Learning How Objects Function via Co-Analysis of Interactions

Ruizhen Hu[1] Oliver van Kaick[2] Bojian Wu[3] Hui Huang[1,3*] Ariel Shamir[4] Hao Zhang[5]

[1]College of Computer Science and Software Engineering, Shenzhen University

[2]Carleton University [3]SIAT [4]The Interdisciplinary Center [5]Simon Fraser University

Abstract

We introduce a co-analysis method which learns a *functionality model* for an object category, e.g., strollers or backpacks. Like previous works on functionality, we analyze object-to-object interactions and intra-object properties and relations. Differently from previous works, our model goes beyond providing a functionality-oriented descriptor for a single object; it prototypes the functionality of a category of 3D objects by co-analyzing typical interactions involving objects from the category. Furthermore, our co-analysis localizes the studied properties to the specific locations, or surface patches, that support specific functionalities, and then integrates the patch-level properties into a *category* functionality model. Thus our model focuses on the *how*, via common interactions, and *where*, via patch localization, of functionality analysis.

Given a collection of 3D objects belonging to the same category, with each object provided within a scene context, our co-analysis yields a set of *proto-patches*, each of which is a patch prototype supporting a specific type of interaction, e.g., stroller handle held by hand. The learned category functionality model is composed of proto-patches, along with their pairwise relations, which together summarize the functional properties of all the patches that appear in the input object category. With the learned functionality models for various object categories serving as a knowledge base, we are able to form a functional understanding of an individual 3D object, without a scene context. With patch localization in the model, functionality-aware *modeling*, e.g, functional object enhancement and the creation of functional object hybrids, is made possible.

Keywords: Shape analysis, co-analysis, functionality analysis, object-to-object interaction, geometric modeling

Concepts: •Computing methodologies → Shape analysis;

1 Introduction

Most man-made objects are designed to serve certain functions. How an object functions can be reflected by how its parts support various usage scenarios either individually or collaboratively, with different object parts often designed to perform different functions. Understanding the functionalities of 3D objects is of great importance in shape analysis and geometric modeling. It has been stipulated that the essential categorization of objects or scenes is by functionality [Stark and Bowyer 1991; Greene et al. 2016]. As well, the most basic requirement for creating or customizing a 3D product is that the object must serve its intended functional purpose.

Figure 1: *We learn how a* category *of 3D objects function. This may lead us to discover that the geometry of a chair could allow it to function as a desk or handcart (top). One could produce* functional hybrids *(bottom), where different functionalities discovered from two objects are integrated into a multi-functional product.*

In this paper, we are interested in learning a *functionality model* for an object *category*, e.g., strollers or backpacks, where the model describes functionality-related properties of 3D objects belonging to that category. Functionality analysis is challenging since how an object or a part therein functions can manifest itself in diverse forms; there is unlikely a generic model of functionality. Similar to previous works on functionality [Kim et al. 2014; Hu et al. 2015], we focus on functionalities of man-made objects that are inferable from their geometric properties, and our analysis is based on *static object-to-object interactions*. However, differently from these works, our functionality model goes beyond providing a functionality-oriented descriptor for a single object; it prototypes the functionality of a category of 3D objects by co-analyzing typical interactions involving objects from the category. Furthermore, our co-analysis localizes the studied properties to the specific locations, or surface patches, that support specific functionalities, and then integrates the patch-level properties into a *category* functionality model. Thus, our model focuses on the *how*, via common interactions, and *where*, via patch localization, of functionality analysis.

Analyzing and manipulating part structures of shapes is the main subject of structure-aware processing [Mitra et al. 2013]. However, the design of an object's parts and their relations reflects functional, aesthetical, and other considerations. Instead of extracting all part properties and relations which are common in a shape collection [Fish et al. 2014], our model recognizes which of the properties, relations, and their combinations are related to functionality and how the functions are performed. To this end, we learn our functionality model by analyzing object-to-object interactions, intra-object geometric properties and part relations, as well as the empty spaces around objects which influence their functionality.

The input to our learning scheme is a collection of shapes belonging to the same object category, where each shape is provided within a

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

SIGGRAPH '16 Technical Papers, July 24-28, 2016, Anaheim, CA.

ISBN: 978-1-4503-4279-7/16/07

DOI: http://dx.doi.org/10.1145/2897824.2925870

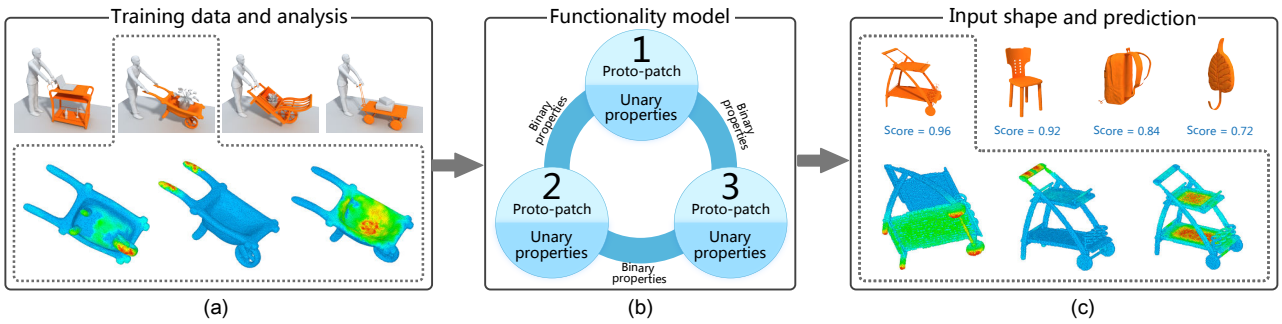


Figure 2: *Overview of the construction and use of our functionality model. (a) Given a set of objects of the same category, where each object is given in the context of a scene, we detect functional patches that support different types of interactions between objects. Example patches are shown as a rainbow color map on the surface of the shape, where values closer to red indicate that a point belongs to the patch with higher probability. (b) We then learn a model that discovers the functionality of the class, describing functionality in terms of proto-patches that summarize the patches in the collection with their properties. (c) Given an unknown object in isolation, we use the model to predict how well the object supports the functionality of the category. This is done by estimating the location of functional patches on the object.*

scene context. To represent the functionalities of an object in our model, we capture a set of patch-level unary and binary functional properties. These functional properties of patches describe the interactions that can take place between a *central* object and other objects, where the full set of interactions characterizes the single or multiple functionalities of the central object.

By means of a co-analysis, we extract the patch properties that are relevant for the functionality of the category. Specifically, the co-analysis yields a set of *proto-patches*, each of which is a patch prototype supporting a specific type of interaction, e.g., stroller handles held by hands. In general, we define a functionality model as a set of proto-patches along with pairwise relations between the proto-patches. Our goal is to learn a category functionality model, or simply, a category functionality, which is composed of proto-patches that summarize the functional properties of all the patches that appear in the training data for the given object category. That said, the localized, patch-level analysis does allow us to define functionality models at varying granularity levels of the objects.

With the learned functionality models for various object categories serving as a knowledge base, we are able to form a functional understanding of an individual 3D object without a scene context. With our patch-based, rather than part-based, analysis, the object does not need to possess any semantic information such as a segmentation. Such an understanding allows us to recognize or discriminate objects from a functionality perspective, to assess functional similarities, and to possibly discover multiple functionalities of the same object; see top row of Figure 1. Furthermore, since our functionality model is localized to the level of patches and current 3D modeling tools operate predominantly at the part-level of objects, functionality-aware *modeling* of 3D objects is possible. For example, we could create *functional object hybrids* by integrating two different functions from two objects and merging them in a way so that the hybrid supports both functions; see bottom of Figure 1.

2 Related work

Structure and functionality. Shape structure is about the arrangement and relations between shape parts, e.g., symmetry, proximity, and orthogonality. In retrospect, many past works on structure-aware analysis [Mitra et al. 2013] are connected to functional analysis, but typically, the connections are either insufficient or indirect for acquiring a functional understanding of shapes. For example, symmetry is relevant to functionality since symmetric parts tend to perform the same functions. However, merely detecting symmetric parts does not reveal what functionalities the parts perform. In addition, not all structural relations are functional. For example, the orthogonality between the seat and back of a chair is an aesthetic property. Thus, although structural relations are useful for the analysis of functionality, a functional understanding of an object or category requires a deeper analysis with additional knowledge acquisition. For example, a single functionality requires a combination of part-to-part relations to be satisfied.

Moreover, structural considerations have also been applied to analyze the arrangements of objects in a scene, benefiting applications such as scene synthesis [Fisher et al. 2012], scene comparison [Xu et al. 2014], and scene understanding [Shi et al. 2016]. In our work, we learn a functionality model via co-analysis of a set of shapes by extracting both intra-shape patch relations and interactions between shapes. We learn combinations of the detected relations that enable specific shape functions.

Meta representation. Recent works generalize shape structures by learning statistics of part relations [Fish et al. 2014] or surfaces [Yumer et al. 2015] via co-analyses. In particular, the meta representation of Fish et al. [2014] provides a prototype of the common structures for a category of 3D objects. The key difference to our work is that meta representations only consider intra-object relations, not all of which are functionality-related; they do not account for object-to-object interactions which are critical to functional analysis. Moreover, both the training and testing data for inferring meta representations come with semantic segmentations, which in some sense, already assume a functional understanding of the object category. Our work analyzes objects at the point and patch level; the objects do not need to be segmented.

Discriminative functionality models. Some past works have focused on categorizing an object based on its function, requiring a method to discriminate between different types of functionality. In the earlier work of Stark and Boyer [1996], a model is handcrafted for a given object category to recognize the functional requirements that objects in the category must satisfy, e.g., the containment of a liquid or stability of a chair. Another class of methods for recognizing functionality is agent-based, where the functionalities are identified based on interactions of an agent with the object [Bar-Aviv and Rivlin 2006; Grabner et al. 2011]. Laga et al. [2013] learn an association between geometric properties and intra-shape part relations of a shape and its functionality, also via a co-analysis. Like meta representations, their work does not consider object-to-object interactions and falls in line with structural co-analysis.

