Self-Supervised Learning of Part Mobility from Point Cloud Sequence

Yahao Shi, Xinyu Cao and Bin Zhou†

State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science and Engineering, Beihang University, Beijing, China

Abstract

Part mobility analysis is a significant aspect required to achieve a functional understanding of 3D objects. It would be natural to obtain part mobility from the continuous part motion of 3D objects. In this study, we introduce a self-supervised method for segmenting motion parts and predicting their motion attributes from a point cloud sequence representing a dynamic object. To sufficiently utilize spatiotemporal information from the point cloud sequence, we generate trajectories by using correlations among successive frames of the sequence instead of directly processing the point clouds. We propose a novel neural network architecture called PointRNN to learn feature representations of trajectories along with their part rigid motions. We evaluate our method on various tasks including motion part segmentation, motion axis prediction and motion range estimation. The results demonstrate that our method outperforms previous techniques on both synthetic and real datasets. Moreover, our method has the ability to generalize to new and unseen objects. It is important to emphasize that it is not required to know any prior shape structure, prior shape category information, or shape orientation. To the best of our knowledge, this is the first study on deep learning to extract part mobility from point cloud sequence of a dynamic object.

CCS Concepts

• Computing methodologies → Shape analysis; Animation; Neural networks;

1. Introduction

In the real world, there often exist a number of dynamic and articulated objects, which can be directly operated using their moving parts. For example, we may need to open a refrigerator door to keep or remove an object. If we expect autonomous agents to correctly interact with such objects, they must have the ability to analyze which part of an object is a moving part. Hence, learning part mobility of 3D objects is beneficial for 3D computer vision [LWL*16] and robotics [HLRB13] and is closely related to the understanding of object affordances [MTFA15], functionality [HSvK18], and interaction using object recognition [LSZ*19] and human motion capture data [RLA*19].

Recently, with the emergence of large 3D labeled datasets and deep learning techniques, several studies have been conducted to make considerable progress in supervised semantic part segmentation, such as PointNet++ [QYSG17] and PointCNN [LBS*18]. However, these segments were often defined by personal subjective experience, which can easily lead to ambiguity. Thus, research on learning moveable part segmentation of 3D objects is of more practical significance, and it helps agents understand the essential features of dynamic objects. Moreover, current research on parsing 3D shape representations into semantic parts faces challenges in processing novel object categories and discovering new functional parts. It is not conducive for agents to explore strange new worlds. Ideally, intelligent agents should be able to parse 3D shapes into previously unseen functional parts from observations of continuous part motion. Mobility-based shape parsing brings a novel perspective to the part determination problem and provides an unambiguous decomposition. Additionally, a mobility-based part segmentation algorithm enables intelligent agents to make better use of man-made objects designed to function or interact with other objects (including humans).

In this study, we are interested in discovering the part mobility of 3D objects by observing their continuous part motion. We define part mobility as motion part segmentation, motion axis prediction and motion range estimation. In previous studies, motion parts have been typically extracted from one or two single static snapshots of dynamic objects. By contrast, we aim to deduce motion part structure from observations of continuous articulation states of an object. The reason is that speculating the motion parts and confirming motion vectors by observing only a few motion states can easily cause confusion. In this study, we adopt point cloud sequences to represent dynamic objects. Point cloud sequences reveal sufficient spatiotemporal information, and they are easier to obtain than large...
well-annotated datasets, owing to recent advances in the techniques of real-time 3D acquisition such as commercial RGB-D cameras.

Automatic motion part induction from point cloud sequences is challenging for several reasons. First, objects differ significantly because of their geometry and pose. Motion directions and motion ranges of motion parts have noticeable differences. Second, a point cloud is unordered. There is no tight point correspondence between adjacent frames of point cloud sequences. Third, we must consider the influence factors of acquisition quality of scan data, including noisy and missing data.

We propose a novel self-supervised deep neural network-based method to address the above mentioned problems, inspired by the traditional methods of Yan and Pollefeys [YP06], which is illustrated in Fig. 1. Points on the same part produce similar motion trajectories. We can calculate a rigid motion hypothesis based on these similar motion trajectories. Following these observations, we treat motion part segmentation as a trajectory clustering problem. Moreover, directly handling point cloud sequences can be difficult in a network without point correspondence. Our method transforms a point cloud sequence into a bunch of trajectories. We design a neural network, PointRNN, to process trajectories, and it can learn latent trajectory feature representations and generate candidate rigid motion hypotheses represented by motion axes and motion ranges. Finally, we merge the trajectories belonging to a similar rigid motion hypothesis by taking these trajectories as input. Finally, we merge these hypotheses and segment motion parts using an iterative algorithm. The results can propagate from the last frame to other frames.

Our method is verified through qualitative and quantitative evaluations on both synthetic and real datasets. Additionally, we conduct ablation experiments to confirm the influence of different loss designs, hyperparameter settings, etc. The results of the comparison experiments demonstrate that our method achieves better performance and requires less time than the traditional method [YLX*16] and deep learning methods [WZS*19], [YHL*18], [YHY*19]. Moreover, our algorithm has the ability to generalize to novel categories.

In summary, a new self-supervised deep learning method is proposed to parse 3D shapes into moving parts, motion axes, and motion ranges from point cloud sequences without labeling or any prior knowledge. Specifically, our method makes three key contributions. First, we introduce a self-supervised method for learning motion part segmentation, motion axis prediction and motion range estimation. Second, we propose a novel neural network called PointRNN which has the capability to process trajectories and extract a feature representation. Third, we demonstrate that the performance of our network is superior state-of-the-art methods, and it can generalize to novel object categories.

2. Related Work

Many 3D shape segmentation approaches have been proposed in previous studies to extract moving parts from an RGB-D sequence, a single point cloud, point cloud sequence, and mesh model. Given a specified RGB-D sequence that contains a dynamic object, attempts have been made in previous studies to recover the 3D scene flow, thus discovering moving parts of 3D objects. Jaimez et al. [JSGC15] proposed a primal–dual algorithm to compute RGB-D flow for estimating heterogeneous and non-rigid motion at a high frame rate. Vogel et al. [VRS14] introduced a method to recover dense 3D scene flow from multiple consecutive frames in a sliding temporal window. To reconstruct the articulated structure, motion part segmentation can be extracted by point correspondence, as shown by Fayad et al. [FRA11]. However, common defects in these approaches are their reliance on RGB color to compute scene flow and inability to handle complex structures or large motions.

