3D Functionality Analysis for Shape Modeling via Partial Matching

by

Yanran Guan

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario
November, 2019

The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

3D Functionality Analysis for Shape Modeling via Partial Matching

submitted by **Yanran Guan**

in partial fulfillment of the requirements for the degree of

Master of Computer Science

Professor Oliver van Kaick, Thesis Supervisor

Professor David Mould, School of Computer Science

Professor Jochen Lang,
School of Electrical Engineering and Computer Science

Professor Lianying Zhao, Chair,
School of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
November, 2019

Abstract

3D Shape modeling is an important research area in computer graphics. Making sure that the modeled shapes are functional can largely facilitate the modeling process, since then the user is able to create more realistic shapes. To ensure that the generated shapes can have complex functionalities, an important requirement for the modeling system is to enable *cross-category* modeling (or *shape hybridization*), i.e., combining shapes from different categories to create shapes with multiple functionalities. However, without a proper method for evaluating the functionality of hybrid shapes, traditional shape modeling methods are not *functionality-aware*, and often produce shapes that are not *functionally plausible*.

In this thesis, we present an analysis method for evaluating the functionality of 3D shapes, especially hybrid shapes with multiple functionalities. Our method is based on *functionality partial matching*, which localizes the functionality analysis down to the partial shape level. We show that functionality partial matching enables functionality analysis for hybrid shapes.

Moreover, we incorporate functionality partial matching into an evolutionary shape modeling framework, which evolves an initial set of shapes through crossover operations at the level of shape parts, making the evolutionary process functionality-aware. We show that our *functionality-aware model evolution* can produce a large and diverse population of functionally plausible hybrid shapes. We also show that the generated hybrid shapes can be used to augment existing 3D shape datasets to train data-driven machine learning methods for shape segmentation.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my thesis supervisor, Dr. Oliver van Kaick, for his insightful and patient guidance throughout my research and writing of this thesis. It is my fortune to have a supervisor who cares so much about my work, and who is not only helpful and knowledgeable but also understanding and kind.

I would like to thank Dr. David Mould and Dr. Jochen Lang for taking the time to review this thesis, and for their valuable suggestions to improve the quality of this work. Also, as a member of the Graphics, Imaging, and Games Lab (GIGL), I would especially like to thank Dr. David Mould and Dr. Oliver van Kaick for their efforts of lab management and their teachings and support.

I would also like to thank my lovely friends and colleagues from GIGL and the School of Computer Science at Carleton University for enriching my life with their joy, help, and companionship during the past years. I hope that our friendship will last forever.

Last but not least, I would like to say thank you to my parents for their continuous support for my study and their constant love throughout my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

3D shape synthesis and modeling is one of the important research areas in computer graphics. During recent years, virtual reality (VR) and augmented reality (AR) have become increasingly popular and accessible, and there has also been a rapid development of deep learning and 3D printing technologies. Therefore, it is necessary to develop methods for 3D content creation, which play key roles in the design and prototyping of real 3D products, as well as in data augmentation for deep learning.

Current modeling methods have mainly focused on preserving appearance, style, and aesthetic aspects of the generated shapes. These methods, mostly modeling shapes from the same *functional category*[1], have largely missed to consider one important criterion which is *functionality* [22]. A man-made object that is the result of a design process typically serves a specific function. Thus, when customizing an existing design or evolving current designs into a new prototype, the most basic requirement is for the final products to serve their intended functional purposes. In addition, rapid developments in geometric deep learning are placing an ever increasing demand for 3D models that can serve as appropriate training data for learning shape spaces. Thus, *functionality-aware* modeling, which is capable of producing *functionally plausible* 3D shapes in large volumes and varieties, is highly desirable. The analysis of 3D object functionality plays a key role in functionality-aware modeling: both in guiding the modeling process and in filtering the modeling results.

In order to generate shapes with specific functionality, another important requirement is that the modeling method should allow *cross-category* modeling (or *shape*

[1]In this thesis, we define *functional 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*.

hybridization), i.e., crossbreeding shapes from different functional categories to create shapes with multiple functionalities, so that the functional properties of shapes coming from different categories can be combined in newly generated shapes. Previous works on co-analysis and data-driven shape processing [61] only work with homogeneous shapes, while cross-category modeling works with a *heterogeneous* shape collection and creates *hybrid* shapes through composing parts from different objects or functional categories. As a result, with shape hybridization, the well-known principle of structure preservation used by several modeling methods [41] can no longer be strictly followed in the generated hybrid shapes, where *structure* refers to the arrangement and relations between shape parts [42]. Some levels of *structure breaking* must be allowed. Structure breaking can potentially create new geometric structures. However, in consequence, the generated hybrid shapes can be hardly categorized into known functional categories. This brings a new challenge: to adapt and enhance functionality models which were designed for discriminative analysis of single categories [23, 25, 47] to serve shape modeling. In particular, we must address the issue that new cross-category hybrid shapes may arise whose functionality models cannot be learned in advance.

Thus, in this thesis, we propose the notion of *functionality partial matching* for analyzing the functionality of hybrid shapes. Based on the functionality models developed by Hu et al. [23] that learn the functionality of objects per *category*, functionality partial matching is a means for localizing the functionality analysis 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 thesis, the *best fit* is defined as a partial shape that provides the highest functionality partial matching score.

We propose different search heuristics for implementing functionality partial matching. Specifically, we evaluate variations of beam search and compare them to baselines such as a full combinatorial search and a simplified search. An example where reverse beam search is used for functionality partial matching is illustrated in Figure 1.1. Based on the functionality partial matching implemented with our heuristics, we further define a scheme for the evaluation of the functionality of a hybrid shape, combining the functionality partial matching scores of known functional categories into a *functional plausibility score* for evaluating plausibility and a

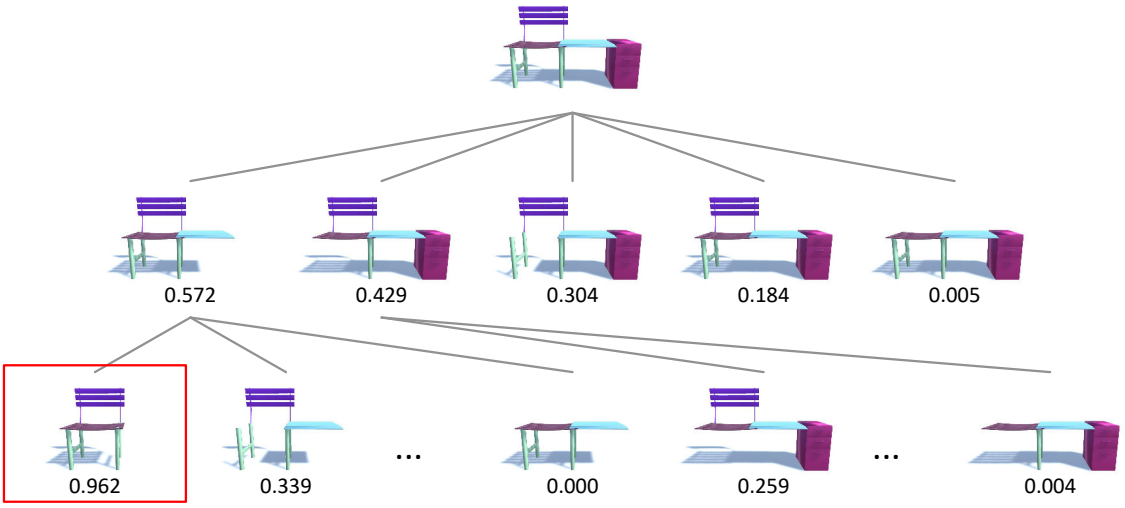


Figure 1.1: An example of functionality partial matching using reverse beam search, where we search for the subset of parts that provides the highest score for the functional category *chair*. We start with the full shape and enumerate its subsets by removing parts one at a time. We only expand a node 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.

multi-functionality score for evaluating the diversity of the generated shapes.

As an important application of our functionality analysis study, we incorporate functionality partial matching into an evolutionary shape modeling framework to develop an exploratory-based shape modeling tool that can evolve a set of 3D objects in a functionality-aware manner, so as to enable *functionality-aware model evolution*. Given an initial population of 3D objects belonging to one or more functional categories, our modeling tool produces generations and generations of functionally plausible hybrids or crossbreeds between parents from preceding generations. In this thesis, we experiment with a dataset of furniture shapes and show that our modeling tool can generate a large amount of hybrid furniture with new designs. For example, we could crossbreed a rocking crib and a chair into a combo which can comfort both a parent and a baby; see Figure 1.2 (top). Our main goal is for the evolutionary modeling tool to produce a large number of functionally plausible 3D prototypes that are both "fit" and "diverse" [63].

The evolutionary approach can be used as a tool for either constrained modeling or open-ended exploration to possibly inspire new shape designs. Specifically, starting

Figure 1.2: Starting from a heterogeneous collection of four objects (left) as initial population, our functionality-aware model evolution can generate a variety of plausible hybrid shapes. Two of them (middle) exhibit strong functional similarity to exiting professional designs (right).

with an initial population of segmented 3D objects, our modeling tool first derives a functional understanding of the input. Then, we evolve the population, where the objects undergo *stochastic* part recombination, mimicking crossovers in evolutionary biology. Our evolution operates at the *part group* level, where a part group consists of one or more related shape parts. Evolution is applied to a set (current population) of shapes, where each modeling operation is a crossover (i.e., an exchange of part groups between two 3D shapes), or mutation (i.e., part insertion into a 3D shape), followed by part (group) deformation and connection to improve plausibility of the offspring shape. At each iteration of evolution, we use functionality partial matching to perform an analysis of the functionality of the offspring shapes, ranking them by their functional plausibility scores as in a design gallery [38, 63]. Then, we select the top k offspring shapes while respecting user-expressed preferences to form the population for the next iteration of evolution; see Figure 6.1. In addition, our tool allows users to restrict or steer the model evolution with constraints defined via functionality labels, such as *sitting*, *leaning*, and *storage*. For example, the user can pick functionality labels and constrain the model evolution to produce offspring shapes possessing the desired functionalities. In summary, our modeling tool includes two types of user interactions as aforementioned: (i) the selection of preferred offspring shapes and (ii) the selection of functionality labels, such that the modeling process is

easy-to-use for users without professional 3D design skills.

Another application of our modeling tool is data augmentation. We show that the hybrid shapes generated by our functionality-aware model evolution can complement existing datasets such as ShapeNet [6] and improve the diversity of training shapes, leading to improved results for data-driven segmentation schemes, especially for *atypical* inputs such as shapes with missing parts.

To summarize, the main contributions of our work are:

1. We adapt the functionality model of Hu et al. [23], that is defined for object categories, to the functionality analysis of hybrid shapes that hold cross-category properties. The key new concept developed is *functionality partial matching*, which enables us to evaluate functional plausibility of new shapes not as a whole, but in parts, with respect to learned functionality models.

2. We apply functionality partial matching into an evolutionary shape modeling framework and develop the first functionality-aware modeling tool for 3D objects, resulting in generations of functionally plausible and diverse offspring shapes in the form of hybrid shapes. Our modeling tool is interactive and involves only *light* user interactions to allow user selection of offspring shapes and constrained modeling.

3. We demonstrate the potential of our functionality-aware shape modeling method for data augmentation. We use the generated hybrid shapes to augment training data for one key application: shape segmentation of partial shapes.

Note that the evolutionary modeling tool was developed in conjunction with Dr. Han Liu, who was a postdoc at Carleton University, and the project was conducted as a collaboration among several researchers [20]. My personal contribution lies in the investigation of the heuristics for functionality partial matching (Contribution 1), integrating the partial matching into the evolutionary approach, and the data augmentation study (Contribution 3).

The remainder of this thesis is organized as follows. In Chapter 2, we review previous literature related to the problem of 3D functionality analysis for hybrid shape modeling. In Chapter 3, we introduce the evolutionary shape modeling framework that we adopt for generating hybrid shapes. In Chapter 4, we describe the proposed method, functionality partial matching for analyzing the functionalities of hybrid

shapes. Chapter 5 presents a comparative study of different algorithms for functionality partial matching. In Chapter 6, we demonstrate the applications of our study, i.e., 3D functionality-aware model evolution and 3D data augmentation, as well as their results. Lastly, Chapter 7 provides our conclusions, and discusses limitations and possibilities for future work.

Chapter 2

Background and Related Work

In this chapter, we first review methods for 3D shape synthesis and modeling, and then discuss recent work that analyzes the functionality of 3D shapes.

2.1 3D Shape Synthesis and Modeling

Classical shape modeling in computer graphics is subject to precise geometric constraints or controls, while fulfilling low-level modeling criteria such as surface smoothness and detail preservation [56]. Recently, much effort has been devoted to structure-preserving shape processing [42], with more emphasis on analyzing and manipulating part structures that are characteristics of man-made shapes. With geometric modeling playing an increasing role in the design and customization of 3D products, the focus of research and development is naturally shifting to higher-level modeling criteria such as creativity and functionality. In this section, we discuss the higher-level shape modeling literature covering four fronts: (i) data-driven assembly-based modeling, (ii) structure-preserving modeling, (iii) evolutionary modeling, and (iv) cross-category modeling.

2.1.1 Data-Driven Assembly-Based Modeling

The idea of "modeling by example" was first proposed by Funkhouser et al. [17], who pioneered the direction of data-driven 3D shape modeling via search-and-assemble schemes applied to object parts [61]. In data-driven assembly-based modeling, parts from 3D object exemplars are substituted into other objects or recombined to form new object prototypes. Due to the large amount of example shapes and parts that are

involved in assembly-based modeling, different mechanisms were developed to decide the suitable part for substitution.

In the work of Funkhouser et al. [17], a system was proposed that allows a user to browse a large database of shapes and recombine parts from the shapes to generate new shapes. The problem of searching for similar parts in the database is addressed by using geometric shape similarity defined between two shapes as the dot product of the voxel rasterization of the first and the square distance transform of the second. Using a similar mechanism, Kraevoy et al. [32] designed a modeling system that creates new shapes by shuffling interchangeable parts between existing shapes. In their work, compatible segmentations of shapes are first computed based on the metrics of part convexity and compactness, to find the correspondences between the interchangeable parts in different shapes. Chaudhuri and Koltun [8] proposed a system that generates data-driven part suggestions that can be added as components to the query shapes. In their work, a histogram similarity measure is used to calculate the correspondence score between the sample points in a database shape and the sample points in the query shape, so that the substitutable parts in the database shape can be found as the part suggestions. Recently, Gonzalez and van Kaick [19] proposed an approach to synthesize new shapes from a collection of fined-grained parts, where the synthesis is guided by a shape template and new shapes are generated via replacing parts in the template with compatible new parts in the part collection. The compatible part replacement is found by a similarity metric in the form of shape energy consisting of a unary term, that measures the similarity between the template and sampled part, and a pairwise term, that measures the consistency between neighboring parts.

Probabilistic reasoning is another approach to define exchangeable parts. Chaudhuri et al. [7] first proposed a probabilistic model using a Bayesian network that learns probabilistic dependencies between part labels, geometric styles, and adjacencies over a set of segmented and labeled shapes. The learned probabilistic model is applied in a shape synthesis system, where a list of suggested parts ranked by relevance is presented to the user for part recombination. In the work of Kalogerakis et al. [29], a component-based probabilistic model, which is composed of a set of variables describing shapes as well as components of shapes, including shape styles, component styles and numbers, and component adjacencies, was learned over a family of input shapes that are consistently segmented and labeled into semantic parts. The parts of the

input shapes can then be assembled into new shapes according to the learned probability distributions. Similarly, Huang et al. [26] presented a shape synthesis method based on learning a probabilistic generative model from a collection of shapes. Differently from the aforementioned works, the method only requires as input one shape per family to be pre-segmented and estimates the part segmentation of shapes by jointly learning a probabilistic deformation model. With the learned shape segmentation, the generative model, which follows the structure of a deep Boltzmann machine [50], encodes the statistical relationships of corresponding surface points and parts in a hierarchical manner. The generative model can be employed for tasks such as optimizing shape correspondence [59] as well as synthesizing new shapes via recombining and deforming parts from the input collection.

Suitable part suggestions for part recombination can also be decided using a fuzzy correspondence, which estimates the similarity relations between shapes and parts. In their evolutionary modeling method, Xu et al. [63] introduced the operators of part crossover and part mutation, that exchange parts across shapes. The method decides which parts should be exchanged according to a fuzzy part correspondence. To define the fuzzy part correspondence, shapes are aligned in an upright orientation [14], and part proportion variations are factored out by anisotropic part scaling [62]. The fuzzy correspondence is then calculated based on the Hausdorff distance between the oriented bounding boxes of two parts. With part crossover and mutation, a large set of diverse 3D shapes, especially man-made shapes, can be created from a small input set. Concurrently, Kim et al. [31] proposed to use fuzzy correspondences between feature points, which is computed from a set of pairwise shape alignments via diffusion maps, to understand similarity relations across 3D shape collections. The similarities then form the basis of an interactive exploration tool that allows a user to browse shapes according to different exploration criteria such as selected regions of an object. The tool returns a sorted list of objects based on their similarity within the selected regions.