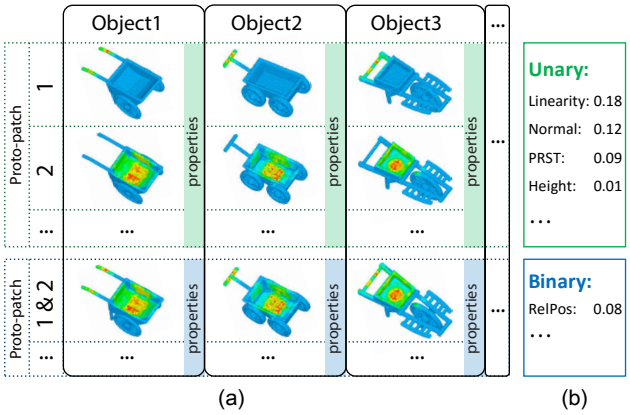


Figure 3: *Our functionality model can be envisioned as the composition of: (a) A collection of functional patches and their unary and binary properties, and (b) a set of weights defining the importance of each property for representing the functionality of the category.*

Our functionality model is learned generically, accounting for both object-to-object and human-to-object interactions. The model is not only discriminative, but also supports functionality-aware modeling. It is a model of the functionality of shapes, rather than just a model of the interacting agents or a set of classification rules.

Shape2Pose and SceneGrok. Inspired by works from vision on object affordances, e.g., [Bar-Aviv and Rivlin 2006; Grabner et al. 2011; Zhu et al. 2014b], recent works in graphics have developed 3D affordance models. Most notably, in Shape2Pose, Kim et al. [2014] analyze the functionality of a 3D object by fitting a suitable human pose to it. Later, SceneGrok [Savva et al. 2014] extends this to scene understanding. The fitting in Shape2Pose is supported by training data composed of human-object pairs for various object categories. A key difference to our work is that the two methods have different goals: Shape2pose aims at predicting a human pose for a given object, while we aim to predict how the object functions based on a richer variety of interactions, including those between a human and an object, learned from different categories.

On a more conceptual level, how an object functions is not always well reflected by interactions with human poses. For example, consider a drying rack with hanging laundry; there is no human interaction involved. Even if looking only at human poses, one may have a hard time discriminating between certain objects, e.g., a hook and a vase, since a human may hold these objects with a similar pose. Last but not the least, even if an object is designed to be directly used by humans, human poses alone cannot always distinguish its functionality from others. For example, a human can carry a backpack while sitting, standing or walking. The specific pose of the human does not allow us to infer the functionality of the backpack.

Interaction context (ICON). Our functional co-analysis builds on the recent work of Hu et al. [2015] on ICON, where the representation and organization of interactions is similar. The key difference is that ICON explicitly describes the functionality of a single object, given in a scene context, while we learn the functionality of an object category. Furthermore, the ICON descriptor is only applicable to describe the functionality of scenes, not shapes in isolation.

3 Overview

In this section, we provide an overview of our functionality model, the learning scheme, and relevant applications; see Figure 2.

Functionality model (Section 4). Since we target a localized analysis of functionality, we need to learn what areas of a shape prevent or contribute to a specific functionality. Thus, we choose to represent functional properties at the level of patches defined on the surface of shapes. Patches are more general than parts since, for example, the top and bottom of a part may support different functional properties. Thus, different regions of a part can be represented with separate patches. Moreover, we seek to create a model that represents the functionalities of man-made objects that are inferable from their geometric properties. Thus, the functional properties that the patches encode are inferred from object-to-object *interactions* present in the geometric arrangements of objects. Note that our work focuses on *static* interactions such as a human holding a handle or an object resting on top of another. Our model does not capture *dynamic* interactions such as a rotating wheel, nor detects functionality unrelated to geometry, e.g., the screen of a TV.

The model itself consists of a set of abstract patches that we call *proto-patches*, as they represent a *patch prototype*. The proto-patches correspond to the different types of interactions that contribute to a specific functionality. They are represented with properties that encode the geometry of the interactions supported by each patch. The patch properties also include a description of the empty space surrounding the objects that is relevant to functionality, and intra-object geometric properties that capture the global arrangement of patches. The combination of all these properties into a model provides a localized explanation of the geometric and structural features of shape patches that are needed to support the functionality of a shape. The model, which is a collection of proto-patches, then encodes the functionality of a category of shapes.

Learning (Section 5). We learn the functionality model via co-analysis of shapes that belong to the same object category. This process discovers the functionality of shapes in a category, and creates the proto-patches. Each input shape, which we call a central object, is provided within a scene context from where we derive the interactions between the central and other objects. Thus, for each scene, the central object is known a priori. We first analyze the interactions in each scene independently and represent them with features derived from geometric entities such as the interaction bisector surfaces and the interaction regions. Although the input shapes belonging to the same category can vary considerably in their geometry, this choice of features encodes the interactions in a manner that is less sensitive to the specific geometry of the shapes. Next, we perform a co-analysis by deriving the functional patches for each shape from the interaction regions and establishing a correspondence between patches that support the same interactions. Finally, we aggregate the properties of all corresponding patches to create the proto-patches.

Prediction and scoring (Section 6). The basic use of our functionality model is to predict whether an unknown shape supports the functionality of a category. However, unknown shapes often appear in isolation, not interacting with other objects in the context of a scene. Thus, we need to define patches on the unknown shape that correspond to the proto-patches in the model, and simultaneously verify if their properties support the functionality of the model. We perform this using an optimization that simultaneously finds the patches and computes a score of how well the shape and its patches support the model functionality. We call this process *functionality scoring*. This optimization serves as a building block to enable several applications. For example, to support functionality-aware enhancement of 3D shapes, the optimization can be used to detect the patches that need to be modified so that the shape better supports a functionality.

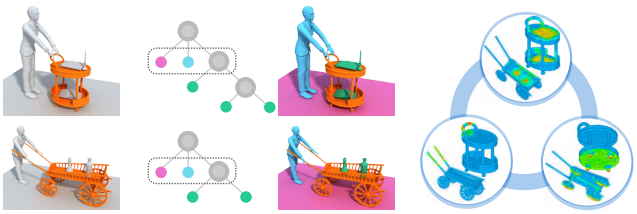


Figure 4: *Learning the functionality model: (a) Given objects in the context of a scene, we compute an ICON hierarchy for each central object (in orange). Only two objects of a larger set are shown. (b) We establish a correspondence between all the hierarchies through a co-analysis, shown with the matching of colors between the hierarchies and the scenes. (c) We collect sets of corresponding functional patches and summarize them with proto-patches. The model is composed of functional properties of the proto-patches and binary properties defined between proto-patches.*

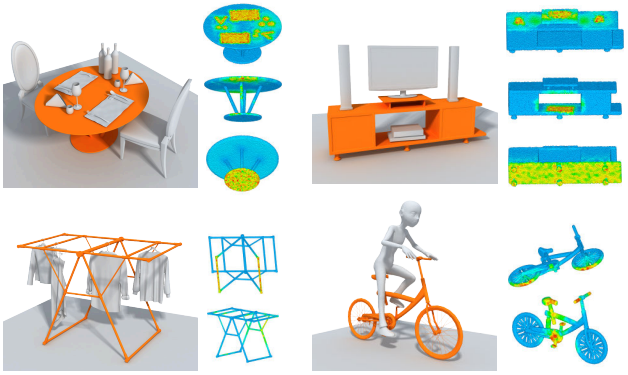


Figure 5: *Examples of functional patches derived from different scenes. Each color map shows the patches corresponding to one of the first-level nodes in the ICON hierarchy of the scene. Note how each node corresponds to patches of the same type of interactions.*

4 Functionality model

Our model can be described as a collection of functional patches originating from the objects in a specific category, as shown in Figure 3. Each object contributes one or more patches to the model, which are clustered together as proto-patches. The model also contains unary properties of the patches, binary properties between patches, and a global set of feature weights that indicate the relevance of each property in describing the category functionality.

More formally, a proto-patch $P_i = \{U_i, S_i\}$ represents a patch prototype that supports a specific type of interaction, and encodes it as distributions of unary properties U_i of the patch, and the functional space S_i surrounding the patch. Our functionality model is denoted as $\mathcal{M} = \{P, B, \Omega\}$, where $P = \{P_i\}$ is a set of proto-patches, $B = \{B_{i,j}\}$ are distributions of binary properties defined between pairs of proto-patches, and Ω is a set of weights indicating the relevance of unary and binary properties in describing the functionality.

We define a set of abstract unary properties $\mathcal{U} = \{u_k\}$, such as the normal direction of a patch, and a set of abstract binary properties $\mathcal{B} = \{b_k\}$, such as the relative orientation of two different patches. We learn the distribution of values for these unary and binary properties for each object in the category. For the i-th proto-patch, $u_{i,k}$ encodes the distribution of the k-th unary property, and for each pair i, j of proto-patches, $b_{i,j,k}$ encodes the distribution of the k-th binary property. The list of unary and binary properties are summarized in Appendix A. Using these properties, the set $U_i = \{u_{i,k}\}$ captures the geometric properties of proto-patch i in terms of the abstract properties $\mathcal{U}$, and similarly the set $B_{i,j} = \{b_{i,j,k}\}$ captures the general arrangement of pairs of proto-patches i and j in terms of the properties in $\mathcal{B}$. Since the functional space is more geometric in nature, S_i is represented as a closed surface. We describe how the distributions and the functional space are derived from the training data in Section 5.

