{T\times|V_S|\times 3}$, where T is the number of time steps.

Output: A kinematic pose tree, i.e. a motion skeleton $\mathcal{MS} = \{J_{MS}, B_{MS}\}$ where J_{MS} is a set of joints in $\mathbb{R}^3$ and bones B_{MS} are the edges connecting the joints, along with its skeletal motion $SM \in \mathbb{R}^{T\times|J_{MS}|\times 3}$.

$\mathcal{CS} = LS(\mathcal{M}_S)$ /* Compute the Local Separators using [1] */

$\mathcal{CS} = merge\_cycles(\mathcal{CS})$

$\mathcal{CS} = prune\_skeleton(\mathcal{CS}, 3)$ /* Now, compute optimal j^* using Eq. (10) */

$j_i^* = \arg\min_{j_i\in\mathbb{R}^3} \sum_{v\in S_i} \lambda_{cpp}\|(j_i - v)\times n(v)\|_2^2 + \lambda_{eucl}\|j_i - v\|_2^2 + \lambda_{smooth}\sum_{k\in N(i)}\|j_i - j_k\|_2^2$

$\mathcal{CS} = connect\_disjoint\_components(\mathcal{CS})$

// Initialization

J_{MS} = $\mathcal{CS}$.functional_nodes()

// Curve Splitting

for *each split* **do**

  for $b \in B_{MS}$ **do**

  $\{\mathcal{R}_b, \mathcal{T}_b\} = Kabsch()$

  $RC(i,k) = \frac{1}{T}\sum_{t=1}^{T}\sum_{v\in C_{ik}}\|Traj_v^t - (\mathcal{R}_b^t v + \mathcal{T}_b^t)\|_2^2$

  $r^* = \arg\min_{r\in C_{ik}} RC(i, r-1) + RC(r, k)$

  $\epsilon_{change} = 1 - \frac{RC(i,r-1) + RC(r,k)}{RC(i,k)}$

  if $\epsilon_{change} > \epsilon_{split}$ **then**

    // Split Bone

    $J_{MS} = J_{MS} \cup \{j_{r^*}\}$

    $B_{MS} = B_{MS} \cup \{(i,r),(r,k)\} \setminus \{(i,k)\}$

  end

end

$W, \{(\mathcal{R}_b, \mathcal{T}_b), b \in B_{MS}\} = SSDR(\mathcal{M}_S, Traj)$

$AM_v^t = \sum_{b\in B_{MS}} W(v,b) * (\mathcal{R}_b^t v + \mathcal{T}_b^t)$

$SM_j^t = j + \sum_{v\in S_j}(AM_v^t - v)$

3.4. Skeleton embedding comparison

Experiment setup: DeformingThings4D dataset contains mesh motion sequences of various animals and humans. In order to have a ground-truth target motion for evaluation, we transfer motion obtained from the single-view depth video back to the same complete mesh from which the video was extracted. Thus the ideal result is to recover the original motion of the complete mesh.

Metrics:: To evaluate the our skeletonization framework for animation transfer, we report 3 metrics in the main paper.

- **Percentage Joints Embedded**: This metric measures the percentage of joints from the source motion skeleton SM that were successfully embedded into the target mesh using Pinocchio [2].

$$PJE = \frac{n(TS)}{n(MS)} * 100 \tag{13}$$

- **Reconstruction Error**: For a complete mesh $CM = V_S, F_S$, we compare the re-targetted motion RM with the original mesh sequence $GrTraj \in R^{T\times n(V_S)\times 3}$. We define the reconstruction error RE as

$$RE = \sqrt{\frac{1}{TN}\sum_{t=0}^{T}\sum_{v\in V_S}\|GrTraj_v^t - RM_v^t\|_2^2} \tag{14}$$

  where T is the number of time steps in the input sequence, and N is the number of vertices.