In the case of a raw 3D point cloud sequence, many studies aimed to establish a point-wise correspondence between consecutive point clouds of an articulated shape [CZ08], [PB11]. Yan and Pollefeys [YP06] cast the problem of motion segmentation of feature trajectories as linear manifold finding problems and proposed a general framework for motion segmentation under affine projections. In addition, Kim et al. [KLAK16] considered that trajectories can be grouped by clustering to separate different motion parts. Yuan et al. [YLX*16] proposed a local-to-global approach to co-segment point cloud sequences of articulated objects into near-rigid moving parts. Most of aforementioned methods require a large amount of computation to achieve better performance. These approaches also require considerable effort in threshold setting from case to case.

Several approaches parse mesh models into moving parts. Mitra et al. [MYY*13] inferred the motion of individual parts and the interactions among parts based on their geometry and a few user-specified constraints. They utilized the results to illustrate the motion of mechanical assemblies. Hu et al. [HLVK*17] introduced a
data-driven approach for learning part mobility based on a defined motion pattern from a single static state of 3D objects. However, these methods are based on a well-defined motion pattern and well-segmented 3D objects.

Recent supervised learning approaches in 3D shape segmentation employ deep network architecture to train a classifier on labeled data represented by a point cloud [QYSG17], volumetric grids [MS15] or spatial data structures [KL17]. Subsequent attempts have been made to extract part rigid motions using a new deep learning technology. Wang et al. [WZS*19] proposed the shape2motion network architecture, which takes a single point cloud as input. Their network aims to simultaneously segment motion parts and predict motion axes based on a large, well-annotated dataset. However, their approach has difficulty in discovering new motion parts and generalizing to novel categories. Their network requires a considerable amount of time for supervised learning using a large dataset. Another type of method, such as that proposed by Yi et al. [YHL*18], infers part motion flow and estimates part correspondence by comparing two different motion states. Unfortunately, they did not perform a specific analysis of motion attributes such as motion axes and motion ranges. Behl et al. [BDG19] and Liu et al. [LQB19] promoted this class of algorithm to identify motion flow in scene data. However, these methods still rely on supervised learning. Yan et al. [YHY*19] introduced RPM-Net to infer movable parts of a single point cloud. They adopted a recurrent neural network (RNN) to segment movable parts by forecasting a temporal sequence motion of 3D objects. In contrast, our network takes point cloud sequences as input instead of a single point cloud. Furthermore, we use an RNN to process each trajectory locally rather than the entire point cloud directly and globally.

Deformable shape registration [SA07] is a basic problem in computational geometry and widely applied to various fields such as computer vision. There are various deformable registration algorithms for point clouds. Many methods are extensions of the classic ICP algorithm [BM92]. Papazov and Burschka [PB11] proposed an algorithm that computes shape transitions based on local similarity transforms, thereby allowing it to model not only as-rigid-as-possible deformations but also shapes on local and global scales.

3. Method

Our method comprises three steps for learning part mobility, as shown in Fig. 1. First, we adopt a deformable registration algorithm using point correspondence to generate trajectories (Sec. 3.1). Second, we propose a network architecture called PointRNN to produce candidate motion hypotheses extracted from trajectories (Sec. 3.2). Third, we design an iterative algorithm for removing redundant motion hypotheses and segmenting a point cloud into several motion parts while considering the match degree between trajectories and motion hypotheses (Sec. 3.3). The second step of our method occurs within the learning pipeline, and the other two steps occur outside it. In this section, we discuss the modules of our approach in detail.

3.1. Trajectory generation

The trajectories are input to our network, PointRNN (Fig. 2). To ensure a fair comparison with Yuan et al. [YLX*16], we use the same trajectory generation method. Thus, we adopt the method of Papazov and Burschka [PB11] based on local similarity transforms to generate motion trajectories, because it is an alternative module to generate trajectories for our method. All the trajectories are cropped to an equal length to facilitate deep neural network processing.

Given two adjacent frames, \(PC_s\) and \(PC_t\), of a point cloud sequence, the registration algorithm includes iterative steps of correspondence and deformation. In terms of the correspondence step, we find the closest point in \(q \in PC_t\) for each point \(p \in PC_s\). We optimize \(q\) by searching over its \(k\)-nearest neighbors to make the match error smooth. Formally, we compute a mapping, \(\pi_{s \rightarrow t}\), by minimizing the Laplacian smoothness energy of the residual field \(\{\pi_{s \rightarrow t}(p) - p : p \in PC_s\}\):

\[
E_{\text{smooth}}(\pi_{s \rightarrow t}) = \sum_{p \in PC_s} \|\delta(p)\|^2,
\]

where the residual Laplacian, \(\delta(p)\), is defined as

\[
\delta(p) = \frac{1}{N(p)} \sum_{q \in N(p)} [(\pi_{s \rightarrow t}(p) - p) - (\pi_{s \rightarrow t}(q') - q')].
\]

Here, \(N(p)\) is the \(k\)-nearest neighbors of \(p\). In terms of the deformation step, we minimize a fitting error between the local neighborhood of \(p \in PC_s\) and its matched counterpart to estimate a similarity transformation, denoted as \((R, t)\), by using the singular value decomposition [SMW06].

\[
E_{\text{fitting}}(R, t) = \sum_{q \in N(p)} \|\pi_{s \rightarrow t}(q') - (Rq + t)\|^2
\]

3.2. Part rigid motion hypothesis extraction

PointRNN network architecture. We propose a novel neural network called PointRNN which has the capacity to process trajectories and extract the feature representation, as illustrated in Fig. 2.

Given a set of trajectories, \(\{\tau_1, \tau_2, ..., \tau_n\} (n = 2048\) in our implementation), \(\tau_i\) has a point set, \(\{x_{i,1}, x_{i,2}, ..., x_{i,t}\} (t = 11\) by default), with \(x_{i,j} \in \mathbb{R}^3\) and a motion direction set, \(\{d_{i,1}, d_{i,2}, ..., d_{i,t}\}\) (\(t = 11\) by default), with \(d_{i,j} \in \mathbb{R}^3\). There is a one-to-one relationship between these two sets. For each trajectory, we employ a Long Short-Term Memory (LSTM) network [HS97] to encode time information, because LSTM has demonstrated high accuracy in tasks related to the processing of sequential and time-series data. Yan et al. [YHY*19] used the PointNet++ encoder to create a global feature for the input point cloud. Then, they fed this feature vector to the LSTM network. Compared to their approach, ours pays more attention to local and detailed features for each trajectory.