The distributions $u_{i,k}$ and $b_{i,j,k}$ can be used to verify how well the property values of an unknown shape agree with the values of the training shapes. Thus, assuming that we have a reasonable set of functional properties, the analysis of likelihood derived from the individual distributions can be aggregated to infer a score of the functionality of a shape, which is described in Section 6.

Our model stores one proto-patch for each type of interaction that objects can support. The functionality of a category is then described by the collection of all proto-patches and their properties. Although the focus of our work is to describe such a *category functionality*, the localized analysis of functionality based on the proto-patches allows us to define functionality at various granularity levels. For example, as shown in Figure 2, the functionality of the handcart category is captured by three proto-patches. A finer grouping of the proto-patches could lead to two functionalities: the patch in the box body of the handcart enables "storage", while the patches on the shafts and wheels provide "locomotion".

5 Learning the functionality model

Given a set of shapes, we initially describe each scene in the input with an *interaction context* (ICON) descriptor [Hu et al. 2015]. We briefly describe this descriptor here for completeness, and then explain how it is used in our co-analysis and model construction, which are illustrated in Figure 4.

Interaction context. ICON encodes the pairwise interactions between the central object and the remaining objects in a scene. To compute an ICON descriptor, each shape is approximated with a set of sample points. Thus, our method can be applied to scenes with shapes that are not necessarily watertight manifolds. Each interaction is described by features of an interaction bisector surface (IBS) [Zhao et al. 2014] and an interaction region (IR). The IBS is defined as a subset of the Voronoi diagram computed between two objects and represents the spatial region between them. The IR is the region on the surface of the central object that corresponds to the interaction captured by the IBS. The features computed for the IBS and IR capture the geometric properties that describe the interaction between two objects, but in a manner that is less sensitive to the specific geometry of the objects.

All the interactions of a central object are organized in a hierarchy of interactions, called the ICON descriptor. The leaf nodes of the hierarchy represent single interactions, while the intermediate nodes group similar types of interactions together; see Figure 4(b). Each ICON descriptor may have multiple associated hierarchies. Thus, to represent the central object, we select the hierarchy that minimizes the average distance to the hierarchies of all the other central objects in the training set for the given category. The tree distance is derived from the quality of a subtree isomorphism, which is computed between two hierarchies based on the IBS and IR descriptors, similarly as described in the work of Hu et al. [2015].

Co-analysis. The goal of our co-analysis is to cluster together similar interactions that appear in different scenes. Given the

ICONs of all the central objects in the input category, we first establish a correspondence between all the pairs of ICON hierarchies. The correspondence for a pair is derived from the same subtree isomorphism used to compute a tree distance. This correspondence is illustrated in Figure 4(b).

Since we aim for a coherent correspondence between all the interactions in the category, we apply an additional refinement step to ensure coherency. We construct a graph where each vertex corresponds to a central object in the set, and every two objects are connected by an edge whose weight is the distance between their ICON hierarchies. We compute a minimum spanning tree of this graph, and use it to propagate the correspondences across the set. We start with a randomly selected root vertex and establish correspondences between the root and all its children (the vertices connected to the root). Next, we recursively propagate the correspondence to the children in a breadth first manner. In each step, we reuse the correspondence already found with the tree isomorphism. This propagation ensures that cycles of inconsistent correspondences between different ICON hierarchies in the original graph are eliminated. The output of this step is a correspondence between the nodes of all the selected ICON hierarchies of the objects.

Patch definition. We define the functional patches based on the interaction regions of each node on the first level of each ICON hierarchy. Due to the grouping of interactions in ICON descriptors, the first-level nodes correspond to the most fundamental types of interactions, as illustrated in Figure 5. Since a node potentially has multiple children corresponding to several interactions and IRs, we take the union of all the interacting objects corresponding to all the children of the node. Hence, we compute the IR for the interaction between the central object and this union of objects. The IRs computed with ICON are not a binary assignment of points on the surface of the object, but rather a weight assignment for all the object's points, where this weight indicates the importance of the point to the specific IR. When computing the functional properties of the patches, we take these weights into consideration. A functional patch is then described by the point weighting and properties of the corresponding IR.

After defining the functional patches, we can extract their properties. The unary properties are related to the interactions of the patches, while the binary properties are pairwise geometric relations between patches. All of the properties that we use are summarized in Appendix A. Note that each sample point on the shape has a set of point-level properties, and each pair of samples has a set of pairwise properties. Then, the patch-level unary properties are computed as histograms of the point-level properties of all the samples in the patch, where we multiply the weight of a point by its point-level property before using the value to create the histogram. The binary patch-level properties are computed as histograms of the pairwise properties of all the points between a pair of patches.

In addition, we extract the functional space that surrounds each patch. To obtain this space for a patch, we first define the *active scene* of the central object as composed of the object itself and all the interacting objects corresponding to the interaction of the IR of the patch. Then, we first bound the active scene using a sphere. Next, we take the union between the sphere and the central object. Finally, we compute the IBS between this union and all the other interacting objects in the active scene. We use a sphere with diameter $1.2\times$the diagonal of the active scene's axis-aligned bounding box, to avoid splitting the functional space into multiple parts. An example of computing the functional space for the patch corresponding to the interaction between a chair and a human is illustrated in Figure 6. In this case, we consider the chair and the human in the computation, but not the ground. The resulting IBS bounds the functional space of the patch.

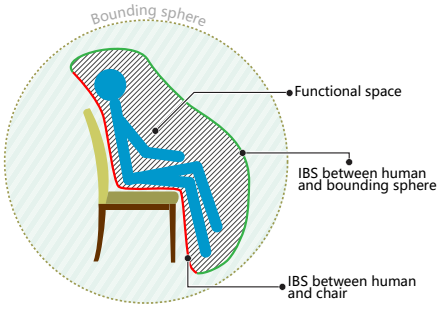


Figure 6: *Computation of the functional space of a chair.*

Model definition. A single proto-patch is defined by a set of patches in correspondence. The distributions $u_{i,k}$ and $b_{i,j,k}$ of the functionality model capture the distribution of the unary and binary properties of proto-patches in the model, respectively. There are different options for representing these distributions. In our case, since the number of training instances is relatively small compared to the dimensionality of the properties, we have opted to represent the distributions simply as sets of training samples. The probability of a new property value is derived from the distance to the nearest neighbor sample in the set. This allows us to obtain more precise estimates in the case of a small training set. If larger training sets were available, the nearest neighbor estimate could be computed with efficient spatial queries, or replaced by more scalable approaches such as regression or density-based approaches.

The functional spaces of all patches in a proto-patch are geometric entities represented as closed surfaces. To derive the functional space S_i of proto-patch i, we take all the corresponding patches and align them together based on principal component analysis. A patch alignment then implies an alignment for the functional spaces, i.e., the spaces are rigidly moved according to the transformation applied to the patches. Finally, we define the functional space of the proto-patch as the intersection of all these aligned spaces.

Learning property weights. Given a set of property weights, we can predict the functional patches and compute the functionality score (defined later in Section 6) for any given shape. However, since different unary and binary properties may be useful for capturing different functionalities, we learn a set of property weights $\Omega_{\mathcal{M}}$ for the model $\mathcal{M}$ of each category. To learn the weights for a model $\mathcal{M}$, we define a metric learning problem: we use our functionality score to rank all the objects in our training set against $\mathcal{M}$, where the training set includes objects from other categories as well. The objective of the learning is then that objects from the model's category should be ranked before objects from other categories. Specifically, let n_1 and n_2 be the number of shapes in the training set that are inside and outside the category of $\mathcal{M}$, respectively. We have $n_1 n_2$ pairwise constraints specifying that the score of a shape inside the category of $\mathcal{M}$ should be smaller than the score of a shape outside. We use these constraints to pose and solve a metric learning problem [Schultz and Joachims 2004; Zhu et al. 2014a].

A challenge is that the score employed to learn the weights is itself a function of the feature weights $\Omega_{\mathcal{M}}$. As defined in Section 6, the score is formulated in terms of distances between predicted patches and proto-patches of $\mathcal{M}$. Due to this reason, we learn the weights in an iterative scheme. In more detail, after obtaining the initial predicted patches for each shape (which does not require weights), we learn the optimal weights by solving the metric learning described above, and then refine the predicted patches with the learned weights by solving a constrained optimization problem. We then

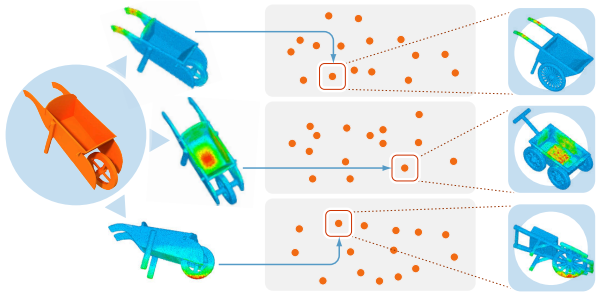


Figure 7: *Prediction of functionality with our model. Given the unknown shape to the left, we locate patches that correspond to proto-patches by finding the most appropriate nearest neighbor patch in the model. We only illustrate the case of unary features here.*

repeat the process with the refined patches until either the functionality score or the weights converge. The details of initial prediction and patch refinement can be found in Section 6. Once we learned the optimal property weights for a model $\mathcal{M}$, they are fixed and used for functionality prediction on any input shape.

6 Functionality prediction

Given a functionality model and an unknown object, we can predict whether the object supports the functionality of the model. More precisely, we can estimate the degree to which the object supports this functionality. To use our model for such a task, we first need to locate patches on the object that correspond to the proto-patches of the model. However, since the object is given in isolation without a scene context from where we could extract the patches, our strategy is to search for the patches that give the best functionality estimation according to the model. Thus, we formulate the problem as an optimization that simultaneously defines the patches and computes their functionality score.