- **Local Error Pose** tests the reconstruction error after rigidly aligning the re-targeted motion with the original motion at each timestep. By removing the global transformation (R_p^t, Tr_p^t) we can observe whether the local pose (deformation of arms, or movement of legs) is accurately being re-targetted. This is similar to local

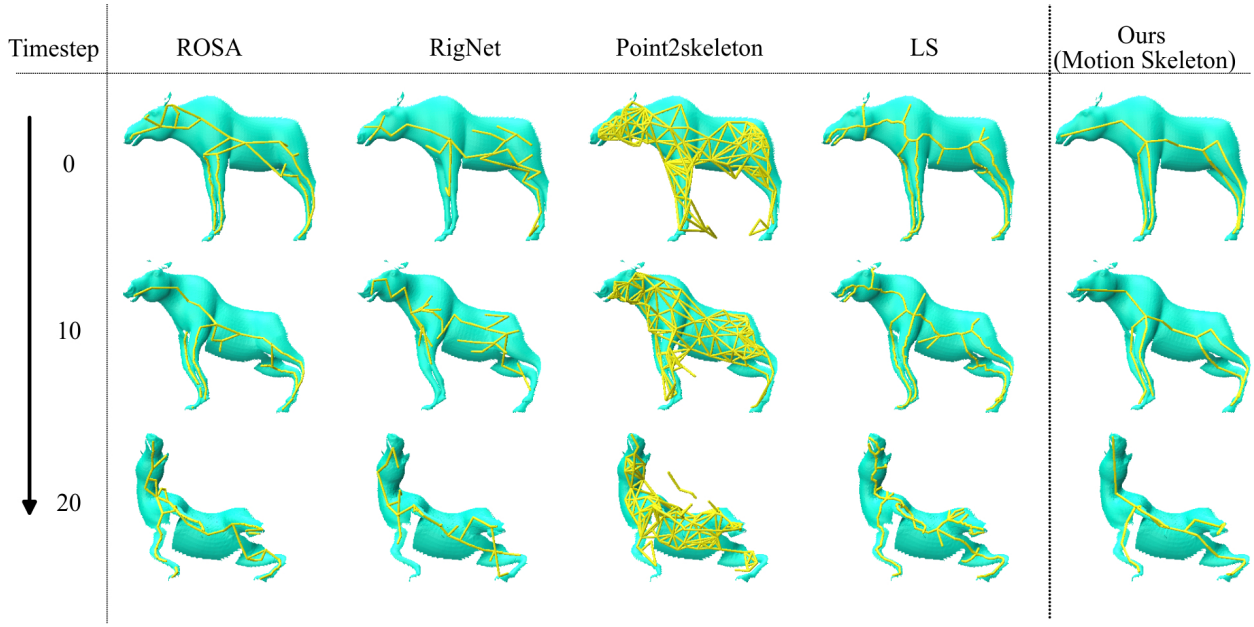


Figure 4. Comparison of skeleton extraction methods (namely, ROSA [20], RigNet [23], Point2Skeleton [10], Local Separators [1]) on Elk(Death sequence) from DeformingThings4D [24]. Point2Skeleton outputs a skeleton mesh instead of a skeleton tree unlike skeleton extracted from local separators that aligns best with the shape of the object. Hence, we choose local seperators as a first step for our skeleton extraction.