Moreover, without explicitly defining a metric of part correspondence, some substructure-based modeling methods have been put forward, where *substructure* refers to a certain subset of shape parts that preserves a special arrangement among the parts. Zheng et al. [65] proposed to detect symmetric functional arrangements (sFarr-s) from shapes, where an sFarr is a symmetric substructure composed of three parts, e.g., a chair seat and two symmetric legs. Their method generates new

shapes by replacing the SFARR-s among existing shapes. As an extension to SFARR-s, Huang et al. [27] introduced the concept of *support substructure*, which is a subset of object parts that provide support and stability to a shape. Shapes with detected support substructures are encoded into graphs that describe the support relations between parts, to enable part reshuffling between different shapes and part rearrangements in the same shape by exchanging compatible supporting or supported components. Similarly, Su et al. [57] proposed to recombine functional substructures from different classes of 3D models using a reference shape that has multi-functional components and a complicated structure. The corresponding functional substructures from shapes stored in a database that match the ones in the reference shape are found by minimizing an energy function that measures the similarity between the substructures. To synthesize new shapes, the pairs of substructures are iteratively selected from the reference and replaced with corresponding database substructures.

The shape modeling method in our work is also data-driven and assembly-based, performing recombination at the part group level. Different from traditional modeling methods, we employ functionality as the constraint for the recombination of part groups, thus incorporating functionality analysis into the shape modeling process. Our method allows to generate shapes with multiple functionalities in a functionality-aware manner. Moreover, all of the methods mentioned above perform part compositions between shapes belonging to the same category, while our method can span different categories.

2.1.2 Evolutionary Modeling

In his landmark book *On the Origin of Species*, Charles Darwin pointed out that all species are evolved from a common ancestor [11]. Inspired by the theory of evolution in biology and the adaptive nature of organisms, the concept of evolutionary computing, which consists in the iterative update of an initial set of candidate solutions by applying recombination and mutation to the solutions [12], has been put forward to solve various practical problems, such as optimization [5], modeling [13], and simulation [58].

The seminal works of Karl Sims [54, 55] introduced evolutionary modeling to the graphics community for the synthesis of novel creatures with desired physical behavior. In his works, each shape is composed of a set of 3D blocks, modeled by a directed graph, which describes the genotype, and a rooted tree, which describes

the phenotype. Both the shape and motion of a creature are evolved simultaneously using genetic algorithms, such as crossover and grafting, for recombining the genotype graphs. Many follow-up works have appeared since. The virtual laboratory Creature Academy proposed by Pilat and Jacob [46], using the same modeling paradigm as Sims' to generate artificial creatures, controls the evolution of artificial creatures through parallel tournament-selection. Lipson and Pollack [35] proposed an evolutionary system for automatic design and manufacturing of robotic forms. Funes and Pollack [16] adopted an evolutionary framework in the design and generation of Lego structures. Moreover, based on mutation-guided evolution, Jon McCormack [39] first proposed to evolve L-system grammars to produce new 3D shapes. The same method is applied in 3D aesthetic modeling by Bergen and Ross [3], who proposed to automatically evolve 3D mathematically aesthetic shapes. In short, the idea of evolutionary modeling has been adopted in the design and generation of multiple forms of objects or structures, apart from the aforementioned examples, also for curvilinear surfaces [21], plants [28], shelters [44], and building architecture [10].

In our work, we adopt an evolutionary framework for shape modeling. Most closely related and inspiring to our modeling method is the "fit and diverse" modeling tool of Xu et al. [63], which evolves a set of 3D shapes via part crossover and mutation, while utilizing a design gallery [38] interface. In "fit and diverse", a crossover refers to the exchange of parts between the parent shapes and a mutation is defined as the deformation of randomly selected parts in an individual shape. Our evolutionary modeling method follows the same evolution paradigm as "fit and diverse", but distinguishes itself from previous works by considering object functionality. Also, "fit and diverse" only evolves shapes of the same category, while our evolution is cross-category, which is further explained in Section 2.1.3.

2.1.3 Cross-Category Modeling

In order to achieve shape hybridization (creating a shape with multiple functionalities), our evolutionary shape modeling framework synthesizes hybrids from initial shapes belonging to different functional categories. However, such cross-category feature is rarely seen in the existing shape synthesis methods. Some substructure-based modeling methods, as introduced in Section 2.1.1, do generate cross-category shapes. For example, the modeling tool developed by Zheng et al. [65] allows the sFarr-s to be transplanted across object categories. The key difference to our method is that

their substructure replacements still *preserve* the objects' overall structures, while we seek to generate objects with structure breaking, which is further discussed in Section 2.1.4. Also, the detected and replaced SFARR-s are rather specific structures. In contrast, our tool for cross-category modeling extracts functional properties of 3D objects from a more generic analysis, and with structure breaking, the shapes resulting from part crossover exhibit much greater variety; see Figure 6.8. The shape reshuffling method introduced in the work of Huang et al. [27], which is based on the detection of support substructures, allows both in-category and cross-category shape synthesis. However, since the structural organization of object parts is already defined by the support substructure, the synthesized shapes will keep the same structure as the existing shapes. Moreover, the modeling method of Su et al. [57], which is also based on substructure transplanting, enables the synthesis of cross-category hybrid shapes under the guidance of a reference shape. In their method, the suitable substructures are selected from database shapes and then recombined into new shapes while reusing the design of the reference shape. Therefore, the main structure of the synthesized shapes is the same as that of the reference.

More closely related to our modeling method is the recent modeling tool by Fu et al. [15] for generating functional hybrids. Their tool also allows cross-category part recombination, but the key difference is that their modeling paradigm is based on fitting 3D parts and shapes to an input human pose. Specifically, their method first identifies groups of candidate shapes which provide *affordances* that support the human pose, i.e., the objects have functionalities compatible with the input pose, and then recombines shape parts to form a well-connected composite shape. In contrast, our functionality analysis is entirely based on shape geometry. More importantly, our model evolution operates on 3D shape collections and allows the generation of a larger volume and variety of functionally plausible hybrids, not just one model to fit a particular affordance constraint.

2.1.4 Structure-Aware Modeling

Shape structure refers to the arrangement and relations between shape parts [42]. Shape structure can reflect the properties of shapes at a high-level, especially for man-made shapes, and can arise from physical constraints, functional or aesthetic designs, and economic considerations. Structure-aware modeling, instead of manipulating lower-level features of a shape, such as local geometric details, focuses on processing

structures of shapes, and can preserve existing structures when shapes are modified, or generate shapes with new structures.

Preserving structural characteristics of shapes, such as symmetry and parallelism, has drawn increasing interest in recent work on shape modeling and shape editing [42]. This type of modeling is naturally performed only over shapes which belong to the same category, since the structures shared among shapes are discovered and preserved. For example, most of the existing assembly-based shape synthesis methods, as discussed in Section 2.1.1, synthesize new shapes by recombining parts from different shapes while fixing the shape structure [7, 8, 17, 19, 26, 27, 29, 57, 63, 65]. Also, structure preservation is important in shape editing. Gal et al. [18] introduced the analyze-and-edit approach in their iWIRES work, where input shapes are first abstracted as a set of descriptive wires, which are then preserved during shape editing. In iWIRES, structural characteristics are considered as the essence of man-made objects. Following up the same idea, numerous methods for structure-preserving shape editing have been proposed, such as the component-wise controller method for shape manipulation of Zheng et al. [66], the interactive tool for retargeting irregular 3D architecture models by Lin et al. [34], the algebraic regularity model for shape editing by Bokeloh et al. [4], and the semantic-based shape deformation method by Yumer et al. [64].

In this context, detection and extraction of structural characteristics, especially symmetric structures [41], plays a key role in structure-preserving processing. Podolak et al. [48] proposed a planar reflective symmetry transform that captures a continuous measure of the reflectional symmetry of a shape with respect to all possible planes. Mitra et al. [40] proposed to detect partial or approximate symmetries based on accumulating local evidence of pairwise symmetries in a transformation space. Simari et al. [53] introduced a method that detects global and local approximate planar symmetries in 3D meshes using a robust statistical approach. They further proposed the folder tree data structure that encodes shape symmetries in the form of non-redundant regions and reflection planes. Apart from symmetry, Pauly et al. [45] presented a computational framework for discovering more general regular geometric structures, involving rotation, translation, and scaling of repetitive elements.

More recently, methods that leverage deep learning for abstracting shape structures have been proposed. In the work of Balashova et al. [1], a structure detector

network, which is trained based on user-selected landmark points on shapes that represent shape structures, was incorporated into a shape synthesis network to guide the generation of shapes with consistent structures. SAGNet of Wu et al. [60] jointly considers the structure and the geometry of shapes when synthesizing shapes, where shape structures are learned as pairwise relationships between the bounding boxes of parts. Similarly, in the StructureNet of Mo et al. [43], shape structures are represented by N-ary trees of parts and the network learns a latent space that encodes shape structures using two types of structure-preserving losses, namely, a symmetry loss that encourages to preserve reflective symmetries in the subtrees of symmetric parts, and an adjacency loss that minimizes the minimum geometry distance between adjacent parts.

Instead of preserving shape structures, our evolutionary modeling tool seeks to generate functionally plausible offspring shapes which may not preserve prominent structures of their parents. Specifically, structure breaking is possible since our functional plausibility score is not always positively correlated with structure preservation — it does not account for structural constraints such as symmetry. As well, the composition of part groups does not need to respect symmetry, e.g., one armrest of a chair can form its own part group. Our evolution is not designed to be strictly functionality-preserving either, as the hybrids can typically break some aspects of the functionalities of their parents. We only consider basic structures of shapes, such as part adjacency and connectivity.

2.2 3D Functionality Analysis

Recently, there has been increasing interest in studying 3D shapes from a functional perspective [22]. Existing works on this topic have all focused on characterizing, comparing, or categorizing 3D objects based on their functionality, where the functionality is typically inferred from the geometry of the shapes and interactions of the shapes with other objects or agents.

The early work of Bar-Aviv and Rivlin [2] introduced a method for classifying functional 3D objects using the simulation of embodied agents, such as virtual humans. Using the simulation of human poses, Kim et al. [30] proposed Shape2Pose, an affordance-inspired shape analysis tool. Given an input 3D shape, Shape2Pose

predicts a corresponding human pose, which is composed of contact points and kinematic parameters, to infer a possible human interaction with this object. Savva et al. [51] also used human poses to predict regions in 3D scenes where actions are likely to take place, to learn the correlation between the geometry and functionality of 3D environments. Furthermore, the probabilistic model proposed by Savva et al. [52] encodes the interactions between a human pose and objects in a scene into a set of prototypical interaction graphs (PiGraphs), which is a human-centric representation of interactions that captures physical contacts and visual attention linkages between the human pose and 3D geometry. Instead of using a specific agent, Pirk et al. [47] introduced a more general means of understanding object interactions. In their work, a descriptor characterizes proximal interactions between a target object and a motion driver, where *motion driver* refers to anything that interacts with a static object, including an agent such as a human pose, also including other objects such as a human hand or wind. The descriptor is computed based on the trajectories of motion particles on the surface of the motion driver while an action takes place.

Hu et al. [25] proposed to analyze object functionality from interactions between multiple objects in scenes rather than analyzing single objects in isolation. The analysis is carried out with a geometric functionality descriptor, the interaction context (ICON) descriptor, that models the functionality of an object within a scene, and takes both object-to-object and human-to-object interactions into account if they are present in the input scene. The ICON descriptor is represented as a tree structure, where the leaf nodes represent the interacting objects and the internal nodes group the leaf nodes with similar interactions. Thus, the similarity between functionalities of two scenes can be estimated by comparing their corresponding ICON trees via subtree isomorphism.

Based on ICON, a co-analysis method was proposed by Hu et al. [23] for functionality analysis of shapes given in isolation. In the co-analysis method, ICON descriptors are first computed for a training set of shapes, deriving structures called *proto-patches*. A proto-patch is a patch prototype supporting a specific type of interaction, and is derived from the interaction regions of all the first level nodes of an ICON hierarchy. The functionality model of an object is then characterized as a composition of its proto-patches, along with their unary and binary properties, and a set of weights defining the relevance of each property in representing the functionality of the category. The model can then be used to predict the functionality of single

objects given in isolation. Since this model forms the basis of our work, we provide more details about it in Section 4.1.1.

Recently, a deep learning based method for shape functionality analysis was proposed by Hu et al. [24], which is also based on an analysis of interactions. In their work, the functional similarity neural network fSIM-NET can, given an input query object in isolation, generate scenes that reveal the functionality most similar to that of the query object.

All of these methods 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, as in a single hybrid shape there may exist multiple partial shapes serving different functionalities. In our work, we base the functionality analysis method on the category functionality model developed by Hu et al. [23]. However, our model is localized to allow functionality partial matching, which is essential when defining the plausibility of hybrid shapes which belong to multiple functional categories.

Note that, as an application of their functionality model, Hu et al. [23] were able to produce some limited forms of hybrid shapes. However, the parent shapes are given and the modeling is manual, with functionality prediction results guiding the user in choosing where hybridization can occur. Most importantly, their hybridization is designed to *preserve* the functionality of a given shape.

Chapter 3

Evolutionary Shape Modeling

We adopt an evolutionary shape modeling framework that generates 3D shapes, possibly with hybrid functionality, from a set of parent shapes. Each parent shape belongs to one functional category only, such as *chair*, *desk*, *cart*, and *shelf*. The evolutionary process follows a *set* evolution paradigm [63] that starts with an initial population of segmented shapes. After pre-processing the input shapes, evolutionary operations are performed over the shapes at the part group level. In this chapter, we give a detailed description about the shape evolution method, including the setup of the evolution and the evolutionary operations.

3.1 Evolution Setup

In this section, we describe the setup of evolution, including the input and output of the method, and the pre-processing of the input.

3.1.1 Input and Output

The input to the evolutionary shape modeling framework, which serves as the initial population for the evolution, is a set of 3D shapes. Each shape comes with a fine-grained segmentation into meaningful parts, where we extract *contact points* between adjacent parts and encode the part-wise connectivity of a shape into a *relation graph* (discussed in Section 3.1.2). We then identify the parts that enable the functionalities of the shapes and assign corresponding functionality labels to them (discussed in Section 3.1.3). Based on the labeled parts, we group together parts with the same functionality label and add their adjacent parts to create part groups, the level at

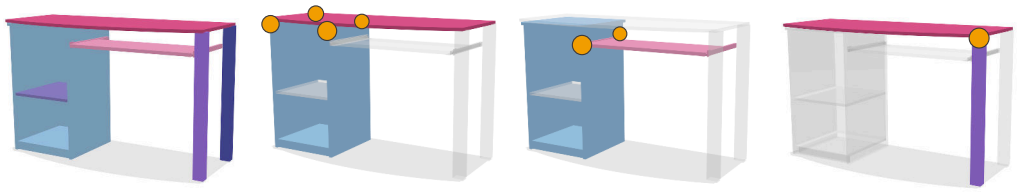


Figure 3.1: Examples of part pairs with different numbers of contact points: four, two, and one contact point.

which the evolution is operated (discussed in Section 3.1.4). The input shapes are also roughly aligned in a consistent manner. The goal of the evolution is to stochastically create a large variety of shapes from a small set of parent shapes, according to guidance from functionality models and possibly the user. Thus, the additional work of segmenting the input shapes is compensated by the fact that the method can then create a large and diverse set of offspring. Iteratively, we perform crossover operations on selected pairs of shapes from the population to produce potential offspring. After several iterations, the output of the evolution is a population of shapes, where each shape combines parts coming from multiple parent shapes, possibly from distinct categories.

3.1.2 Shape Representation

Since the input shapes are given as triangle meshes, we 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. We use the point sets to detect contact points between adjacent parts of each shape, which indicate how the parts connect to each other. As illustrated in Figure 3.1, our method considers 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.

The contact points are extracted in a semi-automatic manner. First, we find the set of points of one part that are the closest to points of the other part according to a small threshold, to define a set of *boundary points*. The threshold is set as 1 % of the bounding box diagonal of the shape. The union of boundary points for two parts are defined as their *boundary region*. Next, for each pair of neighboring parts,

Figure 3.2: Examples of relation graphs for two shapes: a connected graph (left) and a disconnected graph (right).

we detect the pair of closest points on each part within the boundary region and take the midpoint of these points as the contact point. Then, for the pair of neighboring parts having more than one contact points, the user can manually specify additional contact points or adjust them manually.

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. Examples of relation graphs are shown in Figure 3.2. The graphs capture the part-wise connectivity and the structure of the shapes and are used to guide the placement of parts during the evolution.

3.1.3 Functionality Labels

We employ two types of *functionality labels* in the modeling framework: (i) *part labels*, which describe the functionality of a part, e.g., *rolling* for a wheel or *sitting* for a chair seat; (ii) *category labels*, which denote the functionality of an entire category of shapes, e.g., *chair* for shapes that can be used as chairs. To avoid confusion between these two types of labels, we call the former *functionality label* or simply *label* of a part and the latter the *functional category* or simply *category* of the shape.

