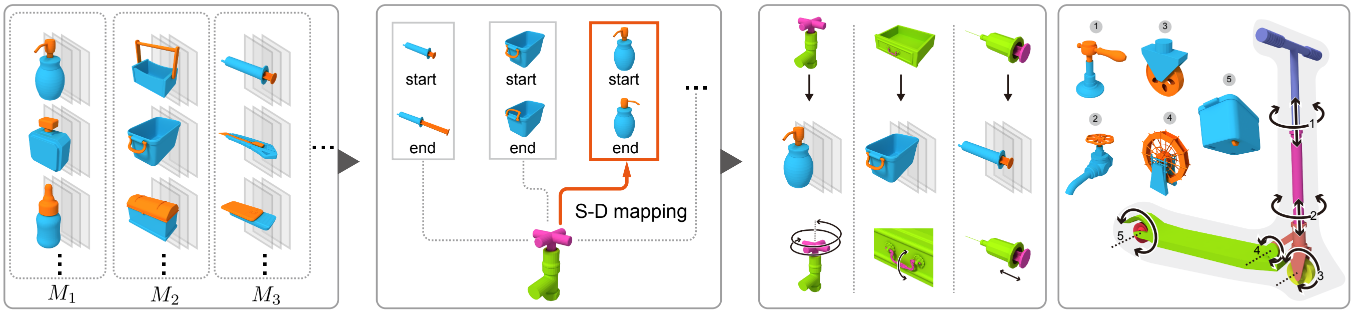
Learning to Predict Part Mobility from a Single Static Snapshot

RUIZHEN HU, Shenzhen University
WENCHAO LI, Shenzhen University
OLIVER VAN KAICK, Carleton University
ARIEL SHAMIR, The Interdisciplinary Center
HAO ZHANG, Simon Fraser University
HUI HUANG, Shenzhen University



(a) Training data (b) Part mobility model (c) Motion prediction and transfer

Fig. 1. We introduce a data-driven approach for learning a *part mobility* model, which enables to predict the motion of parts in a 3D object based only on a *single static* snapshot of the object. The learning is based on a training set of *mobility units* of different motion types, M_1, M_2, . . ., as in (a). Each unit is represented by multiple snapshots over its motion sequence, along with associated motion parameters. The part mobility model (b) is composed of the start and end snapshots of each unit and a *static(snapshot)-to-dynamic(unit) (S-D) mapping* function learned from training data. Given a query 3D shape, shown at the bottom of (b), we find the closest mobility unit from the training set via the S-D mapping (b). Aside from motion prediction, the unit also provides a means to transfer its motion to the query shape, as shown in (c)-left. In (c)-right, we show mobility prediction and transfer on five different parts of a static scooter model, along with the units found via S-D mapping.

We introduce a method for learning a model for the *mobility* of parts in 3D objects. Our method allows not only to understand the dynamic functionalities of one or more parts in a 3D object, but also to apply the mobility functions to *static* 3D models. Specifically, the learned part mobility model can *predict* mobilities for parts of a 3D object given in the form of a *single static* snapshot reflecting the spatial configuration of the object parts in 3D space, and *transfer* the mobility from relevant units in the training data. The training data consists of a set of *mobility units* of different motion types. Each unit is composed of a pair of 3D object parts (one moving and one reference part), along with usage examples consisting of a few snapshots capturing different motion states of the unit. Taking advantage of a linearity characteristic exhibited by most part motions in everyday objects, and utilizing a set of part-relation descriptors, we define a mapping from static snapshots to dynamic units. This mapping employs a *motion-dependent* snapshot-to-unit distance obtained via metric learning. We show that our learning scheme leads to accurate motion prediction from single static snapshots and allows proper motion transfer. We also demonstrate other applications such as motion-driven object detection and motion hierarchy construction.

CCS Concepts: • **Computing methodologies** → **Shape analysis**;

Additional Key Words and Phrases: Functionality analysis, learning part mobility, motion prediction and transfer

ACM Reference format:
Ruizhen Hu, Wenchao Li, Oliver van Kaick, Ariel Shamir, Hao Zhang, and Hui Huang. 2017. Learning to Predict Part Mobility from a Single Static Snapshot. *ACM Trans. Graph.* 36, 6, Article 227 (November 2017), 13 pages.
https://doi.org/10.1145/3130800.3130811

1 INTRODUCTION

Recently in the field of shape analysis, increasing efforts have been devoted to obtaining a *functional* understanding of 3D objects from their geometries and interactions [Hu et al. 2016, 2015; Kim et al. 2014; Pirk et al. 2017; Zhao et al. 2014]. In this setting, the functionality of an object is learned by analyzing how humans or virtual agents may interact with the object and how close-by objects are related to it geometrically. Typically, such knowledge is acquired from static snapshots of the object and its surroundings, e.g., a chair with a human sitting on it, or a table with several objects on top. In a first attempt, Pirk et al. [2017] describe object functionalities by capturing and analyzing dynamic object trajectories, e.g., the motion of a moving agent attempting to sit on a chair. Yet, in all of these previous works, the central object maintains its rigidity.

[*] Corresponding author: Hui Huang (hhzhiyan@gmail.com).

0730-0301/2017/11-ART227 $15.00
https://doi.org/10.1145/3130800.3130811

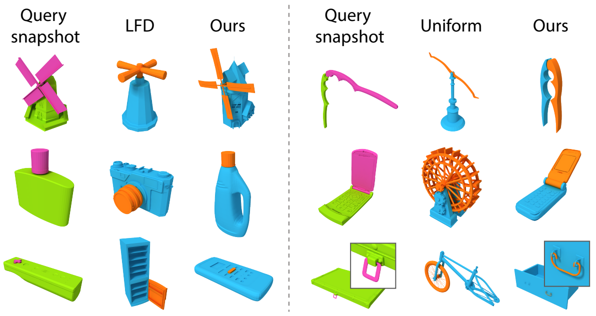


Fig. 2. Comparing mobility unit prediction by using our part mobility model obtained with metric learning ("Ours") to geometry-based retrieval using the LFD descriptor (left) and to our part mobility model but with uniform weights in the snapshot-to-unit distance (right).

In this paper, we are interested in *dynamic* functionalities of 3D objects characterized by the movements of one or more object *parts*, which we term *part mobility*. The presence of part mobilities is ubiquitous in our daily lives, e.g., the opening/closing of the drawers in a chest, or the rotation of the cap of a bottle or the seat of a swivel chair. One could analyze part mobilities from time-varying representations of part motions in 3D. However, the acquisition, as well as distillation and encoding, of dynamic interaction data for functionality analysis is costly and challenging [Pirk et al. 2017]. The intriguing question is whether it is possible to predict *dynamic* part mobilities from only a *single static snapshot* of the spatial configuration of object parts in 3D space. Perhaps even more intriguing would be the prospect of *executing* proper part motions on a given static 3D model. Under these settings, part mobility analysis would involve not only recognizing the type of motion that an object part can undergo, but also inferring appropriate motion parameters to enable the execution of motions.

Without any prior knowledge, the part mobility analysis problem may very well be ill-posed. However, in our daily lives, we, humans, apply motion inferences all the time. In general, we can predict the functionality of unseen objects by prior experiences or knowledge on similar objects. In the same spirit, our work shows that it is possible to infer part mobilities of a static 3D object by analyzing and learning from motion sequences of parts from the same and different classes of objects.

This problem is challenging not only because a mapping from static to dynamic is inevitably under-constrained, but also due to the large geometric diversity of 3D shapes which may support a similar motion. As shown in Figure 1(c), while the handles of the faucet and drawer both involve rotations, the rotations are of different *motion types*. Geometrically, doors, wheels, and handles could all come in different shapes and sizes. Even the same motion type can be parameterized differently based on different geometries and part configurations. Simply searching for geometric similarity between individual snapshots of the dynamic movements means that we need to store examples of all possible motion types, and in all spatial and temporal configurations for each type, which is impractical.

We introduce the idea of learning a specific *metric* for each motion type, which enables a mapping from a given static query snapshot to the closest dynamic *mobility unit* in the training set. The training data consists of a set of mobility units classified by motion types such as wheel rotation, drawer sliding, etc., where each mobility unit is composed of a pair of parts of a 3D object (a moving part, and a part that serves as reference to the motion), along with usage examples consisting of few, typically 2-4, snapshots capturing the geometry configuration in different motion states of the unit. We develop a data-driven approach to learn the static-to-dynamic or *S-D mapping* and our *part mobility model*; see Figure 1. With the learned *snapshot-to-unit* distance, the mapped unit from the query static snapshot not only exemplifies the motion type, e.g., for *motion prediction*, but also comes with the necessary motion parameters to allow *motion transfer* from the dynamic unit to a static 3D object.

We observe that many part mobilities afforded by everyday objects exhibit some form of *linearity*, but in different spaces. For example, drawers undergo linear translations in the spatial domain, while rotations about a hinge are linear in angle space. Linearity makes it possible to characterize and learn part mobilities from few static snapshots. In the "correct" space which reflects the linearity of a motion type, the *sum of distances* from any intermediate snapshot of a linear motion, to the start and end snapshots of the motion, by means of linearity, would remain approximately *invariant*. Thus we can characterize a motion type by the sum of distances to the appropriate start and end states and rely on these distance sums for motion prediction. The remaining challenge however is that we do not know the correct space within which to measure distances.