We assume that similar trajectories should belong to one motion

---

Y. Shi et al. / Self-Supervised Learning of Part Mobility from Point Cloud Sequence
part, as inspired by the traditional clustering method [YP06]. Thus, we design a Sample&Group module to learn the spatial features with their underlying motion. Specifically, we employ a LSTM network to encode time information for each trajectory at first. To encode spatial information, we first use iterative farthest point sampling to obtain a subset of trajectories, \( \{ \tau_1, \tau_2, \ldots, \tau_m \} \), such that \( \tau_i \) is the most distant trajectory (in metric distance) from set \( \{ \tau_1, \tau_2, \ldots, \tau_{i-1} \} \). The metric distance function between two trajectories, \( (\tau_i, \tau_j) \), is as follows:

\[
dis(\tau_i, \tau_j) = \frac{1}{t} \sum_{1 \leq m \leq t} \| x_{i,m} - x_{j,m} \|^2. \tag{4}
\]

We adopt Euclidean distance rather than Hausdorff distance or Ferche distance, because the latter two metrics require significantly more computations without yield improved performance. Next, we search a neighborhood trajectory set, \( \{ \tau_i, \tau_{i+1}, \ldots, \tau_{i+k} \} \), for each sampled trajectory, \( \tau_i \), using the \( k \)-nearest method, where the metric distance between \( \tau_i \) and \( \tau_m \) (\( 1 \leq m \leq k \)) does not exceed a certain threshold \( \varepsilon \). Then, we obtain a number of overlapping partitions of trajectories. Finally, we aggregate the features of the trajectory set as regional spatial features and express them as features of \( \tau_i \). In our method, \( k \) and \( \varepsilon \) are tunable parameters in different layers.

We design an Interpolate module to propagate global spatial features to single trajectory features with distance-based interpolation and skip-connection. We achieve feature propagation by interpolating features from the \( k \)-nearest trajectory set (\( k = 3 \)), \( \{ \tau_j, \tau_{j+1}, \ldots, \tau_{j+k} \} \), to each trajectory, \( \tau_j \). We use the inverse distance-weighted average based on the \( k \)-nearest neighbors in interpolation. Finally, we feed these interpolated trajectory features into a Multi-layer Perceptron (MLP) Network to encode single trajectory features.

By observing most objects in the real world, we found that the number of motion parts is finite and small. Therefore, we assume that the number of motion parts is no more than 10 based on our dataset. We design a Sample&Predict module to generate part rigid motion hypotheses. Similar to the Sample&Group module, we obtain \( m (m = 64) \) trajectory sets using farthest point sampling and \( k \)-nearest method (\( k = 32 \)). We consider that these sets can cover all the motion parts of 3D shapes. We use pooling to obtain aggregated spatial features for each trajectory set. Then, we obtain temporal feature using a LSTM network. Finally, we feed their temporal and spatial features to a MLP Network. The network outputs a motion axis including a start point (\( \mu \)) and direction (\( \omega \)), motion ranges including a shifted distance (\( \Phi \)), and a rotation angle (\( \Theta \)) (Fig. 3).

**Self-supervised loss function design.** To reconstruct the motion sequences, we compute the motion matrix, \( \mathbf{M} \in \mathbb{R}^{4 \times 4} \), by taking advantage of network output. The motion matrix is composed of a rotation matrix and a translation matrix (\( \mathbf{M} = \mathbf{R} + \mathbf{t} \)). \( \mathbf{t} \) is easy to generate by using \( \omega \) and \( \Phi \). The problem of solving \( \mathbf{R} \) can be reduced to transforming the point set rotating around the given axis (including \( \mu \) and \( \omega \)) into the point set rotating around the z-axis of the standard coordinate system, \( \mathbf{O}_{xyz} \). The basic idea of deriving this matrix is to divide the problem into few known simple steps. First, we align the start of the given axis with the coordinate origin and move the point set. Next, we rotate the given axis and the point set such that the axis lies in the yOz coordinate planes. Then, we rotate the given axis and the point set such that the axis is aligned with the z-axis. We use one of the fundamental rotation matrices to rotate the point sets using \( \Theta \). Finally, we undo steps three to one.
We can rebuild the last frame ($PC_t$) from the first frame ($PC_1$) by pre-multiplication ($\hat{M}$ on the left) in our neural network. To apply the backpropagation algorithm, we implement the partial derivative for each network output in our neural network. Thus, we define the first loss function, called the rebuild loss function, as follows.

$$L_{\text{rebuild}} = \|PC_t - PC_1 \cdot \hat{M}\|^2 \quad (5)$$

For each trajectory set in the Sample&Predict module, we solve an absolute orientation problem to obtain an approximate solution of rigid motion ($\hat{M}$). Our solution for the absolute orientation problem is based on the method of Myronenko and Song [MS09]. In addition, we guarantee that the determinant of $\hat{M}$ is 1 to ensure that $\hat{M}$ has a rotation matrix other than the reflection matrix. Then, we extract rotation matrix $R \in \mathbb{R}^{3 \times 3}$ from $\hat{M}$. Similarly, we can obtain $\hat{R}$ from $\hat{M}$. We define the relative pose estimation loss function, which was used to measure the angular distance between $\hat{R}$ and $R$ in the study by Suwajanakorn et al. [SSTN18].

$$L_{\text{pose}} = 2 \arcsin \left(\frac{1}{2\sqrt{2}} \|\hat{R} - R\|^2\right) \quad (6)$$

We found that the network has a slow convergence and easily falls into a local optimum solution using only $L_{\text{rebuild}}$ and $L_{\text{pose}}$. The reason for this is that it is possible to generate the same motion parts segment and optimization

$\hat{R} = (\cos \hat{\Theta})I + (\sin \hat{\Theta})[\hat{\omega}]_x + (1 - \cos \hat{\Theta})((\hat{\omega} \otimes \hat{\omega})) \quad (7)$

where $[\hat{\omega}]_x$ is the cross product matrix of $\hat{\omega}$, $\hat{\omega} \otimes \hat{\omega}$ is the outer product, and $I$ is the identity matrix. We can easily obtain $\hat{\Theta}$ and $\hat{\omega}$ from equation (7).

$$\hat{\Theta} = \arccos \left(\frac{\text{tr}(\hat{R}) - 1}{2}\right), \quad (8)$$

where $\text{tr}(\hat{R})$ is the trace of $\hat{R}$.