Part Labels

Before starting the evolution, the parts of each shape are labeled with functionality labels. These serve as constraints for the evolutionary operations. To obtain the

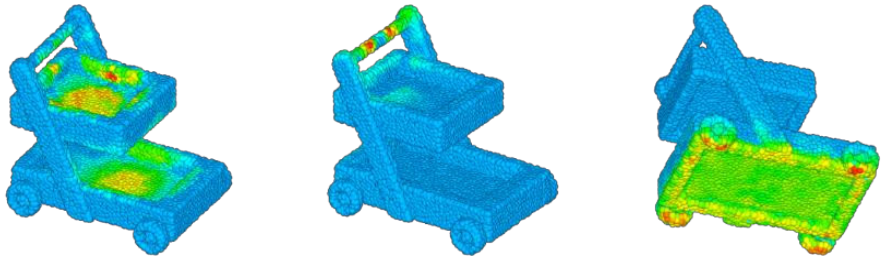


Figure 3.3: The functional patches predicted for a shape belonging to the *cart* category. From left to right are *storage*, *grasping*, and *rolling*, respectively. The image is courtesy of Hu et al. [23].

functionality labels for shapes of known categories, we first predict *functional patches* over the initial population of shapes with the functionality models of Hu et al. [23]. The prediction provides a weight field over the input point cloud, which defines the probability that each point belongs to a patch that provides a certain functionality, such as *sitting* or *leaning*. These patches associated with a functionality are called *functional patches*. An example of predicted functional patches is visualized in Figure 3.3, and more details about the functionality models and prediction are given in Chapter 4.

Given the weight fields predicted by the functionality models for each functional patch, we assign to a part the functionality label of the functional patch that has the highest sum of weights in the part. If the sum of weights is below a threshold of 0.5 for all the functional categories (where the weights for a functional patch sum up to 1), then we leave the part unlabeled. If the input shape belongs to an unknown functional category, the user can manually annotate its parts with custom functionality labels, e.g., *rolling* and *rocking*. Note that some parts may remain unlabeled. We keep track of symmetries among shape parts by storing them in a list. In the evolutionary modeling framework, symmetries are manually specified by the user, but any existing method for automatically detecting part symmetries [41] can be incorporated into the framework.

Category Labels

We assign a single functional category to each shape. Then, when shapes are evolved from two parents, they receive all the functional categories from their parent shapes. The functional categories indicate which category functionality models should be

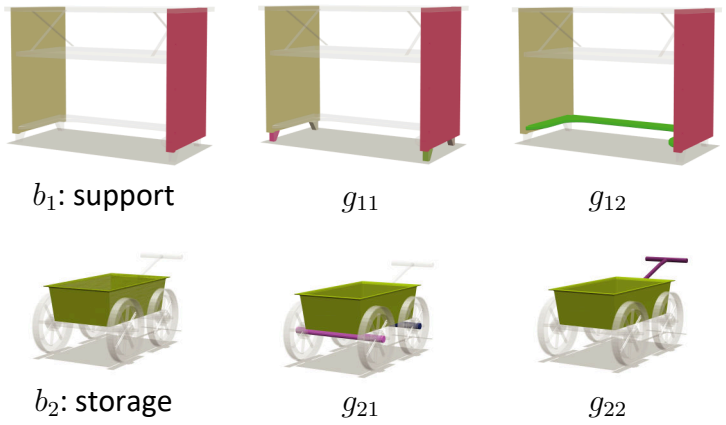


Figure 3.4: Examples of part groups derived from base groups. Given base group b_1, with functionality label *support*, we derive part groups g_{11} and g_{12} by adding the colored parts to b_1, where b_1 is formed by symmetric and disconnected parts. Similarly, we derive g_{21} and g_{22} from b_2.

used for evaluating the functionality of the shapes, which we describe in more detail in Section 3.2.

3.1.4 Part Groups

Our shape evolution is executed by crossover operations defined at the part group level, where each part group consists of one or more parts from a shape in the population and represents a partial shape taken from the original shape. We create multiple candidate part groups for each shape by enumerating possible combinations of its parts. We use a heuristic based on the functionality part labels to form the part groups, which helps us to avoid the combinatorial explosion of enumerating all possible part groups.

Specifically, we first group together adjacent parts with the same functionality label, which form base groups from which we build the final part groups. For each of these base groups, we create several part groups by adding different combinations of parts that are adjacent to the base group, with a breadth-first search. Note that the expanded parts need to be either all unlabeled or share a same label, which can be distinct from the initial label. A base group by itself is also considered a valid part group. Part groups created in this manner are restricted to one or two functionality

labels, and thus form meaningful structures for exchanging and recombining the functionalities of the input shapes. We also allow symmetric parts to form base groups for further enumeration, implying that part groups can be disconnected. Finally, individual symmetric parts can also form the basis of part groups, so that structure breaking is allowed in the evolution. Figure 3.4 shows examples of part groups created for two different types of base groups of a shape.

3.2 Shape Evolution

Our shape evolution is implemented by the evolutionary operation called *crossover* that operates at the part group level (discussed in Section 3.2.1), and guided by user preferences, including preferred functionality labels and preferred offspring shapes, also including a selection of shape diversity (discussed in Section 3.2.2). Starting from an initial population $\mathcal{G}_0$, the evolution iteratively generates offspring shapes for generation $\mathcal{G}_i$ from its previous generation $\mathcal{G}_{i-1}$, where all the generated shapes preserve the user-selected functionality labels specified by set $\mathcal{L}_{\text{user}}$. The evolution stops when a preset maximum number of iterations $i_{\max}$ is reached. During the production of $\mathcal{G}_i$, we try all the possible pairs of shapes, $\mathcal{S}_A$ and $\mathcal{S}_B$ ($\mathcal{S}_A \neq \mathcal{S}_B$), from $\mathcal{G}_{i-1}$ as parent shapes to generate offspring shapes. If all the user-selected functionality labels are preserved in the parent shapes, the crossover degenerates to the exchange of part groups. If missing functionality labels are detected, we insert a part group that possesses the missing functionality into the parent shape. We apply a diversity selection to ensure that the generated shapes are geometrically *diverse*. As soon as one generation is produced, we allow the user to select preferred offspring shapes as parent shapes for the next generation. After the evolution is finished, a set $\mathcal{G}_{\text{evolved}}$ is presented to the user that contains shapes from all the evolved generations. The flow of our shape evolution is described in Algorithm 1. More details about the evolutionary operations and the constraints used in the evolution are discussed below.

3.2.1 Evolutionary Operations

A crossover is defined between two part groups g_A and g_B, anchored on shapes $\mathcal{S}_A$ and $\mathcal{S}_B$, respectively. The crossover 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

Algorithm 1: Shape evolution

Input: $\mathcal{G}_0$, $\mathcal{L}_{\text{user}}$, $i_{\max}$
Output: $\mathcal{G}_{\text{evolved}}$

```
1  function ShapeEvolution(G_0, L_user, i_max)
2      G_evolved ← ∅
3      i ← 1
4      while i ≤ i_max do
5          G_i ← ∅
6          foreach S_A ∈ G_{i−1} do
7              L_A ← functionality labels in S_A
8              L_missing ← L_user \ L_A
9              foreach S_B ∈ G_{i−1} do
10                 if S_A ≠ S_B then
11                     if L_missing = ∅ then
12                         foreach g_A ∈ all part groups of S_A do
13                             foreach g_B ∈ all part groups of S_B do
14                                 if g_A is unlabeled or has no functionality label in
                                    L_user then
15                                     S_offspring ← exchange g_A on S_A for g_B
16                                     G_i ← G_i ∪ {S_offspring}
17                     if L_missing ≠ ∅ then
18                         foreach g ∈ all part groups of S_B do
19                             if g has a functionality label in L_missing then
20                                 S_offspring ← insert g into S_A
21                                 G_i ← G_i ∪ {S_offspring}
22         G_i ← DiversitySelection(G_i)
23         G_i ← UserSelection(G_i)
24         G_evolved ← G_evolved ∪ G_i
25         i ← i + 1
26     return G_evolved
```

shape $\mathcal{S}_B$. We also allow in some cases g_A (or g_B) to be 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 $\mathcal{S}_A$. We call these two different types of crossover *part group exchange* and *part group insertion*. Deformation of part groups may be necessitated after a crossover to fulfill geometric constraints. Note that we do not define a *part group removal* operation since we start the evolution with relatively simple shapes that we evolve into more complex ones, so that removal of functionality is not needed. Figure 3.5

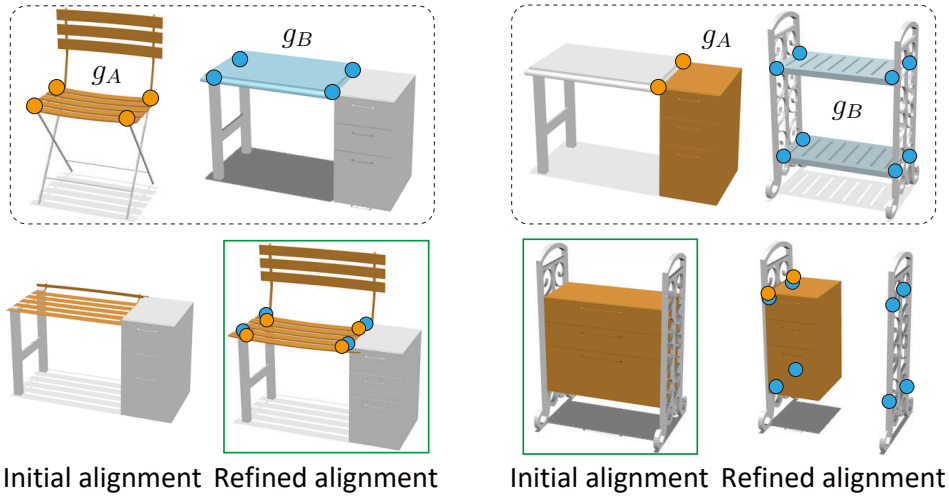


Figure 3.5: Two examples of part group exchange. After a crossover that replaces the part group g_B by g_A, we perform an initial placement of g_A based on bounding box alignment. This is followed by a refinement deformation that aligns contact points (blue and orange circles). The refinement may be reverted back if some of the contact points are not brought into proximity (right column example).

illustrates examples of part group exchange, while Figure 3.7 illustrates part group insertion.

Part Group Exchange

After exchanging the part groups of two shapes, we position the parts according to a set of heuristics. Suppose without loss of generality that we are replacing g_B with g_A in shape $\mathcal{S}_B$. We first perform an initial alignment of g_A, so that we can use spatial proximity to establish a correspondence between the contact points of g_A and contact points that previously connected to parts in g_B. Then, we refine the placement of g_A so that corresponding contact points are brought into contact with each other.

For the initial alignment, we describe g_A and g_B with axis-aligned bounding boxes. We translate g_A so that the center of its box aligns with the center of g_B. Next, we scale the longest axis of g_A to align it with the corresponding axis of g_B, and scale the other axes proportionally to maintain the aspect ratio of g_A's bounding box.

For the refined alignment, we match each contact point in g_A to the closest contact point in $\mathcal{S}_B$, after the initial alignment. If g_A and g_B have different numbers of contact

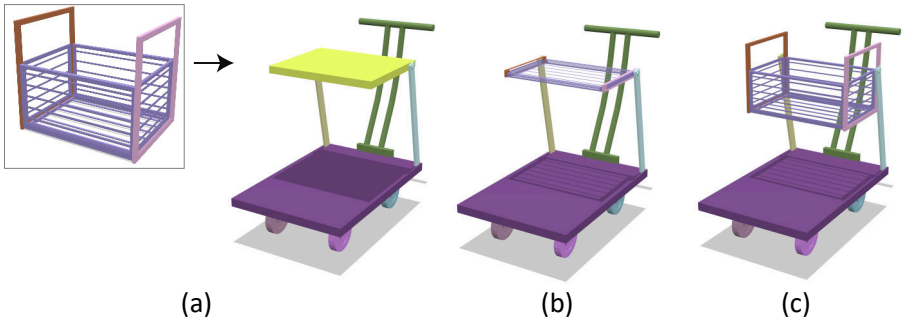


Figure 3.6: Restoring the scale of parts after a crossover. When performing the crossover between the entire basket and the yellow part group of the cart shown in (a), the basket is placed and deformed as shown in (b). Since the part placement significantly deforms the parts, which possess functionality labels, we restore their proportions.

points n_A and n_B, respectively, we define the match only for the $n = \min(n_A, n_B)$ closest contact points. Then, we derive the transformation that best aligns the matching contact points. In the following operations, only points that were matched are considered. First we derive a translation that aligns the centroid of the contact points in g_A to the centroid of the corresponding contact points in shape $\mathcal{S}_B$. Next, we scale g_A by the average of the scalings needed to align each contact point in g_A to its matching contact point in $\mathcal{S}_B$. This provides a transformation that best aligns the part groups in a least-squares sense, according to a term which is the sum of squared errors that penalize discrepancies of position between each pair of contact points, similarly to the optimization proposed for part placement by Kalogerakis et al. [29]. Note that, since the input shapes are roughly pre-aligned, the transformation does not include a rotation, so that the evolved shapes have the same alignment as the input ones. This assumption of pre-alignment may cause the evolved shapes to have less variations, but simplifies the functionality analysis, as the functionality models that we use require the input shapes to be consistently oriented [23].

The two steps used in part placement are illustrated in the bottom row of Figure 3.5. If the refined alignment is suboptimal, meaning that the distance of any of the contact points to its closest contact point in the other part group is too large, according to a threshold, then we revert back to the initial alignment. This provides a more meaningful part placement as shown in the right column of Figure 3.5, since in this example the refined alignment fails because some of the blue contact points are

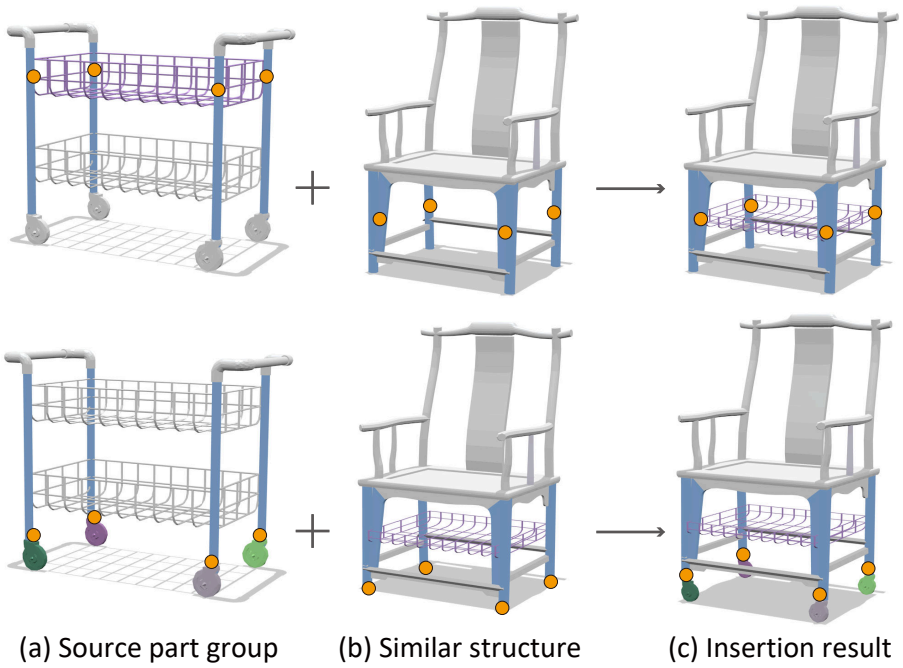


Figure 3.7: Two examples of part group insertion. Given a part group in (a), we find a region in the target shape in (b) with a structure similar to the context of the part group in the source shape in (a). For example, the context consists of four support structures (the legs) adjacent to the basket on the top row, or one support structure (one leg) adjacent to each wheel on the bottom row. We insert the part group in this region as shown in (c).

too far from the yellow ones. The proximity threshold is set as 5 % of the bounding box diagonal of the shape.

During the refined alignment, the proportions of parts can change, which could leave some parts unrecognizable. Thus, we restore the proportions of parts that possess functionality labels, if their scaling passes a threshold. Specifically, if the scaling of the y-axis or z-axis of a part's bounding box relative to the x-axis is greater than an empirical factor of 3, we restore the scale of the affected axis. The effect of this step is illustrated in Figure 3.6.

Part Group Insertion

For inserting a part group g into a shape $\mathcal{S}_A$, we look for a region in $\mathcal{S}_A$ with a similar structure to the context of g in its source shape and place g in this region. Specifically,

we record the relative position (translation vectors) of g to all of its adjacent parts in its source shape that have functionality labels, where we denote the set of labels of the adjacent parts as $\mathcal{L}_{\text{adjacent}}$. Next, we locate a region in $\mathcal{S}_A$ which has a similar relative position to the same set of labels $\mathcal{L}_{\text{adjacent}}$. This is formally defined as the location in $\mathcal{S}_A$ with the average of translation vectors that is the closest to the average recorded in the source shape. Only the labels in $\mathcal{L}_{\text{adjacent}}$ that exist in shape $\mathcal{S}_A$ are considered in the average. We choose this location as the position to insert g. Two examples of part group insertion are shown in Figure 3.7. Note that, if the location in $\mathcal{S}_A$ is already occupied by a part group g_A, then we perform a standard crossover to exchange g_A for g. Since insertion operations are unconstrained and could be applied add infinitum, they are mainly used to add missing functionalities (discussed in Section 3.2.2).