With the motion types exhibiting much diversity in their temporal and geometric characteristics, we resort to machine learning, and specifically metric learning, to approximate the correct spaces for measuring distances. The S-D mapping requires measuring distances from a query snapshot of a 3D object to mobility units of other objects. We define the snapshot-to-unit distance as the sum of distances from the query snapshot to the start and end snapshots of target units. Under the linearity assumption, this sum should be invariant for snapshots within the unit. Moreover, this sum for snapshots with the same type of motion should be smaller than the sum for snapshots with other types of motion. Thus, the problem of defining a snapshot-to-unit distance is reduced to defining a meaningful distance between snapshots based on our chosen set of descriptors. However, if we simply apply uniform weighting of the descriptors for all motion types, we would obtain unsatisfactory prediction results, as shown in Figure 2 (right). On the other hand, using different descriptor weights learned from data, we can define a distance that approximates the correct motion space. Indeed, our metric learning allows us to define a *motion-dependent* distance for each motion type.

We show that our learning scheme is able to predict dynamic part mobilities from single static snapshots with high accuracy and the predicted dynamic units provide information on the motion parameters to allow effective motion transfer; see Figure 1(c). Our prediction model also enables other applications, such as motion-driven object detection in 3D scenes, as well as understanding the mobility of an entire object, forming a motion hierarchy of the object parts. Such a motion hierarchy can be used either to generate animations of objects, or to create static scenes with objects in different motion states.

2 RELATED WORK

Object functionality from static snapshots or agents. In comparison to previous works on functionality analysis, the interactions we consider are dynamic (the mobility of parts) and are not limited to those involving human agents. For example, affordance-based methods simulate a human agent to predict the functionality of objects [Grabner et al. 2011; Kim et al. 2014], or to recognize the regions of a scene to enable a human to perform certain actions [Savva et al. 2014, 2016]. Thus, although some of these methods involve the dynamics of human interactions, they do not extend to more general types of object affordances. The interaction context descriptor [Hu et al. 2015] and functionality models learned for categories [Hu et al. 2016] consider more general object-object interactions. However, these object-object interactions are static in nature. In contrast, we analyze the part-part interactions for part mobility prediction.

Object functionality from dynamic interactions. The recent work of Pirk et al. [2017] performs functionality inference from dynamic interaction data. The key difference to our work is that they characterize functionalities of *static* objects by analyzing dynamic interactions, e.g., how a cup can be used in the dynamic action of drinking coffee. Similarly, Hermans et al. [2013] introduce an approach to learn the dynamic interactions that a robot can perform with rigid objects, in the form of pushing an object to displace or rotate the object. However, the analyzed objects in these approaches are not dynamic themselves. As a consequence, their analyses are performed at the object level, and not at the part level as in our work. Moreover, dynamic interaction data is not only difficult to acquire and process, but may also be unavailable altogether.

Another line of works in the literature target the capture of dynamic interactions. Kry and Dinesh [2006] propose a method to acquire the details of hand interactions. Their work focuses on the use of specialized hardware for acquiring the interactions, and does not leverage the motion information to represent the functionality of objects. Recent works in computer vision aim at capturing the functionality of tools [Zhu et al. 2015] or representing general human interactions [Wei et al. 2017]. However, the focus of these works has been on recognition — the derived functionality representations are not intended for grouping or transferring part mobility.

Motion inference from geometry. In earlier work along this direction, Gelfand and Guibas [2004] infer slippage motions from the geometry of shapes. Although the method is applied to shape segmentation rather than motion analysis, the slippage analysis discovers sliding motions that kinematic objects can undertake. In subsequent work, Xu et al. [2009] employ slippage analysis to discover joints in articulated models. Moreover, Mitra et al. [2010] analyze mechanisms to infer their motion. They predict the possible motion of mechanical parts from their geometry and spatial configuration, and use the result of the analysis to illustrate the motion of mechanical assemblies. Guo et al. [2013] follow a similar approach to illustrate the disassembly of 3D models, while Shao et al. [2013] create animated diagrams from a few concept sketches. Although these methods can be used to automatically discover the mobility of parts, we remark that most of the 3D shapes available in online repositories are not modeled with all the mechanical parts needed to infer their motion. Similarly, the animation of sketches requires user assistance to provide the missing motion information.

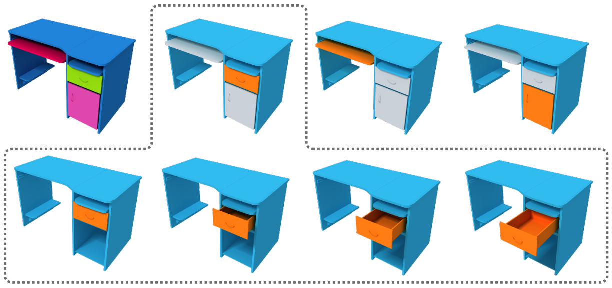
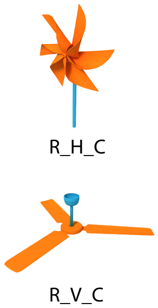


Fig. 3. Training data setup: given a segmented shape, shown on the top-left, we define several mobility units by pairing the parts into (moving, reference) pairs, as shown on the top row. The bottom row shows several snapshots of the motion of the highlighted unit.

Part mobility from indoor scenes. Sharf et al. [2013] build a *mobility tree* to summarize the support relations between objects or parts in a scene, and their relative mobility. First, the input scene is searched for repeated instances of objects. Next, given a repeated model detected in distinct configurations, the method discovers possible motions the model can undergo. One limitation of this approach is that it relies on the occurrence of repeated models in the input scene, appearing in different states of motion, e.g., open and closed drawers. Thus, the detected mobility cannot be easily transferred to objects that do not appear in the scene, since the motion is discovered separately for each instance. In contrast, we learn a model that groups together similar types of motion coming from different objects, and can then be used to transfer the motion to new objects, which do not necessarily have the same geometry as the analyzed models.

Mobility fitting from motion sequences. Li et al. [2016] capture the part mobility of an articulated object by searching for a set of joints that determine the motion. Bypassing geometry reconstruction of the model, they directly solve an optimization to find the joints and motion parameters. On the other hand, Pero et al. [2016] identify parts that move together in video sequences to extract articulated object parts, while Stückler et al. [2015] use random forest classifiers and expectation-maximization to identify rigid parts. In comparison to our work, these approaches are restricted to articulated models, and require dynamic data in the form of a scan or video sequence.

Tevs et al. [2012] present a method for reconstructing dynamic shapes which enables the acquisition of more general motions than just articulated parts. Their input is also required to be a scan sequence with a considerable number of intermediate snapshots. In recent work, Xue et al. [2016] synthesize future frames from single input images using a cross convolutional network. However, their method requires a considerable amount of dynamic data for training, in the order of tens of thousands of image pairs, and the training of a deep neural network. In contrast, our method requires the learning of the S-D mapping, which can be accomplished with significantly less data.



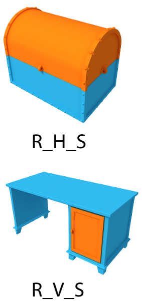


Fig. 4. Classification of the training units into motion types. We show one example unit for each motion type. See text for labels.

3 TRAINING DATA SETUP

The input to the training is a set of shapes in upright orientation and with parts segmented into separate geometries. The parts are grouped into *mobility units*, where each unit is composed of a *moving* part and a *reference* part, e.g., a drawer that moves and the furniture frame that serves as a reference for the motion. The complete input consists of a few static configurations of each mobility unit, which we call *snapshots*. The snapshots are given at different states of the motion, e.g., drawer fully-open, drawer half-open, etc. An example of the input is shown in Figure 3. We ensure that the start and end snapshots of each unit are included, and order all snapshots in a unit according to the chronological motion of the parts.

Each unit in our dataset is associated with a set of motion parameters represented as a quadruple $(t, \mathbf{a}, \mathbf{p}, \mathbf{r})$, where t is the transformation type, $\mathbf{a}$ and $\mathbf{p}$ are the direction and position of the motion axis, and $\mathbf{r}$ is the transformation range, stored as a start and end position (for translations) or angle (for rotations) relative to the center of mass of the moving part and the upright axis.

The units can then be roughly classified into 8 different motion types, according to the first three parameters. The classification is based on labeling the units according to their transformation type (translation, rotation, or both, denoted as T, R, and TR), the general direction of the translation or rotation axes (horizontal or vertical, denoted as H and V), and the location of the axes (close to the center of the units or to one of their sides, denoted as C or S). Note that the transformation range parameter, which is determined by the start and end snapshots, is not used for the motion classification since it is more related to the semantics of the given object, while our classification is mainly intended for facilitating the learning of a motion-dependent distance measure. We show one example unit for each motion type in Figure 4. We provide this classification as an initial grouping to our method, and also use the motion type and the associated motion parameters as ground-truths to evaluate the results of motion prediction and transfer, respectively, in a cross-validation scheme.

