

# Parallel Beam Search for Functionality Partial Matching

Yanran Guan  
School of Computer Science  
Carleton University  
Ottawa, Canada K1S 5B6  
yanran.guan@carleton.ca

December 16, 2020

## Abstract

In the context of 3D shape modeling, functionality partial matching refers to the process of localizing the functionality analysis of 3D shapes from the category/object level down to the patch/part level, which can help today’s cross-category shape modeling systems that generate multi-functional shapes to evaluate the functionality of the generated shapes. In this work, we study a parallel beam search algorithm that enables functionality partial matching, based on the idea of level synchronization. The results show that, the parallel beam search can largely speed up the process of functionality partial matching as compared to a sequential beam search.

## 1 Introduction

Recently, there has been increasing interest in studying 3D shapes from a functional perspective [7]. However, the existing studies about functionality analysis for 3D shapes are primarily targeted at discriminating different types of functionalities or interactions. Thus, it is difficult to apply them without significant modifications to analyze hybrid shapes, which can be created by cross-category shape modeling systems [25, 3, 6] and may carry functionalities from different functional categories<sup>1</sup>. Therefore, the notion of functionality partial matching [5, 6] is proposed that enables the localization of functionality analysis of 3D shapes from the category/object level down to the patch/part level. Specifically, functionality partial matching refers to the search for the subset of parts, i.e., partial shape, from a hybrid shape that best fits to one of the functional categories provided by a pre-learned set of category functionality models. In this work, we use the functionality models developed by Hu et al. [8] that learn the functionality of objects per category.

In functionality partial matching, a shape is considered as a connected graph with each node denoting a part of the shape and each edge representing the connection between two parts. Thus, functional partial matching can be seen as a graph/tree traversal process that finds out all functionally plausible combinations of parts within a shape. Recently, a functionality partial matching method was proposed based on the beam search [5]. Beam search is a generalized breadth-first search (BFS) that only expands its search tree at the most “promising” nodes at each depth level, using a heuristic function. However, the current

---

<sup>1</sup>In this work, we define *functional category* or simply *category* as the object category that groups objects serving the same functionality, e.g., all chairs belong to the functional category *chair*, while all desks belong to the functional category *desk*.

implementation of functionality partial matching uses a sequential beam search which is unoptimized and time-consuming. Thus, based on the idea of level synchronization [24, 1, 12], similar to a parallel BFS, we develop a parallel beam search to improve the running time of functionality partial matching.

## 2 Literature Review

In this section, we review the previous research that is related to our work, including the recent work that analyzes the functionality of 3D shapes, the beam search algorithm and its applications, and the studies about parallel BFS.

### 2.1 Functionality Analysis in 3D Shape Modeling

Analyzing the functionality of 3D shapes [7] is a widely studied area in 3D shape modeling. Existing works on this topic have all focused on characterizing, comparing, or categorizing 3D objects based on their functions, which are typically inferred from their geometry and interactions with other objects or agents [11, 18, 10, 8, 17, 9]. In this work, we utilize the category functionality models developed by Hu et al. [8], of which each model computes a score for an input shape that describes how well the shape satisfies the functionality of a category. Furthermore, we localize the model to shape parts/part groups instead of directly applying it to an entire object, so as to enable functionality partial matching.

### 2.2 Beam Search and Its Applications

Beam search is a heuristic graph search algorithm and was first used in the HARP speech recognition system by Bruce Lowerre [14]. The search tree of beam search is expanded in a breadth-first manner but it uses a heuristic function to select the nodes to be expanded at each level of its search tree. The set of selected nodes at each depth level of the search tree is called the beam. Due to its tractability and its efficiency in terms of both time and space complexity, beam search is usually applied in the case where the solution space of the graph is relatively large, e.g., in speech recognition systems [14, 4] and machine translation systems [21]. Recently, the sequence-to-sequence model in natural language processing [20] also uses a beam search for decoding. In our work, we use a heuristic function that cuts off the nodes with low scores at each step of tree expansion and preserves the most promising high score nodes.

### 2.3 Parallel BFS Algorithms

The BFS is a graph search algorithm that explores the nodes of a graph level by level, where the level is defined as the set of leaf nodes that have the same depth to the root node of the search tree. Researches about parallel BFS algorithms cover its implementations for different architectures, e.g., for multi-cores [24, 1, 19, 12], for GPUs [13, 16, 22], and for distributed systems [15, 23]. One of the key ideas of parallelizing a BFS is based on level synchronization [24, 1, 12]. Specifically, level synchronization makes sure that at each iteration only nodes having the same depth level are expanded to the next level. Moreover, the parallelism of a BFS can be further improved by using a multi-set data structure to replace the queue in traditional BFS algorithms [12]. Our method is based on

the beam search, which can be seen as a special form of BFS. We also follow the idea of level synchronization to parallelize a beam search.

### 3 Our Method

In this section, we introduce our solution to parallel beam search for functionality partial matching, including the input to our method, the functional plausibility modeling method, and the functionality partial matching method.

#### 3.1 Input Shapes

The input data to our method is a set of 3D hybrid shapes generated by a shape evolution framework [6]. We briefly introduce the representation of the hybrid shapes and the evolution process.

##### 3.1.1 Shape Representation

The input shapes are given as triangle meshes and we also represent them as sets of points uniformly sampled over the shape surfaces, to ensure that the analysis is not affected by the non-uniformity of the tessellations. Each shape comes with a fine-grained segmentation into meaningful parts. We use the point sets to detect contact points between adjacent parts of each shape, which indicate how the parts connect to each other. There are three types of contact points: (i) contact points that connect four corners of a part; (ii) contact points that connect two extremities of a part; (iii) single contact points. These types of contact points cover the majority of part connections and allow us to appropriately position parts by defining automatic connection rules. Given all the contact points, a shape is abstracted as a relation graph, where each part corresponds to a node in the graph, and two nodes are connected by an edge if the two parts possess at least one contact point in common. The graphs capture the part-wise connectivity and the structure of the shapes.