$$\hat{\omega} = \frac{1}{2 \sin \hat{\Theta}} \begin{pmatrix} \hat{R}[2,1] - \hat{R}[1,2] \\ \hat{R}[0,2] - \hat{R}[2,0] \\ \hat{R}[1,0] - \hat{R}[0,1] \end{pmatrix} \quad (9)$$

We extract $\hat{i} \in \mathbb{R}^3$ from $\hat{M}$, where $\hat{i}$ is a vector from the centroid of $PC_1$ to that of $PC_i$. $\hat{\Phi}$ is the length of $\hat{i}$ projected to $\hat{\omega}$.

$$\hat{\Phi} = \frac{\|\hat{i} \cdot \hat{\Phi}\|}{\|\hat{\omega}\|^2} \quad (10)$$

Finally, we solve linear algebraic equations to obtain $\hat{\mu}$, where $I$ is the identity Matrix.

$$\hat{\mu} = (\hat{R} - I)^{-1} \cdot (\hat{\Phi} \cdot \hat{\omega} - \hat{i}) \quad (11)$$

In the case of the general situation in which the start of an axis is not at the coordinate origin, we can obtain the same result. Then, we define the axis loss function as follows:

$$L_{\text{axis}} = \|\hat{\Theta} - \Theta\|^2 + \|\hat{\Phi} - \Phi\|^2 + \cosine(\hat{\omega}, \omega) + \frac{1}{3} \|\hat{R} - I\|_2^2 + \mu - (\hat{\Phi} \cdot \hat{\omega} - \hat{i})\|_2^2 \quad (12)$$

where $\cosine(\hat{\omega}, \omega)$ is the cosine distance between $\hat{\omega}$ and $\omega$. Given that $(\hat{R} - I)^{-1}$ is difficult to compute, network output $\mu$ should satisfy $(\hat{R} - I) \cdot \mu = (\hat{\Phi} \cdot \hat{\omega} - \hat{i})$ rather than directly regressing $\mu$ by $\hat{\mu}$. All formula deductions and practical calculation procedures are presented in the appendix.

The final loss function is the weighted sum of $L_{\text{rebuild}} (5)$, $L_{\text{pose}} (6)$, and $L_{\text{axis}} (12)$.

$$L_{\text{total}} = \alpha_1 \cdot L_{\text{rebuild}} + \alpha_2 \cdot L_{\text{pose}} + \alpha_3 \cdot L_{\text{axis}} \quad (13)$$

In our implementation, $\alpha_1 = \alpha_2 = \alpha_3 = 1$ by default. In addition, we set a threshold for each loss function to tolerate error and improve robustness, because approximate solutions of $\hat{\mu}$, $\hat{\omega}$, $\hat{\Phi}$, and $\hat{\Theta}$ usually have errors with respect to the ground truth, especially in real scan data.

### 3.3. Motion parts segment and optimization

We obtain $m$ part rigid motion hypotheses from PointRNN. These hypotheses are redundant, such that we must merge similar motion hypotheses. We define that there is no movement if $\Theta$ and $\Phi$ are no more than $\varepsilon_\Theta$ (0.1 radian by default) and $\varepsilon_\Phi$ (0.05 length by default), respectively, considering noisy data and computation errors. Further, we define rotation as $\Theta > \varepsilon_\Theta$ and $\Phi > \varepsilon_\Phi$ and translation by contrast. The combination of rotation and translation satisfies the conditions that $\Theta > \varepsilon_\Theta$ and $\Phi > \varepsilon_\Phi$. We merge the remaining motion axes by comparing the diversity of $\mu$, $\omega$, $\Phi$, and $\Theta$.

We define the metric distance between trajectory $t$ and motion hypotheses as the match degree. We use the motion hypotheses to rebuild $\hat{t}_i$, Then, the metric distance has two terms: rebuild loss and direction cosine distance.

$$\text{dis}_{\text{match}} = \frac{1}{t} \sum_{1 \leq m \leq t} \left(||\hat{x}_{i,m} - \hat{x}_{i,m}||^2 + \alpha \cdot \cosine(d_{i,m}, \hat{d}_{i,m})\right) \quad (14)$$

$\alpha = 0.2$ in our implementation. Then, we obtain refined motion axes and motion parts using an iteration algorithm. Inspired by Non-Maximum Suppression, we first set the refined motion axis set, $S$, as empty. We assign the trajectories to candidate axes. Then, we choose the candidate axis that has the most votes. If $S$ is empty, we add this candidate axis to $S$. If $S$ is not empty, we compute whether trajectories belonging to this candidate axis can also belong to an axis in $S$. If not, we add this candidate axis to $S$. We iterate these steps until all trajectories are assigned to a motion axis.
We optimize coarse segments by examining whether the label of a trajectory is the same as most of the labels of its $k$-nearest neighbors.

4. Experiments

In this section, we first introduce the benchmark datasets used in the experiments (Sec. 4.1). We then compare our approach with previous state-of-the-art methods on the tasks of motion part segmentation, motion attribute estimation, and 3D flow determination (Sec. 4.2). We also replace the PointRNN backbone with two different network designs to evaluate the effect of PointRNN (Sec. 4.3). Finally, we conduct ablation experiments of the loss function design, transfer ability, hyper parameters, etc. (Sec. 4.4). Further analyses and visualizations are provided in the appendix.

4.1. Dataset settings

We evaluate our methods on three different datasets, including two synthetic sets (f-data, s-data) and one real set (r-data). A fake dataset (f-data) is generated randomly and automatically. f-data enables the neural network to learn the common motion patterns of random trajectories. We leverage two annotated datasets including the Motion dataset [WZS*19] and PartNet [MZC*19] to construct the second synthetic dataset (s-data). Xiang et al. [XQM*20] enriched the PartNet dataset with motion attributes. We choose 31 categories including cabinet, lamp, and window to generate motion sequences. For the real data (r-data), we select certain reasonable motion sequences in the RBO dataset [MMEB18]. Moreover, Yi et al. [YHL*18] provided a small real scan dataset. Additionally, we scan some real data sequences in-house. More details on these datasets are provided in the appendix.