4 METRIC LEARNING

The key to enable the S-D mapping is to learn a distance between a snapshot and a unit. To ensure accurate motion prediction for unseen snapshots, we learn a separate distance for each motion type. In the following, we first detail the different distance measures we use in our method, and then describe the metric learning.

Snapshot descriptors. We use a set of descriptors to represent the configuration of the moving and reference parts appearing in a snapshot. First, we capture the interaction between the two parts with the *interaction bisector surface* or IBS [Zhao et al. 2014]. The IBS is a subset of the Voronoi diagram computed between the two objects, which captures the spatial region where the objects interact with each other. Moreover, as the geometry of the parts themselves is relevant to their motion, we also capture the regions on the surfaces of the objects that correspond to their IBS, called the *interaction regions* or IRs [Hu et al. 2015]. We represent the IBS and IR with the same descriptors as in Hu et al. [2015]. We also represent the relative configuration of the parts with the RAID descriptor [Guerrero et al. 2016], which captures inter-region relations based on the spatial distribution of point-to-region relationships. The descriptors used to encode IBS and IR and the details on how we adapted RAID to our setting can be found in Appendix A.

Distance measures. We will define three distance measures that are used by our mobility model.

Snapshot-to-snapshot distance. The distance between two snapshots is a weighted combination of N individual descriptor distances:

$$D^S_W(s_i, s_j) = \sum_{f=1}^{N} w_f D^S_f(s_i, s_j), \tag{1}$$

where $D^S_f(s_i, s_j)$ is the distance between snapshots s_i and s_j for the f-th descriptor, normalized to the range $[0, 1]$, $w_f \in W$ lies in the range $[0, 1]$ and is the weight for descriptor f, and $\sum_f w_f = 1$. Since the sum of weights is one, D^S_W is also in the range $[0, 1]$. Note that this distance depends on the chosen weights W. If we choose different weights, we can obtain different distance measures as will be described below.

Snapshot-to-unit distance. This measure compares a snapshot to a unit by combining our linearity assumption with the snapshot distance defined above:

$$D^{SU}(s, u_j) = \frac{1}{2}\left(D^S_{W_j}(s, s^j_1) + D^S_{W_j}(s, s^j_m)\right), \tag{2}$$

where s is an arbitrary snapshot, s^j_1 and s^j_m are the start and end snapshots of unit u_j, respectively, and m is the number of snapshots used to represent each unit. The snapshot-to-unit distance is the main tool used in the S-D mapping, where we compute the distances of a snapshot to the units in the training data, and select the unit closest to the snapshot. Note that we use the weights W_j learned for the unit u_j when invoking the snapshot-to-snapshot distance D^S, since we do not necessarily know the source unit of the snapshot s, especially if s is a query during motion prediction.

Unit-to-unit distance. We also define a distance between two units, which is used to cluster similar types of units as explained below:

$$D^U(u_i, u_j) = \frac{1}{m} \sum_{s^i_k \in u_i} D^{SU}(s^i_k, u_j). \tag{3}$$

This distance is asymmetric since it considers the snapshots of unit u_i and the weights W_j learned for the unit u_j when invoking D^{SU}.

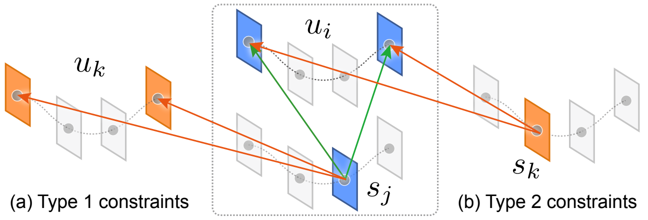


Fig. 5. Metric learning constraints: (a) Type 1 between a snapshot and two units: $D^{SU}(s_j, u_i) < D^{SU}(s_j, u_k)$; (b) Type 2 between two snapshots and a unit: $D^{SU}(s_j, u_i) < D^{SU}(s_k, u_i)$. Different snapshot colors indicate different motion types. The distances indicated with green lines should be smaller than the distances of orange lines.

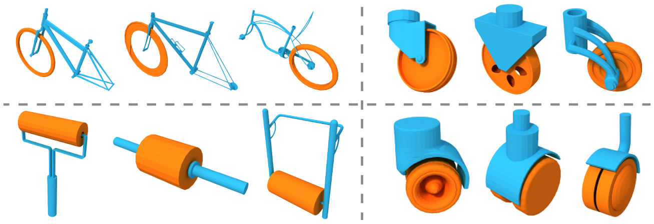


Fig. 6. Selected clusters of motion type R_H_C, one per cell, obtained by performing affinity propagation clustering according to our unit distance measure. Note how the units in a cluster have similar local geometry and part interactions.

Metric learning. The goal of this step is to learn a different set of weights for each motion type, as each type of motion may be better described by different descriptors. The weights W for the snapshot distance D^S are learned from a set of constraints derived from the snapshots in the training data and their specified motion types. While learning the weights, we take into account the effect that the weights have when comparing snapshots to units with D^{SU}. Thus, the constraints used in the learning ensure that units and snapshots with the same type of motion are kept closer to each other than to units or snapshots with a different type of motion. We achieve this with two types of constraints.

Suppose that we have three different mobility units u_i, u_j and u_k, where any of their snapshots can be denoted as s_i, s_j and s_k, respectively. Let us assume that units u_i and u_j belong to the same motion class, while unit u_k is from another class. The Type 1 constraints (illustrated in Figure 5(a)) capture the notion that snapshots (e.g., s_j) should be kept closer to units with the same type of motion as themselves (e.g., u_i), rather than to units with a different type of motion (e.g., u_k). Therefore, we derive a constraint on comparing one snapshot to two different units: $D^{SU}(s_j, u_i) < D^{SU}(s_j, u_k)$.

Type 2 constraints (illustrated in Figure 5(b)) capture the notion that the distance from snapshots to units of the same motion type should be smaller than the distance from snapshots to units of a different type. Therefore, we derive a constraint on comparing two snapshots to the same unit: $D^{SU}(s_j, u_i) < D^{SU}(s_k, u_i)$.

For realizing the S-D mapping, it is sufficient to perform the metric learning only with Type 1 constraints, as they ensure that the nearest neighbor unit provided by the snapshot-to-unit distance is meaningful. However, as we demonstrate in Section 7, there are applications that benefit from Type 2 constraints, as they require the distance from different snapshots to a unit to be comparable.

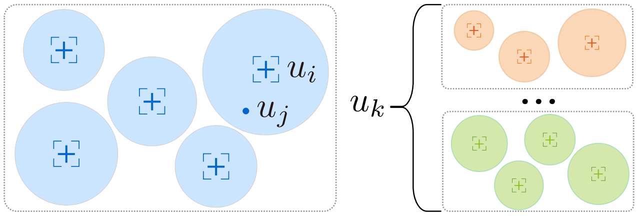


Fig. 7. Subsampling the constraints using finer clustering of motion types. Each motion type is drawn in a different color and shown in a dashed square, while the clusters of units in a motion type are the colored circles, where the centers are marked by crosses. The only constraints we use for snapshots of a given unit u_j involve the unit's cluster center u_i and cluster centers of other motion types (e.g., u_k).

Clustering of units. If the training data is large, the number of derived constraints may be prohibitive for learning the distance in practice. Thus, we systematically subsample a tractable number of constraints. To subsample, we first cluster each input motion type into smaller groups of similar units (see Figure 6 for examples). This allows us to reduce the number of constraints by defining constraints only in terms of clusters, where the number of clusters is significantly lower than the number of units. Each cluster is composed of finer variations of the same type of motion.

We estimate the similarity between units with the unit distance measure defined in Eq. 3, which considers the motion and geometry of the part interactions of the units, while assuming equal weights for all descriptors. As we will see in Section 6, equal weights do not provide the best indication of motion similarity, but lead to a reasonable clustering of units. We perform the clustering with the affinity propagation method [Frey and Dueck 2007]. The advantage of this method is that it does not depend on a good initial guess to yield reasonable clustering results, and it automatically selects the number of clusters based on the distance between units and an estimate of how likely each unit is a cluster center. We set this estimate for a unit as the median distance from the unit to all other units in the set. The output of this process is a clustering of mobility units for each motion type, and a unit selected as the center of each cluster, which are then used for defining the constraints.

Constraint subsampling. To define the constraints that involve snapshots s_j of any unit u_j in our dataset, we choose u_i to be the center of u_j's cluster, and take u_k to be the center of one of the clusters of a different motion type; see illustration in Figure 7. Since we use the nearest neighbor to define the S-D mapping, it is sufficient to ensure that any snapshot is close to its cluster center. Thus, we do not use additional constraints between snapshots and other cluster centers within the same motion type. This subsampling reduces the total number of constraints significantly.

Optimization. We use the constraints defined above to learn the distance with a method similar to that of Schultz and Joachims [2003], where the main difference is that we constrain the sum of weights with respect to each motion to be one.