##### 3.1.2 Shape Evolution

The shape evolution is implemented by the evolutionary operation that operates at the part group<sup>2</sup> level. There are two different types of evolutionary operations called *part group exchange* and *part group insertion*. Given two part groups  $g_A$  and  $g_B$  anchored on shapes  $\mathcal{S}_A$  and  $\mathcal{S}_B$ , a part group exchange results in two possible offspring shapes. In one offspring,  $g_A$  is replaced by  $g_B$  on shape  $\mathcal{S}_A$ . In the other,  $g_B$  is replaced by  $g_A$  on shape  $\mathcal{S}_B$ . A part group insertion can be seen as a special case of part group exchange, where  $g_A$  (or  $g_B$ ) is the null set, so that one of the offspring shapes would be  $\mathcal{S}_B$  with  $g_B$  deleted and the other is the result of inserting  $g_B$  into  $\mathcal{S}_A$ . With this framework based on the evolutionary operations, we can produce a large amount of hybrid shapes through generations of evolution.

#### 3.2 Functional Plausibility Modeling

We use the functionality modeling method of Hu et al. [8], which comprises category functionality models of 15 functional categories, as the backbone method for our functional

---

<sup>2</sup>A part group consists of one or more parts from a shape in the population and represents a partial shape taken from the original shape.

plausibility modeling. Each category functionality model computes a category score for an input shape represented by point cloud, which describes how well a shape satisfies the functionality of the category. To make scores derived from different category functionality models comparable with each other, we apply a score normalization based on the training data to the category scores. More details about the category functionality model can be found in Appendix A.

Apart from the category functionality model, we also include the shape validity verification in our method, which includes the verifications for part-wise connectivity, physical stability, and functional space. Specifically, for part-wise connectivity, we check if the relation graph that represents the shape, as described in Section 3.1.1, has isolated subgraphs or not. For physical stability, we approximate the stability of a shape by verifying whether the center of mass of the partial shape falls within the convex polygon formed by all the ground touching points of the shape, similar to the idea of static stability of shapes [2]. And lastly, for functional space, we require that the shape to have enough space to support the functionality of a functional category. For example, the seat of a chair, which provides the *sitting* functionality for the whole shape, should not be blocked by other parts of the chair. We use the functional space extracted from the models of Hu et al. [8] to verify if a shape has adequate space to perform a functionality.

### 3.3 Functionality Partial Matching

Given a hybrid shape evolved from parent shapes belongs to different functional categories, it may not fit well a single functional category and cannot be modeled by category functionality models. We stipulate that a cross-category shape is functionally plausible as long as it partially supports the functionality of one or multiple functional categories. Thus, we arrive at the notion of functionality partial matching, where we derive the functionality score of a hybrid shape by aggregating the scores of partial matches of parts of the shape to different functional categories.

#### 3.3.1 Beam Search for Functionality Partial Matching

We use a beam search to efficiently find the subset of parts with the highest score. The search tree of beam search is built in a breadth-first manner. It starts the search at the root node, explores all of the successor nodes of the current node at the present depth prior to moving on to the nodes at the next depth level, and sorts the successor nodes in increasing order of heuristic score. Unlike a common breadth-first search, the search tree of beam search stores only a predetermined number of best successor nodes with highest heuristic scores at each depth level, and expands only to the next level from the best successor nodes. In the context of functionality partial matching, starting from the entire hybrid shape, the beam search expands its search tree by removing one part at a time from the current shape, until it converges to an optimal partial match. In Figure 1, we show an example of beam search that converges to a chair. More visualized examples can be found in Appendix B.

#### 3.3.2 Parallelization of Beam Search

To implement parallelism for the beam search, we also adopt the idea of level synchronization, which means that the threads must expand all nodes in one level before expanding any nodes in the next level. Thus, to enable the parallel beam search, we remove parts from

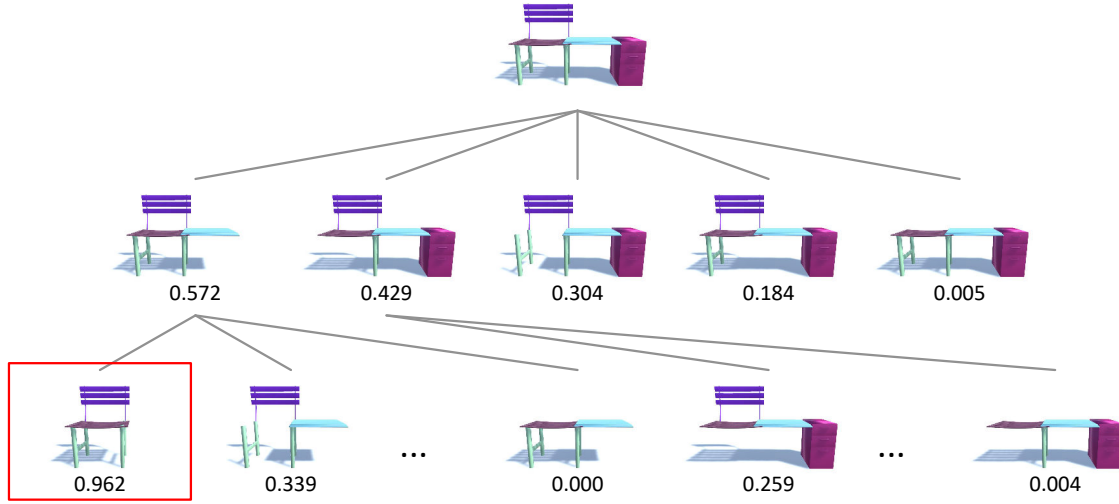


Figure 1: An example of functionality partial matching using beam search, where we search for the subset of parts that provides the highest score for the functional category *chair*. We start with the full shape and enumerate its subsets by removing parts one at a time. We only expand a node if it is promising, i.e., it has one of the top two functionality scores, as the beam width is 2. The node from level 2 highlighted in red is returned as the best partial match, since its children (omitted from the figure) have lower scores.

the shapes in parallel when processing partial shapes in each beam, but update the beam in a sequential manner after we finish processing a beam.