3.2.2 Constraints in Evolution

The evolution consists in applying part group exchange operations to generate candidate offspring shapes, and then performing additional part group insertion operations to guarantee the presence of certain functionalities. In order to make the generated offspring shapes less random and more reasonable, a few constraints are employed that can guide the process of evolution, including constraints specified by the user and the selection of diversity.

User Constraints

The user can control the evolution with two guiding mechanisms: (i) setting constraints on the functionality part labels and (ii) selecting preferred offspring shapes for further evolution. Specifically, the user selects a set of part functionalities that should appear in the offspring shapes, and the evolutionary process ensures that the shapes generated by crossovers possess all of the specified labels. After one generation of shapes has been evolved, the user can select preferred shapes from the population, which are then used as input for evolving another generation.

For satisfying the user constraints, we verify if all the shapes possess parts with the functionality labels listed in the constraints. If a shape $\mathcal{S}_A$ does not possess all the labels, we insert the missing labels through additional crossover operations. Specifically, for each missing functionality in $\mathcal{S}_A$, we randomly select a part group g that possesses the missing functionality, from the pool of all part groups derived from

the parent shapes. Next, we add the part group to $\mathcal{S}_A$ with one of two operations: (i) exchanging g with a part group g_A in $\mathcal{S}_A$ which is unlabeled or has a label that is not in the set of constraints; (ii) inserting g into $\mathcal{S}_A$ with part group insertion. We remark that insertion operations are mainly used to add missing functionality to offspring shapes, as otherwise they could be applied ad infinitum in an unconstrained setting. If the necessary part group insertions cannot be performed, then the candidate shape is discarded.

Diversity Selection

Given that several of the offspring shapes can be geometrically similar if they were created from similar part groups, for each generation of evolution, we only keep the offspring shapes that are diverse in terms of their geometry. Specifically, we compute the geometric similarity between all pairs of shapes according to the light field descriptor (LFD) [9], which gives an indication of the global similarity of shapes based on light fields. The light field of a 3D object describes the radiometric properties of light around the object and is represented by a 4D function interpreted from a set of 2D images [33]. The LFD captures shape features from the light field rendered from cameras positioned on the 20 vertices of a regular dodecahedron around a shape. Once the LFDs are computed for all offspring shapes, we perform farthest point sampling according to the LFD distances between the shapes, to keep only the top 50 % most distinct shapes. Finally, these shapes are presented to the user according to the ranking of their functional plausibility, as described in Chapter 4.

Chapter 4

Functionality Analysis

In this chapter, we describe the method for analyzing the functionalities of hybrid shapes, including the category functionality models that calculate the functionality scores for each functional category (i.e., the category scores). We also describe the shape validity verification that defines the geometrical and physical validity of shapes, the partial matching algorithms that find the best functional-matching partial shapes from the hybrid shapes, and lastly, the functionality scoring schemes that evaluate the functional plausibility and multi-functionality of hybrid shapes.

4.1 Functional Plausibility Modeling

Our functional plausibility analysis is based on the functionality modeling method of Hu et al. [23], which comprises functionality models of 15 functional categories. Each 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.

4.1.1 Category Functionality Model

As illustrated in Figure 4.1, 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

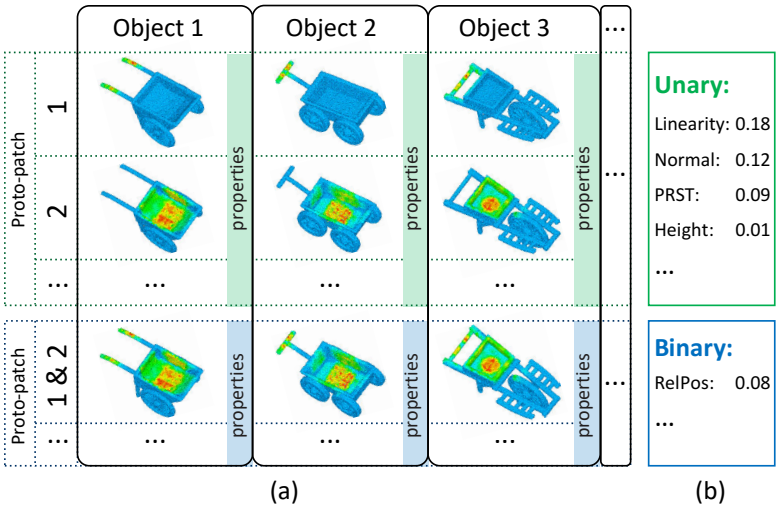


Figure 4.1: The category functionality model can be defined as: (a) a set of proto-patches with their unary and binary properties, and (b) a set of weights that indicate the relevance of each property in describing the functionality. The image is courtesy of Hu et al. [23].

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 4.2. A brief description of the model is given below based on the original paper of Hu et al. [23].

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 Ω denotes the set of weights that indicate the relevance of the unary and binary properties in describing the functionality.

Furthermore, for the i^{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^{th} unary property of the i^{th} proto-patch. $B_{i,j}$ comprises different binary properties, such as the relative orientation and relative position between

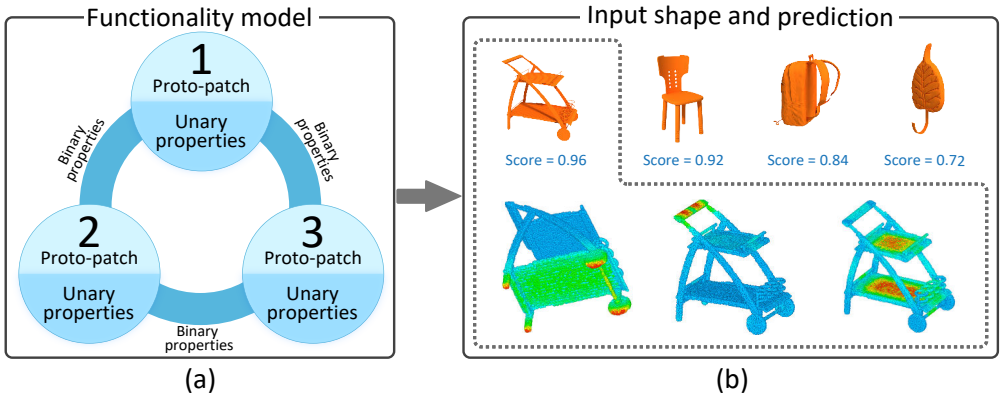


Figure 4.2: Overview of the use of the category functionality model. (a) A category functionality model describes functionality in the form of proto-patches that summarize the functional patches into collections of their properties. (b) Given an unknown shape as input, the model computes a category score that measures how well the shape supports the functionality of the category. The image is courtesy of Hu et al. [23].

two patches, denoted as $B_{i,j} = \{b_{i,j,k}\}$, where $b_{i,j,k}$ encodes the distribution of the k^{th} binary property between the i^{th} and j^{th} proto-patches.

Model Learning

The category functionality models are learned with a set of training shapes in scene contexts described by their ICON descriptors [25]. An ICON descriptor, as illustrated in Figure 4.3, 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.

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

Figure 4.3: 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. [25].

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 π_i on the unknown shape that corresponds to the proto-patch P_i of the model $\mathcal{M}$, we compute the spatial extent of π_i, which is encoded as a n-dimensional weight field W_i, that describes how well the property values of π_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 π_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:

$$d(W, \mathcal{M}) = d_u(W, \mathcal{M}) + d_b(W, \mathcal{M}), \tag{4.1}$$

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 π_i defined by W_i on the unknown shape to the proto-patch P_i is measured by comparing their

patch-level properties:

$$d_{u_k}(W_i, u_{i,k}) = \|\mathbb{D}_{u_k}(W_i) - \mathcal{N}(u_{i,k})\|_F^2 , \tag{4.2}$$

where $\|\cdot\|_F$ is the Frobenius norm of a vector, $\mathbb{D}_{u_k}(W_i)$ is the patch-level feature descriptor of π_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 π_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 π_i to P_i is simply formulated as the sum of all property distances:

$$\begin{aligned} d_u(W_i, P_i) &= \sum_{u_k} \alpha_k^u d_{u_k}(W_i, u_{i,k}) \\ &= \sum_{u_k} \alpha_k^u \|\mathbb{D}_{u_k}(W_i) - \mathcal{N}(u_{i,k})\|_F^2 , \end{aligned} \tag{4.3}$$

where α_k^u is the weight for property u_k in Ω.

Thus, given an initial assumption for W_i and its nearest neighbors, the location of π_i and its spatial extent W_i can be refined iteratively through performing gradient descent of the distance function defined by (4.3).

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

$$\begin{aligned} d_u(W, \mathcal{M}) &= \sum_i \sum_{u_{i,k}} \alpha_k^u d_{u_k}(W_i, u_{i,k}) \\ &= \sum_i \sum_{u_{i,k}} \alpha_k^u \|\mathbb{D}_{u_k}(W_i) - \mathcal{N}(u_{i,k})\|_F^2 . \end{aligned} \tag{4.4}$$

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

$$\begin{aligned} d_b(W, \mathcal{M}) &= \sum_{i,j} \sum_{b_{i,j,k}} \alpha_k^b d_{b_k}(W_i, W_j, u_{i,k}) \\ &= \sum_{i,j} \sum_{b_{i,j,k}} \alpha_k^b \|\mathbb{D}_{b_k}(W_i, W_j) - \mathcal{N}(b_{i,j,k})\|_F^2 , \end{aligned} \tag{4.5}$$

where α_k^b is the weight for property b_k in Ω, $\mathbb{D}_{b_k}(W_i, W_j)$ is a set of histograms that describe the pairwise properties between π_i and π_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

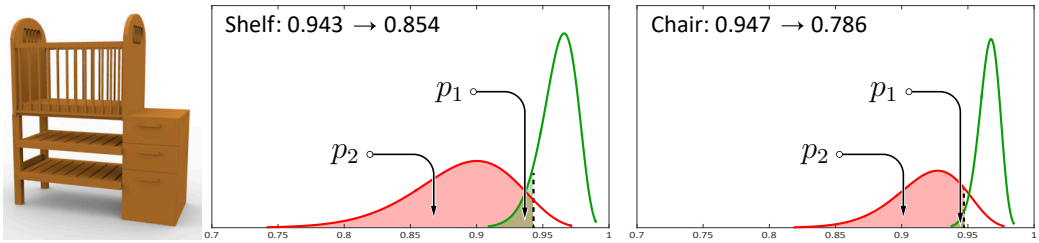


Figure 4.4: Normalization of category scores for the shape on the left. The two numbers on the top of each graph are the scores of the shape for the *shelf* and *chair* categories before and after normalization. The score distributions for training shapes inside/outside the corresponding category are drawn in green and red, respectively. The probabilities p_1 and p_2 represent the percentages of shapes having lower scores in the corresponding distributions. We see that, even though the original scores for the two categories are quite close, the normalized score for the *chair* category becomes much smaller than that of the *shelf* category, since shapes inside the *chair* category have relatively high scores.

of the unary and binary distances as well as the feature descriptors can be found in Appendix A.

4.1.2 Score Normalization

The category scores computed with the models of Hu et al. [23] are based on the similarity between the input shape being evaluated and training shapes in the functional category of the model. Since the functionality models are trained separately per category, scores for different categories are not directly comparable. To address this limitation and obtain scores that are comparable across categories, we normalize the scores for each category according to the training data.

Given a category functionality model, we compute the scores for all the shapes in the training data. Next, we compute the cumulative distribution of the scores of shapes inside the category, and also of the scores of shapes outside the category, obtaining cumulative distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. Given an unknown query shape, we compute its score with the category functionality model, and compute its normalized score as $w_1 \times p_1 + w_2 \times p_2$, where p_1 is the probability of the shape's score according to $\mathcal{D}_1$, and p_2 according to $\mathcal{D}_2$. The probability p_1 is an approximation of the percentage of shapes inside the category that have scores lower than the query, and represents the probability of the query being inside the category. Similarly, p_2

represents the probability of the shape not being outside the category. Thus, the query is more likely to belong to the category when both p_1 and p_2 are large, which can be captured by a weighted sum of these two probabilities. The weights w_1 and w_2 are defined as the percentages of shapes inside and outside the category over all the shapes in the training data. They indicate the reliability of the two distributions, respectively. Figure 4.4 shows one example to illustrate how the scores are normalized and more meaningful than the original scores.

4.2 Shape Validity Verification

A shape should be geometrically and physically valid in addition to being functionally plausible. We perform the validity verification according to the following criteria.

4.2.1 Part-Wise Connectivity

In order to ensure that all parts are connected in the valid shapes, we include the verification of part-wise connectivity into the validity evaluation. Since we use a relation graph to represent each shape, as described in Section 3.1.2, the part-wise connectivity can be reduced to the graph connectivity, for which we only need to check if the graph has isolated subgraphs or not. For example, in Figure 3.2, the relation graph on the left represents a shape with all parts connected, while the one on the right represents a disconnected shape.

4.2.2 Physical Stability

The mass arrangement of a valid shape should be balanced so that the shape does not fall due to gravity. Similar to the idea of static stability of shapes introduced by Fu et al. [14], we verify a shape's stability in an approximate manner.

We first calculate the center of mass of the shape by averaging the center points of the bounding box diagonals of all parts of the shape and calculate the convex polygon formed by all *ground touching points* of the shape. A threshold of 1 % is set to decide the ground touching points, i.e., we find the minimum z-coordinate $z_{\min}$ among all points in the shape and mark all points that have z-coordinate smaller than $z_{\min} + 1\,\%$ as ground touching points. We then check whether the projection of the center of mass on the ground falls inside the convex boundary formed by the ground

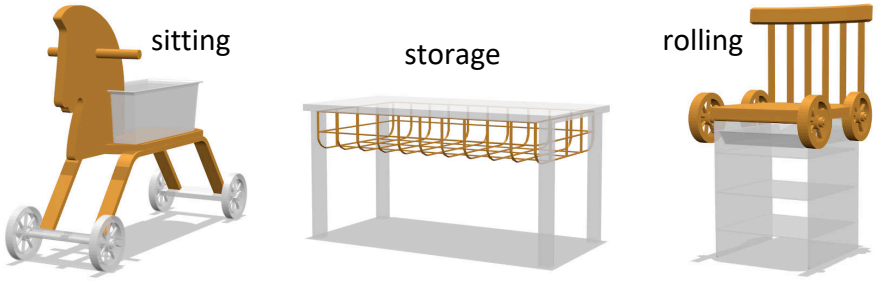


Figure 4.5: Shapes not having functional space due to obstruction of parts.

touching points. The shape is considered as physically stable if the projection is inside the convex boundary. Note that this physical stability check assumes that parts are static and have constant density. Thus, it is not able to approximate stability for shapes with movable parts or composed of various physical materials.

4.2.3 Functional Space

The shape should have enough functional 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. Some examples of shapes not having enough functional space are shown in Figure 4.5. We use the functional space extracted from the models of Hu et al. [23] to verify if a shape has adequate space to perform a functionality.

4.3 Functionality Partial Matching

Given that the parent shapes of an offspring may not be of the same functional category, the evolutionary modeling can generate cross-category hybrid shapes that do not fit well a single functional category and cannot be modeled by category functionality models. Moreover, one of the goals of the evolutionary modeling is to generate shapes with surprising functionality, possibly mixing the functionality of different functional categories. Thus, we need to take multiple categories into account when evaluating a hybrid shape.

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

Algorithm 2: Combinatorial search

```
Input: S, F
Output: scores
function CombinatorialSearch(S,F)
    category_scores ← compute category scores of S
    foreach f ∈ F do scores[f] ← Normalize(category_scores[f])
    foreach g ∈ power set of all parts do
        if g is part-wise connected and physically stable then
            category_scores ← compute category scores of g
            foreach f ∈ F do
                if scores[f] < Normalize(category_scores[f]) then
                    scores[f] ← Normalize(category_scores[f])
    foreach f ∈ F do
        if S does not have functional space for f then
            scores[f] ← 0
    return scores
```

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. Specifically, we combine the category functionality models with partial shape searching algorithms to extract the functional partial shapes from hybrid shapes that best match the functional categories, so that the category scores of the best-matching partial shapes can be used to reveal the functional plausibility of the entire shape. Technically speaking, a combinatorial search, which traverses through all partial shapes from a hybrid shape, can give the optimal analysis result. However, due to its high computational complexity, we propose other search heuristics as low-cost alternatives, namely, forward beam search, reverse beam search, and a simplified search.

4.3.1 Combinatorial Search

The combinatorial search, as the term suggests, takes all partial shapes of a shape into consideration. Specifically, for a shape $\mathcal{S}$ with n parts, the combinatorial search first computes all the category scores for the entire shape $\mathcal{S}$. Then, it traverses through all its partial shapes represented by the $2^n - 2$ part groups, and computes the category scores for the partial shapes which are part-wise connected and physically stable.

During the combinatorial search, the category scores are calculated for all functional categories $\mathcal{F}$ provided by the category functionality models. If the shape $\mathcal{S}$ does not have adequate functional space to support the functionality of category f in $\mathcal{F}$, the score for f is set to 0.