More details on the constraint subsampling and optimization are given in the supplementary material.

Algorithm 1 *MotionTransfer*$(q, u, n, l) \rightarrow (t, \mathbf{a}, \mathbf{p}, \mathbf{r})$

```
Input: query snapshot $q$, top retrieved unit $u$, maximum iteration number
    $n$, initial sample step size $l$
Output: motion parameters $(t, \mathbf{a}, \mathbf{p}, \mathbf{r})$ of $q$
$\mathcal{M} \leftarrow$ GenerateCandidateParams($q$, $u$)
bestDist $\leftarrow \infty$,  bestM $\leftarrow \emptyset$
foundSample $\leftarrow$ false,  iterNum $\leftarrow 1$,  checkValidity $\leftarrow$ true
while foundSample = false do
    if iterNum = $n$ then
        checkValidity $\leftarrow$ false
    end if
    for each $m \in \mathcal{M}$ do
        $\mathcal{S} \leftarrow$ SampleMotion($q$, $m$, $l$, checkValidity)
        if $\mathcal{S} \neq \emptyset$ then
            foundSample $\leftarrow$ true
        end if
        for each $s \in \mathcal{S}$ do
            $d \leftarrow D^{SU}(s, u)$      /* snapshot-to-unit distance */
            if $d <$ bestDist then
                bestDist $\leftarrow d$
                bestM $\leftarrow m$
            end if
        end for
    end for
    $l \leftarrow l/2$
    iterNum $\leftarrow$ iterNum + 1
end while
return bestM
```

5 MOTION PREDICTION AND TRANSFER

To perform the motion prediction, given a query snapshot q from an unknown unit, we use the snapshot-to-unit distance defined in Eq. 2 to compare the query to the units in the training data, and select the unit u that is the most similar to q. After that, we can transfer the motion from u to the parts of the query q. The goal is to find a set of motion parameters for q, such that the motion of q is consistent with the parameters of u, but is adapted to the geometry of q's parts. Note that the motion parameters of training units are not used during the prediction, but for motion transfer only.

Motion transfer overview. The general idea of the motion transfer procedure is to generate candidate motion parameters for q, and select the candidate parameters that provide the best fit with the motion of u. More specifically, we sample candidate parameters for q according to the motion parameters of u. To verify the quality of the fit, we generate additional snapshots for q using the motion defined by the candidate parameters, and compute the distances from the new snapshots to the unit u, according to the snapshot-to-unit distance. Finally, we select the parameter set that generated the snapshot with the smallest distance, and use it to define the motion on q. An overview of the motion transfer is presented in Algorithm 1. The steps of the method are explained in more detail as follows.

Candidate motion parameters. To generate the candidate sets of motion parameters for q, we determine the four motion parameters one by one. First of all, the transformation type of all the candidates should be exactly the same as the transformation type of u.

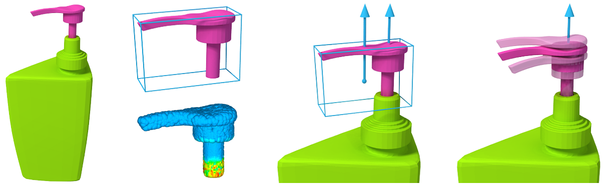


Fig. 8. Motion transfer: we sample different candidates for the motion axis of a query snapshot, based on the OBB and the IR of the moving part in the snapshot. We then select the best candidate as the motion axis of the query shape.

For the candidate axis, we observe that most man-made objects possess some kind of symmetry and the transformation axis direction and position of the movable parts are usually highly related to the symmetry. Thus, we first compute the symmetry-aligned oriented bounding box (OBB) of the moving part in q, with the OBB computation of Fish et al. [2014]. Then, based on the axis direction label of u, which is either "horizontal" or "vertical", each edge of the OBB which follows the prescribed direction provides a candidate direction for the translation or rotation axis. We assume that the input shapes are upright-oriented, thus an edge direction can be easily classified as vertical or horizontal by computing the angle between the edge direction and the upright direction.

For each candidate axis direction, we further sample multiple candidate positions based on the axis position label of u. For "side" axes, we take all the edges of the OBB that are parallel to the candidate direction and use the edge centers to determine the position of the axis. For "central" axes, we select two points to generate two candidate axes passing through those points. One point is the center of the OBB, while the other is the weighted center of the interaction region (IR) on the moving part, computed according to the weights of points on the shape that indicate the likelihood of the points belonging to the IR [Hu et al. 2015]. One candidate selection example is illustrated in Figure 8.

For the transformation range, since the shapes are not aligned, we cannot directly transfer the motion range from u to q. Thus, we transfer the extent of the range temporarily, and determine the exact motion range for q during motion sampling, as explained below. In the case of rotations, we transfer the rotation angle extent, defined as the difference between the start and end angles. This strategy does not apply to translations, since translation depends on the scale of the shapes, which can differ among units. Thus, for translations, we define the extent as the length of the translation vector from the start and end positions of the motion. Then, we compute the ratio between the extent and the projection length of the moving part along the motion axis of u. Finally, the extent for q should be the projection length of its moving part along the candidate axis, scaled by the ratio.

Motion sampling and selection. To generate additional snapshots for q using a candidate axis, we treat q as the center of the motion and extend the motion to each "side" of q according to the motion axis and transformation type. That is, we either translate the moving

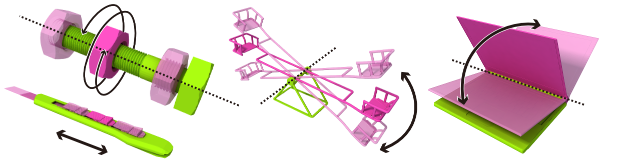


Fig. 9. Examples of motion ranges estimated for test snapshots with different motion types. We show the start and end snapshots of the estimated motion (drawn with transparency) for the given snapshot (drawn in solid color).

part along the two possible directions of a translation axis starting from q, or rotate the part around a rotation axis at q into the two rotational directions. The motion range of q is determined from how far the expansion went; see Figure 9.

During the expansion from q, we sample the geometry to create additional snapshots and check if the newly sampled snapshots are *valid* or not. Specifically, we start with a sample interval distance l, and create a new motion snapshot a distance l away from q to each side of q. The snapshot is created by either translating the moving part l units along the translation axis and sampling direction, or by rotating the moving part an angle of l degrees around the rotation axis. Next, if a snapshot is valid, we attempt to continue the expansion along its side. We stop when no further expansion can be done.

If no valid snapshots were found for any of the candidate axes, we divide the step size l by half to sample a finer set of motion snapshots. In this way, we adaptively adjust the step size to find valid motions, while preferring large motions first. The initial l is set as 0.4 times the motion range extent of u. If after the maximum number of iterations $n = 3$ of adaptive sampling we still cannot find any valid snapshot for all the candidates, we disable the validity checking and sample two snapshots for each candidate axis on both sides of q.

A snapshot is *valid* if the moving part remains connected to the reference part during the motion, but without significant intersection between the parts. In more detail, a snapshot is valid in two cases: (i) If the two parts collide, most of the intersection should be confined to the IR of the reference part, since it is expected that collisions can happen in this region. (ii) If there is no collision, the moving and reference parts should remain as close as possible. The closest distance between the moving and reference parts in the query snapshot provides a reasonable threshold for this closeness constraint. In our implementation, a sampled snapshot is valid as long as the two parts are not further apart than twice this threshold. If the threshold is zero, the parts should remain connected during the motion.

6 RESULTS AND EVALUATION

In this section, we demonstrate the use of our part mobility model for motion prediction and transfer, and evaluate different aspects of the learned part mobility model. We show additional example applications of our model in the next section.

Dataset. We evaluate our method on a set of 368 mobility units with diverse types of motion encountered in our daily lives. As described in Section 3, we use the classification of the units into motion types (Figure 4) and their motion parameters as a ground-truth to perform a quantitative evaluation. The full dataset and the classification of units into motion types are shown in the supplementary material.

Dataset preparation. We collected the shapes in our dataset mostly from the ShapeNetCore repository, and complemented this sample with shapes from other repositories like SketchUp, to ensure that the dataset contains a variety of motions. Then, we manually segmented the shapes into parts. To further complement the shapes with motion information, we implemented a tool to manually add motion parameters to shape parts, so that motion sequences can be generated automatically from the parameters. A non-expert user can employ the tool to select a unit from a segmented shape and manually specify the motion parameters. The construction of our dataset took about 5 hours with this tool, with less than 1 minute per shape on average.

By sampling snapshots from the motion sequences, we create the sparse set of two snapshots used for training, and denser sets of snapshots used as ground-truth in the evaluation. Note that, although our dataset would allow us to also train a model with a denser set of samples, using a sparse set of samples can provide comparable results as we explore in Section 3 of the supplementary material. Moreover, there are many sources of data for which performing such manual motion assignment would be prohibitive, e.g., for processing large-scale datasets or 3D scans which are not as clean and easy to manipulate as our models. In these more difficult scenarios, using our method is beneficial for reducing the amount of manual effort required to collect the motion sequences.