As shown in Algorithm 1, we define the parallel beam search for the partial match that fits the functional category  $f$  as a function taking three input arguments, which is written as  $\text{ParallelBeamSearch}(\mathcal{S}, f, w)$ , where  $\mathcal{S}$  is the entire shape,  $f$  refers to the functional category to be matched, and  $w$  refers to the beam width, which is an integer value denoting the predetermined number of best successor nodes at each expansion of depth level of the search tree. In the algorithm, **Normalize** refers to the process of score normalization based on the training data, as introduced in Section 3.2. Finally, the beam search returns the score  $s$  as the functionality partial matching score.

Prior to starting the beam search, we check if the functional category  $f$  has enough functional space in  $\mathcal{S}$ . If not, the score is returned as 0. We then compute the category score of  $f$  for the entire shape  $\mathcal{S}$ . If the normalized score exceeds a threshold  $\theta = 0.9$ , we directly return it as the final functionality partial matching score, as we find through experiments that shapes with a category score higher than 0.9 already support well the functionality of the concerned functional category.

During the beam search for the best partial shape matching functional category  $f$ , we use three node lists to manage the search process, namely, the **beam** list, the **closed** list, and the **open** list. The **beam** list stores the part groups at each level of the search tree with highest category scores for  $f$ , i.e., the nodes to be expanded to the next depth level of the search tree, **closed** stores the part groups that have been visited, and **open** stores the part groups of the current level of the search tree in decreasing order of scores. Partial shapes with disconnected parts that arise during the search are not considered further in the search. Shapes that are physically unstable are still considered in the search, as such instability

---

**Algorithm 1:** Parallel beam search

---

**Input:**  $S, f, w$   
**Output:**  $s$

```
1 function ParallelBeamSearch( $S, f, w$ )
2   if  $S$  does not have functional space for  $f$  then return 0
3    $s \leftarrow$  compute the category score of  $f$  for  $S$ 
4    $s \leftarrow \text{Normalize}(s)$ 
5   if  $s \geq 0.9$  then return  $s$ 
6   beam  $\leftarrow \{S\}$ 
7   closed  $\leftarrow \{S\}$ 
8   while beam  $\neq \emptyset$  do
9     open  $\leftarrow \emptyset$ 
10    foreach  $g \in$  beam do in parallel
11      foreach  $p \in$  all parts of  $S$  do in parallel
12        successor  $\leftarrow$  remove  $p$  from  $g$ 
13        if successor  $\in$  closed then continue
14        if successor is not part-wise connected then continue
15         $s' \leftarrow$  compute the category score of  $f$  for successor
16        if successor is not physically stable then
17           $s' \leftarrow s' - 0.1$ 
18        open  $\leftarrow$  open  $\cup \{\text{successor}\}$ 
19        closed  $\leftarrow$  closed  $\cup \{\text{successor}\}$ 
20        if  $s < \text{Normalize}(s')$  then
21           $s \leftarrow \text{Normalize}(s')$ 
22        if  $s \geq 0.9$  then return  $s$ 
23    beam  $\leftarrow \emptyset$ 
24    while open  $\neq \emptyset$  and  $|\text{beam}| < w$  do
25       $g \leftarrow$  part group in open with highest score of  $f$ 
26      beam  $\leftarrow$  beam  $\cup \{g\}$ 
27      open  $\leftarrow$  open  $\setminus \{g\}$ 
28  return  $s$ 
```

---

can be temporary. However, for each partial shape that is not physically stable, we reduce its score to ensure that the stable shapes appear at the front of the **open** list. Also, note that in a parallel beam search, we use thread-safe collections to implement the **closed** and **open** lists, as we are accessing them in the parallel loops. In this implementation, we use concurrent dictionaries for implementing the two lists. But for **beam**, we use a normal list, as it is updated sequentially. We use the same threshold score  $\theta = 0.9$  to determine the termination of the beam search, i.e., as soon as a partial shape with normalized score of the functional category  $f$  higher than  $\theta$  is found, the beam search for  $f$  is stopped. If no partial shape that has a score higher than  $\theta$  can be found during the beam search, the beam search proceeds until it cannot be further expanded.

### 3.3.3 Complexity Analysis

The time and space consumption of the beam search algorithm is dependent on the heuristic function and the beam width, where an inaccurate heuristic function may lead the expansion of the search tree to undesired nodes. In the worst case, a beam search can be led all the


























Shape	Combinatorial search		Parallel beam search	
	 $s_{\text{chair}} = 0.96$	 $s_{\text{desk}} = 0.98$	 $s_{\text{chair}} = 0.96$	 $s_{\text{desk}} = 0.98$
	 $s_{\text{chair}} = 0.96$	 $s_{\text{shelf}} = 0.94$	 $s_{\text{chair}} = 0.96$	 $s_{\text{shelf}} = 0.93$
	 $s_{\text{chair}} = 0.93$	 $s_{\text{cart}} = 0.98$	 $s_{\text{chair}} = 0.90$	 $s_{\text{cart}} = 0.98$
	 $s_{\text{desk}} = 0.98$	 $s_{\text{shelf}} = 0.96$	 $s_{\text{desk}} = 0.98$	 $s_{\text{shelf}} = 0.96$
	 $s_{\text{cart}} = 0.96$	 $s_{\text{shelf}} = 0.97$	 $s_{\text{cart}} = 0.92$	 $s_{\text{shelf}} = 0.96$

Table 1: Comparison of the partial matches found by the combinatorial search and the parallel beam search, with  $w = 2$ .

way to the deepest level of the search tree, which results in a computational complexity of  $O(w\delta_{\max})$  for the number of tree expansions, with  $\delta_{\max}$  being the maximum depth of the search tree. Considering that the beam search for a shape with  $n$  parts can expand to at most  $n$  nodes at each step of tree expansion, the upper bound of the complexity for the number of nodes is  $O(wn)$  for each level and  $O(wn\delta_{\max})$  for the entire search for both time and space. In a parallel beam search where  $m$  threads are working simultaneously, the upper bound of the complexity at each level can be reduced to  $O(\frac{wn}{m})$ . Thus, for a parallel beam search, if  $m < wn$ , the upper bound of its complexity is  $O(\frac{wn\delta_{\max}}{m})$ , and if  $m \geq wn$ , the upper bound of its complexity is  $O(\delta_{\max})$ .