The combinatorial search takes two input arguments, namely, the shape $\mathcal{S}$ and all functional categories $\mathcal{F}$, written as `CombinatorialSearch(`$\mathcal{S}$`,`$\mathcal{F}$`)`, and returns a list `scores` as the functionality partial matching scores that stores a category score for each functional category. The pseudo-code of the combinatorial search is shown in Algorithm 2, The set of all part groups can be seen as the power set of the set of all parts. In order to efficiently derive all part groups from shape $\mathcal{S}$, we first calculate the total number of subsets, which is 2^n, and then iterate from number 1 to number $2^n - 1$. During each iteration, we analyze the current number in binary form from the 1^{st} to the n^{th} digit. If the value of the k^{th} digit ($k \in \{1, 2, \ldots, n\}$) is 1, then the part with index k will be included in the part group. Otherwise, if the k^{th} digit is 0, the part with index k will be excluded from the part group.

According to the algorithm described above, the computational complexity of the combinatorial search is $O(2^n)$ for both time and space.

4.3.2 Beam Search

The combinatorial search, which traverses through all possible part sets of a shape, is a naive algorithm for partial shape search and can be computationally expensive when the shapes are composed of a large number of parts. Therefore, based on the beam search algorithm, we propose partial matching methods that can be seen as efficient alternatives to the combinatorial search.

Beam search is a heuristic graph search algorithm and was first used in the HARPY speech recognition system by Bruce Lowerre [36]. It is usually applied in the case where the solution space of the graph is relatively large. In order to reduce the space and time complexity of the search, beam search uses a *heuristic function* to select the nodes to be expanded at each level of its search tree. 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. The set of high score nodes at each depth level of the search tree is called the *beam*.

The search tree of beam search is built in a breadth-first search 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.

As shown in Algorithm 3, within the context of functionality partial matching, we define the beam search process as a function taking four input arguments, written as `BeamSearch(`$\mathcal{S}$`,`$\mathcal{F}$`,`w`,start)`, where $\mathcal{S}$ is the entire shape, $\mathcal{F}$ refers to all functional categories, 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, and `start` is the list of initial partial shapes represented by part groups where the beam search starts, defining the state of the root node. Finally, the beam search returns the list `scores` as the functionality partial matching scores.

The functionality models of Hu et al. [23] provide 3D functionality analysis for 15 different kinds of functional categories. However, considering that a hybrid shape is the offspring of two parent shapes, it is not necessary to perform beam search to find the partial shapes that fit each functional category separately. Given a shape $\mathcal{S}$, we only compute the category scores for the following functional categories: (i) the set $\mathcal{F}_{\text{parents}}$ of functional categories of the parent shapes of $\mathcal{S}$ and (ii) categories that have a category score for the entire shape not exceeding a threshold $\theta = 0.9$, 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. To avoid repeating computation, during beam search, we use a hash table `visited` that maps the binary number `hash` denoting each part group to its category scores. If a computed part group is visited again in further beam search, we only need to look up the corresponding scores from the `visited` table.

During each 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 consider further in the search. Shapes that are physically unstable

Algorithm 3: Beam search

Input: $\mathcal{S}$, $\mathcal{F}$, w, `start`
Output: `scores`

```
1  function BeamSearch(S,F,w,start)
2      category_scores ← compute category scores of S
3      foreach f ∈ F do  scores[f] ← Normalize(category_scores[f])
4      foreach f ∈ F_parents do
5          if scores[f] ≥ 0.9 then  continue
6          beam ← start
7          closed ← start
8          while beam ≠ ∅ do
9              open ← ∅
10             foreach g ∈ beam do
11                 foreach p ∈ all parts of shape S do
12                     successor ← remove p from g
13                     if successor ∈ closed then  continue
14                     if visited.HasKey(successor.hash) then
15                         category_scores ← visited[successor.hash]
16                     else
17                         if successor is not part-wise connected then  continue
18                         category_scores ← compute category scores of successor
19                         if successor is not physically stable then
20                             foreach f ∈ F do
21                                 category_scores[f] ← category_scores[f] − 0.1
22                         successor.category_scores ← category_scores
23                         visited[successor.hash] ← category_scores
24                     open ← open ∪ {successor}
25                     closed ← closed ∪ {successor}
26                     if scores[f] < Normalize(category_scores[f]) then
27                         scores[f] ← Normalize(category_scores[f])
28                     if scores[f] ≥ 0.9 then  goto 4
29             beam ← ∅
30             while open ≠ ∅ and |beam| < w do
31                 g ← part group in open with highest score of f
32                 beam ← beam ∪ {g}
33                 open ← open \ {g}
34     foreach f ∈ F do
35         if S does not have functional space for f then
36             scores[f] ← 0
37     return scores
```

are still considered in the search, as such instability 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.

Before the beam search starts, the category scores of all functional categories $\mathcal{F}$ are calculated for the entire shape $\mathcal{S}$. The beam search can then be conducted in two different directions: the reverse direction and the forward direction. A reverse beam search starts from the part group that represents the entire shape and expands the search tree to the next depth level by removing one part from the current part group. A forward beam search, on the contrary, starts from a list of initial part groups, and adds one part to each successor part group during the expansion of the search tree. We use the same threshold score $\theta = 0.9$ defined before to determine the termination of the beam search, i.e., as soon as a partial shape with normalized score of the current functional category f higher than θ is found, the beam search for the current category is stopped and the search proceeds to the next functional category in $\mathcal{F}_{\text{parents}}$. If no partial shape that has a score higher than θ can be found during the beam search, the beam search proceeds until it cannot be further expanded. The functional categories that do not have enough functional space in $\mathcal{S}$ are detected, and their scores are set to 0. We show the pseudo-code of a reverse beam search in Algorithm 3, where the `start` list comprises the entire shape. For a forward beam search, we only need to change line 12 in Algorithm 3 to add a part p to part group g.

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 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 computational complexity for the number of nodes is $O(wn\delta_{\max})$ for both time and space.

4.3.3 Simplified Search

To further mitigate the computational complexity of partial matching, we introduce a less costly algorithm for partial matching that only takes constant time and space, which we call *simplified search*. As shown in Algorithm 4, given a shape $\mathcal{S}$, the

Algorithm 4: Simplified search

```
Input: S, F
Output: scores
function SimplifiedSearch(S,F)
    category_scores ← compute category scores of S
    foreach f ∈ F do
        scores[f] ← Normalize(category_scores[f])
    foreach g ∈ part sets of S from parent shapes do
        if g is part-wise connected and physically stable then
            category_scores ← compute category scores of g
            foreach f ∈ F do
                if scores[f] < Normalize(category_scores[f]) then
                    scores[f] ← Normalize(category_scores[f])
    foreach f ∈ F do
        if S does not have functional space for f then
            scores[f] ← 0
    return scores
```

simplified search takes 3 sets of parts into consideration: the entire shape and each part group coming from one of its parent shapes. We compute category scores of all functional categories $\mathcal{F}$ for the shapes represented by the 3 sets of parts. Similarly as for the beam search, the scores of shapes that do not have enough functional space in $\mathcal{S}$ are set to 0.

4.4 Functionality Scoring

The category scores only reflect the functional plausibility of hybrid shapes for each functional category separately. To comprehensively evaluate the functionalities of hybrid shapes, we perform functionality scoring by aggregating the per-category scores in two manners: considering functional plausibility and multi-functionality.

4.4.1 Functional Plausibility Score

Given the best category scores of a shape computed for each of the relevant functional categories through partial matching and normalized as described above, we

integrate the scores into a single number that indicates the shape's cross-category functional plausibility by simply taking the maximum of the scores. Although different manners of integrating the scores are possible, the maximum score indicates the best partial functionality that the shape possesses, and thus sets a lower bound for the functionality of the shape. An alternative approach such as the sum of scores would be biased by the categories that possess low scores because the shape does not support their functionality. However, such low-score categories do not necessarily indicate that the shape is not functional. One limitation of the maximum is that it does not indicate whether a shape has multiple functionalities. To take into account such multi-functional shapes, the score described next can be used as an alternative to the plausibility score.

4.4.2 Multi-Functionality Score

The multi-functionality score is intended to capture the number of functionalities that a shape can possess. We define the score as the number of different functional categories that are partially supported by the shape. Specifically, we count the number of functional categories for which the shape has a high score, determined by whether the maximum of the functionality partial matching score for the category is above a threshold θ. We use the threshold value $\theta = 0.9$ (the same threshold that is used for terminating the beam search in Section 4.3.2).

Chapter 5

Experimental Evaluation

In this chapter, we discuss the results of experiments comparing different functionality partial matching algorithms for the analysis of functionality of hybrid shapes. We first compare the results of partial matching, using the combinatorial search as a baseline, to that of full matching, to justify the necessity of partial matching in functionality analysis for hybrid shapes. Next, we compare the three low-cost partial matching algorithms, i.e., reverse beam search, forward beam search, and simplified search, and analyze the results of these methods with different sets of parameters. Lastly, we present a comparative summary of the functionality scores and execution time of the three low-cost partial matching algorithms.

To compare the different partial matching algorithms, we perform controlled experiments where we evaluate the functionality of a set of hybrid shapes using the partial matching algorithms. These hybrid shapes were obtained by evolving an initial set of 4 parent shapes belonging to 4 functional categories, namely, *chair*, *desk*, *cart*, and *shelf*, which are included in the functionality models of Hu et al. [23], using the evolution method described in Chapter 3. From all the evolved offspring shapes, 5 are selected for the experiments. All the execution times presented in this and the following chapters are measured on a workstation with an Intel Core i7-6700 3.40 GHz quad-core processor. Our implementation is based on C# and MATLAB, and is largely unoptimized; we believe that it can be sped up significantly with refactoring.

5.1 Partial Matching vs. Full Matching

Table 5.1 presents the functionality analysis results provided by a full matching scheme, i.e., computing category scores for the entire hybrid shapes, and a partial

Table 5.1: Comparison between full matching and partial matching.

	Shape		Partial shape 1		Partial shape 2		Number of shapes evaluated
1		$s_{\text{chair}} = 0.44$ $s_{\text{desk}} = 0.00$		$s_{\text{chair}} = 0.96$		$s_{\text{desk}} = 0.98$	37
2		$s_{\text{chair}} = 0.73$ $s_{\text{shelf}} = 0.69$		$s_{\text{chair}} = 0.96$		$s_{\text{shelf}} = 0.94$	150
3		$s_{\text{chair}} = 0.90$ $s_{\text{cart}} = 0.57$		$s_{\text{chair}} = 0.93$		$s_{\text{cart}} = 0.98$	136
4		$s_{\text{desk}} = 0.63$ $s_{\text{shelf}} = 0.38$		$s_{\text{desk}} = 0.98$		$s_{\text{shelf}} = 0.96$	134
5		$s_{\text{cart}} = 0.68$ $s_{\text{shelf}} = 0.53$		$s_{\text{cart}} = 0.96$		$s_{\text{shelf}} = 0.97$	376

matching scheme, i.e., computing category scores for all partial shapes derived by a combinatorial search. In the first column of the table, i.e., "Shape", we show the hybrid test shape and the category scores computed for the entire shape. In columns "Partial shape 1" and "Partial shape 2", we show the visualized best partial shapes that fit the functional categories of the parent shapes as well as their corresponding category scores. In column "Number of evaluated shapes", we present the total number of shapes evaluated during the combinatorial search.

From Table 5.1 we can conclude that, compared to full matching, partial matching can explicitly find the best-matching partial shapes of the hybrid shapes to specific functional categories and thereby provide more reasonable category scores. For example, Shape 1, which is evolved from one parent shape from the *chair* category and another from the *desk* category, possesses the functionalities from the functional categories of both parents, as seen in the visualization of the shape. The functionality evaluation of the entire shape gives low category scores of $s_{\text{chair}} = 0.44$ and $s_{\text{desk}} = 0.00$. However, the combinatorial search of Shape 1 successfully captures the partial shapes that fit the categories *chair* (Partial shape 1) and *desk* (Partial

shape 2). The partial matching provides high category scores of $s_{\text{chair}} = 0.96$ and $s_{\text{desk}} = 0.98$ that better describe the functionalities of the hybrid shape.

In a few cases, such as for hybrid shapes that have functional patches that support a certain category, it is also possible that the category score computed for the entire shape is high. For example, the category score of the *chair* category for Shape 3 reaches $s_{\text{chair}} = 0.90$ on the full shape.

In these experiments, on average, the combinatorial search of each shape traverses through 166.6 partial shapes and takes 4674 s. Note that all time evaluations for partial matching in this chapter include the time for generating partial shapes.

5.2 Partial Matching Evaluation

In this section, we perform a detailed comparative analysis between reverse beam search, forward beam search, and simplified search. We evaluate the three partial matching algorithms on the same experimental data, and test the beam search algorithms with different parameter settings in order to find the best setting.

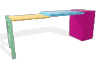
5.2.1 Reverse Beam Search

The reverse beam search has only one parameter, the beam width w, that can be tuned. Different choices of beam width can result in search trees of different sizes (incurring higher or lower computational cost) and also result in different partial matching results (closer or further away from the optimal solution provided by the combinatorial search). In this section, we experiment with three different beam widths, namely, 1, 2, and 3. We present, for each test shape and beam width, the visualizations of the functional partial matches captured by the reverse beam search and their corresponding category scores. We also present the total number of partial shapes evaluated in each search. Note that we do not experiment with beam widths larger than 3, since these would create trees that are too large to fit in memory for the shapes used in our experiments, which are composed of 8.2 parts on average.

Beam Width Selection

The results of functionality analysis provided by the reverse beam search with beam width of 1 are illustrated in Table 5.2. We find that a beam width of 1 is not sufficient

Table 5.2: Results of reverse beam search, with $w = 1$.

	Shape	Partial shape 1	Partial shape 2	Number of shapes evaluated
1		 $s_{\text{chair}} = 0.96$	 $s_{\text{desk}} = 0.88$	17
2		 $s_{\text{chair}} = 0.96$	 $s_{\text{shelf}} = 0.93$	23
3		 $s_{\text{chair}} = 0.90$	 $s_{\text{cart}} = 0.98$	7
4		 $s_{\text{desk}} = 0.88$	 $s_{\text{shelf}} = 0.97$	27
5		 $s_{\text{cart}} = 0.92$	 $s_{\text{shelf}} = 0.96$	10

for finding the optimal functional partial matches of some hybrid shapes. Because the heuristic scores of the partial shapes at each level of the search tree are temporary, the partial shape with highest score may not be expanded to the best partial match. For example, the search fails to find the best partial shapes that match *desk* for Shape 1 and Shape 4, which are illustrated as Partial shape 2 of Shape 1 and Partial shape 1 of Shape 4. Their functionality partial matching score of the *desk* category is $s_{\text{desk}} = 0.88$, which is below the threshold $\theta = 0.9$.

Moreover, the reverse beam search often cannot find the best partial matches found by the combinatorial search. For example, Partial shape 2 of Shape 2, a partial shape matching the *shelf* category, gets a category score of $s_{\text{shelf}} = 0.93$, which is slightly lower than $s_{\text{shelf}} = 0.94$ derived by the combinatorial search. Due to the threshold $\theta = 0.9$ that we set for terminating the search process, the expansion of the search tree can stop before the best partial match is found. The same situation also happens to Partial shape 1 of Shape 3, and Partial shape 1 and Partial shape 2 of Shape 5. However, by visually inspecting the aforementioned partial shapes, we find that such inaccuracy is acceptable, as the form of these partial shapes can still

Table 5.3: Results of reverse beam search, with $w = 2$.

	Shape	Partial shape 1		Partial shape 2		Number of shapes evaluated
1			$s_{\text{chair}} = 0.96$		$s_{\text{desk}} = 0.98$	17
2			$s_{\text{chair}} = 0.96$		$s_{\text{shelf}} = 0.93$	32
3			$s_{\text{chair}} = 0.90$		$s_{\text{cart}} = 0.98$	7
4			$s_{\text{desk}} = 0.98$		$s_{\text{shelf}} = 0.97$	37
5			$s_{\text{cart}} = 0.92$		$s_{\text{shelf}} = 0.96$	10

support the functionality of the matched categories.

The results of functionality analysis provided by the reverse beam search with beam widths of 2 and 3 are illustrated respectively in Table 5.3 and Table 5.4. Compared to the search with beam width of 1, widths of 2 and 3 can find more of the optimal partial shapes. For example, as shown in Table 5.3, the best partial matches to *desk*, which cannot be found by the reverse beam search with beam width of 1, are found by the reverse beam search with beam width of 2 for Shape 1 (Partial shape 2) and for Shape 4 (Partial shape 1), with category score of $s_{\text{desk}} = 0.98$.

On the other hand, a larger beam width can result in a larger computational cost. For example, as shown in Table 5.4, the number of shapes evaluated for Shape 2 is 40, which presents an increase of 25.0 % from 32 in Table 5.3, and an increase of 73.9 % from 23 in Table 5.2. The same situation also happens to Shape 1 and Shape 2. The average numbers of partial shapes evaluated by the reverse beam search with beam width of 1, 2, and 3 are respectively 16.8, 20.6, and 24.2, and correspondingly, the average execution times are 447 s, 534 s, and 626 s. The average execution time of the reverse beam search increases respectively by 19.4 % and 17.2 %, when the beam

Table 5.4: Results of reverse beam search, with $w = 3$.