Snapshot-to-unit distance measure. As the key contribution of our paper is a new snapshot-to-unit distance which enables the mapping between the static and dynamic domains, we first evaluate the overall performance of the distance measure we learned. We perform a 10-fold cross validation experiment, that is, we divide the units in the training data into 10 folds of equal size and evaluate the distance measure when taking each fold as the test set while training on the other nine folds. For this and the subsequent experiments, we use four snapshots per unit for training, and nine snapshots per unit for testing. We evaluate the distance measure in two directions, i.e., snapshot-to-unit and unit-to-snapshot directions. For the snapshot-to-unit direction, given a test snapshot, we rank all the training units according to their distance to the given snapshot and then verify whether the units with same motion type are consistently ranked before units with other motion types. For the unit-to-snapshot direction, given a training unit, we rank all the test snapshots according to the distance measure, and evaluate the ranking.

We evaluate the quality of the ranking with the ranking consistency (RC) measure [Hu et al. 2016]. We take the snapshot-to-unit direction as an example to explain the computation of the RC measure, while the unit-to-snapshot direction can be derived in a similar manner. The RC evaluates in a quantitative manner how well a set of retrieved items is ordered according to ground-truth labels, by comparing the relative ordering between pairs of items. We define

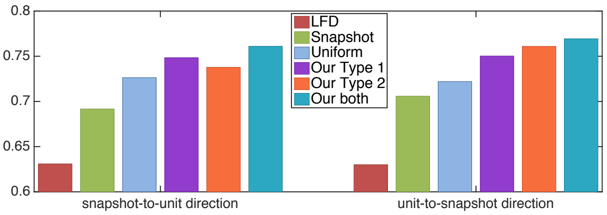


Fig. 10. Ranking consistency for different snapshot-to-unit distances. Please refer to the text for details.

the RC for a test snapshot s as:

$$\mathrm{RC}(s) = \sum_{u_i \in \mathcal{I}} \sum_{u_j \in \mathcal{O}} C(D^{SU}(s,u_i), D^{SU}(s,u_j)) \Big/ |\mathcal{I}||\mathcal{O}|, \quad (4)$$

$$\text{where} \quad C(d_i, d_j) = \begin{cases} 1, & \text{if } d_i < d_j, \\ 0, & \text{otherwise,} \end{cases} \quad (5)$$

with $\mathcal{I}$ being the set of training units with the same motion type as s, and $\mathcal{O}$ the set of training units with different motion types. The RC ranges in [0, 1] and captures the global quality of the ranking according to the ground-truth. Note that the RC tests whether positives are ranked higher than negatives and thus is equivalent to the area under the receiver operating characteristic (ROC) curve [Steck 2007].

We compute the average RC for test snapshots and report it as the graph on the left of Figure 10, while the result for the unit-to-snapshot direction is shown on the right. In this evaluation, we study the effect that using different types of weights and metric learning constraints have on the model. The bar labeled *Uniform* denotes the accuracy of motion prediction when using a set of equal descriptor weights for the snapshot distance. *Our Type 1* and *Our Type 2* denote the cases when only Type 1 or Type 2 constraints are used, and finally *Our both* denotes the use of both constraints for the metric learning. We note that the RC is best preserved when both constraints are included in the optimization (*Our both*).

Comparison to a baseline method. To further evaluate our distance measure, we compare it to other distance choices. First, we compare to a baseline that uses only the geometry of shape parts for prediction. Specifically, we use the light field descriptor (LFD) to compute the distance between two snapshots, which is taken as the sum of the LFD descriptor distances between the moving and reference parts of the two snapshots. Since all the snapshots of a unit have the same moving and reference parts, it is sufficient to use one snapshot as representative of each unit. Then, we match a query snapshot to the unit in the training set with the lowest distance to its representative snapshot, and assign the motion of the unit to the query snapshot. The result is shown in Figure 10 and denoted *LFD*.

Metric learning applied to the snapshot-to-snapshot distance. To show the effectiveness of our linear assumption on the S-D mapping, we also evaluate the scenario where we apply the metric learning directly to the snapshot-to-snapshot distance, instead of the snapshot-to-unit distance. We then derive a snapshot-to-unit distance by exhaustively computing the distance from the query to each snapshot of a unit with the optimized snapshot-to-snapshot distance, and picking the unit corresponding to the snapshot with the minimum distance. The result is denoted as *Snapshot* in Figure 10.

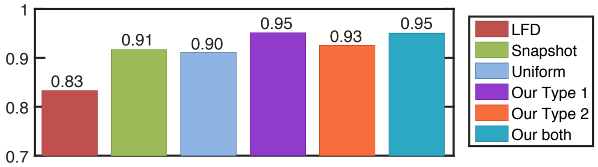


Fig. 11. Motion prediction accuracy for our method when using different sets of weights. We also include a comparison to a baseline method (LFD). See the text for details.

Table 1. Size (in number of units) and average prediction accuracy for all motion types.

Motion type	**R_H_C**	**R_H_S**	**T_H**	**TR_H**
Cluster size	85	71	54	28
Prediction accuracy	0.98	0.91	0.95	0.98

Motion type	**R_V_C**	**R_V_S**	**T_V**	**TR_V**
Cluster size	21	29	16	64
Prediction accuracy	0.90	0.96	0.86	0.98

Discussion. From these comparisons, we conclude that our part mobility model takes into consideration not only the geometry of the parts, but also the correct motion that the parts can possess. Thus, our method provides more accurate results when compared to the LFD baseline that only considers the geometry of parts. With the linearity assumption that simplifies the distance measure between a snapshot and a unit, our method not only saves time during training and when computing the distance measure, but also provides better performance than when the snapshot-to-snapshot distance is optimized directly, since only very few snapshots are sampled for each training unit. Finally, when using different types of weights and metric learning constraints, we observe that the metric learning has a benefit in the prediction accuracy when compared to uniform weights. The best prediction accuracy is obtained when both types of constraints are incorporated into the learning.

Motion prediction. Since our main focus is to use S-D mapping for motion prediction and transfer, we specifically evaluate the motion prediction accuracy of the top retrieved unit for a test snapshot, which is the most relevant for motion prediction. We perform the same 10-fold cross validation experiment described above and verify the accuracy of motion prediction for all the test snapshots in a fold. The accuracy is evaluated according to the ground-truth labels, and we report the average accuracy for the ten tested folds. The comparison results with different distance measures are shown in Figure 11. We see that the best prediction accuracy of 0.95 is obtained when either constraints of Type 1 or both types of constraints are satisfied. Using only the snapshot-to-snapshot distance or uniform weights provides a slightly lower accuracy (0.91), while LFD has the lowest accuracy (0.83). Figure 2 shows a few visual examples of units predicted by our model, in comparison to other options.

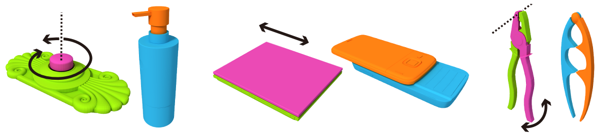


Fig. 12. Failure cases for motion prediction. Each example is the query snapshot on the left and the predicted unit to the right. As can be seen, a wrong type of motion is inferred.

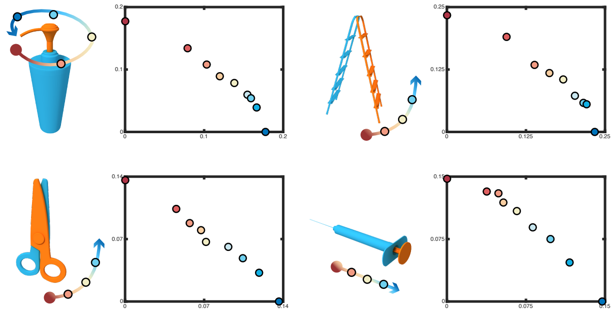


Fig. 13. Linearity for the snapshots of the units shown to the left of each graph, where the distances to the start and end snapshots of the unit are indicated along the x and y axis, respectively. We observe that the sum of distances remains almost constant.

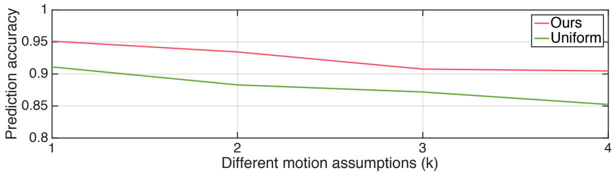


Fig. 14. Average accuracy for motion prediction when using different motion assumptions for defining the snapshot-to-unit distance.

Table 1 shows the size and prediction accuracy for each motion class in our dataset. We observe that the accuracy for all motion types is above 0.86. The lowest accuracies appear in the classes R_H_S, R_V_C and T_V, where T_V has the smallest set of training data, which makes the prediction unstable compared to other motion types. Figure 12 presents a few examples of incorrect predictions for these classes. We see how the geometry and interaction of parts in the query and unit are similar, although the motion supported by the parts is different. We also evaluate the different components and parameters of our model as follows.