## 4 Results and Evaluation

In this section, we discuss the results and evaluation of the functionality partial matching method based on parallel beam search. We first show the accuracy of the partial shapes found by our functionality partial matching, then we evaluate the performance and scalability of the parallel beam search.

### 4.1 Accuracy of Partial Matches

To evaluate the accuracy of our functionality partial matching, we compare the partial matches found by the parallel beam search to those found by a combinatorial search. Because the combinatorial search is a brute-force search that takes all the possible part combinations of a shape into consideration, we can see the partial shapes found by it as ground truth matches.

We show this comparison in Table 1. Each of the five hybrid shapes shown in the table is evolved from two parent shapes, so we show partial shapes that fit the two parent categories. We use a parallel beam search with a beam width of 2, as 2 is the experimentally best beam width for functionality partial matching, providing a trade-off between the analytical accuracy and the execution time [5]. We can see from this comparison that the beam search sometimes may not find the best partial matches as found by the combinatorial search, such as the shelf found for the second shape, the chair found for the third shape, and the cart and the shelf found for the fifth shape. However, we consider such inaccuracy to be

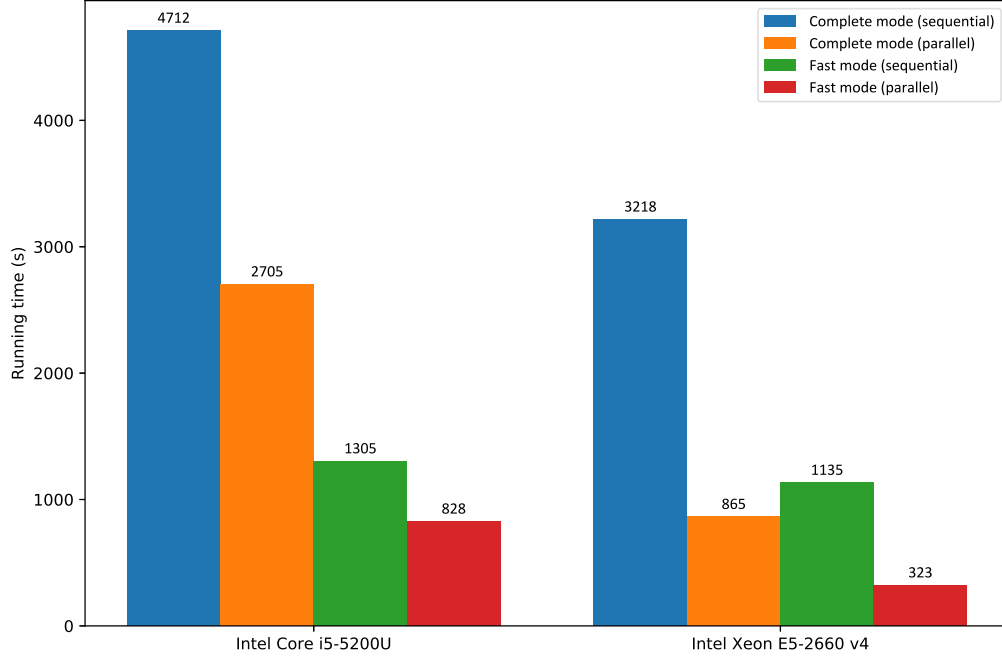


Figure 2: Comparison of the average running time of the parallel beam search and sequential beam search.

acceptable, as the form of these partial shapes can still support the functionalities of the matched categories.

## 4.2 Performance and Scalability

We evaluate the performance of our parallel beam search by comparing its average running time to a sequential beam search, based on the five hybrid shapes shown in Table 1. We show this comparison in Figure 2. We use two different processors, an Intel Core i5-5200U dual-core 4-thread processor and an Intel Xeon E5-2660 v4 14-core 28-thread processor. We test our implementation in two modes: a fast mode which means we stop the search based on the threshold score 0.9 and a complete mode which means the search tree is completely expanded until the beam is empty. We see speedups of 1.58 and 1.74 for the parallel beam search in respectively the fast mode and complete mode using Intel Core i5-5200U. And we see speedups of 3.51 and 3.72 for the parallel beam search in respectively the fast mode and complete mode using Intel Xeon E5-2660 v4.

## 5 Conclusion

In this paper, we describe a parallel beam search algorithm that is used to enable functionality partial matching. We show that, with parallelization implemented for beam search, the process of functionality partial matching can be largely sped up. However, the current implementation of the category functionality model of Hu et al. [8], where the computational cost mainly lies in the calculation of feature distances between patches on point clouds (see Appendix A for more details), still remains unoptimized. So, in future



work, we can work on improving the implementation of the category functionality model to further mitigate the running time of functionality partial matching.