For practical reasons, we will define a functionality distance $\mathcal{D}$ instead of a functionality score. The distance measures how far an object is from satisfying the functionality of a given category model $\mathcal{M}$, and its values are between 0 and 1. The functionality score of a shape can then be simply defined as $\mathcal{F} = 1 - \mathcal{D}$.

Single functionality patch. Let us first look at the case of locating a single patch π_i on the unknown object, so that the patch corresponds to a specific proto-patch P_i of the model. We need to define the spatial extent of π_i on the object and estimate how well the property values of π_i agree with the distributions of P_i. We solve these two tasks with an iterative approach, alternating between the computation of the functionality distance from π_i to P_i, and the refinement of the extent of π_i based on a gradient descent.

We represent an object as a set of n surface sample points. The shape and spatial extent of a patch π_i is encoded as a column vector W_i of dimension n. Each entry $0 \leq W_{p,i} \leq 1$ of this vector indicates how strongly point p belongs to π_i. Thus, in practice, the patches are a probabilistic distribution of their location, rather than discrete sets of points.

Let us assume for now that the spatial extent of a patch π_i is already defined by W_i. To obtain a functionality distance of π_i to the proto-patch P_i, we compute the patch-level properties of π_i and compare them to the properties of P_i. As described in Section 5, we use the nearest neighbor approach for this task. In detail, given a specific abstract property u_k, we compute the corresponding descriptor for patch π_i that is defined by W_i, and denoted $\mathbb{D}_{u_k}(W_i)$. Next, we find its nearest neighbor in distribution $u_{i,k} \in P_i$, denoted $\mathcal{N}(u_{i,k})$. The functionality distance for this property is given by

$$\mathcal{D}_{u_k}(W_i, u_{i,k}) = \|\mathbb{D}_{u_k}(W_i) - \mathcal{N}(u_{i,k})\|_F^2, \tag{1}$$

where $\|\cdot\|_F$ is the Frobenius norm of a vector. This process is illustrated in Figure 7. In practice, we consider multiple nearest neighbors for robustness, implying that the functionality distance is a sum of distances to all nearest neighbors, i.e., we have a term like the right-hand of Eq. 1 for each neighbor. However, to simplify the notation of subsequent formulas, we omit this additional sum.

When considering multiple properties, we assume statistical independence among the properties and formulate the functionality distance for patch W_i as the sum of all property distances:

$$\mathcal{D}_u(W_i, P_i) = \sum_{u_k} \alpha_k^u \, \|\mathbb{D}_{u_k}(W_i) - \mathcal{N}(u_{i,k})\|_F^2, \tag{2}$$

where α_k^u is the weight learned for property u_k in $\Omega_{\mathcal{M}}$, as explained in Section 5. $\mathcal{D}_u(W_i, P_i)$ then measures how close the patch defined by W_i is to supporting interactions like the ones supported by proto-patch P_i.

Now, given the nearest neighbors for patch π_i, we are able to refine the location and extent of the patch defined by W_i by performing a gradient descent of the distance function given by Eq. 2. This process is repeated iteratively similar to an expectation-maximization approach: starting with some initial guess for W_i, we locate its nearest neighbors, compute the functionality distance, and then refine W_i. The iterations stop when the change in the functionality distance is smaller than a given threshold.

Next, we first explain how this formulation can be extended to include multiple patches as well as the binary properties of the model $\mathcal{M}$. Then, we describe how we obtain the initial guess for patches and the refinement.

Multiple patches and binary properties. We represent multiple patches on a shape by a matrix W of dimensions $n \times m$, where m is the number of proto-patches in the model $\mathcal{M}$ of the given category. A column W_i of this matrix represents a single patch π_i as defined above. We formulate the distance measure that considers multiple patches and binary properties between them as:

$$\mathcal{D}(W, \mathcal{M}) = \mathcal{D}_u(W, \mathcal{M}) + \mathcal{D}_b(W, \mathcal{M}), \tag{3}$$

where $\mathcal{D}_u$ and $\mathcal{D}_b$ are distance measures that consider the distributions of unary and binary properties of $\mathcal{M}$, respectively.

We use the functionality distance of a patch defined in Equation 2 to formulate a term that considers the unary properties of all the proto-patches in the model:

$$\begin{aligned} \mathcal{D}_u(W, \mathcal{M}) &= \sum_i \sum_{u_{i,k}} \alpha_k^u \, \mathcal{D}_{u_k}(W_i, u_{i,k}) \\ &= \sum_i \sum_{u_{i,k}} \alpha_k^u \, \|\mathbb{D}_{u_k}(W_i) - \mathcal{N}(u_{i,k})\|_F^2. \end{aligned} \tag{4}$$

The patch-level descriptors for patches are histograms of point-level properties (Appendix A). Since we optimize the objective with an iterative scheme that can change the patches π_i in each iteration, it would appear that we need to recompute the histograms for each patch at every iteration. However, for each sample point on the shape, the properties are immutable. Hence, we decouple the point-level property values from the histogram bins by formulating the patch-level descriptors as $\mathbb{D}_{u_k}(W_i) = \mathbb{B}_k \, W_i$, where

$\mathbb{B}_k \in \{0,1\}^{n_k^u \times n}$ is a constant logical matrix that indicates the bin of each sample point for property u_k. The dimension n_k^u is the number of bins for property u_k, and n is the number of sample points of the shape. $\mathbb{B}_k$ is computed once, based on the point-level properties of each sample. This speeds up the optimization as we do not need to update the matrices $\mathbb{B}_k$ at each iteration, and only update the W_i's, that represent each patch π_i.

The unary distance measure thus can be written in matrix form as

$$\mathcal{D}_u(W, \mathcal{M}) = \sum_{u_k} \alpha_k^u \, \|\mathbb{B}_k W - N_k\|_F^2, \tag{5}$$

where $N_k = [\mathcal{N}(u_{1,k}), \mathcal{N}(u_{2,k}), \ldots, \mathcal{N}(u_{m,k})]$.

Similarly, the binary distance measure can be written as

$$\begin{aligned}
\mathcal{D}_b(W, \mathcal{M}) &= \sum_{i,j} \sum_{b_{i,j,k}} \alpha_k^b \, \mathcal{D}_{b_k}(W_i, W_j, b_{i,j,k}) \\
&= \sum_{i,j} \sum_{b_{i,j,k}} \alpha_k^b \sum_{l=1}^{n_k^b} (W_i^T \mathbb{B}_{k,l}^b \, W_j - \mathcal{N}(b_{i,j,k})_l)^2 \\
&= \sum_{b_k} \sum_{l=1}^{n_k^b} \alpha_k^b \sum_{i,j} (W_i^T \mathbb{B}_{k,l}^b \, W_j - \mathcal{N}(b_{i,j,k})_l)^2 \\
&= \sum_{b_k} \sum_{l=1}^{n_k^b} \alpha_k^b \|W^T \, \mathbb{B}_{k,l}^b \, W - N_{k,l}^b\|_F^2,
\end{aligned} \tag{6}$$

where α_k^b is the weight learned for property b_k in $\Omega_{\mathcal{M}}$, $\mathbb{B}_{k,l}^b \in \{0,1\}^{n \times n}$ is a logical matrix that indicates whether a pair of samples contributes to bin l of the binary descriptor k, n_k^b is the number of bins for property k, and $N_{k,l}^b = [\mathcal{N}(b_{i,j,k})_l; \forall i,j] \in \mathbb{R}^{m \times m}$, where $\mathcal{N}(b_{i,j,k})_l$ is the l-th bin of the histogram $\mathcal{N}(b_{i,j,k})$. Note that both $\mathbb{B}_{k,l}^b$ and $N_{k,l}^b$ are symmetric.

Optimization. To estimate the location of the patches and their scores efficiently, we first compute an initial guess $W^{(0)}$ for the functional patches using the point-level properties only. Then, we find the nearest neighborhoods N_k and $N_{k,l}^b$, and optimize W to minimize $\mathcal{D}(W, \mathcal{M})$ of Eq. 3.

Initial prediction. We use regression to predict the likelihood of any point in a new shape to be part of each proto-patch. In a pre-processing step, we train a random regression forest [Breiman 2001] (using 30 trees) on the point-level properties for each proto-patch P_i. For any given new shape, after computing the properties for the sample points, we can predict the likelihood of each point with respect to each P_i. We set this as the initial $W_i^{(0)}$.

Refinement. Next, we find the nearest neighbors of the predicted patches for every property in the proto-patch, and refine W by performing a gradient descent to optimize Eq. 3. We set two constraints on W to obtain a meaningful solution: $W \geq 0$ and $\|W_i\|_1 = 1$. We employ a limited-memory projected quasi-Newton algorithm (PQN) [Schmidt et al. 2009] to solve this constrained optimization problem since it is efficient for large-scale optimization with simple constraints. To apply PQN, we need the gradient of the objective function, which we derive in Appendix B. Although the gradient can become negative, the optimization uses a projection onto a simplex to ensure that the weights satisfy the constraints [Schmidt et al. 2009; Niu et al. 2015]. The optimization stops when the change in the objective function is smaller than 0.001.

Output. The result of the optimization is a set of patches that are located on the input shape and represented by W. Each patch W_i corresponds to proto-patch P_i in the model. Using these patches, we obtain two types of functionality distance: (i) The *global* functionality distance of the object, that estimates how well the object supports the functionality of the model; and, (ii) The functionality distance of each patch, which is of a *local* nature and quantifies how well W_i supports the interactions that proto-patch P_i supports. This gives an indication of how each portion of the object contributes to the functionality of the whole shape.