Linearity assumption. In Figure 13, we investigate our assumption that the distances from snapshots to the their units exhibit a form of linearity. We see in these examples with different types of motion that, after learning, the snapshot-to-unit distance exhibits the assumed linearity. In general, we observe a similar result for the other types of motion in our dataset.

Moreover, we compare our linearity assumption to other possible motion assumptions. More specifically, we define snapshot-to-unit distances that consider different k as follows:

$$D_k^{SU}(s, u_j) = \frac{1}{2}\left(\left(D_{W_j}^{S}(s, s_1^j)\right)^k + \left(D_{W_j}^{S}(s, s_m^j)\right)^k\right). \qquad (6)$$

Note that our previous definition in Eq. 2 is the special case when $k = 1$. We show the comparison to these distances in Figure 14. We see that the linearity assumption provides the best prediction results, while our metric learning is able to improve the prediction accuracy under all of the different motion assumptions.

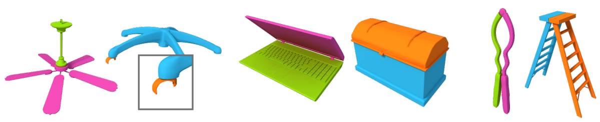


Fig. 15. Three prediction results when certain semantic classes are missing from the training data. For each pair, the query snapshot is shown to the left and the predicted unit to the right.

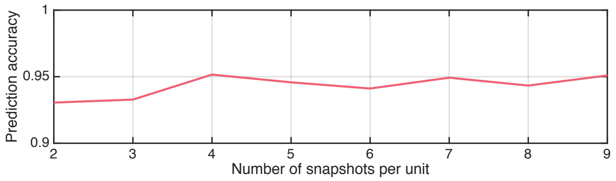


Fig. 16. Average accuracy for motion prediction in a cross-validation experiment. We measure the accuracy when performing the metric learning with different numbers of snapshots per unit.

Prediction for missing semantic classes. One question related to our model is whether we can find a suitable motion for a snapshot even if its exact semantic class is not present in the training set. To study this scenario, we perform an experiment where we remove an entire semantic class from the training data, and then verify what motion is predicted for snapshots from this removed class. We present a few example results in Figure 15. We see that, although we do not have the correct semantic class in the model, we map the snapshots to classes where the motion is relevant and has similar parameters, such as a rotating fan mapped to rotating wheels on a chair's legs.

Number of training snapshots per unit. We perform an experiment to investigate the effect that using different numbers of snapshots m per training unit has in the accuracy of motion prediction. To ensure that the results for different m are comparable, we always represent test units with nine snapshots when computing the prediction accuracy. The results are shown in Figure 16. We observe that the average accuracy lies around 0.95 when using four or more snapshots per training unit. Using more than four snapshots does not improve the accuracy much more for the types of motion encountered in our dataset.

Effect of training set size. We investigate the relation between the size of the training set and the accuracy of motion prediction. For this experiment, we perform cross-validation experiments as explained above, except that we vary the number of units in the training set to be a percentage of the entire dataset. We then evaluate the accuracy of prediction on a fixed set of 10% of units randomly sampled from the dataset and left out for testing. As observed in

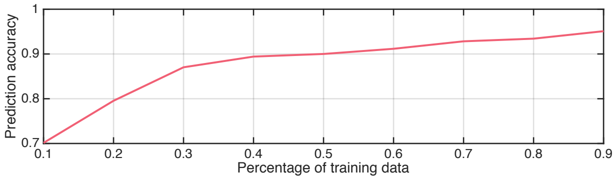


Fig. 17. Average accuracy for motion prediction when using training sets of increasing size, where more units are included.

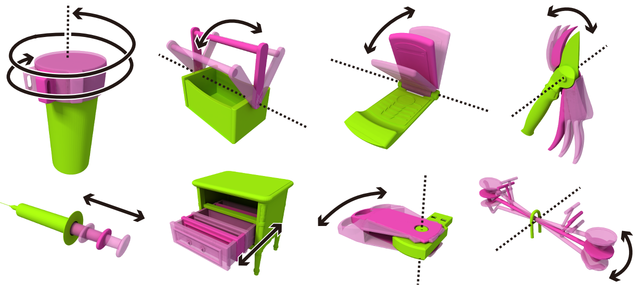


Fig. 18. Examples of motion transfer obtained with our method after predicting the correct motion type for each snapshot. The transformation axes (for rotation or translation) are denoted with the dashes lines, while the motion is indicated by the arrows.

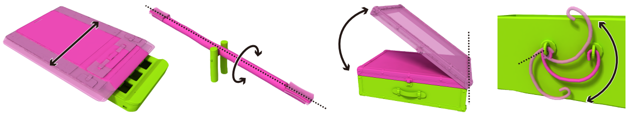


Fig. 19. Failure cases of motion transfer where the transformation axes were flipped.

Figure 17, with approximately 40% of our dataset (148 units), we obtain an accuracy of 0.9 or higher for the selected random sample. Thus, we can obtain a general model that covers a variety of part mobility types with only 148 units stored in the model.

Motion transfer. We evaluate the quality of the motion transfer to verify that, after correctly predicting the motion type for a snapshot, we can also successfully transfer the actual motion from the retrieved unit. Figure 18 shows examples of motion transfer results. We see how motions with different types of transformations and axes can be transferred successfully to diverse objects. The supplementary video accompanying this submission shows animated examples of results.

We evaluate the quality of the motion transfer by comparing the transferred transformation axis to the ground-truth axis with two measures. First, we measure the distance between the supporting lines of each axis. However, to obtain a distance that takes the parts into account, we consider only the portions of the lines that lie inside the bounding box of the snapshot, in fact measuring the distance between two line segments. This ensures that we get an accurate distance even for co-planar lines that have a zero distance because of intersection points far from the snapshot. Next, we also evaluate the angle between the transferred and ground-truth axes. We evaluate these two measures for all the snapshots of our dataset also in a cross-validation scheme with 10 folds.

We find that the distance between axes is on average 0.03, with a standard deviation of 0.08, where the shapes are normalized so that the moving part fits into a bounding box that has a diagonal of length 1. Thus, the distance is small as it corresponds to 3% of the part's bounding box diagonal, implying that the errors are more on a finer scale. The angle between the axes is on average 8.14 degrees with a deviation of 32.93. Further analysis reveals that the errors for angles typically concentrate around 0 and 90 degrees, where 90 degrees is the case where the two possible horizontal axes are flipped with each other. We found that 92% of the errors concentrate around 0 degrees, with a deviation of 4.37, implying that the angle is well approximated. The remaining 8% of the errors are flipped axes. Figure 19 shows representative failure cases.

To obtain more insight into the performance of each step of the motion transfer, we also pose the motion transfer as a classification task and evaluate the accuracy of the transfer. First, we set parameter difference thresholds of 5.0 and 0.3 for the transformation axis direction and position, respectively. Then, given a snapshot, if the distance between the transferred parameters and the ground-truth parameters is below the given thresholds for the two parameters, and the transformation type is the same as in the ground-truth, then we classify the motion transfer of the snapshot as correct. We then compute the transfer accuracy for the candidate parameter sets and for the final selected motion. To evaluate the candidate generation, we verify whether any of the generated candidates is classified as correct. To evaluate the final motion, we simply verify whether the final candidate selected is correct.

The accuracy of the candidate generation step in isolation is 0.96. Given that the initial accuracy for the prediction of motion type using the S-D mapping is 0.95, the overall accuracy of candidate generation is $0.91 = 0.95 * 0.96$. Moreover, the accuracy of the final transfer after selecting one candidate is 0.94 in isolation, or $0.86 = 0.91 * 0.94$ when considering the previous steps. Thus, we observe that, for each step, the accuracy of the prediction is relatively high. However, the error accumulates providing a final performance of 0.86.

Timing. The entire learning takes around 11 minutes on our dataset of 368 mobility units with 4 snapshots per unit, where the clustering for constraint subsampling takes 14s, and the metric learning takes 617s with both types of constraints. With only one type of constraint, the learning takes around 69s for Type 1 constraints and 422s for Type 2. When only constraints of Type 2 are used, parallelism can be explored to reduce the timing to 70s. Mapping a query snapshot to the most similar unit after learning is fast, taking only 1ms. The timing for motion transfer depends on the number of candidate sets of motion parameters we generate. It typically takes less than 30 seconds to transfer the motion from one unit to a snapshot when 6 candidate sets are sampled. The motion transfer is highly parallelizable and holds the potential for a significant speed-up.

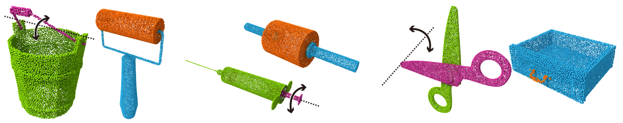


Fig. 20. Failure cases for motion prediction on shape surfaces. Each example is the query snapshot on the left and the predicted unit to the right. As can be seen, a wrong type of motion is inferred.