4.2. Comparison with state-of-the-art methods

We test our method against four alternatives, including both non-learning and learning approaches. Specifically, we compare our method with the traditional method of Yuan et al. [YLX*16] and three network-based approaches including Yi et al. [YHL*18], Wang et al. [WZS*19], and Yan et al. [YHY*19]. To the best of our knowledge, we are the first to propose an approach using deep learning to extract part mobility from a point cloud sequence. To ensure a fair comparison, we implement a space-time co-segmentation baseline following that of Yuan et al. [YLX*16]. Because their method also needs trajectories as input, we employ the same trajectory generation algorithm [PB11]. Yi et al. [YHL*18] proposed a neural network architecture with three modules that propose correspondences, estimate 3D deformation flows, and perform segmentation. Their method takes as input a pair of point clouds representing two different articulation states to segment motion parts and estimate 3D flows. To better provide a contrasting experiment between their approach and ours, we set $t = 2$ in our network settings. That is, we also take a pair of point clouds as input to train and test our network. In terms of the approach of Wang et al. [WZS*19], we train and evaluate their network and ours using the same training and testing data. However, it must be mentioned that their network requires a single point cloud as input. Considering this difference, the data for their network are randomly sampled frames from point cloud sequences. For the comparison between Yan et al. [YHY*19] and our method, the data for their approach are the first frame of the point cloud sequence, because their approach segments motion parts by predicting a temporal sequence from a single point cloud.

We use seven metrics to evaluate motion part segmentation and motion axis prediction. We use IoU to evaluate motion part segmentation. We measure the Minimum Distance (MD), Orientation Error (OE) and Type Accuracy (TA), as introduced in [WZS*19], which are used for measuring the distance, angle, and motion type accuracy, respectively, between the predicted motion axis line and the ground truth. Moreover, we employ Rand Index (RI) and End-Point-Error (EPE) to evaluate motion part segmentation and 3D flows in comparison with Yi et al. [YHL*18]. To compare the method of Yan et al. [YHY*19] with our method, we also use the mean Average Precision (mAP) as defined in their paper to measure the segmentation accuracy.

Table 1: Comparison between our method and those of Yuan et al. [YLX*16] and Wang et al. [WZS*19] in terms of IoU, MD, OE, and TA. A higher IoU value indicates more complete motion part segmentation. Lower values of MD and OE and higher TA values indicate more accurate axis prediction.

| Method                  | IoU | MD  | OE  | TA  |
|-------------------------|-----|-----|-----|-----|
| Yuan et al. [YLX*16]    | 0.71| -   | -   | -   |
| Wang et al. [WZS*19]    | 0.67| 0.051| 0.055| 0.92|
| Ours                    | 0.87| 0.032| 0.027| 0.99|

Yuan et al. [YLX*16] adopted a clustering method to obtain local segments and propagated these segments to the neighboring frames. Finally, they merged all frames using a space–time segment grouping technique to obtain the final motion part segmentation. However, their method tends to have degraded performance when dealing with tiny motion parts. Moreover, their approach has difficulty in processing 3D shapes with many diverse large motions. Table 1 shows that our approach outperforms theirs in terms of IoU. They did not exploit the fact that motion parts have local characteristics. They generated a motion hypothesis by randomly choosing a number of trajectory triplets. This resulted in the creation of several fragments that were difficult to group, especially in tiny parts. Additionally, the parameters of their approaches were different depending on the case.
Figure 5: Results of our approach compared with the ground truth on both synthetic and real datasets. Rows 1–5 show the results on the synthetic dataset and row 6 shows the results on the real dataset. We show three frames, first, middle, and last, of the point cloud sequence. All results are shown in the first frame.

Wang et al. [WZS*19] parsed 3D shapes into part mobility by observing a single static point cloud based on a large, well-labeled dataset. Their method can only predict motion parameters of 3D shapes that have been seen in the training data, because of the supervised learning based on a large, well-annotated dataset. It seems very easy to get confused in extracting motion parts from a single snapshot. For example, it would be very difficult to discover whether a door opens on the right or left when it is closed. More importantly, their network is likely to consider that is not a motion part when a door is closed. They used a similar matrix to segment motion parts. However, this type of method is also not conducive to dealing with tiny parts. In contrast to their method, which directly predicts an axis as a regression problem, we minimize the cosine distance for $\omega$ and solve linear algebraic equations to compute $\mu$, which makes our result more robust and stable than theirs (Table 1).

Yi et al. [YHL*18] aimed to discover motion parts of objects and estimated 3D flows by analyzing the underlying articulation states and geometry of shapes. Their network architecture alternates between correspondence, deformation flow, and segmentation prediction iteratively in an ICP-like fashion. Theirs is a supervised learning method, which estimates 3D point-wise flows at first and then segments the motion parts. In contrast to their method, we compute the motion axes and motion ranges, and then segment motion parts using these motion parameters. The results demonstrate that our method achieves higher IoU and RI, and lower EPE than theirs.

Table 2: Comparison between our method and that of Yi et al. [YHL*18] in terms of IoU, RI, and EPE. Higher IoU and RI values indicate more complete motion part segmentation. A lower EPE indicates more accurate flow prediction.

| Method         | IoU | RI  | EPE  |
|----------------|-----|-----|------|
| Yi et al. [YHL*18] | 0.71 | 0.80 | 0.029 |
| Ours           | 0.83 | 0.87 | 0.024 |

Yan et al. [YHY*19] predicted a temporal sequence of point-wise displacements from the input shape using LSTM. Then, the RPM-Net used these displacements to learn all the movable points. Next, they obtained a pointwise distance matrix by computing the Euclidean distance between these learned point features. Finally, they selected a clustering method to separate the points in the motion parts according to a distance matrix. They tended to study the part mobility from point cloud sequence using LSTM, but they still used a single point cloud. Table 3 shows that the results obtained from these models are close to each other. Although their method
deal with motion segmentation well, it has the same defect as that of Wang et al. [WZS 19], which is due to supervised learning. It is difficult to imagine a motion sequence from a single point cloud.

Table 3: Comparison our method and that of Yan et al. [YHY 19] in terms of IoU and mAP. Higher IoU and mAP values indicate more complete motion part segmentation.

| Method          | IoU  | mAP |
|-----------------|------|-----|
| Yan et al. [YHY 19] | 0.86 | 0.76 |
| Ours            | 0.87 | 0.77 |

4.3. Analysis of the effects of different network designs.

Compared to the contrast experiments, there are three additional novel metrics, Θ, Φ, and re, presented in Sec. 4.3 and Sec. 4.4. Our approach also estimates motion ranges such that Θ and Φ are errors between the network outputs and ground truth. re is the error of rebuilding the last frame from the first frame, which is used for a comprehensive evaluation. Moreover, we employ the model size to compare the size of different models and timing, which includes training time and testing time, to evaluate the efficiency of the algorithm.