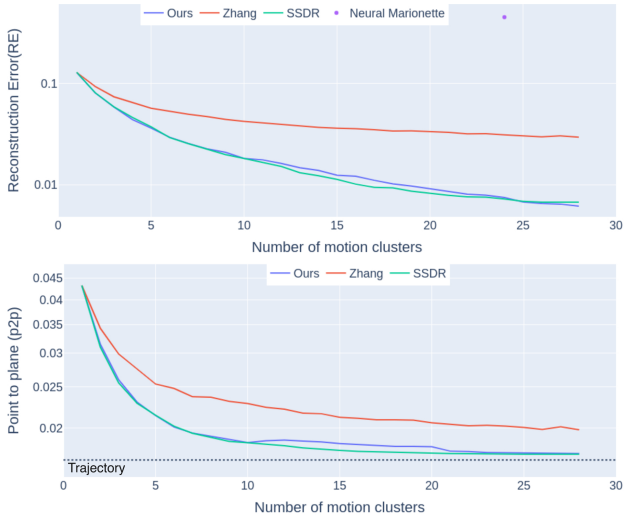


Figure 5. Quantitative comparison of skeletonization with baselines, Zhang et al. [25], SSDR [6], Neural Marionette [4] in terms of two geometric metrics, reconstruction error and point-plane distance to evaluate the quality of the resulting articulated motion AM of the incomplete source mesh.

3D seen in human pose estimation papers.

$$LPE = \sqrt{\frac{1}{TN}\sum_{t=0}^{T}\sum_{v\in V_S}\|GrM_v^t - (R_p^t * RM_v^t + Tr_p^t)\|_2^2} \quad (15)$$

where locally aligned re-targetted motion is obtained by performing procrustes' analysis between GrM^t and RM^t

Failure cases: Fig. 6 shows a few failure cases of our algorithm. These occur mainly because the approach depends on the Pinocchio framework for skeletal embedding. The failure in Fig. 6 (a) is due to large pose variations between the source and target, and Fig. 6 (b) is due to a mismatch in the global orientation of poses between source and target.

4. Cost Comparison with Existing Techniques

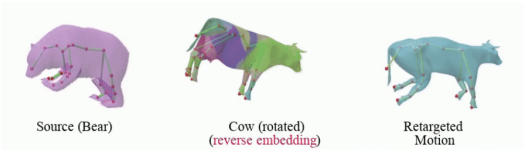
The existing professional motion capture techniques[3] use a complex setup of multiple cameras to record 3D positions and actions of performers, which in turn are used to animate the digital character. The volume of animation data generated at low latency, and complex movement capture comes at a prohibitive cost for small production houses. Some of them include *Motion Capture NYC*[4] where the rent cost ranges from $\$4,000$ a day + $\$20$ a second on top. Meta Motion [5] sells the motion capture systems for price range between $\$800$ to $\$250000$. Marker-less motion capture the software costs $\$99$ and the system costs $\$2500$ and the mocap suit

[3] https://studio.knightlab.com/results/oscillations-vr/oscillations-mocap-comparison/

[4] http://www.motioncapturenyc.com/cost

[5] https://metamotion.com/FAQ/prices.html

(a) Our approach fails in situations of large pose variations between source and target.



(b) In addition, the global orientation of the target shape should be aligned with the source shape

Figure 6. Transfer4D failure cases from motion retargeting. Note: mainly because the approach depends on the Pinocchio framework for skeletal embedding.

Step	Time(seconds)	CPU (GB)	GPU (GB)
NRR	3629.71	10.54	5.37
Skeletonization	264.52	8.67	0.51
Skeleton Embedding	45.04	0.9	-
Motion Retargeting	16.15	0.67	-

Table 2. Computation cost of our pipeline on the bear to cow example.

alone costs $2500 [6]. High FPS Camera costs between $500 to $500. To facilitate democratization, we utilize the depth video streams that can be used to deform the corresponding target meshes. The potential benefits are to reduce in the turnaround time for the animator, and a reduction in the cost of motion capture by a huge factor with some tradeoff in high fidelity reconstruction achieved by sophisticated motion capture systems with markers. A *Kinect V.2* would cost $\sim$ $499 and the newer version *Azure Kinect* is priced at $300. Other frugal alternatives include depth cameras with Intel Real Sense priced at $200 [7]. Our solution would just need a single-view depth camera and target mesh without a rig to animate, meaning the rough expense would be around $\$100 - 300$.

5. Computation Cost

We give a ballpark estimate of the cost of using our pipeline using the "bear (drinking)" example. Tab. 2 highlights the time, RAM usage, and GPU memory usage taken by each step. The video contains 120 frames. The extracted incomplete mesh contains 70,240 vertices and 138,300 faces. The embedded graph contains 903 nodes. The motion skeleton contains 22 joints. The target mesh (cow) contains 21,158 vertices and 42,312 faces.