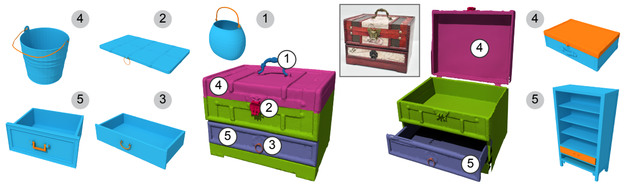


Fig. 21. Motion prediction for two scans of a toolbox. The unit predicted for each pair of parts is indicated by a number.

7 APPLICATIONS

In this section, we explore different applications of the motion prediction and our part mobility model.

Prediction from surface representations. In situations where a 3D object is only available as a segmented surface model, we show that we can still use our mobility model for motion prediction. For this type of shape representations, the internal structures of the object were not modeled, e.g., the internal structure of a drawer is missing and we only have a patch with the outer front of the drawer. To evaluate the performance of our method in this scenario, we learn the model also with surface representations, and then evaluate the prediction accuracy with the same cross-validation procedure as for models with internal structures. In practice, we use the same dataset as before for the learning, except that we only sample points from the visible portions of the shapes to compute the descriptors, which simulates the absence of internal geometry. We obtain an average prediction accuracy of 0.94 when performing the learning with 4 snapshots per unit. We note that this is only 1% lower than the average prediction on full shapes with internal structures. Figure 20 shows a few failure cases for the prediction. For example, a syringe is predicted to have the motion of a roll, since the surface geometry is similar, although the motion is a translation and not a rotation.

An extension to work with real scanned data with annotated segmentations is also possible, which can be promising for robotic applications. The added challenge would come from surface imperfections and incomplete data. Figure 21 shows an example of the predicted motion for different units of a shape scanned in two different configurations. The scans were obtained with an Artec Space Spider scanner, which directly provides a reconstructed triangle mesh as output.

The prediction over surface shapes is potentially applicable to fill in the missing geometry in these shapes. In Figure 22, we show a few examples where we manually completed the missing geometry and then automatically assigned motion to the new parts based on the prediction. Automating such motion-driven shape completions would be an interesting future work.

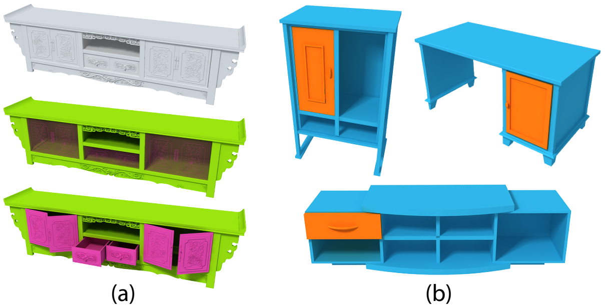


Fig. 22. Completing internal structures from surface representations. (a) The input shape, the segmented surface, rendered with transparency to show the empty space inside, and the final result with completed internal parts. (b) Units predicted for the surface of the door and drawers.

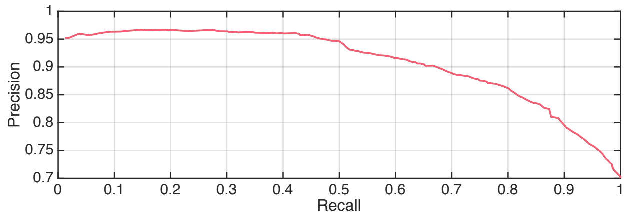


Fig. 23. Precision and recall of snapshot retrieval with our model, which can be potentially used for object detection. Note the high precision for recall rates under 0.5.

Object retrieval. Instead of detecting objects based on geometry only, we can also consider what motions may be applied to the objects, to search for objects that are most suitable for a specific task. Such an application can be useful in a recognition setting, e.g., a robot searches for the top objects that can satisfy a certain type of motion, such as a handle rotation, which could indicate the location of a door handle. We motivate the potential of our model for such an application by evaluating the retrieval of snapshots according to a given motion type. Figure 23 reports the recall and precision when ranking all test snapshots in our dataset according to their distance to a motion type, which is defined as the minimal snapshot-to-unit distance to the training units with such motion type, also in a cross-validation setting. We observe that the precision is quite high, over 0.95, for recalls under 0.5.

Motion hierarchies of shapes. The basic use of our model is to predict the mobility of snapshots composed of pairs of parts. We can further use the model to predict the motion of all parts in a shape, and encode it in the form of a motion hierarchy that reveals the dynamic properties of the shape. Given a segmented shape, we first create a connectivity graph for the parts of the shape. Next, we predict the motion for every possible edge in this graph, which corresponds to two adjacent parts. Since we have no information on which is the moving part and which one is the reference, we predict the motion for both possibilities, and select the one with the lowest snapshot-to-unit distance as the moving part. Finally, we select the part that was never chosen as a moving part in a pair as the root of the hierarchy, while edges of the hierarchy are

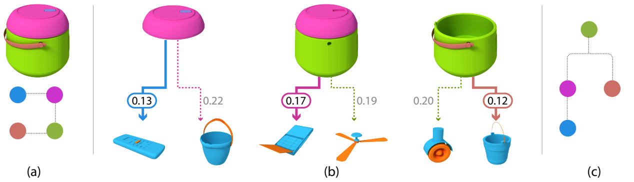


Fig. 24. Construction of the motion hierarchy for a shape. (a) We start with a connectivity graph for the shape parts. (b) We predict the motion for every pair of parts, determining which part is moving or reference in the pair. (c) From the information of all pairs, we establish the root node and motion hierarchy of the shape.

derived by propagating the reference-to-moving part relations we just identified. The process is illustrated in Figure 24. Note how the chosen units have the same type of motion as the query part pairs.

In Figure 25, we present examples of the motion predicted for the parts of selected shapes and their corresponding hierarchies. Note how we are able to find the most relevant motion for the parts.

8 DISCUSSION, LIMITATION, AND FUTURE WORK

We introduced a part mobility model and a supervised method to learn it based on metric learning. We showed that the model can be learned from few static snapshots of mobility units, not requiring the use of dense snapshot sequences capturing the motion of the units. Moreover, we showed with a detailed analysis that the learning creates a meaningful model of part mobilities, which can be used for various applications, including motion prediction, motion-driven object detection, and motion hierarchy construction. The key ingredient of the model is the S-D mapping from static snapshots to dynamic mobility units, which can be used to predict the motion of static pairs of parts appearing on one 3D snapshot of an object.

Limitations. The approach we have developed represents a first step in the direction of learning part mobilities of shapes. Thus, the model has limitations arising from our assumptions that helped simplify the learning scheme. First, we focus on mobility units composed only of pairs of parts. Some objects in the real world may possess units composed of three or more parts, such as those from complex mechanical assemblies, which cannot be modeled with our current approach. Moreover, we assume that each unit satisfies the assumption of linearity, and thus bilinear part mobilities, such as a ball-in-a-socket joint, or nonlinear part mobilities are not captured by our model. In addition, when the model is extended to encode the mobility of an entire shape in the form of a motion hierarchy, we assume that the motion of each unit is independent from each other. Thus, we do not capture the co-dependency of mobility units, e.g., when opening an umbrella, the parts that compose the frame of the umbrella can only move in a coordinated manner.

Another technical limitation of our learning pipeline is that we assume that the input shapes are segmented. Thus, the use of the model in a fully-automatic system for motion prediction will depend on the quality of automatic segmentation. In terms of the evaluation and applications, we only showed preliminary results for motion prediction on scans or surface shapes. We created training data by sampling points on the visible surfaces of the models, to simulate complete scans with the absence of internal geometry. It would be ideal to use real scans for training, whose geometry and part relationships can be quite different from clean data. However, to collect such data, we must face typical difficulties with scan completion and registration. Especially for dynamic motion data, the visible portions of the surfaces change during their motion, requiring the acquisition and merging of multiple scans. More research is required to obtain a fully reliable prediction in such scenarios. Also, additional research is needed on the problem of automatically completing the missing geometry of shape surfaces, which we consider an orthogonal problem to motion prediction.

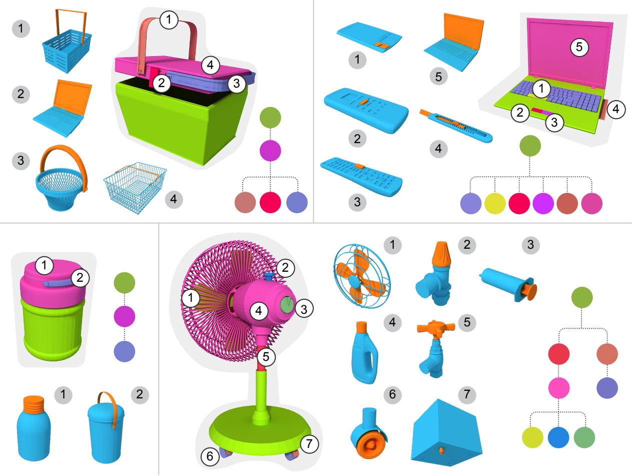


Fig. 25. Examples of motion prediction for all the parts of different shapes, and the corresponding motion hierarchies for the shapes. The query shapes are shown in the middle of the various groups, and the units in the training data closest to different pairs of parts are indicated by numbers. The colors of nodes in the hierarchy indicate the correspondence to the shape parts.