To verify the effectiveness of PointRNN on the tasks of motion part segmentation and motion attribute estimation, we design two baselines to replace the network backbone. We adopt a slightly modified version of that of Liu et al. [LYB19] to build baseline1. We combine t point clouds to obtain a single point cloud. Next, we feed this single point cloud to PointNet++ to extract features. Finally, we minimize the loss function Ltotal to estimate motion axis and motion range. Simultaneously, we design baseline2 by referring to Yan et al. [YHY 19]. We extract t global features per frame using a shared PointNet++. Then, we adopt LSTM to encode the time information. Similarly, we use Ltotal as a loss function. For more details about these baselines, please refer to the appendix. The results show that we can achieve better performance by using PointRNN, because our network can extract local and detailed spatiotemporal features.

Table 4: Comparison of our network with two other network designs in terms of IoU, MD, OE, Θ, Φ, re, and TA. Higher IoU indicates more complete motion part segmentation. Lower MD, OE, Θ, Φ, and higher TA indicate more accurate axis prediction. re is a comprehensive evaluation of motion part segmentation and axis prediction, where a lower re value is better.

| Loss Function | MD     | OE     | Θ     | Φ     | re    | TA    |
|---------------|--------|--------|-------|-------|-------|-------|
| Ltotal        | 0.027  | 0.014  | 0.047 | 0.023 | 0.020 | 0.97  |
| Lrebuild + Lpose | 0.032  | 0.016  | 0.053 | 0.022 | 0.045 | 0.94  |
| Lpose + Laxis  | 0.031  | 0.017  | 0.051 | 0.036 | 0.086 | 0.90  |
| Laxis          | 0.035  | 0.017  | 0.058 | 0.026 | 0.061 | 0.96  |

4.4. Ablation experiments

Effect of different loss function designs. To analyze the consequences of different loss function designs, we experiment with an ablated version of our network with four combinations of all loss functions. Different loss designs mainly influence the network performance with respect to axis generation. Thus, we train and test our network on f-data to focus on examining the quality of the predicted motion axes. Table 5 lists the results of including Laxis, Lrebuild + Lpose, Lpose + Laxis, and Ltotal. The results demonstrate that our method achieves optimal performance for motion axis estimation, thereby verifying its optimal effect when using Lrebuild, Lpose, and Laxis together. The reason for this is that it is possible to generate the same M using two different parameter sets of μ, ω, Φ and Θ. There is an interactive constraint between two loss functions.

Table 5: Effectiveness of different loss function designs. IoU of f-data is not needed. Lower MD, OE, Θ, Φ, re, and higher TA indicate more accurate axis prediction.

Figure 6: Our results are compared with those of Yuan et al. [YLX 16] and Wang et al. [WZS 19].
Comparison between our method and that of Myronenko and Song [MS09]. Our method is self-supervised, because our data are not manually annotated, either in training or testing. Moreover, the supervised information is automatically generated from the characteristic distribution of the data. We solve an absolute orientation problem to generate a motion matrix \( \hat{M} \) using the method of Myronenko and Song [MS09]. Then, we parse the supervised information \((\mu, \omega, \Phi, \Theta)\) from \( \hat{M} \). We adopt the neural network approach rather than the traditional method of Myronenko and Song [MS09] because the traditional method is easily influenced by noise, which would interfere with and alter the degree of accuracy of the motion matrix. The neural network approach has better stability and robustness in complex situations (e.g., an object has many motion parts and the data are noisy). In the experiment of loss function design, it is reflected from the side that the network output is more robust than only using supervised information parsed from Myronenko and Song [MS09], because the result of \( L_{\text{total}} \) is better than that of \( L_{\text{pose}} + L_{\text{axis}} \). In addition, we designed an experiment to compare our method with that of Myronenko and Song [MS09], and the results presented in Table 6 show that directly using supervised information as an output is better than \( s\text{-data} \) than ours, but worse in \( f\text{-data} \), because \( s\text{-data} \) is more complex and noisy than \( f\text{-data} \).

Table 6: Comparison between our method and that of Myronenko and Song [MS09] in terms of the performance of EPE, where a lower EPE value is better.

| Method                        | Dataset | \( f\text{-data} \) | \( s\text{-data} \) |
|-------------------------------|---------|---------------------|---------------------|
| Myronenko and Song [MS09]     |         | 0.018               | 0.027               |
| Ours                          |         | 0.021               | 0.023               |

Performance on three datasets. We train and test our network using three different datasets. The performance of these datasets is reported in Table 7. These three datasets have unique features. The difficulty of \( f\text{-data} \) stems from its randomness. Each element of \( f\text{-data} \) has totally different motion parameters, including motion axes and motion ranges. \( s\text{-data} \) has various categories. Most 3D shapes in \( s\text{-data} \) have two or more motion parts. \( r\text{-data} \) generally has a partial single view of 3D shapes with more noise. The results demonstrate that not only can our approach learn motion parameters from synthetic data, but it can also be applied to noisy real data.

Table 7: Performance of our method on three datasets in terms of IoU, MD, OE, \( \Theta_e \), \( \Phi_e \), \( r_e \), and TA. A higher IoU value indicates more complete motion part segmentation. Lower MD, OE, \( \Theta_e \), \( \Phi_e \), and higher TA indicate more accurate axis prediction. \( r_e \) is a comprehensive evaluation of motion part segmentation and axis prediction, where a lower value is better.

| Dataset | IoU | MD   | OE   | \( \Theta_e \) | \( \Phi_e \) | \( r_e \) | TA   |
|---------|-----|------|------|---------------|-------------|--------|------|
| \( f\text{-data} \) | -   | 0.027| 0.014| 0.047         | 0.023       | 0.020  | 0.97 |
| \( s\text{-data} \) | 0.87| 0.032| 0.027| 0.046         | 0.027       | 0.023  | 0.99 |
| \( r\text{-data} \) | 0.79| 0.058| 0.036| 0.079         | 0.037       | 0.058  | 0.99 |

Analysis on the length of point cloud sequence. Considering that our motivation is to learn part mobility from a point cloud sequence, the number of frames \( t \) is an important hyper parameter in our approach. Thus, we must determine how many frames are sampled from one motion sequence. To achieve this, we conduct an ablation experiment on \( t \) to discuss the impact of the length of the point cloud sequence. We reduce the number of frames from 17 to 3 for a sufficient verification. The results are depicted in Fig. 7. As the number of frame increases, the performance gain is consistent but more and more slow. Hence, we adopt 11 frames as input to balance the performance and efficiency.