[6] https://www.rokoko.com/en/products/smartsuit-pro

[7] https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d435i.html

References

[1] Andreas Bærentzen and Eva Rotenberg. Skeletonization via local separators. *ACM Trans. Graph.*, 40(5), sep 2021. 4, 5, 6

[2] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3):72–es, July 2007. 5

[3] Aljaz Bozic, Pablo Palafox, Michael Zollöfer, Angela Dai, Justus Thies, and Matthias Nießner. Neural non-rigid tracking. 2020. 2

[4] Bae Jinseok, Jang Hojun, Min Cheol-Hui, Choi Hyungun, and Young Min Kim. Neural marionette: Unsupervised learning of motion skeleton and latent dynamics from volumetric video. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, February 2022. 6

[5] J. Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23:462–466, 1952. 2

[6] Binh Huy Le and Zhigang Deng. Smooth skinning decomposition with rigid bones. *ACM Trans. Graph.*, 31(6), 2012. 6

[7] Hao Li, Robert W. Sumner, and Mark Pauly. Global correspondence optimization for non-rigid registration of depth scans. In *Proceedings of the Symposium on Geometry Processing*, SGP '08, page 1421–1430, Goslar, DEU, 2008. Eurographics Association. 2, 3

[8] Xueqian Li, Jhony Kaesemodel Pontes, and Simon Lucey. Neural scene flow prior. *Advances in Neural Information Processing Systems*, 34, 2021. 3

[9] Yang Li and Tatsuya Harada. Lepard: Learning partial point cloud matching in rigid and deformable scenes. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1, 2, 3

[10] Cheng Lin, Changjian Li, Yuan Liu, Nenglun Chen, Yi-King Choi, and Wenping Wang. Point2skeleton: Learning skeletal representations from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4277–4286, June 2021. 4, 6

[11] Wenbin Lin, Chengwei Zheng, Jun-Hai Yong, and Feng Xu. Occlusionfusion: Occlusion-aware motion estimation for real-time dynamic 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1736–1745, June 2022. 2, 3

[12] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, 34(6):1–16, 2015. 2

[13] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, 2015. 3

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 2

[15] 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

[16] Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6), Nov. 2017. 2

[17] Olga Sorkine and Marc Alexa. As-Rigid-As-Possible Surface Modeling. In Alexander Belyaev and Michael Garland, editors, *Geometry Processing*. The Eurographics Association, 2007. 2

[18] Robert W. Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Trans. Graph.*, 26(3):80–es, jul 2007. 1

[19] Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. Mean curvature skeletons. *Comput. Graph. Forum*, 2012. 4

[20] Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or. Curve skeleton extraction from incomplete point cloud. *ACM Trans. Graph.*, 28(3), jul 2009. 4, 6

[21] Zachary Teed and Jia Deng. Tangent space backpropagation for 3d transformation groups. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2

[22] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *Proceedings of the IEEE International Conference on Computer Vision*, 2019. 3

[23] Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. Rignet: Neural rigging for articulated characters. *ACM Trans. on Graphics*, 39, 2020. 4, 6

[24] Li Yang, Takehara Hikari, Taketomi Takafumi, Zheng Bo, and Matthias Nießner. 4dcomplete: Non-rigid motion estimation beyond the observable surface. *IEEE International Conference on Computer Vision (ICCV)*, 2021. 1, 6

[25] Quanshi Zhang, Xuan Song, Xiaowei Shao, Ryosuke Shibasaki, and Huijing Zhao. Unsupervised skeleton extraction and motion capture from 3d deformable matching. *Neurocomputing*, 100:170 – 182, 2013. Special issue: Behaviours in video. 6

[26] Silvia Zuffi, Angjoo Kanazawa, David Jacobs, and Michael J. Black. 3D menagerie: Modeling the 3D shape and pose of animals. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2