	Shape	Partial shape 1		Partial shape 2		Number of shapes evaluated
1			$s_{\text{chair}} = 0.96$		$s_{\text{desk}} = 0.98$	19
2			$s_{\text{chair}} = 0.96$		$s_{\text{shelf}} = 0.93$	40
3			$s_{\text{chair}} = 0.90$		$s_{\text{cart}} = 0.98$	7
4			$s_{\text{desk}} = 0.98$		$s_{\text{shelf}} = 0.97$	45
5			$s_{\text{cart}} = 0.92$		$s_{\text{shelf}} = 0.96$	10

width is increased from 1 to 2 and from 2 to 3.

Based on the results described above, we can conclude that, as far as our test shapes are concerned, experimentally 2 is the best beam width for the reverse beam search. The beam width of 2 not only allows the reverse beam search to find reasonable partial matches, but also requires less execution time, which can be seen as a trade-off between the analytical accuracy and the execution time of the method.

5.2.2 Forward Beam Search

The forward beam search has two parameters that can be tuned: the beam width and the partial shapes from which the search starts. Similar to the reverse beam search, we find through experiments that 2 is the best beam width for the forward beam search, when evaluating our test shapes. Therefore, in this section, we only discuss the optimization of the initial partial shapes when using a beam width of 2. The initial partial shapes for the search can be defined by a parameter that selects the number of parts n_{parts} that the initial partial shapes should contain. The forward

Table 5.5: Results of forward beam search, with $n_{\text{parts}} = 1$.

	Shape	Partial shape 1		Partial shape 2		Number of shapes evaluated
1			$s_{\text{chair}} = 0.96$		$s_{\text{desk}} = 0.98$	18
2			$s_{\text{chair}} = 0.96$		$s_{\text{shelf}} = 0.94$	45
3			$s_{\text{chair}} = 0.90$		$s_{\text{cart}} = 0.96$	42
4			$s_{\text{desk}} = 0.98$		$s_{\text{shelf}} = 0.45$	42
5			$s_{\text{cart}} = 0.92$		$s_{\text{shelf}} = 0.95$	51

beam search starts from the set of all partial shapes with n_{parts}, i.e., the shapes derived from all possible combinations of n_{parts}. We experiment with three different values for n_{parts}: 1, 2, and 3. The largest n_{parts} is set to 3 because the parent shapes in our dataset are composed of at least 3 parts, so that n_{parts} will not exceed the number of parts in parent shapes. For the experimental results, we present for each test shape the category scores of the functional partial matches captured by the forward beam search and the total number of partial shapes evaluated in each search.

Initial Partial Shapes Selection

The functionality analysis results of forward beam search starting from the partial shapes represented by 1-combinations, 2-combinations, and 3-combinations of parts are respectively shown in Table 5.5, Table 5.6, and Table 5.7.

If a partial shape does not contain enough parts, its heuristic functionality score, which is based on the functional patch detected in the shape, can be very low, as no patch that supports any functionality can be found in such a partial shape. Therefore,

Table 5.6: Results of forward beam search, with $n_{\text{parts}} = 2$.

	Shape	Partial shape 1		Partial shape 2		Number of shapes evaluated
1			$s_{\text{chair}} = 0.96$		$s_{\text{desk}} = 0.98$	14
2			$s_{\text{chair}} = 0.96$		$s_{\text{shelf}} = 0.94$	39
3			$s_{\text{chair}} = 0.90$		$s_{\text{cart}} = 0.96$	37
4			$s_{\text{desk}} = 0.98$		$s_{\text{shelf}} = 0.45$	41
5			$s_{\text{cart}} = 0.90$		$s_{\text{shelf}} = 0.94$	75

the ranking of these scores can be unreasonable, which can further result in unreasonable partial shapes found by the search. For example, as shown in Table 5.5 and Table 5.6, the forward beam search starting from 1-combinations and 2-combinations cannot find a reasonable partial shape for *shelf* for Shape 4, of which the category score is $s_{\text{shelf}} = 0.45$. This issue can be addressed with the forward beam search starting from initial partial shapes having more parts, for example, in the forward beam search starting from 3-combinations, as shown in Table 5.7, a more reasonable partial shape for *shelf* can be found for Shape 4 with a higher category score $s_{\text{shelf}} = 0.96$, as illustrated with Partial shape 2 of Shape 4.

On the other hand, since a threshold score $\theta = 0.9$ is used to stop the expansion of the search tree, for many hybrid shapes, the forward beam search cannot find the best partial shapes found by the combinatorial search, for example, Partial shape 1 and Partial shape 2 of Shape 3, Partial shape 2 of Shape 4, and Partial shape 1 and Partial shape 2 of Shape 5. Although such inaccuracy is acceptable, this phenomenon is still much more common in the forward beam search as compared to the reverse

Table 5.7: Results of forward beam search, with $n_{\text{parts}} = 3$.

	Shape	Partial shape 1		Partial shape 2		Number of shapes evaluated
1			$s_{\text{chair}} = 0.96$		$s_{\text{desk}} = 0.98$	14
2			$s_{\text{chair}} = 0.96$		$s_{\text{shelf}} = 0.94$	43
3			$s_{\text{chair}} = 0.90$		$s_{\text{cart}} = 0.96$	39
4			$s_{\text{desk}} = 0.98$		$s_{\text{shelf}} = 0.96$	32
5			$s_{\text{cart}} = 0.92$		$s_{\text{shelf}} = 0.95$	66

beam search.

There is no obvious correlation that can be found between the execution time of the forward beam search and the number of parts of the initial partial shapes in our experiments. The average number of shapes evaluated in the forward beam search starting from 1-combinations, 2-combinations, and 3-combinations of parts are respectively 39.6, 41.2, and 38.8, which correspond to execution times of 1123 s, 1440 s, and 1274 s.

The experimental results show that, the initial partial shapes of the forward beam search should contain at least 3 parts to ensure that reasonable partial matches can be found for each functional category.

5.2.3 Simplified Search

The simplified search provides a higher computing speed at the cost of lower analytical accuracy. For each test shape, a simplified search evaluates 3 shapes: the entire shape and each partial shape coming from the parents of the test shape. Table 5.8 presents the functionality analysis results of the simplified search. In the table, we present

Table 5.8: Results of simplified search.

	Shape	Partial shape 1		Partial shape 2	
1			$s_{\text{chair}} = 0.08$		$s_{\text{desk}} = 0.98$
2			$s_{\text{chair}} = 0.96$		Parts not connected
3			$s_{\text{chair}} = 0.00$		$s_{\text{cart}} = 0.86$
4			$s_{\text{desk}} = 0.98$		$s_{\text{shelf}} = 0.31$
5			Parts not connected		Parts not connected

the visualizations of the partial shapes captured by the simplified search and their corresponding category scores.

For some of the hybrid shapes, the best partial shapes for the parent functional categories can be found by a simplified search, such as Partial shape 2 of Shape 1 and Partial shape 1 of Shape 2. However, most of the partial shapes from the parent shapes possess issues that prevent good partial matches to their functional categories. For example, in Table 5.8, Partial shape 1 of Shape 1 and Partial shape 2 of Shape 4 are physically unstable, with category scores of $s_{\text{chair}} = 0.08$ and $s_{\text{shelf}} = 0.31$, respectively, and Partial shape 1 and Partial shape 2 of Shape 3 are incomplete, with category scores of $s_{\text{chair}} = 0.00$ and $s_{\text{cart}} = 0.86$, respectively.

Another important issue shown in the results of the simplified search is that the captured partial shapes may not be part-wise connected, which prevents the category scores from being computed for these partial shapes, such as Partial shape 2 of Shape 2 and Partial shape 1 and Partial shape 2 of Shape 5.

Note that, for a test shape, the category scores computed for the partial shapes do not necessarily represent its final functional plausibility score, as the simplified search

Table 5.9: Analysis of partial matching schemes.

	Shape	Reverse beam search			Forward beam search			Simplified search		
		s_p	s_m	t	s_p	s_m	t	s_p	s_m	t
1		0.98	2	420	0.98	2	457	0.98	1	56
2		0.96	2	796	0.96	2	1407	0.96	1	54
3		0.98	2	221	0.96	2	1586	0.90	0	68
4		0.98	2	950	0.98	2	1042	0.98	1	54
5		0.96	2	284	0.95	2	1878	0.68	0	58

also takes into account the entire shape. For example, the functional plausibility score of Shape 3 is derived from the category score of *chair* for the entire shape, which is $s_{\text{chair}} = 0.90$.

5.2.4 Best Partial Matching Algorithm

In this section, we summarize the functionality analysis results of the reverse beam search, the forward beam search, and the simplified search, to determine the best method to be used in practice. The results of each algorithm are analyzed based on the best parameter settings as discussed in Section 5.2.1 and Section 5.2.2, i.e., the reverse beam search uses a beam width of 2, and the forward beam search uses a beam width of 2 and starts the tree expansion from initial partial shapes with 3 parts. In Table 5.9, for each test shape, we summarize the functionality analysis results of the different partial matching algorithms in three aspects: (i) the functional plausibility score s_p, which is derived from the highest best category score captured by the partial

matching algorithm, as described in Section 4.4.1; (ii) the multi-functionality score s_m, which is derived from the number of category scores that exceed the threshold $\theta = 0.9$, as described in Section 4.4.2; (iii) the execution time t of the partial matching algorithm, in seconds.

From Table 5.9 we find that, in terms of the functional plausibility score s_p, the reverse beam search and the forward beam search present similar results. The average functional plausibility scores derived by the reverse beam search and the forward beam search are 0.97 and 0.96. The slightly higher average score derived by the reverse beam search indicates that the partial matches captured by the reverse beam search are a bit more reasonable than those captured by the forward beam search. However, the simplified search cannot capture as reasonable partial matches as the other two algorithms. The average functional plausibility score derived by the simplified search is 0.90.

In terms of the multi-functionality score s_m, both the reverse beam search and the forward beam search can capture the number of functionalities that the test shapes possess. Each test shape is generated from two parent shapes of single functional category, so that the multi-functionality score of each test shape is 2. However, the simplified search is unable to reveal the multi-functionality of the test shapes. The multi-functionality score computed by the simplified search is either 1 or 0.

From the perspective of execution time, the reverse beam search and the forward beam search take on average 534 s and 1274 s, respectively. The reverse beam search not only presents a higher analytical accuracy, but also is 58.1 % faster than the forward beam search. Although the simplified search takes only 58 s on average, being much faster than the reverse beam search, it does not provide reasonable analysis results for most of the test shapes.

Therefore, according to this summary, we can conclude that the reverse beam search with beam width of 2 is the best partial matching scheme, of which a visualized example is illustrated in Figure 1.1, providing both reasonable analysis results and execution time. More visualized examples can be found in Appendix B.

Chapter 6

Applications

In this chapter, we present two applications of functionality partial matching, namely, 3D functionality-aware model evolution and 3D data augmentation for shape segmentation.

6.1 3D Functionality-Aware Model Evolution

Functionality-aware model evolution is the key application presented in this thesis that benefits from functionality analysis. Without proper functionality analysis, the evolutionary shape modeling framework described in Chapter 3 can still generate a large amount of shapes that are not functionally plausible, as shown in Figure 6.7(a), even when incorporating functionality labels to constrain the evolutionary process. With functionality partial matching integrated into the evolutionary framework and the functional plausibility score described in Section 4.4.1, unreasonable shapes generated during the evolutionary process are filtered out. Based on this method, we develop a functionality-aware modeling tool that enables functionality-aware model evolution, which is capable of generating a large volume and variety of shapes that are functionally plausible, including hybrid shapes.

In this section, we present the results of the functionality-aware modeling tool used in different modeling scenarios as well as the evaluation and analysis of the modeling results. We also provide comparisons to existing modeling tools.

We experiment with the functionality-aware modeling tool using the parameter settings described in preceding chapters. The experiments were conducted on 3D objects belonging to 15 functional categories including *chair*, *desk*, *cart*, and *shelf*.

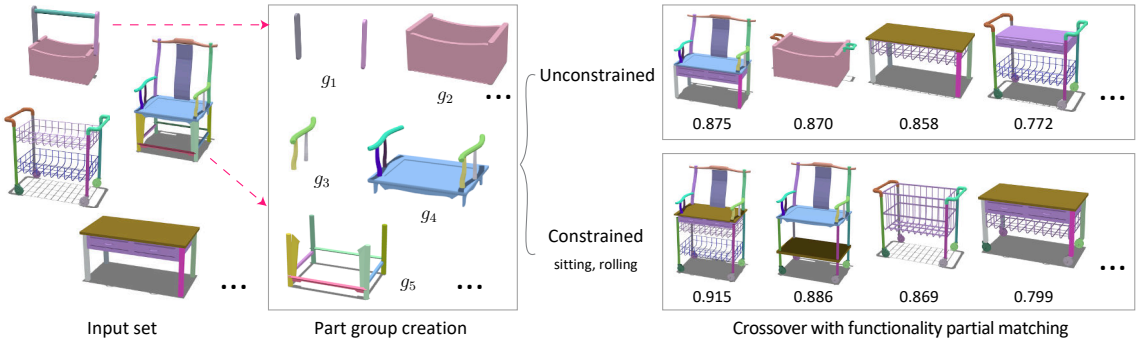


Figure 6.1: Starting from a set of segmented 3D shapes, we construct part groups for each shape, where a part group g_i is composed of a combination of one or more shape parts (left). We apply crossover and mutation operations, i.e., part group exchange and part group insertion, as described in Chapter 3, to create a variety of novel shapes (right). The evolutionary process can be performed in an unconstrained or constrained manner. In the constrained evolution, the user prescribes functionalities that should appear in the output shapes, e.g., *sitting* and *rolling*. The resulting shapes are then ranked according to their functional plausibility. Finally, the user can select shapes to be part of the next generation for further evolution.

The 15 functional categories are adopted from the work of Hu et al. [23]. We also predefine 10 different functionality part labels, such as *sitting*, *placement*, *storage*, and *rolling*. Evolving one generation with random crossovers from a population of 4 shapes with a pool of 17 part groups (including null part groups) and 8 functionality labels can generate 31 shapes. With user constraints (constraining with two functionality labels) that filter out the unwanted functionalities, on average 6 offspring shapes are produced during the generation. If a simplified search is adopted for functionality partial matching, the time needed to evolve one generation from this population is about 6 min, where evolutionary operations take about 1 min (about 16.7 % of the computation time for a generation) and functionality partial matching takes 5 min (the remaining 83.3 % of the computation). The time needed to compute the functional plausibility score of a single offspring is 50 s on average with the simplified search. If a reverse beam search is employed, the time for evolving one generation from the same population is about 55 min, where evolutionary operations take about 1 min (about 1.8 % of the computation time for a generation) and functionality partial matching takes 54 min (the remaining 98.1 % of the computation). The time needed to compute the functionality scores of a single offspring with the reverse beam search is on average 9 min. The largest population we tested includes 17

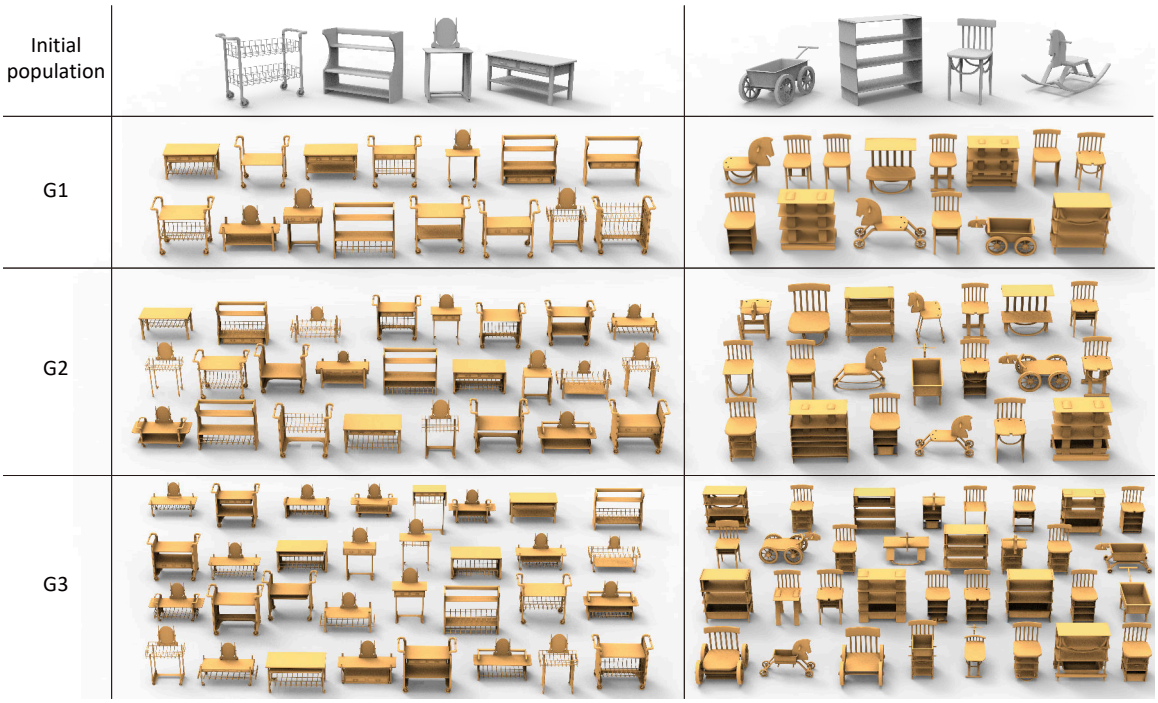


Figure 6.2: A gallery of modeling results from unconstrained evolution obtained with our functionality-aware approach. The first row shows two initial populations of four objects each, one per column. The next three rows show selected offspring from subsequent generations. The generations contain 78, 113, and 52 shapes in total for the set in the first column, and 31, 96, and 45 shapes for the set in the second column.

shapes, taking 2.2 h to evolve one generation (with user constraints and the simplified search). Pre-processing of an initial population of 4 shapes takes on average 15 min, including functional proto-patch mapping and part group creation.