Analysis on transfer ability. Here, we aim to verify that our network can estimate the motion axes by learning the feature representation of trajectories other than only by learning geometrical characteristics. Our approach is a self-supervised deep learning algorithm. It can abstract out motion patterns from trajectories, which can enhance the ability of generalization. Therefore, we design an experiment in which we train our network on \( f\text{-data} \), and then test it on \( s\text{-data} \). We present three categories with different levels of complexity results in Table 8. The results demonstrate that the network can produce correct motion axes, even though it did not see any real objects. This implies that our network can learn motion patterns from trajectories and our algorithm indeed has the capability of generalization.

Table 8: Experiment on the generalization ability. Our method can handle unseen categories, because motion patterns and type are learned by the network.

| category | IoU   | MD   | OE   | \( \Theta_e \) | \( r_e \) | TA   |
|----------|-------|------|------|---------------|--------|------|
| Fan      | 0.89  | 0.014| 0.014| 0.032         | 0.028  | 1    |
| Laptop   | 0.95  | 0.018| 0.013| 0.035         | 0.021  | 1    |
| Scissor   | 0.79  | 0.022| 0.017| 0.087         | 0.031  | 1    |

Effect of trajectory aggregation: The aggregation of trajectories is of great importance in PointRNN, as it describes a motion...
part consisting of a group of similar motion trajectories. Thus, it is significant to analyze how different aggregation designs influence performance. We report five different sizes with two types of aggregations: max and average, as shown in Fig. 8. As the aggregation radius \((k)\) increases, PointRNN improves until it peaks at approximately 32. The results demonstrate that it is difficult to learn the features when we cluster too few trajectories in a local region. In contrast, a larger region usually contains more than one motion part and results in decreased performance. Moreover, max pooling achieves better results than average by aggregating the features in local regions.

**Analysis on the quality of trajectory.** We also discuss the impact of different qualities of trajectory. Given a point cloud sequence, we first generate a set of trajectories. Then, we add different levels of random noise to these trajectories. Fig. 9 shows the relationship between \(IoU\) and noise. Our method maintained more than 0.7 \((IoU)\) after adding 0.04 random noise. As the noise degree increases, the performance declines sharply. The results demonstrate that our method can tolerate noise to a certain extent, but it will fail when under the addition of excessive noise.

Table 9: Model size and processing time. Our model is smaller and faster.

| Method           | Model size | Timing     |
|------------------|------------|------------|
| Yuan et al. [YLX*16] | -          | -180s      |
| Wang et al. [WZS*19] | 86.7MB     | 38h/7.32s  |
| Yi et al. [YHL*18]   | 78MB       | 35.5h/4.88s|
| Yan et al. [YHY*19]  | 153.3 MB   | 13.2h/0.55s|
| Ours              | 25.2MB     | 11.6h/0.43s|

**Model size and speed.** Our proposed model is highly efficient, as it leverages sparsity in point clouds by using clusters and does not require many stages like those of Wang et al. [WZS*19] and Yi et al. [YHL*18]. Compared to previous methods (Table 9), our model is more than 3\(\times\) smaller in size and more than 10\(\times\) times faster than those of Wang et al. [WZS*19] and Yi et al. [YHL*18]. While the method in Yuan et al. [YLX*16] is a non-learning method, it costs approximately 3 min to segment a 3D shape without giving motion parameters. Furthermore, our model is more than 6\(\times\) smaller in size than that of Yan et al. [YHY*19].

### 4.5. Limitations and future work

There are several avenues for future research. First, our method is based on trajectories such that if the quality of trajectories is extremely poor with many fragments, the performance decreases. Second, if two parts have exactly the same motion, e.g., two drawers in a cabinet have the same motion, our method considers that they have the same motion and should be one motion part. Third, for hierarchical mobility extraction, we can segment the 3D object into different motion parts, but the motion axis describes a compound movement. For example, the motion of a bulb holder is a composite movement with that of a lamp post. In the future, it would be interesting to infer parts and motions and discover common articulation patterns from various of sensor data. Moreover, our method relies on a pre-registration of point clouds using an ICP-like optimization. There are also many deep learning methods for point cloud registration, such as [PMR*20]. It is deserved to design a network encompassing both trajectory generation and motion attribute estimation in an end-to-end fashion in future work. To facilitate future research and reproduce our method more easily, the source code and dataset are available on Github: [https://github.com/AGithubRepository/PartMobility](https://github.com/AGithubRepository/PartMobility).

### 5. Conclusion

In this paper, we introduce a self-supervised method for parsing 3D shapes into several motion parts with motion parameters from point cloud sequences. We first transform point cloud sequences into trajectories. Then, we propose a novel neural network architecture called PointRNN to extract the feature representations from trajectories and produce motion hypotheses. Finally, we propose an...
iterative algorithm for segmenting motion parts. In contrast to previous studies, our approach first predicts part motion parameters including motion axes and motion ranges, and then segments motion parts guided by these motion hypotheses. We experimentally demonstrated that our approach yields significantly better results compared with both the traditional and network-based methods.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. This work was supported in part by National Key Research and Development Program of China (2018YFC0831003 and 2019YFF0302902), National Natural Science Foundation of China (61902014 and U1736217 and 61932003), and Pre-research Project of the Manned Space Flight (060601).

References

[Bell] BELONGIE S.: Rodrigues’ rotation formula. From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein. https://mathworld.wolfram.com/RodriguesRotationFormula.html.