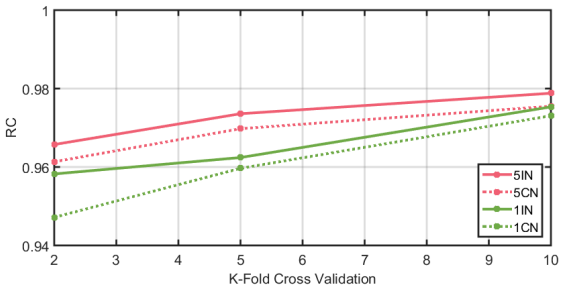


Figure 8: *Evaluation of functionality models learned, in terms of the ranking consistency (RC). Note how the ranking of shapes is well-preserved even when half the dataset is used for training and half for testing, independently of the settings for the selection of nearest neighbors (different curves).*

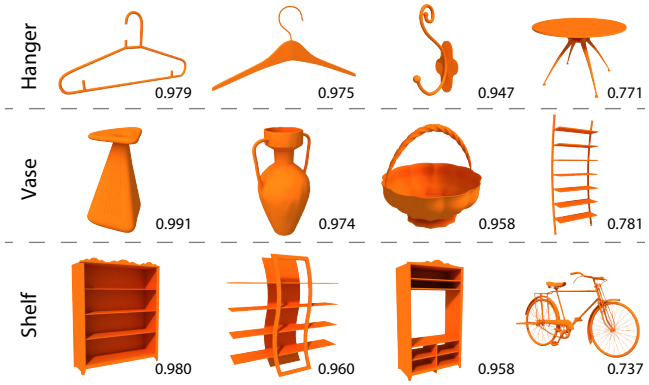


Figure 9: *Functionality scores computed for objects in our dataset. The score of a shape in a given row is predicted with the model of the category written at the beginning of the row.*

7 Results and evaluation

We first evaluate the construction of the functionality model, and then give examples of applications where the model can be used.

Datasets. We test our functionality model on 15 classes of objects, where each class has 10-50 central objects, with 608 objects and their scenes in total. The classes are: Backpack, Basket, Bicycle, Chair, Desk, Drying Rack, Handcart, Hanger, Hook, Shelf, Stand, Stroller, Table, TV Bench, and Vase. The full datasets are shown in the supplementary material. We selected classes that cover a variety of interaction types (1-3 interactions for each central object), and where the interactions of the shapes can be inferred from their geometry. Our dataset contains shapes from the datasets of Kim et al. [2014], Fisher et al. [2015], and Hu et al. [2015], with additional shapes that we collected. We assume the input shapes are consistently upright-oriented both for learning and prediction.

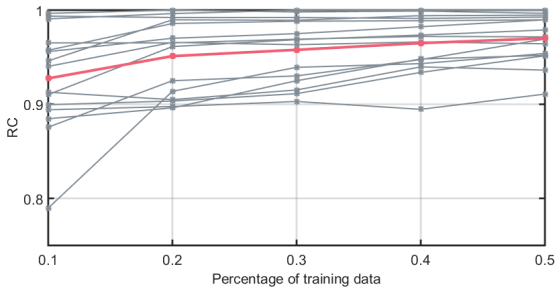


Figure 10: *Effect of the training set size on the ranking consistency (RC). The red line is the average for all classes, while the gray lines are individual classes. Note how, with 20% of the shapes in the dataset, we are already able to obtain a high-quality model.*

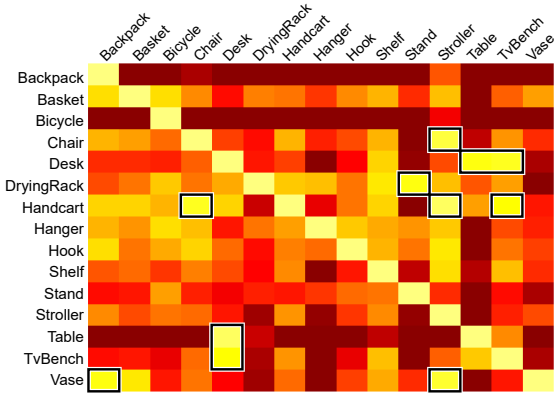


Figure 11: *Amount of correlation between different object categories according to the learned functionality models, where the colormap ranges from red to yellow. Note how the categories with similar functionality have the highest correlation (the outlined cells).*

Evaluation of the functionality model. We evaluate different aspects of the model construction, starting with the accuracy of the learned models. The goal of our optimization is to learn a distance measure from a shape to a category. The learned measure should be low when the functionality of a shape is close to that of the model's category, and high when the functionality is far from that of the category. Thus, a natural way of evaluating the accuracy of the model is verifying how well the distance measure satisfies this consistency requirement. This requirement is equivalent to asking for the preservation of a ranking of shapes ordered by their distance, where the first shapes that appear in the ranking are of the same class as the model. Thus, we evaluate the quality of the ranking in a quantitative manner with the *ranking consistency* (RC):

$$\mathrm{RC}(\mathcal{M}) = \sum_{s_i \in \mathcal{I}} \sum_{s_j \in \mathcal{O}} C(\mathcal{D}(s_i, \mathcal{M}), \mathcal{D}(s_j, \mathcal{M})) \;/\; |\mathcal{I}||\mathcal{O}|, \quad (7)$$

where

$$C(d_i, d_j) = \begin{cases} 1, & \text{if } d_i < d_j, \\ 0, & \text{otherwise,} \end{cases} \quad (8)$$

with $\mathcal{I}$ being the set of test shapes in the same category as the model, $\mathcal{O}$ the set of shapes outside the category, and $D(s_i, \mathcal{M})$ the functional distance of model $\mathcal{M}$ for shape s_i. The RC varies in $[0, 1]$ and measures how often each individual shape in the category of the model is ranked before shapes that are not in the model's category, capturing the global consistency of the ranking.

We compute the RC with different levels of k-fold cross-validation, that is, the dataset is divided into k sets of shapes, with $k - 1$ sets being used for training a model and one set being used to evaluate the RC. This allows us to evaluate the RC in more general settings with different sizes of training and test sets. Note that, for each category of shapes, we train a model using shapes from inside and outside the class, as we also need negative examples to learn the model's weights. Thus, the folds used for training and testing contain models from all the categories.

In Figure 8, we see a plot of the average RC for all classes in our dataset. Note in the graph how, as the size of the training set increases, the quality of the ranking also increases. The RC obtained with $k = 2$ is already over 0.94, implying that high-quality models can be obtained with our optimization when using half of our dataset as a training set, which is in the order of 300 shapes. Figure 9 shows a few qualitative examples of the functionality scores obtained for different shapes inside and outside of a category. We observe that shapes inside a given category, or with a similar functionality, have high scores close to 1, while shapes outside the category have lower scores around 0.7. Note that, since we always optimize the patches to yield the highest functionality score, the scores for unrelated shapes are not zero, but typically cluster around 0.7. Nevertheless, the relative ranking is preserved as the scores for functionally-related shapes are close to 1.

Selection of nearest neighbors. We investigate different ways of using nearest neighbors to access the training data, and study how these choices affect the accuracy of the results. We investigate the use of different numbers of neighbors, specifically, 1 and 5 nearest neighbors, and the use of the neighbors in a coherent and incoherent manner. In the formulation of the unary and binary terms of our objective, given in Equations 5 and 6, we consult the nearest neighbors for each property independently, which corresponds to using the neighbors in an incoherent manner. In the coherent setting, we incorporate an additional constraint in the optimization to ensure that all the properties are consulted from the same selected set of 1 or 5 neighbors, implying that we do not treat the properties independently. We see in Figure 8 that these settings do not have a significant influence on the results, but there is a clear trend where multiple incoherent neighbors give the best results. We also see that the optimization is insensitive to outliers, since the performance is quite stable even when only one nearest neighbor is used.

Training set size. We also investigate in more detail how the size of the training set affects the accuracy of the model, to establish what number of objects is sufficient to learn a satisfactory model. In detail, we analyze how the average RC of a model changes with respect to the training set size, starting with much smaller training sets than in the previous experiment. We explore the use of only 10% to 50% of the shapes in our dataset for training. We compute the RC on a separate test set composed of 10% of the dataset. The result of this experiment as an average for all classes is shown in Figure 10, laid over the results for each class. We observe that a training set using 10% of the shapes in our dataset (in the order of 60 shapes), already has an average RC of over 0.9, implying that this is a sufficient number of shapes for training an accurate model.

Weight learning. When inspecting the weights learned for each property, we observe that the weights are different in each category, demonstrating that some properties are more relevant than others for capturing different functionalities. Moreover, there are no properties with a weight of zero for all classes, both for the unary and binary properties, implying that all the pre-defined properties are useful for a range of different categories. We show a plot of the weight of each property in the supplementary material.

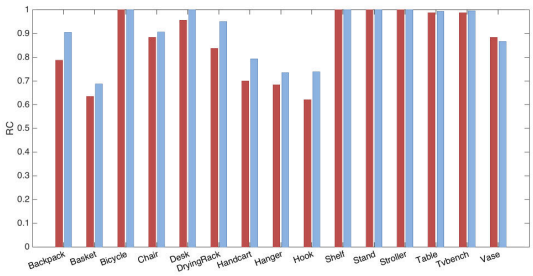


Figure 12: *Results of a user study where the bars show the agreement of our functionality score with the score derived from users. The agreement is computed for each category in terms of the RC.*

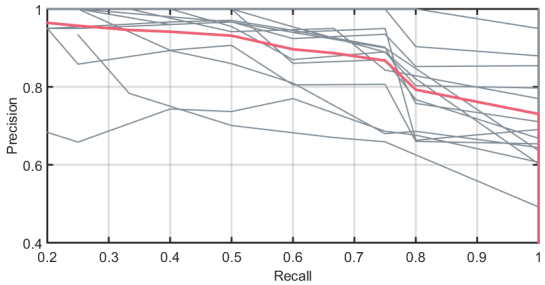


Figure 13: *Object recognition performed with our functionality model. The plot shows the precision-recall of the object rankings given by the models of different classes. The red line is the average for all classes, while the gray lines are individual classes. The closer the lines are to point* $(1, 1)$*, the higher the ranking quality.*

Matching and prediction. We analyzed the matching of interactions and tree isomorphism of ICON descriptors used by our method for training. We observed that they are quite robust in practice, being correct in the large majority of cases. This is the case since we are mainly interested in the correct matching of nodes at higher levels of the hierarchies, which is robustly implied by multiple interactions. Given that different scenes can have different numbers of interacting objects, the matching of individual nodes is not used by our method. Thus, any inaccuracies at the finer level of the matching do not have detrimental effects on the results. Moreover, we also observed that the prediction of patches on shapes with high functionality scores is robust, leading to meaningful predictions.