## References

- [1] David A. Bader and Kamesh Madduri. Designing multithreaded algorithms for breadth-first search and *st*-connectivity on the Cray MTA-2. In *Proceedings of the International Conference on Parallel Processing*, pages 523–530, 2006.
- [2] Hongbo Fu, Daniel Cohen-Or, Gideon Dror, and Alla Sheffer. Upright orientation of man-made objects. *ACM Transactions on Graphics*, 27(3):42:1–42:7, 2008.
- [3] Qiang Fu, Xiaowu Chen, Xiaoyu Su, and Hongbo Fu. Pose-inspired shape synthesis and functional hybrid. *IEEE Transactions on Visualization and Computer Graphics*, 23(12):2574–2585, 2017.
- [4] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 6645–6649, 2013.
- [5] Yanran Guan. 3D functionality analysis for shape modeling via partial matching. Master’s thesis, Carleton University, 2019.
- [6] Yanran Guan, Han Liu, Kun Liu, Kangxue Yin, Ruizhen Hu, Oliver van Kaick, Yan Zhang, Ersin Yumer, Nathan Carr, Radomir Mech, and Hao Zhang. FAME: 3D shape generation via functionality-aware model evolution. *IEEE Transactions on Visualization and Computer Graphics*, early access.
- [7] Ruizhen Hu, Manolis Savva, and Oliver van Kaick. Functionality representations and applications for shape analysis. *Computer Graphics Forum*, 37(2):603–624, 2018.
- [8] Ruizhen Hu, Oliver van Kaick, Bojian Wu, Hui Huang, Ariel Shamir, and Hao Zhang. Learning how objects function via co-analysis of interactions. *ACM Transactions on Graphics*, 35(4):47:1–47:12, 2016.
- [9] Ruizhen Hu, Zihao Yan, Jingwen Zhang, Oliver van Kaick, Ariel Shamir, Hao Zhang, and Hui Huang. Predictive and generative neural networks for object functionality. *ACM Transactions on Graphics*, 37(4):151:1–151:13, 2018.
- [10] Ruizhen Hu, Chenyang Zhu, Oliver van Kaick, Ligang Liu, Ariel Shamir, and Hao Zhang. Interaction context (ICON): Towards a geometric functionality descriptor. *ACM Transactions on Graphics*, 34(4):83:1–83:12, 2015.
- [11] Vladimir G. Kim, Siddhartha Chaudhuri, Leonidas Guibas, and Thomas Funkhouser. Shape2Pose: Human-centric shape analysis. *ACM Transactions on Graphics*, 33(4):120:1–120:12, 2014.
- [12] Charles E. Leiserson and Tao B. Schardl. A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). In *Proceedings of the ACM Symposium on Parallelism in algorithms and architectures*, pages 303–314, 2010.

- [13] Gu Liu, Hong An, Wenting Han, Xiaoqiang Li, Tao Sun, Wei Zhou, Xuechao Wei, and Xulong Tang. FlexBFS: A parallelism-aware implementation of breadth-first search on GPU. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 279–280, 2012.
- [14] Bruce T. Lowerre. *The HARPY Speech Recognition System*. PhD thesis, Carnegie Mellon University, 1976.
- [15] Huiwei Lu, Guangming Tan, Mingyu Chen, and Ninghui Sun. Reducing communication in parallel breadth-first search on distributed memory systems. In *Proceedings of the IEEE International Conference on Computational Science and Engineering*, pages 1261–1268, 2014.
- [16] Enrico Mastrostefano and Massimo Bernaschi. Efficient breadth first search on multi-GPU systems. *Journal of Parallel and Distributed Computing*, 73(9):1292–1305, 2013.
- [17] Sören Pirk, Vojtech Krs, Kaimo Hu, Suren Deepak Rajasekaran, Hao Kang, Yusuke Yoshiyasu, Bedrich Benes, and Leonidas J. Guibas. Understanding and exploiting object interaction landscapes. *ACM Transactions on Graphics*, 36(3):31:1–31:14, 2017.
- [18] Manolis Savva, Angel X. Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. SceneGrok: Inferring action maps in 3D environments. *ACM Transactions on Graphics*, 33(6):212:1–212:10, 2014.
- [19] Daniele Paolo Scarpazza, Oreste Villa, and Fabrizio Petrini. Efficient breadth-first search on the Cell/BE processor. *IEEE Transactions on Parallel and Distributed Systems*, 19(10):1381–1395, 2008.
- [20] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.
- [21] Christoph Tillmann and Hermann Ney. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational linguistics*, 29(1):97–133, 2003.
- [22] Dominik Tödling, Martin Winter, and Markus Steinberger. Breadth-first search on dynamic graphs using dynamic parallelism on the GPU. In *Proceedings of the IEEE High Performance Extreme Computing Conference*, pages 1–7, 2019.
- [23] Koji Ueno, Toyotaro Suzumura, Naoya Maruyama, Katsuki Fujisawa, and Satoshi Matsuoka. Efficient breadth-first search on massively parallel and distributed-memory machines. *Data Science and Engineering*, 2(1):22–35, 2017.
- [24] Yang Zhang and Eric A. Hansen. Parallel breadth-first heuristic search on a shared-memory architecture. In *AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*, 2006.
- [25] Youyi Zheng, Daniel Cohen-Or, and Niloy J. Mitra. Smart variations: Functional substructures for part compatibility. *Computer Graphics Forum*, 32(2):195–204, 2013.

## A Category Functionality Model

As illustrated in Figure 3, the category functionality model consists of a set of proto-patches, where a proto-patch aggregates example patches of the same type. The model also stores the geometric properties of proto-patches, including unary properties associated with each proto-patch and binary properties associated with each pair of proto-patches, and a set of weights that indicate the relevance of each property in describing the functionality. The properties are common geometric descriptors such as overall shape and normal distribution. An overview of the use of the category functionality model is illustrated in Figure 4. A brief description of the model is given below based on the original paper of Hu et al. [8].

### A.1 Model Definition

Formally, a proto-patch  $P_i$  is defined as  $P_i = \{U_i, S_i\}$ , where  $U_i$  are the distributions of unary properties associated to the proto-patch, and  $S_i$  is the functional space that surrounds the proto-patch. A category functionality model is denoted as  $\mathcal{M} = \{P, B, \Omega\}$ , where  $P$  denotes the set of proto-patches, i.e.,  $P = \{P_i\}$ ,  $B$  denotes the distributions of binary properties between pairs of proto-patches, i.e.,  $B = \{B_{i,j}\}$ , and  $\Omega$  denotes the set of weights that indicate the relevance of the unary and binary properties in describing the functionality.