[BM92] BESL P. J., MCKAY N. D.: A method for registration of 3-d shapes. IEEE Trans. Pattern Anal. Mach. Intell. 14, 2 (Feb. 1992), 239–256. 3

[BPDG19] BEHIL A., PASCHALIDOU D., DONNE S., GEIGER A.: Pointflownet: Learning representations for rigid motion estimation from point clouds. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019). 3

[CLZ98] CHANG W., ZWICKER M.: Automatic registration for articulated shapes. Computer Graphics Forum 27, 5 (2008), 1459–1468. 2

[FRA11] FAYAD J., RUSSELL C., AGAPITO L.: Automated articulated structure and 3d shape recovery from point correspondences. In 2011 International Conference on Computer Vision (Nov 2011), pp. 431–438. 2

[HLRB13] HERMANS T., LI F., REHG J. M., BOBICK A. F.: Learning contact locations for pushing and orienting unknown objects. In 2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids) (Oct 2013), pp. 435–442. 1

[HLVK*17] HU R., LI W., VAN KICX O., SHAMIR A., ZHANG H., HUANG H.: Learning to predict part mobility from a single static snapshot. ACM Trans. Graph. 36, 6 (Nov. 2017), 227:1–227:13. 2

[HS97] HOCHEPITRE S., SCHMIDHUBER J.: Long short-term memory. Neural Computation 9, 8 (1997), 1735–1780. 3

[HSK18] HU R., SAVA M., VAN KICX O.: Functionality representations and applications for shape analysis. Computer Graphics Forum 37, 2 (2018), 603–624. 1

[JSJC15] JAIMEZ M., SOUAI M., GONZALEZ-JIMENEZ J., CREAMERS D.: A primal-dual framework for real-time dense rgb-d scene flow. In 2015 IEEE International Conference on Robotics and Automation (ICRA) (May 2015), pp. 98–104. 2

[KL17] KLOKOV R., LEMPTSKY V.: Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In The IEEE International Conference on Computer Vision (ICCV) (Oct 2017). 3

[KLAK16] KIM Y., LIM H., AHN S. C., KIM A.: Simultaneous segmentation, estimation and analysis of articulated motion from dense point cloud sequence. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (Oct 2016), pp. 1085–1092. 2

[LBS*18] LI Y., BU R., SUN M., WU W., DI X., CHEN B.: Pointconv: Convolution on x-transformed points. In Advances in Neural Information Processing Systems 31, Curran Associates, Inc., 2018, pp. 820–830. 1

[LQG19] LIU X., QI C. R., GUIBAS L. J.: Flownet3d: Learning scene flow in 3d point clouds. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019). 3

[LSZ*19] LIU M., SHI Y., ZHENG L., XU K., HUANG H., MANOCHA D.: Recurrent 3d attentional networks for end-to-end active object recognition. Computational Visual Media 5, 1 (2019), 92–104. 1

[LWL*16] LI H., WANG G., LI H., SHARF A., XU K., CHEN B.: Mobility fitting using 4d ransac. Computer Graphics Forum 35, 5 (2016), 79–88. 1

[LYB19] LIU X., YAN M., BOHG J.: MeteorNet: Deep learning on dynamic 3d point cloud sequences. In The IEEE International Conference on Computer Vision (ICCV) (October 2019). 8

[MMEB18] MARTIN-MARTIN R., EPPNER C., BROCK O.: The rbo dataset of articulated objects and interactions, 2018. 6

[MS9] MYRONENKO A., SONG X. B.: On the closed-form solution of the rotation matrix arising in computer vision problems. CoRR abs/0904.1613 (2009). 5

[MS15] MATURANA D., SCHERER S.: 3d convolutional neural networks for landing zone detection from lidar. In 2015 IEEE International Conference on Robotics and Automation (ICRA) (May 2015), pp. 1374–1381. 1

[MTFA15] MYERS A., TEO C. L., FERMILLER C., ALOIMONOS Y.: Affordance detection of tool parts from geometric features. In 2015 IEEE International Conference on Robotics and Automation (ICRA) (May 2015), pp. 1374–1381. 1

[MYY*13] MITRA N. J., YANG Y.-L., YAN D.-M., LI W., AGRAWALA M.: Illustrating how mechanical assemblies work. Commun. ACM 56, 1 (Jan. 2013), 106–114. 2

[MZC*19] MO K., ZHU S., CHANG A. X., YI L., TRIPATHI S., GUIBAS L. J., SU H.: PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019). 6

[PB11] PAPAZOV C., BURSCHKA D.: Deformable 3d shape registration based on local similarity transforms. Computer Graphics Forum 30, 5 (2011), 1493–1502. 2, 3, 6

[PMR*20] PAIS G., MIRALDO P., RAMALINGAM S., GOVIND V., NASCIMENTO J., CHULLAPPA R.: 3dregnet: A deep neural network for 3d point registration. 2020 IEEE/ICVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020), 7191–7201. 10

[QYSG17] QI C. R., YI L., SU H., GUIBAS L. J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 5099–5108. 1, 3

[RLA*19] ROBERTS R., LEWIS J. P., ANIYO K., SEO J., SEOL Y.: Optimal and interactive keyframe selection for motion capture. Computational Visual Media 5, 2 (2019), 172–191. 1

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling, 2007. 3

[SMW06] SCHAFFER S., MCPHAIL T., WARREN J.: Image deformation using moving least squares. ACM Trans. Graph. 25 (07 2006), 533–540. 3

[SSTN18] SUWAJANAKORN S., SNAVELY N., TOMPSON J. J., NOROUIZI M.: Discovery of latent 3d keypoints via end-to-end geometric reasoning. In Advances in Neural Information Processing Systems 31. Curran Associates, Inc., 2018, pp. 2059–2070. 5

[VRS14] VOGEL C., ROTH S., SCHINDLER K.: View-consistent 3d scene flow estimation over multiple frames. In Computer Vision – ECCV 2014 (Cham, 2014), Springer International Publishing, pp. 263–278. 2

[WZS*19] WANG X., ZHOU B., SHI Y., CHEN X., ZHAO Q., XU K.: Shape2motion: Joint analysis of motion parts and attributes from 3d shapes. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019). 2, 3, 6, 7, 8, 10
[XQM*20] Xiang F., Qin Y., Mo K., Xia Y., Zhu H., Liu F., Liu M., Jiang H., Yuan Y., Wang H., Yi L., Chang A. X., Guibas L. J., Su H.: SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020).

[YHL*18] Yi L., Huang H., Liu D., Kalogerakis E., Su H., Guibas L.: Deep part induction from articulated object pairs. *ACM Trans. Graph.* 37, 6 (Dec. 2018), 209:1–209:15.

[YHY*19] Yan Z., Hu R., Yan X., Chen L., van Kaick O., Zhang H., Huang H.: Rpm-net: Recurrent prediction of motion and parts from point cloud. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA 2019)* 38, 6 (2019), 240:1–240:15.

[YLX*16] Yuan Q., Li G., Xu K., Chen X., Huang H.: Space-time co-segmentation of articulated point cloud sequences. *Computer Graphics Forum* 35, 2 (2016), 419–429.

[YP06] Yan J., Pollefeys M.: A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *Computer Vision – ECCV 2006* (Berlin, Heidelberg, 2006), Springer Berlin Heidelberg, pp. 94–106.

submitted to *COMPUTER GRAPHICS Forum* (10/2020).