Learning of *functionality*. One may argue that our method merely learns object categories based on interaction and contextual attributes. However, we remark that our model discovers the functionality of a given category, and separates functional properties of the objects from properties related mainly to the geometry of the shapes. Thus, using our models can reveal similar functionalities in objects from other categories. To demonstrate this claim and assess how well our models discover the functionality of different categories, we start by computing the amount of correlation between the classes according to the functionality distances. The rationale behind this experiment is that categories with similar functionality will be more correlated, with shapes of both classes having a low functionality distance to each category.

We evaluate the correlation in terms of a correlation matrix between all pairs of categories in our dataset. To compute an entry (i, j) of this matrix, we apply the model of class i to predict the score of shapes in class j. Next, we obtain the average distance of all shapes in class j, which provides the closeness of class j to class i in terms of functionality. Figure 11 shows the inverse of the distances for all the classes in our dataset, where larger values (shown in yellow) imply more correlation.

We observe that objects that naturally have a similar function, such as desks, tables and TV benches, or strollers and handcarts, have the most correlation. We also see that objects with totally different functionality, such as desks and hangers, or chairs and stands, have practically no correlation. Perhaps more interestingly, tables and shelves, although being typically composed of flat surfaces, have low correlation, as the interactions involved in the functionality of these shapes are of a different nature.

User study on functional similarity. To demonstrate more conclusively that we discover the functionality of shapes, i.e., the functional aspects of shapes that can be derived from the geometry of their interactions, we conducted a small user study with the goal of verifying the agreement of our model with human perception. Specifically, we verified the agreement of our functionality scores with scores derived from human-given data. In order to do this, we created queries consisting of a central object A appearing in the context of a scene, and a second object B in isolation. We randomly sampled 10% of the shapes from our dataset as objects B, and compare to the 15 categories in our dataset (objects A). Next, we presented a random set of such queries to each study participant. We asked users to rate, in a scale from 1 to 5, how well B substitutes object A in the scene in terms of performing the same function. To reduce any ambiguity in the understanding of the function of object A, we in fact showed four objects from the same category as A in different scenes, to help the users in generalizing the functionality of object A. Example queries are shown in the supplementary material. We collected 60 queries from each user; we had 72 users.

To evaluate the agreement between the user ratings and our scores, we use the RC. Recall that the RC measures the quality of a ranking in terms of pairs, where one shape in the pair is from the same category as the model and the other shape is from outside this category. Thus, for a specific category, we create such pairs of shapes according to our functionality score, where we determine if a shape is inside or outside the category with the category thresholds described below in the application of functional similarity. Then, we use the RC to verify the agreement of the user ratings with the pairs defined by our functionality score.

Figure 12 shows the agreement for each category, where the red bars denote the agreement estimated on all the collected data, while the blue bars denote the agreement after cleaning some of the user data. To remove unreliable data, we compute the standard deviation of the rating given to each shape and category pair by all users, and remove any query responses where the deviation is larger than 1 (since ratings range from 1 to 5). The average RC for all classes is 0.86 and 0.90, before and after filtering, respectively.

We see in the plot that, users agree at least 80% with our model for 12 out of 15 categories. We analyzed the responses for the categories with lower agreement, and conjecture that there are two main reasons for the results. First, users may recognize a common functionality in different types of interactions. For example, users seem to believe that drying racks can also function as hooks. This is reasonable since we can hang clothes on both types of objects. However, the way that clothes are hung on both classes is different (horizontally or on a hook), which leads to two different types of interactions. Moreover, users may also be able to perform partial matching of objects, so that they may believe for example that a table can hold objects just as well as baskets in static scenarios, while in general baskets are more suitable for storing objects when we would like to transport these objects.

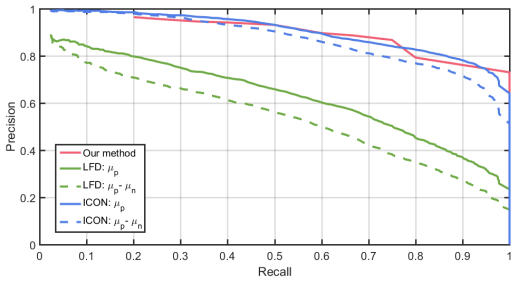


Figure 14: *Comparison of our functionality model to ICON and LFD, in terms of the precision-recall of retrieval simulated on our dataset. Note the better performance of our model and ICON over LFD, where ICON requires an input scene for each shape, while our model predicts the functionality of shapes given in isolation.*

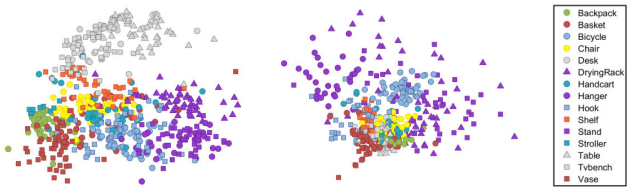
8 Applications

In this section, we demonstrate potential applications enabled by our functionality model. In particular, the spatial localization afforded by our model allows it to be applicable in several modeling tasks, while previous functionality models such as affordance models were designed for discrimination.

Recognition. To use our approach for determining the categories of shapes, we can directly use the ranking of shapes provided by each model to enable a form of shape retrieval, and evaluate the ranking in terms of a precision-recall plot. First, we divide our dataset into training and test sets. Next, we order all the shapes in our test set according to the functionality distance computed with a category model. Finally, given a target number of shapes t, we take the t shapes with the lowest distances and verify how many of them are of the same category as the model, counting the precision and recall of this simulated retrieval.

Figure 13 shows the average precision-recall plots, for this experiment, for all the classes in our dataset, laid over the individual plots for each category. The plots for individual classes are shown with labels in the supplementary material. We see that the accuracy of recognition is high, with a precision of over 0.8 for a recall of up to 0.8 on average. The classes with precision-recall under the average are Desk, Drying Rack, Handcart, and Table, which are some of the classes that have similar functionality to other classes, explaining the lower recognition rates due to class correlation.

We also compare our method with the retrieval provided by the ICON descriptor [Hu et al. 2015] and the lightfield descriptor (LFD) [Chen et al. 2003], which serves as a baseline comparison to our method. Since we perform retrieval with a functionality model learned from a training set, to provide a fair comparison with LFD and ICON, we also use the training sets for retrieval with these descriptors. Specifically, for a given class, we compute the average distance from all the models of this class in the training set to each shape in the test set, which we denote μ_p. Next, we rank the test shapes according to μ_p and measure the precision and recall of the simulated retrieval. We also evaluate an alternative approach where we make use of negative examples of the class, as in the training of our model. We compute the average distance to all the negative examples in the training set, denoted μ_n. Then, we rank the test shapes based on $\mu_p - \mu_n$, to retrieve shapes that are close to the training shapes of the same class but far from the negative examples of the class. Note that, in the case of ICON, we perform this experiment with the scenes provided with the shapes, as this descriptor does not operate on individual objects.



(a) (b)

Figure 15: *Embedding of the shapes in our dataset obtained with multi-dimensional scaling, according to our functionality distance in (a), and the similarity of lightfield descriptors in (b).*

Figure 14 shows the results of this experiment. We observe that our approach and ICON provide better retrieval results than LFD, as these two approaches take into account the interactions between objects. ICON and our model have a similar performance as both approaches represent interactions in a similar manner, although we recall that ICON needs a context scene to be provided for each test shape, while our functionality model predicts the functionality of shapes in isolation.

Functionality similarity. We derive a measure to assess the similarity of the functionality of two objects. Given a functionality model and an unknown object, we can verify how well the object supports the functionality of a category. Intuitively, if two objects support similar types of functionalities, then they should be functionally similar, such as a handcart that supports similar interactions as a stroller. However, the converse is not necessarily true: if two objects do not support a certain functionality, it does not necessarily imply that the objects are functionally similar. For example, the fact that both a table and a backpack cannot be used as a bicycle does not imply that they are functionally similar. Thus, when comparing the functionality of two objects, we should take into consideration only the functionalities that each object likely supports. To perform such a comparison, we decide whether an object supports a certain functionality only if its functionality score, computed with the corresponding model, is above a threshold.

More specifically, since we learn 15 different functionality models based on our dataset, we compute 15 functionality scores for any unknown shape. We concatenate all the scores into a vector of functional similarity $F_S = [f_1^S, f_2^S, ..., f_n^S]$ for shape S, where $n = 15$. We then determine whether the shape supports a given functionality by verifying if the corresponding entry in this vector is above a threshold. We compute the thresholds for each category based on the shapes inside the category using the following procedure. We perform a leave-one-out cross validation, where each shape is left out of the model learning so that we obtain its unbiased functionality score. Next, we compute a histogram of the predicted scores of all the shapes in the category. We then fit a Beta distribution to the histogram and set the threshold t_i, for category i, as the point where the inverse cumulative distribution function value is 0.01.