6.1.1 Modeling Scenarios

Our functionality-aware modeling tool can perform model evolution under two modeling scenarios: (i) *unconstrained modeling*, where an open-ended exploration is conducted over an initial set of input shapes without any user intervention; (ii) *constrained modeling*, where a few user constraints are incorporated, such as the selection of functionalities that the output shapes should possess. In both evolutionary modeling scenarios, the evolved shapes are presented to the user according to the descending order of functional plausibility score. An overview of our modeling tool is

illustrated in Figure 6.1. Unless otherwise indicated, we mimic the process in which users would handpick interesting offspring from a large and diverse set of shapes, and show for each figure in this section the selected shapes from the evolved populations.

Unconstrained Modeling

The results of unconstrained evolution are shown in Figure 6.2. It can be observed that many of the evolved shapes are truly cross-category, as they combine multiple functionalities from two or more parents, e.g., the movable shelves and table-shelves in the first column, or the sittable shelves in the second column. The more generations are evolved, the more shapes with complex and mult-functional features are generated. For example, from the third generation (G3) for both columns, we can find hybrid shapes that have parts from all four parent shapes of the initial population.

We also observe in the results that our tool can handle shapes that do not belong to the predefined functional categories. For example, in the second column of Figure 6.2, we see a horse-shaped toy in the initial population that does not belong to any of the predefined functional categories. However, by manually labeling the toy's parts with appropriate functionality labels such as *rocking* and *sitting*, our evolution can combine the parts from this object with other shapes to create offspring shapes that preserve these functionalities. Finally, we note that when the functionality of a parent shape is transferred to an offspring, the functionality is preserved without obstructions to the functional spaces.

Constrained Modeling

Our tool works most effectively when the evolution is guided by user constraints. In the constrained modeling scenario, the user can first come up with a design goal such as "generating shapes that enable storage and transportation", and then select appropriate functionality labels as constraints to ensure that these functionalities appear in the offspring shapes. During evolution, our tool provides concrete examples of shapes satisfying the constraints. The user can examine the generated shapes to select and further evolve the preferred ones.

We show the results of constrained evolution in Figure 6.3, in which we present three generations of offspring shapes evolved from one set of input shapes. It can be

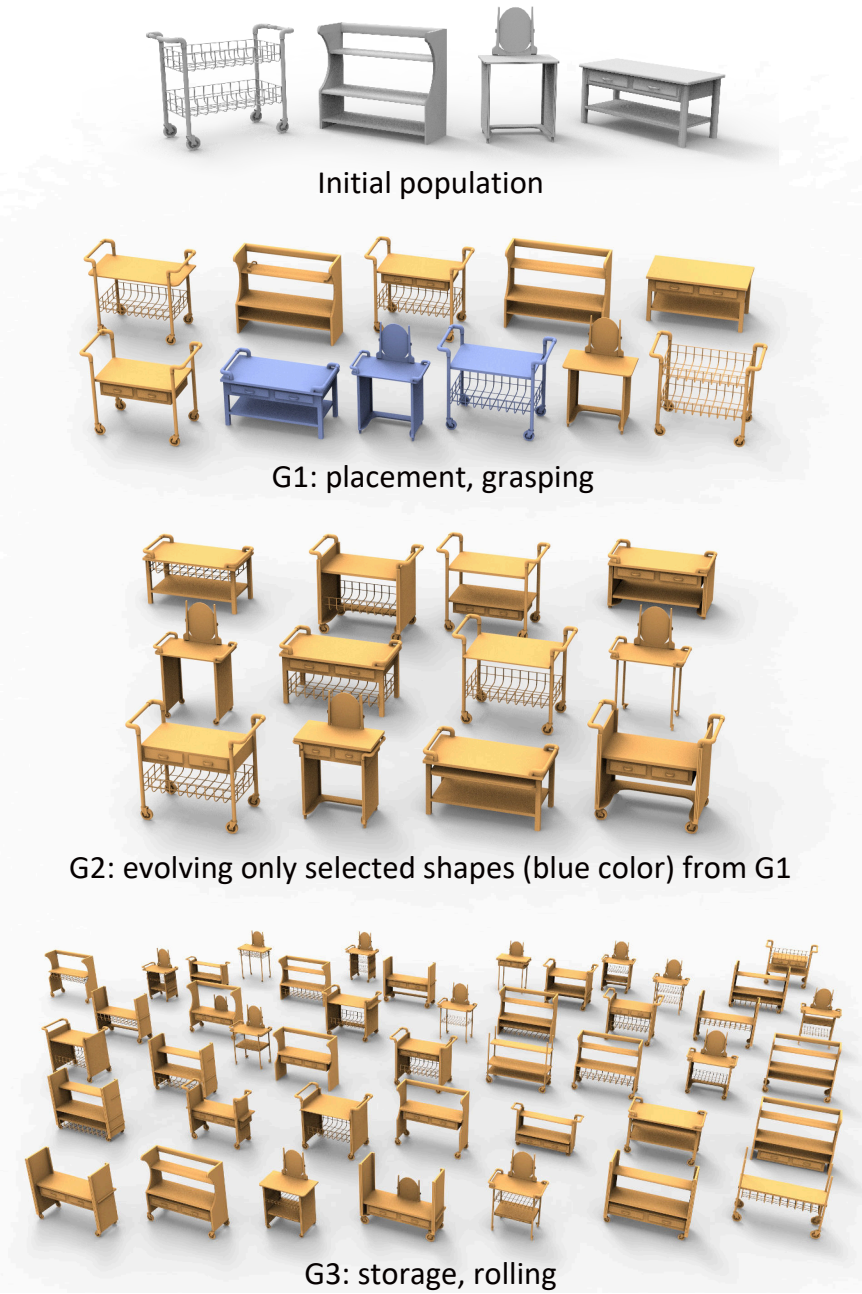


Figure 6.3: Results of constrained evolution by our functionality-aware modeling tool. The user evolves the initial population by constraining the offspring shapes with the functionality labels *placement* and *grasping*, obtaining the first generation (G1). The user then selects the shapes marked in blue in the first generation (G1) to be further evolved, to get the second generation (G2). Finally, the functionality constraints *storage* and *rolling* are included as new preferences into the evolution of all the shapes, to obtain the third generation (G3).

Figure 6.4: Starting from a heterogeneous collection of four shapes (in gray) as initial population, our functionality-aware modeling tool is able to generate a variety of offspring shapes (in yellow) with a combination of constrained and unconstrained evolution. Some of the offspring shapes exhibit forms of cross-category structure breaking.

observed that the user-specified functionalities are preserved in all of the offspring shapes but are enabled in various manners. For example, in offspring shapes from the first generation (G1) and the second generation (G2), the functionalities of *placement* and *grasping* are enabled by combining *placement* of shelf planks or desktops with *grasping* of handles; in offspring shapes from the third generation (G3), the functionalities of *storage* and *rolling* are enabled by adding wheels to storage objects like baskets and shelves, with functionalities like *placement* and *grasping* also inherited from the parent shapes in the second generation (G2).

Moreover, Figure 6.4 presents a set of hybrid shapes generated with a combination of constrained evolution and unconstrained evolution. In the first generation, *placement* is the only constraint specified by the user for evolution, to ensure that the generated shapes can be used to hold objects. Then, a few of the resulting shapes are selected by the user and are further evolved in an unconstrained manner so as to obtain more object variations. This demonstrates the flexibility of the evolution approach where different types of modeling scenarios can be combined to guide the evolution. We also note that this varied set with 19 shapes was obtained by evolving only four input shapes.

6.1.2 Evaluation and Comparisons

We evaluate our evolutionary modeling tool from different perspectives, including the ability to break structures, the scalability of the evolutionary process, and the ranking

Table 6.1: Statistics of structure breaking in the evolved shapes.

Input set	M	G	$\%\beta$
Figure 6.2 left	4	24	45 %
Figure 6.2 right	4	17	6.4 %
Figure 6.4	4	21	25 %

of the functionality scores of the evolved shapes. We also compare the evolved shapes provided by our method to those provided by the methods of Zheng et al. [65] and Fu et al. [15].

Structure Breaking

It can be observed from the evolutionary modeling results that, the ability to break structures, such as symmetries, during part composition is a unique feature of our cross-category modeling tool, which allows the evolution to introduce variations in the structure of the generated shapes. In Table 6.1, we report percentages of offspring shapes from our model evolution which exhibit breaking of symmetries possessed by their parents: M denotes the number of 3D objects in the input set, G denotes the total number of part groups obtained from the input set, and $\%\beta$ denotes the average percentage of the offspring shapes produced by the evolution which break symmetries from their parents. The percentage of offspring shapes exhibiting symmetry breaking can vary considerably, e.g., 45 % for the set shown on the left of Figure 6.2, 6.4 % for the set on the right of Figure 6.2, and 25 % for the set shown in Figure 6.4.

Scalability

We evaluate the scalability of our evolution in two aspects: (i) the number of shapes evolved and (ii) the number of functionalities given by the user as constraints. In Figure 6.5, we see that functionally plausible shapes are produced from requesting not only a small number of functionalities such as 2 or 3, as shown in Figure 6.5(a) and Figure 6.5(b), but also a larger number of functionalities such as 5, as shown in Figure 6.5(c). We also observe that the generated shapes are true hybrid shapes that can serve multiple functionalities. For example, the hybrid chairs in Figure 6.5(c),

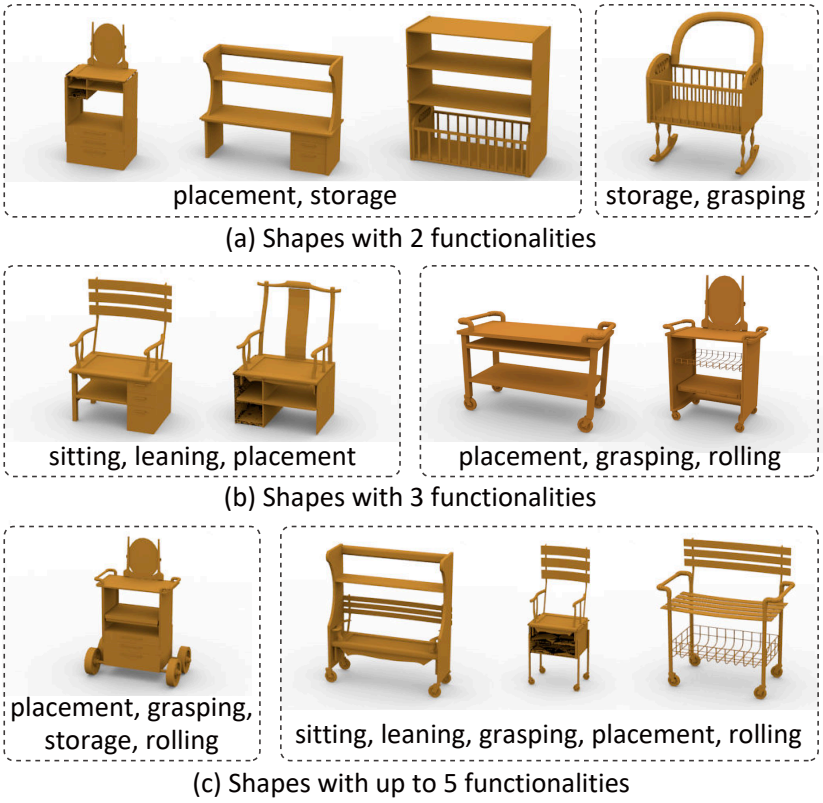


Figure 6.5: Results of model evolution where the objects are constrained to possess 2, 3, and up to 5 functionalities.

besides enabling *sitting* and *leaning*, also enable the *placement* of objects, and *rolling* and *grasping* for transportation.

In Figure 6.6, we present three sets of hybrid shapes obtained by evolving the same initial population of 17 shapes by choosing different sets of functionality constraints, namely, *sitting* and *leaning*, *placement* and *storage*, and *rolling* and *grasping*. We use a larger input population here than those used in the previous experiments so that the scalability of the method can be assessed. From the results, we observe that the evolution generates large sets with hundreds of novel shapes which are all constrained by the user guidance.

Ranking Scores

When a large number of offspring shapes are generated, as shown in Figure 6.2, Figure 6.3, Figure 6.4, and Figure 6.6, a ranking measure is especially important so that

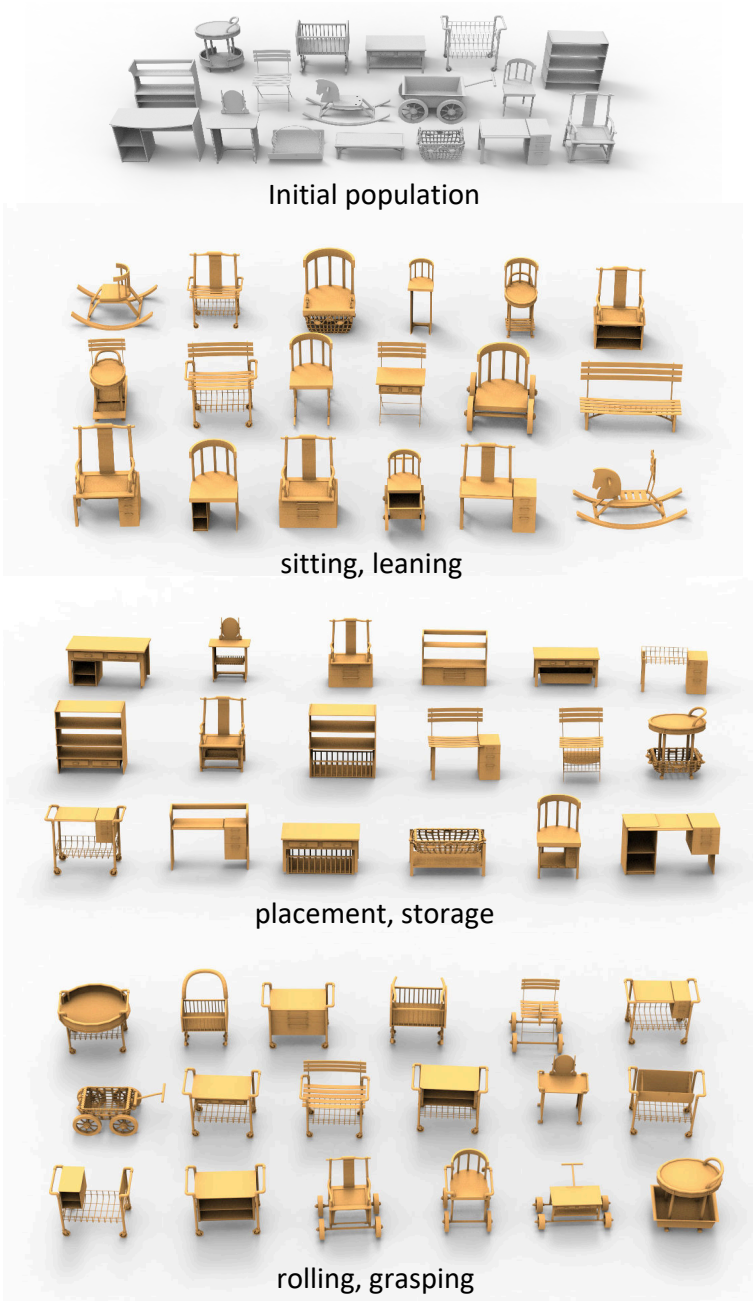


Figure 6.6: Results of model evolution from a large initial population (17 shapes) and with various functionality constraints to demonstrate scalability. The following populations are generated: *sitting* + *leaning* with 141 shapes, *placement* + *storage* with 234 shapes, and *rolling* + *grasping* with 258 shapes. Only the top 18 shapes for each set are shown, according to the ranking by functional plausibility.

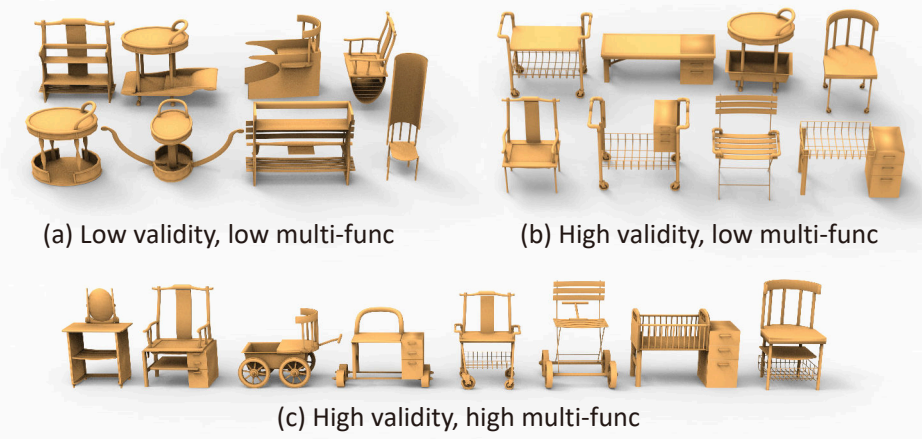


Figure 6.7: Comparison of ranking scores. Three sets of objects with different levels of priority for plausibility and multi-functionality measures, i.e., from low (a) to high (c).

the most plausible objects are presented to the user first. In Figure 6.7, we compare the functional plausibility and multi-functionality measures as choices for ranking the shapes from Figure 6.6. We observe a noticeable difference in the objects with different scores, where objects with high values for the two measures appear more functional than objects with low scores. In addition, objects with high multi-functionality scores tend to combine several functionalities. Thus, these measures enable the user to save time by only inspecting the most promising prototypes. The user, focusing either on general functional validity or mixing of multiple functionalities, can choose one measure over the other depending on whether the focus of the exploration is to obtain objects that are functionally plausible in general, or to retrieve objects that mix multiple functionalities.