Furthermore, for the  $i^{\text{th}}$  proto-patch,  $U_i$  is a set that consists of different unary properties, such as normal, height, and linearity, i.e.,  $U_i = \{u_{i,k}\}$ , where  $u_{i,k}$  encodes the distribution of the  $k^{\text{th}}$  unary property of the  $i^{\text{th}}$  proto-patch.  $B_{i,j}$  comprises different binary properties, such as the relative orientation and relative position between two patches, denoted as  $B_{i,j} = \{b_{i,j,k}\}$ , where  $b_{i,j,k}$  encodes the distribution of the  $k^{\text{th}}$  binary property between the  $i^{\text{th}}$  and  $j^{\text{th}}$  proto-patches.

### A.2 Model Learning

The category functionality models are learned with a set of training shapes in scene contexts described by their ICON descriptors [10]. An ICON descriptor, as illustrated in Figure 5, organizes the interactions of the target object into a tree structure. The correspondence between all the pairs of ICON hierarchies can be computed through subtree isomorphism so as to cluster shapes with similar interactions into functional categories. Then, for shapes from the same category, the functional patches are collected from the first level nodes of their ICON hierarchies. The functional patches are then summarized into proto-patches by extracting their distributions of unary and binary properties that are represented as histograms of point-level property values, and functional spaces that are represented as closed surfaces.

### A.3 Category Scoring

With the category functionality models, we can make functionality predictions for unknown shapes. Given a model and an input shape, we use the distributions of unary and binary properties of the proto-patches to locate corresponding patches on the unknown shape. Then, based on the proto-patches, we define the functionality distance  $d$  that measures how far the shape is from satisfying the functionality defined by the model. The distance  $d$  is a value between 0 and 1. Thus, we define the category score of a shape as  $s = 1 - d$ .

Specifically, each input shape is represented as a set of  $n$  surface sample points. To capture the patch  $\pi_i$  on the unknown shape that corresponds to the proto-patch  $P_i$  of the

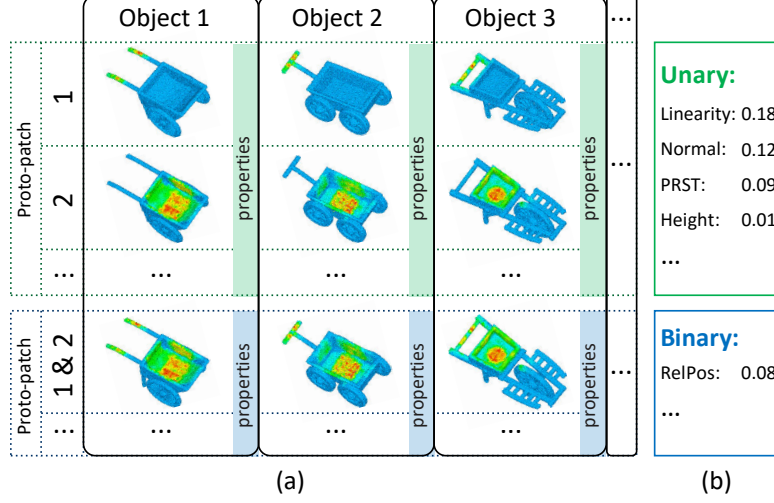


Figure 3: The category functionality model can be defined as: (a) a set of proto-patches with their unary and binary properties, and (b) a set of weights that indicate the relevance of each property in describing the functionality. The image is courtesy of Hu et al. [8].

model  $\mathcal{M}$ , we compute the spatial extent of  $\pi_i$ , which is encoded as a  $n$ -dimensional weight field  $W_i$ , that describes how well the property values of  $\pi_i$  agree with the distributions of  $P_i$ . Each entry  $0 \leq W_{i,j} \leq 1$  of  $W_i$  denotes how strong point  $j$  belongs to  $\pi_i$ . Thus, the spacial extents of multiple patches can be represented as an  $n \times m$  matrix  $W$ , where  $m$  is the number of proto-patches in  $\mathcal{M}$ .

Therefore, the functionality distance can be simply formulated as:

$$d(W, \mathcal{M}) = d_u(W, \mathcal{M}) + d_b(W, \mathcal{M}), \quad (1)$$

where  $d_u$  and  $d_b$  are the unary and binary distances that consider respectively the distributions of unary and binary properties of  $\mathcal{M}$ . The computation of  $d_u$  and  $d_b$  are explained in detail below.

Considering a specific unary feature  $u_k$ , the distance from the patch  $\pi_i$  defined by  $W_i$  on the unknown shape to the proto-patch  $P_i$  is measured by comparing their patch-level properties:

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

where  $\|\cdot\|_F$  is the Frobenius norm of a vector,  $\mathbb{D}_{u_k}(W_i)$  is the patch-level feature descriptor of  $\pi_i$  for the unary feature  $u_k$ , which comprises a set of histograms that describe the point-level properties of all points in the patch  $\pi_i$  for  $u_k$ , and  $\mathcal{N}(u_{i,k})$  is the nearest neighbor of  $\mathbb{D}_{u_k}(W_i)$  in distribution  $u_{i,k} \in U_i$ .

Assuming that the properties are independent of one another, the functionality distance from  $\pi_i$  to  $P_i$  is simply formulated as the sum of all property distances:

$$\begin{aligned} d_u(W_i, P_i) &= \sum_{u_k} \alpha_k^u d_{u_k}(W_i, u_{i,k}) \\ &= \sum_{u_k} \alpha_k^u \|\mathbb{D}_{u_k}(W_i) - \mathcal{N}(u_{i,k})\|_F^2, \end{aligned} \quad (3)$$

where  $\alpha_k^u$  is the weight for property  $u_k$  in  $\Omega$ .