The functionality distance between two shapes is then defined as

$$\mathscr{D}(S_1, S_2) = \sum_{i=1}^{n} \phi\left(f_i^{S_1}, f_i^{S_2}, t_i\right) \;/\; |J|, \tag{9}$$

where

$$\phi(x, y, t) = \begin{cases} \|x - y\|_2, & \text{if } \max(x, y) \geqslant t, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

The function ϕ considers a functionality only if either S_1 or S_2 supports it, while $J = \{i | \min(f_i^{S_1}, f_i^{S_2}) \geqslant t_i, i = 1, ..., n\}$ is the set of functionalities that are supported by both S_1 and S_2.

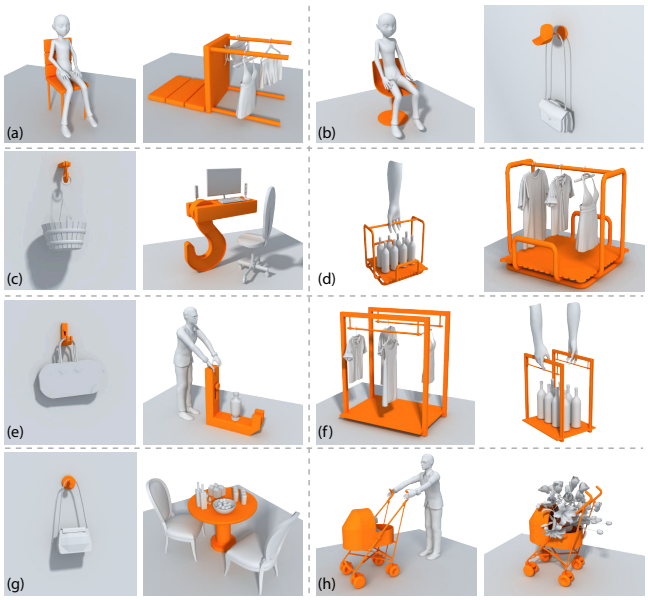


Figure 16: *Scale selection with our functionality model: since objects can perform various functionalities when at different scales, we use our model to select the proper object scale for a scene.*

In Figure 15, we show a 2D embedding of all the shapes in our dataset obtained with multi-dimensional scaling, where the Euclidean distance between two points approximates our functionality distance between two shapes. We compare it to an embedding obtained with the similarity of lightfield descriptors of the shapes. Note how, in our embedding, the shapes are well distributed into separate clusters, while the clusters in the lightfield embedding have significant overlap. Moreover, the overlaps in the embedding of our distance occur mostly for the categories that have functional correlation, as shown before by the correlation matrix in Figure 11.

Detection of multiple functionalities. As shown in Figure 1, a chair may serve multiple functions, depending on its pose. To discover such multiple functionalities for a given object using the functionality models learned from our dataset, we sample various poses of the object. For each functionality model learned of a category, the object pose that achieves the highest functionality score is selected. Moreover, based on patch correspondence inferred from the prediction process, we can also scale the object so that it can replace an object belonging to a different category, in its contextual scene. Figure 16 shows a set of such examples. For each pair, we show on the left the original object in a contextual scene to provide a contrast; the scene context is not used in the prediction process. On the right, we show the scaled object serving a new function. We believe that this type of exploration can potentially inspire users to design objects that serve a desired functionality while having a surprising new geometry.

Functionality enhancement. Given a functionality model $\mathcal{M}$, if a shape has a relatively high functionality score according to the model, but still below the corresponding threshold of the category, we can guide a user in enhancing the shape so that it better supports the functionality. For each predicted functional patch, we find the most similar patch in the dataset, i.e., the nearest neighbor patch. Then, we take the geometry of the nearest neighbor patch and blend it onto the region of the shape corresponding to the predicted patch. Note that such an enhancement is only meaningful when the shape's functionality is close to that of the model, since otherwise the predicted patches may be meaningless. Figure 17 shows an example of this application, where we transfer the geometry of patches of the handcart category to enhance a chair, so that it better serves as a handcart. Note that, in this example, the user manually adjusted the blending of the geometry, while the selection of geometry and patches is automatic.

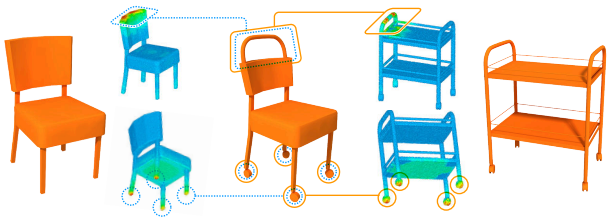


Figure 17: *Functionality enhancement: the chair on the left is enhanced by patch transfer so that it can serve as a handcart.*

Functional hybrids. We can also use the proto-patches to guide a user in creating objects that are *functional hybrids*, preventing the modeling from being an exhaustive trial-and-error process. For example, given a table and shelf, we can guide the creation of a shape that more effectively blends the functionality of these two objects. Given two shapes S_1 and S_2 to hybridize, and models of their corresponding categories, we first detect the functional patches of each shape. Next, we analyze the prediction results to suggest regions of the shapes where the users can add or blend patches to preserve the shape functionalities. We provide two types of hybridizing suggestions: (i) A functional patch of shape S_1 can be attached to regions of S_2 that do not support any functional patch, so that the patches of S_2 are not damaged, and at the same time the functional space of S_1 is not obstructed. (ii) If two patches, one from each shape, serve the same functionality, we can merge them together so that we obtain a single patch on the hybrid shape that serves this functionality.

Figure 18 shows examples of hybrids created with this guidance process. Note how, given two objects, the functionality of both objects is preserved in the hybrid due to the localized guidance offered by the functionality model. The example in cell (a) is created with suggestion type (i), where it is detected that the back of the chair is not involved in any type of interaction, and so we can attach a hook to it without damaging the functionality of the chair. Similarly, example (c) is obtained by blending the shelf to the edge of the table, since when the shelf is attached to that portion of the table, the functionality is affected the least. In contrast, (f) shows a user-provided case that has a low score according to our model, since the functionality of the shelf is obstructed when someone is sitting at the table. In the third column of Figure 1, we see a hybrid obtained with suggestion type (ii), where a vase and table are blended by merging their support patches that interact with the floor.

9 Conclusion, limitation, and future work

In this work, we are mainly concerned with the "where" and "how" of functionality analysis of 3D objects. Beyond obtaining a description of functionality, e.g., [Kim et al. 2014; Hu et al. 2015], which can discriminate between objects, we are interested in learning how a function is performed, by discovering the interactions responsible for the function and the surface patches that contribute to the function. Through co-analysis, we learn a functionality model for a given object category. The learned category functionality models allow us to both infer the functionality of an individual object and perform functionality-aware shape modeling.

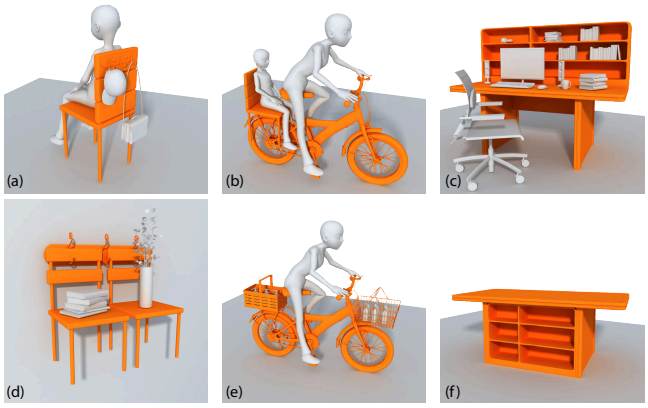


Figure 18: *Functional hybrids created with guidance from our functionality model in (a)-(e). Note how the functionality of the original objects is preserved in the hybrids, in contrast to the user-given configuration in (f).*

Limitations. Our functionality analysis is entirely based on reasoning on the geometry of shapes. As a result, our model could recognize a backpack as a vase, as shown in Figure 19. In this case, perhaps only a touch to feel the material would make the right distinction. Indeed, we learned from users' feedback that sometimes, their judgement on functionality is influenced by a recognition of material. Geometrically speaking, a drying rack as the one shown in Figure 16(f), when properly scaled, can be put on one's back as a backpack. But that functionality is hardly recognized since most people would assume the rack is made of metal.

The "how" in our functionality analysis is limited to extracting information from *static* configurations of object interactions. It would be an entirely new pursuit to understand the "how" by observing and learning from dynamic human-to-object and object-to-object interactions. SceneGrok [2014] is a step towards this direction. Last but not the least, while localizing functionality analyses to the patch level is a strength of our work, we only learn patch-level properties and pairwise relations between patches. This may prevent us from discovering certain global properties related to functionality, e.g., the instability of the hanger in Figure 16(c).

Future work. An interesting future problem is to examine how the proto-patches obtained from our co-analysis can be combined to build functionality models at *varying granularity levels*. For example, each of the rolling, storage, and support functionalities of the handcarts is supported by a combination of proto-patches for that category. These distinctive functionalities, which may help relate objects between different categories, are not studied by our current work. We believe that a study of higher-order, or even *hierarchical*, relations between the proto-patches is worth considering, so is *cross-category* functionality analysis. These pursuits would enrich and strengthen all the functionality-aware analysis and modeling applications we have discussed in this paper.

Acknowledgements

We would like to thank all the reviewers for their comments and suggestions. This work was supported in part by grants from NSFC (61522213, 61528208, 61379090), 973 Program (2014CB360503, 2015CB352501), Guangdong Science and Technology Program (2015A030312015, 2014B050502009, 2014TX01X033), Shenzhen Innovation Program (JCYJ20151015151249564), NSERC (611370, 2015-05407) and ISF-NSFC (2216/15).