Future work. Some of the limitations discussed above are interesting directions for future work. For example, bilinear or nonlinear part mobilities may be reduced to a composition of multiple linear mobilities, which would allow us to also model these types of motion. Learning the dependency among mobility units in shapes with complex mechanisms may be achieved with a learning-based method that incorporates cues derived from the geometry of the shapes [Mitra et al. 2010]. Recently, there have been many works on learning CNNs for semantic segmentation of 3D models and RGBD scans, which can potentially provide segmented input to our method. The extraction of motion hierarchies for entire shapes may also benefit from a method that combines part segmentation with mobility prediction, so that parts are extracted based on their potential motion. We also believe that more applications that make use of our snapshot-to-unit distance can be explored, especially applications that benefit from both types of constraints used in the learning of the distance measure. Finally, we expect part mobility analysis to be an essential task for VR/AR and robotics applications, where an extension of our learning and processing framework applied to real-time depth or RGBD scans can play a critical role.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable comments. This work was supported in part by NSFC (61602311, 61522213), 973 Program (2015CB352501), Guangdong Science and Technology Program (2015A030312015), Shenzhen Science and Technology Program (JCYJ20170302153208613), Natural Science Foundation of Shenzhen University (827-000196), NSERC Canada (611370, 611649, 2015-05407), and gift funds from Adobe Research.

REFERENCES

Noa Fish, Melinos Averkiou, Oliver van Kaick, Olga Sorkine-Hornung, Daniel Cohen-Or, and Niloy J. Mitra. 2014. Meta-representation of Shape Families. *ACM Trans. on Graphics* 33, 4 (2014), 34:1–11.

Brendan J. Frey and Delbert Dueck. 2007. Clustering by Passing Messages Between Data Points. *Science* 315 (2007), 972–976.

Natasha Gelfand and Leonidas J. Guibas. 2004. Shape Segmentation Using Local Slippage Analysis. In *Proc. SGP*. 214–223.

H. Grabner, J. Gall, and L. Van Gool. 2011. What makes a chair a chair?. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*. IEEE, 1529–1536.

Paul Guerrero, Niloy J. Mitra, and Peter Wonka. 2016. RAID: A Relation-augmented Image Descriptor. *ACM Trans. on Graphics* 35, 4 (2016), 46:1–12.

Jianwei Guo, Dong-Ming Yan, Er Lid, Weiming Dong, Peter Wonka, and Xiaopeng Zhang. 2013. Illustrating the disassembly of 3D models. *Computers & Graphics* 37, 6 (2013), 574–581.

T. Hermans, F. Li, J. M. Rehg, and A. F. Bobick. 2013. Learning contact locations for pushing and orienting unknown objects. In *Int. Conf. on Humanoid Robots*. IEEE, 435–442.

Ruizhen Hu, Oliver van Kaick, Bojian Wu, Hui Huang, Ariel Shamir, and Hao Zhang. 2016. Learning How Objects Function via Co-Analysis of Interactions. *ACM Trans. on Graphics* 35, 4 (2016), 47:1–12.

Ruizhen Hu, Chenyang Zhu, Oliver van Kaick, Ligang Liu, Ariel Shamir, and Hao Zhang. 2015. Interaction Context (ICON): Towards a Geometric Functionality Descriptor. *ACM Trans. on Graphics* 34, 4 (2015), 83:1–12.

Vladimir G. Kim, Siddhartha Chaudhuri, Leonidas Guibas, and Thomas Funkhouser. 2014. Shape2Pose: Human-Centric Shape Analysis. *ACM Trans. on Graphics* 33, 4 (2014), 120:1–12.

Paul G. Kry and Dinesh K. Pai. 2006. Interaction Capture and Synthesis. *ACM Trans. on Graphics* 25, 3 (2006), 872–880.

Hao Li, Guowei Wan, Honghua Li, Andrei Sharf, Kai Xu, and Baoquan Chen. 2016. Mobility Fitting using 4D RANSAC. *Computer Graphics Forum* 35, 5 (2016), 79–88.

Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. 2010. Illustrating How Mechanical Assemblies Work. *ACM Trans. on Graphics* 29, 4 (2010), 58:1–12.

Luca Del Pero, Susanna Ricco, Rahul Sukthankar, and Vittorio Ferrari. 2016. Discovering the physical parts of an articulated object class from multiple videos. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*. IEEE.

Sören Pirk, Vojtech Krs, Kaimo Hu, Suren Deepak Rajasekaran, Hao Kang, Bedrich Benes, Yusuke Yoshiyasu, and Leonidas J. Guibas. 2017. Understanding and Exploiting Object Interaction Landscapes. *ACM Trans. on Graphics* 36, 3 (2017), 31:1–14.

Manolis Savva, Angel X. Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. 2014. SceneGrok: Inferring Action Maps in 3D Environments. *ACM Trans. on Graphics* 33, 6 (2014), 212:1–10.

Manolis Savva, Angel X. Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. 2016. PiGraphs: Learning Interaction Snapshots from Observations. *ACM Trans. on Graphics* 35, 4 (2016), 139:1–12.

Matthew Schultz and Thorsten Joachims. 2003. Learning a distance metric from relative comparisons. *Advances in neural information processing systems (NIPS)* 16 (2003).

Tianjia Shao, Wilmot Li, Kun Zhou, Weiwei Xu, Baining Guo, and Niloy J. Mitra. 2013. Interpreting Concept Sketches. *ACM Trans. on Graphics* 32, 4 (2013), 56:1–10.

A. Sharf, H. Huang, C. Liang, J. Zhang, B. Chen, and M. Gong. 2013. Mobility-Trees for Indoor Scenes Manipulation. *Computer Graphics Forum* 33, 1 (2013), 2–14.

Harald Steck. 2007. Hinge rank loss and the area under the ROC curve. In *European Conference on Machine Learning*. Springer, 347–358.

Jörg Stückler, Benedikt Waldvogel, Hannes Schulz, and Sven Behnke. 2015. Dense Real-time Mapping of Object-class Semantics from RGB-D Video. *J. Real-Time Image Process.* 10, 4 (2015), 599–609.

Art Tevs, Alexander Berner, Michael Wand, Ivo Ihrke, Martin Bokeloh, Jens Kerber, and Hans-Peter Seidel. 2012. Animation Cartography – Intrinsic Reconstruction of Shape and Motion. *ACM Trans. on Graphics* 31, 2 (2012), 12:1–15.

P. Wei, Y. Zhao, N. Zheng, and S. C. Zhu. 2017. Modeling 4D Human-Object Interactions for Joint Event Segmentation, Recognition, and Object Localization. *IEEE Trans. Pattern Analysis & Machine Intelligence* 39, 6 (2017), 1165–1179.

Weiwei Xu, Jun Wang, KangKang Yin, Kun Zhou, Michiel van de Panne, Falai Chen, and Baining Guo. 2009. Joint-aware Manipulation of Deformable Models. *ACM Trans. on Graphics* 28, 3 (2009), 35:1–9.

Tianfan Xue, Jiajun Wu, Katherine L Bouman, and William T Freeman. 2016. Visual Dynamics: Probabilistic Future Frame Synthesis via Cross Convolutional Networks. In *Advances in neural information processing systems (NIPS)*.

Xi Zhao, He Wang, and Taku Komura. 2014. Indexing 3D Scenes Using the Interaction Bisector Surface. *ACM Trans. on Graphics* 33, 3 (2014), 22:1–14.

Y. Zhu, Y. Zhao, and S. C. Zhu. 2015. Understanding tools: Task-oriented object modeling, learning and recognition. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*. IEEE, 2855–2864.

A SNAPSHOT DESCRIPTORS

To represent the IBS and IRs of parts in a snapshot of a mobility unit, we compute a set of geometric descriptors as proposed by Zhao et al. [2014]. We compute: (i) The point feature histogram (PFH) descriptor, which is encoded as the mean and variance of the relative rotation between each pair of normals in the IBS. The PFH captures the general geometry of the IBS; (ii) A histogram that collects the angle between the IBS normals and the +z vector (Dir). This histogram captures the general direction of the boundary between the parts, since the input shapes are upright-oriented along the +z axis; (iii) A histogram that captures the distribution of distances between the IBS and the parts (Dist). These descriptors are also computed for the IRs by considering the surface of each IR rather than the IBS, as proposed in the work of Hu et al. [2015]. In addition, we compute a histogram of height values along the +z axis for each IR.

To adapt RAID to our setting, we extend it from 2D to 3D. We replace pixels by voxels and circles by spheres in the descriptor construction. For the space partitioning within each sphere, we use the three canonical planes to subdivide 3D space, obtaining 8 spatial partitions. Using an inner and an outer sphere, we obtain 16 partitions in total. Given that we have a larger dimensionality than in the original work, we sample 40, 000 points for histogram computation instead of 10, 000, and compare descriptors with the earth-mover's distance rather than the $L1$ metric.