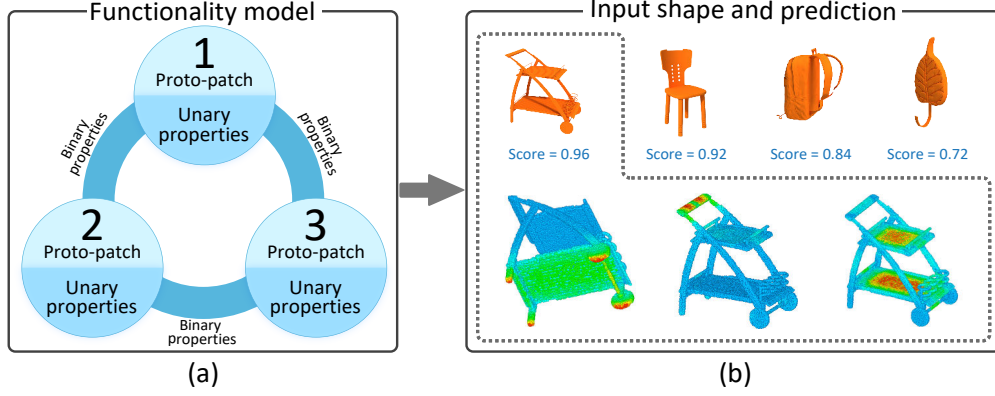


Figure 4: Overview of the use of the category functionality model. (a) A category functionality model describes functionality in the form of proto-patches that summarize the functional patches into collections of their properties. (b) Given an unknown shape as input, the model computes a category score that measures how well the shape supports the functionality of the category. The image is courtesy of Hu et al. [8].

Thus, given an initial assumption for  $W_i$  and its nearest neighbors, the location of  $\pi_i$  and its spatial extent  $W_i$  can be refined iteratively through performing gradient descent of the distance function defined by (3).

With all the refined patches located, the distance that considers the unary properties of all the proto-patches is formulated as:

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

Similar to the computation of unary distance, the binary distance is written as:

$$\begin{aligned}
 d_b(W, \mathcal{M}) &= \sum_{i,j} \sum_{b_{i,j,k}} \alpha_k^b d_{b_k}(W_i, W_j, u_{i,k}) \\
 &= \sum_{i,j} \sum_{b_{i,j,k}} \alpha_k^b \|\mathbb{D}_{b_k}(W_i, W_j) - \mathcal{N}(b_{i,j,k})\|_F^2,
 \end{aligned} \tag{5}$$

where  $\alpha_k^b$  is the weight for property  $b_k$  in  $\Omega$ ,  $\mathbb{D}_{b_k}(W_i, W_j)$  is a set of histograms that describe the pairwise properties between  $\pi_i$  and  $\pi_j$ , and  $\mathcal{N}(b_{i,j,k})$  is the nearest neighbor of  $\mathbb{D}_{b_k}(W_i, W_j)$  in distribution  $b_{i,j,k} \in B_{i,j}$ . More details about the computation of the unary and binary distances as well as the feature descriptors can be found in the next section.

#### A.4 Computation of Functionality Distance

The descriptor  $\mathbb{D}_{u_k}(W_i)$  in (2) is formulated by decoupling the point-level property values from the bins of the histogram that represents  $u_k$ , written as:

$$\mathbb{D}_{u_k}(W_i) = \mathbb{B}_k W_i, \tag{6}$$



Figure 5: The ICON descriptor (right) that describes the interactions of the table in orange in the scene (left). Using the ICON descriptor, the interactions of the table are organized into a tree structure, where the leaf nodes (blue and green nodes) represent the interacting objects and the internal nodes (gray nodes) group the leaf nodes with similar interactions. The image is courtesy of Hu et al. [10].

where  $\mathbb{B}_k \in \{0, 1\}^{n_k^u \times n}$  is a constant logic 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.

Therefore, based on (4), the unary distance is computed as:

$$d_u(W, \mathcal{M}) = \sum_i \sum_{u_{i,k}} \alpha_k^u \|\mathbb{B}_k W_i - \mathcal{N}(u_{i,k})\|_F^2. \quad (7)$$

Similarly, based on (5), the binary distance is computed as:

$$\begin{aligned} d_b(W, \mathcal{M}) &= \sum_{i,j} \sum_{b_{i,j,k}} \alpha_k^b \sum_{l=1}^{n_k^b} \left( W_i^T \mathbb{B}_{k,l}^b W_j - \mathcal{N}(b_{i,j,k})_l \right)^2 \\ &= \sum_{b_k} \sum_{l=1}^{n_k^b} \alpha_k^b \sum_{i,j} \left( W_i^T \mathbb{B}_{k,l}^b W_j - \mathcal{N}(b_{i,j,k})_l \right)^2 \\ &= \sum_{b_k} \sum_{l=1}^{n_k^b} \alpha_k^b \left\| W^T \mathbb{B}_{k,l}^b W - N_{k,l}^b \right\|_F^2, \end{aligned} \quad (8)$$

where  $\mathbb{B}_{k,l}^b \in \{0, 1\}^{n \times n}$  is a logical matrix that indicates whether a pair of samples contributes to bin  $l$  of the histogram representing  $b_k$ ,  $n_k^b$  is the number of histogram bins, and  $N_{k,l}^b = [\mathcal{N}(b_{i,j,k})_l; \forall i, j] \in \mathbb{R}^{m \times m}$ , where  $m$  is the number of proto-patches in  $\mathcal{M}$  and  $\mathcal{N}(b_{i,j,k})_l$  is the  $l^{\text{th}}$  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.

## B Examples of Beam Search

Given a hybrid shape possessing multiple functionalities, Figure 6, Figure 7, and Figure 8 show the complete beam search with a beam width of 2 conducted on the shape and the optimal partial shapes found for three functional categories: *chair*, *desk*, and *shelf*.