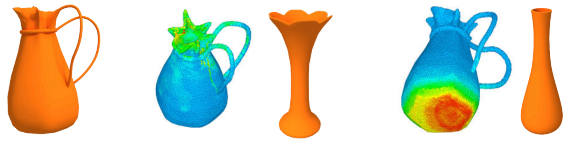


Figure 19: *The backpack (left) is recognized as a vase, as its geometry supports similar interactions as vases, shown with the detected patches and nearest neighbors in the middle and right.*

References

BAR-AVIV, E., AND RIVLIN, E. 2006. Functional 3D object classification using simulation of embodied agent. In *British Machine Vision Conference*, 32:1–10.

BREIMAN, L. 2001. Random forests. *Machine learning 45*, 1, 5–32.

CHEN, D.-Y., TIAN, X.-P., SHEN, Y.-T., AND OUHYOUNG, M. 2003. On visual similarity based 3D model retrieval. *Computer Graphics Forum (Proc. of Eurographics) 22*, 3, 223–232.

FISH, N., AVERKIOU, M., VAN KAICK, O., SORKINE-HORNUNG, O., COHEN-OR, D., AND MITRA, N. J. 2014. Meta-representation of shape families. *ACM Trans. on Graphics 33*, 4, 34:1–11.

FISHER, M., RITCHIE, D., SAVVA, M., FUNKHOUSER, T., AND HANRAHAN, P. 2012. Example-based synthesis of 3D object arrangements. *ACM Trans. on Graphics 31*, 6, 135:1–11.

FISHER, M., LI, Y., SAVVA, M., HANRAHAN, P., AND NIESSNER, M. 2015. Activity-centric scene synthesis for functional 3D scene modeling. *ACM Trans. on Graphics 34*, 6, 212:1–10.

GRABNER, H., GALL, J., AND VAN GOOL, L. 2011. What makes a chair a chair? In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*, 1529–1536.

GREENE, M. R., BALDASSANO, C., BECK, D. M., AND FEI-FEI, L. 2016. Visual scenes are categorized by function. *Journal of Experimental Psychology: General 145*, 1, 82–94.

HU, R., ZHU, C., VAN KAICK, O., LIU, L., SHAMIR, A., AND ZHANG, H. 2015. Interaction context (ICON): Towards a geometric functionality descriptor. *ACM Trans. on Graphics 34*, 4, 83:1–12.

KIM, V. G., CHAUDHURI, S., GUIBAS, L., AND FUNKHOUSER, T. 2014. Shape2Pose: Human-centric shape analysis. *ACM Trans. on Graphics 33*, 4, 120:1–12.

LAGA, H., MORTARA, M., AND SPAGNUOLO, M. 2013. Geometry and context for semantic correspondence and functionality recognition in manmade 3D shapes. *ACM Trans. on Graphics 32*, 5, 150:1–16.

MITRA, N., WAND, M., ZHANG, H., COHEN-OR, D., AND BOKELOH, M. 2013. Structure-aware shape processing. In *Eurographics State-of-the-art Report (STAR)*.

NIU, B., WANG, J., AND WANG, H. 2015. Bacterial-inspired algorithms for solving constrained optimization problems. *Neurocomputing 148*, 54–62.

SAVVA, M., CHANG, A. X., HANRAHAN, P., FISHER, M., AND NIESSNER, M. 2014. SceneGrok: Inferring action maps in 3D environments. *ACM Trans. on Graphics 33*, 6, 212:1–10.

SCHMIDT, M., VAN DEN BERG, E., FRIEDLANDER, M. P., AND MURPHY, K. 2009. Optimizing costly functions with simple constraints: A limited-memory projected quasi-Newton algorithm. In *Proc. Int. Conf. AI and Stat.*, 456–463.

SCHULTZ, M., AND JOACHIMS, T. 2004. Learning a distance metric from relative comparisons. *Advances in neural information processing systems (NIPS)*, 41.

SHI, Y., LONG, P., XU, K., HUANG, H., AND XIONG, Y. 2016. Data-driven contextual modeling for 3D scene understanding. *Computers & Graphics 55*, 55–67.

STARK, L., AND BOWYER, K. 1991. Achieving generalized object recognition through reasoning about association of function to structure. *IEEE Trans. Pattern Analysis & Machine Intelligence 13*, 10, 1097–1104.

STARK, L., AND BOWYER, K. 1996. *Generic Object Recognition Using Form and Function*. World Scientific.

XU, K., MA, R., ZHANG, H., ZHU, C., SHAMIR, A., COHEN-OR, D., AND HUANG, H. 2014. Organizing heterogeneous scene collection through contextual focal points. *ACM Trans. on Graphics 33*, 4, 35:1–12.

YUMER, M. E., CHAUDHURI, S., HODGINS, J. K., AND KARA, L. B. 2015. Semantic shape editing using deformation handles. *ACM Trans. on Graphics 34*, 4, 86:1–12.

ZHAO, X., WANG, H., AND KOMURA, T. 2014. Indexing 3D scenes using the interaction bisector surface. *ACM Trans. on Graphics 33*, 3, 22:1–14.

ZHU, P., HU, Q., ZUO, W., AND YANG, M. 2014. Multi-granularity distance metric learning via neighborhood granule margin maximization. *Information Sciences 282*, 321–331.

ZHU, Y., FATHI, A., AND FEI-FEI, L. 2014. Reasoning about object affordances in a knowledge base representation. In *Proc. Euro. Conf. on Computer Vision*.

A Unary and binary properties

We list here the properties used in the functionality model, where we assume that the input shapes are consistently upright-oriented. Some of the properties are similar to the ones used by Kim et al. [2014] and Hu et al. [2015].

We first describe the point-level unary properties. We take a small geodesic neighborhood of a point and compute the eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ and corresponding eigenvectors μ_i of the neighborhood's covariance matrix. We then define the features:

$$L = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3};\ P = \frac{2(\lambda_2 - \lambda_3)}{\lambda_1 + \lambda_2 + \lambda_3};\ S = \frac{3\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3};$$

which indicate how linear-, planar- and spherical-shaped the neighborhood of the point is. We also use the neighborhood to compute the mean curvature at the point and average mean curvature in the region. In addition, we compute the angle between the normal of the point and the upright direction of the shape, and angles between the covariance axes μ_1 and μ_3 and the upright vector. The projection of the point onto the upright vector provides a height feature. Finally, we collect the distance of the point to the best local reflection plane, and encode the relative position and orientation of the point in relation to the convex hull. For this descriptor, we connect a line segment from the point to the center of the shape's convex hull and record the distance of this segment and the angle of the segment with the upright vector, resulting in a 2D histogram. To capture the functional space, we record the distance from the point to the first intersection of a ray following its normal, and encode this as a 2D histogram according to the distance value and angle between the point's normal and upright vector. The distances are normalized by the bounding box diagonal of the shapes and, if there is no intersection, the distance is set to the maximum value 1.

The patch-level unary properties are then histograms capturing the distribution of the point-level properties in a patch, as explained in Section 5. We use histograms composed of 10 bins, and $10 \times 10 = 100$ bins specifically for 2D histograms.

For the binary properties, we define two properties at the point-level: the relative orientation and relative position between pairs of points. For the orientation, we compute the angle between the normal of two points. For the position, we compute the length and the angle between the line segment defined between two points and the upright vector of the shape. The patch-level properties derived for two patches i and j are then 1D and 2D histograms, with 10 and $10 \times 10 = 100$ bins, respectively.

B Gradient for optimization of the objective

We derive the gradient of our objective function here, which we need for the optimization with PQN. The unary distance measure can be re-expressed as the following smooth function:

$$\begin{aligned}\mathcal{D}_u(W, \mathcal{M}) &= \sum_{u_k} \omega_k^u \left\| \mathbb{B}_k W - N_k \right\|_F^2 \\ &= \sum_{u_k} \omega_k^u \operatorname{tr}\big((\mathbb{B}_k W - N_k)^T (\mathbb{B}_k W - N_k)\big) \\ &= \sum_{u_k} \omega_k^u \big(\operatorname{tr}(W^T \mathbb{B}_k^T \mathbb{B}_k W) \\ &\quad - 2\operatorname{tr}(W^T \mathbb{B}_k^T N_k) + \operatorname{tr}(N_k^T N_k)\big).\end{aligned} \tag{11}$$

The gradient of the unary term is then given by:

$$\bigtriangledown_W \mathcal{D}_u(W, \mathcal{M}) = 2 \sum_{u_k} \omega_k^u \, \mathbb{B}_k^T (\mathbb{B}_k W - N_k). \tag{12}$$

When considering multiple neighbors N_k, we simply sum the gradient for each neighbor, due to the additive property of gradients.

Similarly, the binary distance measure can be re-expressed as the following smooth function:

$$\begin{aligned}\mathcal{D}_b(W, \mathcal{M}) &= \sum_{b_k} \sum_{l=1}^{n_k^b} \omega_k^b \left\| W^T \mathbb{B}_{k,l}^b W - N_{k,l}^b \right\|_F^2 \\ &= \sum_{b_k} \sum_{l=1}^{n_k^b} \omega_k^b \operatorname{tr}\big(W^T (\mathbb{B}_{k,l}^b)^T W W^T \mathbb{B}_{k,l}^b W \\ &\quad - 2W^T (\mathbb{B}_{k,l}^b)^T W N_{k,l}^b + (N_{k,l}^b)^T N_{k,l}^b\big).\end{aligned} \tag{13}$$

Since both $\mathbb{B}_{k,l}^b$ and $N_{k,l}^b$ are symmetric, the gradient of the binary term is then given by:

$$\bigtriangledown_W \mathcal{D}_b(W, \mathcal{M}) = 4 \sum_{b_k} \sum_{l=1}^{n_k^b} \omega_k^b \, \mathbb{B}_{k,l}^b W (W^T \mathbb{B}_{k,l}^b W - N_{k,l}^b). \tag{14}$$