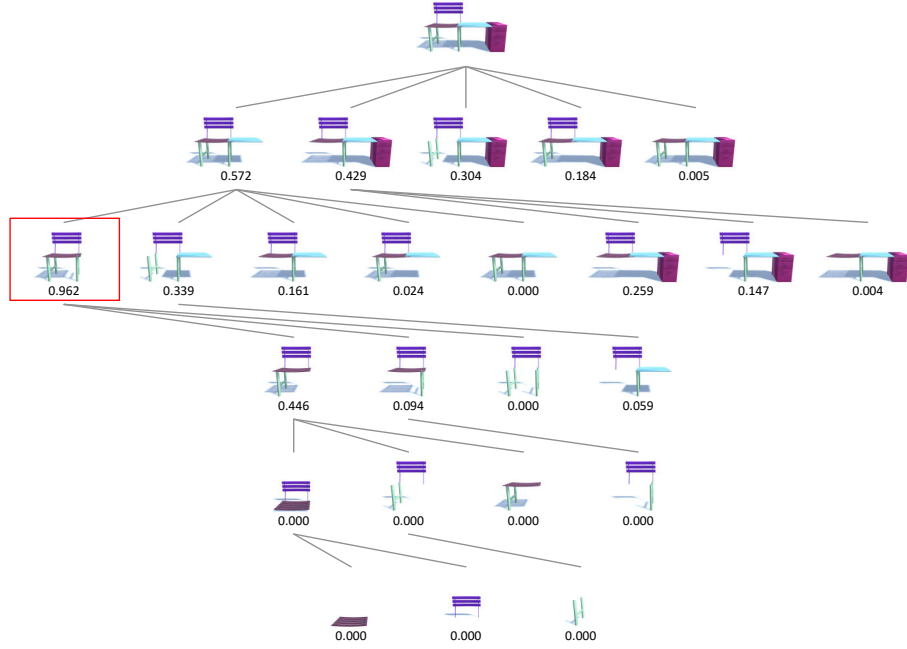


Figure 6: Beam search for functionality partial matching, where we search for the subset of parts that provides the highest score for the *chair* category.

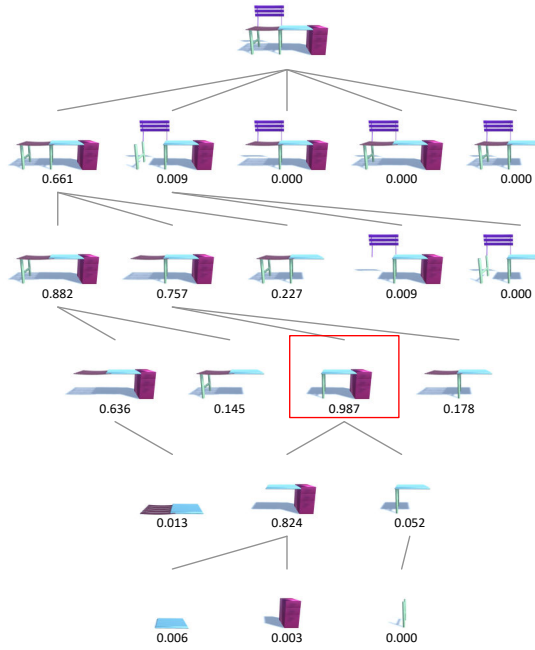


Figure 7: Beam search for functionality partial matching, where we search for the subset of parts that provides the highest score for the *desk* category.

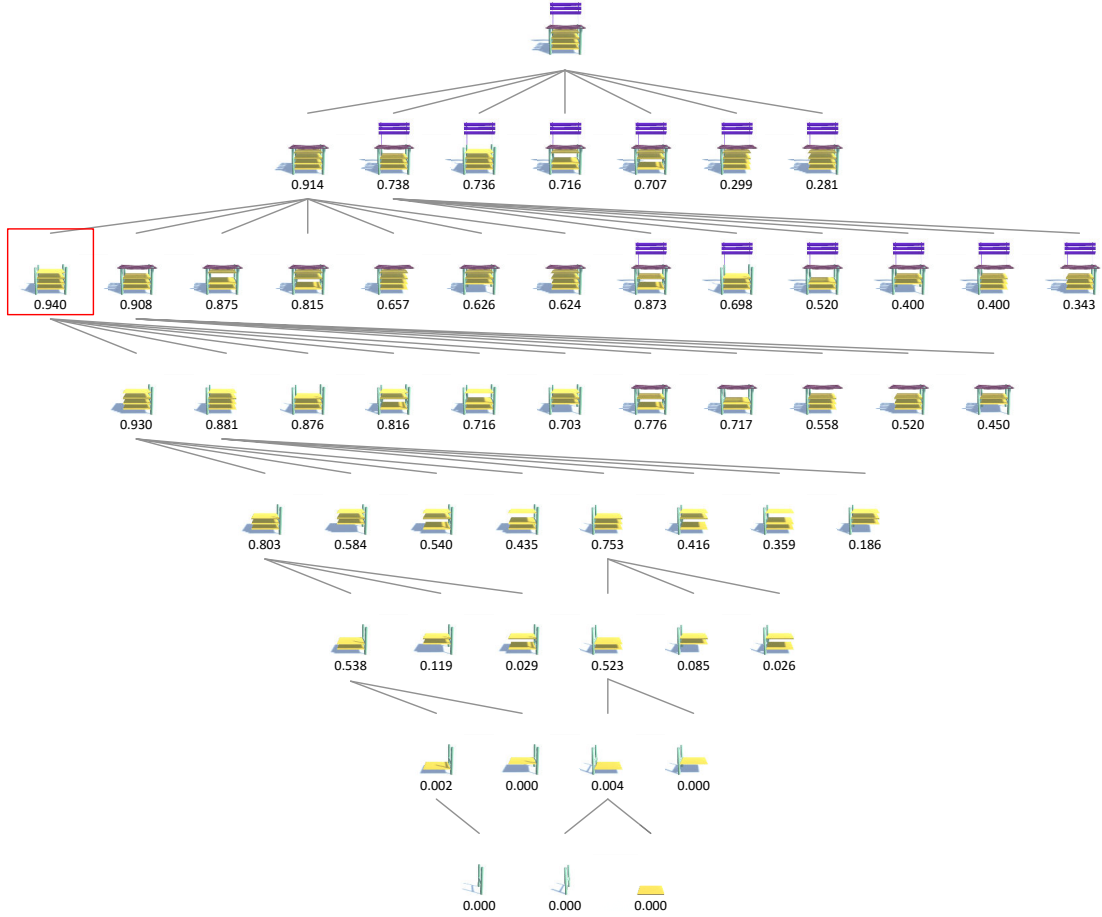


Figure 8: Beam search for functionality partial matching, where we search for the subset of parts that provides the highest score for the *shelf* category.