Comparisons

In Figure 6.8, we compare our method to that of Zheng et al. [65] by applying our model evolution to one of their input sets, with the same shape segmentations. Recall that their method is designed to preserve a specific type of three-part symmetric support substructure (sFarr) in the input and limited to six combination rules, while our method is more generic and even allows symmetry breaking. We observe that our generic hybridization/crossover approach, i.e., without explicitly modeling or enforcing any specific symmetries or support structures, can also generate the types of

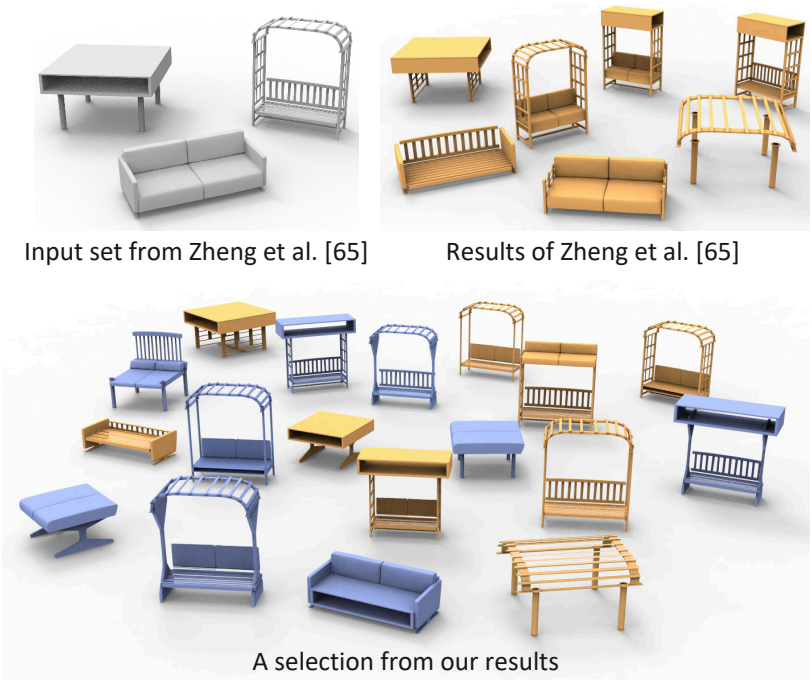


Figure 6.8: A comparison of our shape generation results to those from Zheng et al. [65], on an input set from their work. The set of offspring shapes generated by our method contains not only shapes producible by their method (shapes in yellow), but also other shapes (in blue) which their method cannot produce for various reasons discussed in the text.

shapes that their method can. At the same time, their method is able to generate cross-category hybrids, but the functionality that is explicitly preserved in all of the generated shapes involves mainly support structures that are captured by symmetric functional substructures. On the other hand, our modeling tool is able to achieve more general part recombinations and produce shapes exhibiting larger functional variations, e.g., the combination of *sitting* and *storage* that appears in the couch hybridized with a bench and a shelf, where two different parts are attached to the supporting structure. Moreover, we remark that their method is able to generate interesting variations for this input set where all shapes have symmetric support structures, but their method is not applicable to more general shapes like the asymmetric desks that we evolved in Figure 6.4 or Figure 6.6.

In Figure 6.9, we compare our method to that of Fu et al. [15], which can synthesize a cross-category, functional hybrid to fulfill the affordance constraint defined by an input human pose. The comparison results demonstrate that our evolutionary

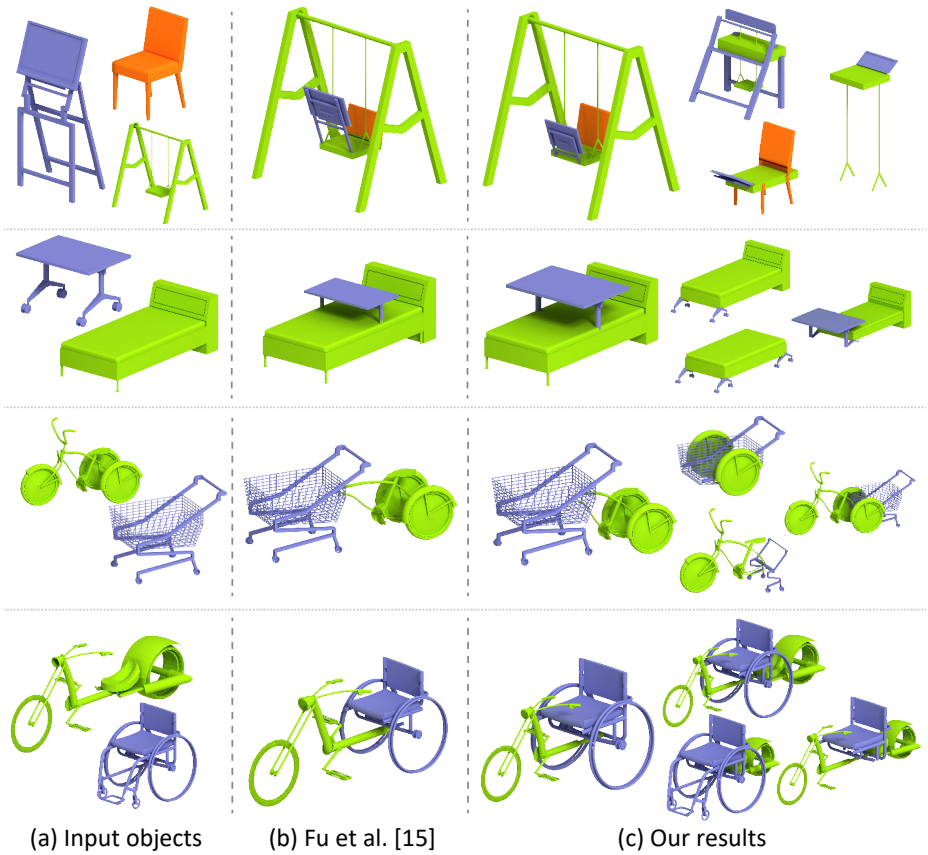


Figure 6.9: Comparison to functional hybrid generation by Fu et al. [15]. In each row, we show the 3D shapes identified by their method (left) that match a human pose and the hybrid shape produced (middle). Using the same 3D shapes as the initial population, our method is able to generate a more diverse set of hybrids (right), including one that well resembles the outcome from their method, without a human pose as constraint.

modeling tool is able to produce similar hybrids, without specifying a human pose or explicit affordance constraints, as well as other hybrids which are also functionally plausible. On the other hand, an input human pose does narrow down the search for potential parent shapes and part placements; the resulting synthesized shape would more closely serve a specific target functionality.

6.2 3D Data Augmentation

We have shown that our functionality-aware model evolution is able to generate a large and diverse population of functionally plausible offspring shapes. However,

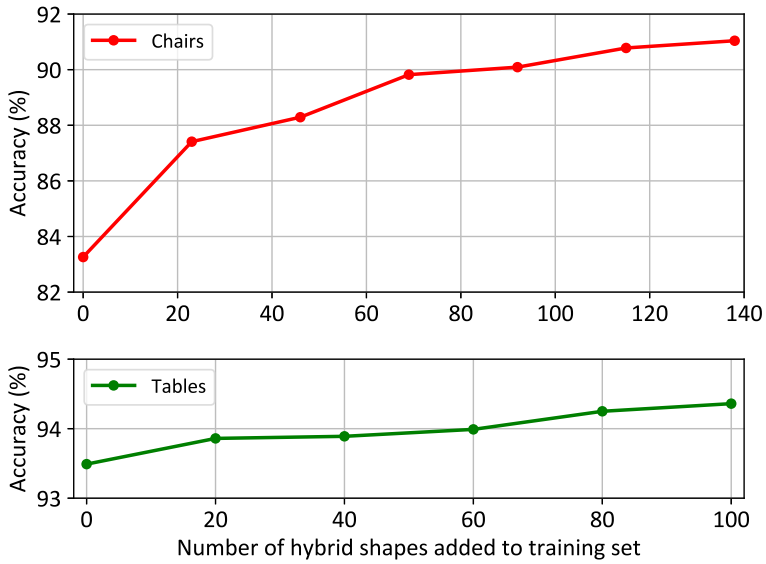


Figure 6.10: Results showing improved accuracy via data augmentation for learning shape segmentation, using PointNet, for two sets of shapes. The ShapeNet training set is augmented progressively with shapes evolved using our tool. Please refer to the text for details.

from the perspective of modeling and design exploration, there are usually only a few shapes that can inspire new designs in an evolved population. On the other hand, data augmentation, which is aimed at boosting the performance of learning-based shape analysis schemes, is a venue where we can utilize most, if not all, of the evolved shapes from our modeling tool. We expect these shapes to serve as useful training data since they are both plausible, partially resembling potential test shapes, and diverse, providing a better coverage of the distribution of test shapes.

To demonstrate the potential of our method for data augmentation, we use the generated offspring to augment training data for one key application: shape segmentation of partial shapes. To create a set of labeled shape segmentations, we assign labels to the already segmented parent shapes, which usually constitute a small set that can be manually labeled by a human in a short time. Then, we keep the labels assigned to each part as we evolve the parents into a large set of segmented and labeled shapes. We use this data to augment the training data for shape segmentation.

Specifically, we evolve two input populations. The first one is composed of 7 chairs and 9 shapes from other classes that add diversity to the set, resulting in

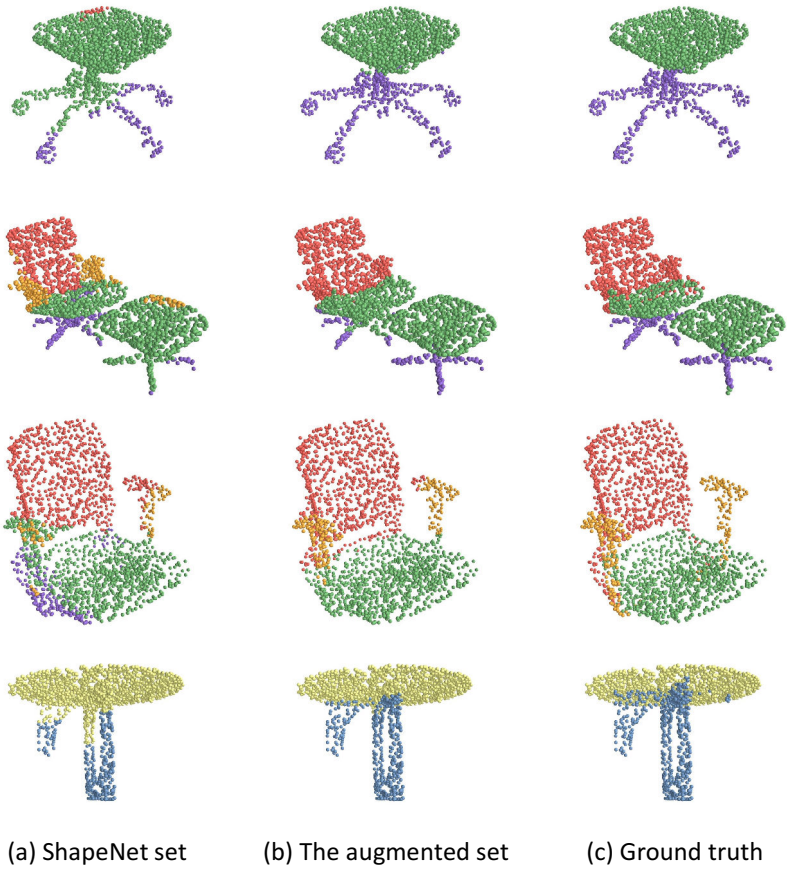


Figure 6.11: Visual results of PointNet segmentation on partial test shapes, using ShapeNet training set vs. the augmented training set (ShapeNet + our shapes).

an evolved population of 138 hybrid chairs, since we constrain the offspring to the functionality of chairs. The second one is composed of 10 tables and 5 shapes from other classes, resulting in 100 hybrid tables. For the experiments, we use PointNet [49] for segmenting point clouds, where we uniformly sample our shapes with 2048 points. To show how our data can aid in learning to segment diverse shapes, especially when partial matching is needed, we create a test set of partial shapes, where parts are randomly removed from ShapeNet [6] shapes. We augment the ShapeNet training set used by PointNet with our training shapes composed of hybrid shapes, and evaluate the predicted segmentations on the test set of partial shapes.

Figure 6.10 shows the results of data augmentation with increasing numbers of shapes from our training set, where we evaluate the segmentations by measuring the

label accuracy. We observe how each additional batch of shapes helps to improve the accuracy on the chair set, with a gain of around 8 %. The accuracy is also slightly improved for tables, which typically have a simpler structure. Moreover, the data points with $x = 0$ represent the accuracy obtained when only the ShapeNet training set is used for learning. We notice how the accuracy is much lower than when our more diverse shapes are used for training, demonstrating that the diverse, plausible shapes do provide additional information for the learning of the deep network.

We confirm this reasoning by visually inspecting the results, such as the set of examples shown in Figure 6.11. Note how the segmentations of these partial shapes have considerable errors when only the ShapeNet set is used for training. On the other hand, we obtain a much refined segmentation, closer to the ground truth, when our evolved shapes are used for data augmentation. For example, the chair-stool combo in the second row is missing the side handles, but the ShapeNet data alone still leads to a segmentation with this label. In the third row, the leg label appears below the seat, while with our data we obtain a better prediction of chair seats and back rests.

Chapter 7

Conclusion

We present a novel functionality analysis method, based on functionality partial matching, for evaluating functionalities of hybrid shapes. With functionality partial matching, the category functionality models of Hu et al. [23] can be localized to the part/patch level to enable finding the partial shape from a hybrid shape that best fits to a known functional category.

We incorporate our functionality analysis into the "fit and diverse" set evolution framework of Xu et al. [63] and present the first functionality-aware and user-in-the-loop modeling tool to evolve a set of 3D objects, aimed at producing large and diverse sets of functionally plausible offspring. Rather than restricting part exchange to objects within the same category, as done by Xu et al. [63] and all the works so far on structure-preserving modeling [41], our method excels at generating cross-category hybrids while allowing structure breaking. In the end, our evolutionary modeling tool makes a promising step towards 3D content generation to generate a large number of shapes with intra-class and inter-class variety, producing generations and generations of within-category shapes and cross-category hybrids via controlled stochastic part exchange. The generated shapes show the potential for data augmentation that improves 3D shape segmentation with atypical inputs.

7.1 Limitations

There is still plenty of room to improve our current method from a technical perspective. Our main technical limitations stem from our set goal of only producing rough design prototypes, where the emphasis has been on functional properties of the offspring shapes, not their precise geometries. Specifically, our part exchange

procedure only applies non-uniform part group scaling, while free-form deformations should allow a richer variety of offspring. Moreover, our current part connection mechanisms still lack an understanding of shape semantics and are unable to resolve topological mismatches. At the same time, our current functional plausibility score and constrained evolution only offer a starting point for further investigation and development. Last but not the least, we provide the option of computing plausibility scores with a detailed functionality partial matching. However, the computation relies on an expensive search procedure. Thus, directions for future work include more efficient manners of detecting partial functionalities of shapes, including the use of learning-based methods.

Moreover, real-world design tasks are often characterized by various spatial, aesthetic, and functionality constraints. Our current modeling tool is inherently constrained by selection of the initial population, the available functionality models, as well as user preferences expressed during a modeling session. Our modeling tool does not incorporate other types of aesthetic constraints, such as style.

7.2 Future Work

In future work, besides addressing the above limitations, we would also like to explore other functionality modeling paradigms. For example, instead of performing functionality-preserving style transfer, as in the work by Lun et al. [37], we can invert the problem to style-preserving *functionality transfer*: to transfer the extracted functionalities from a given shape to another shape, while preserving its stylistic features. Our evolution-based modeling offers a partial solution to this problem. Specifically, a part exchange between two shapes is implicitly transferring functionalities associated with the exchanged part groups between the shapes.

Another interesting problem to study is *functional analogy*, i.e., to synthesize a new shape $\mathcal{S}_B{}'$ from an input $\mathcal{S}_B$ by adding functionalities to $\mathcal{S}_B$, so that the pair, $\mathcal{S}_B$ and $\mathcal{S}_B{}'$, has functionality differences analogous to those between two given shapes $\mathcal{S}_A$ and $\mathcal{S}_A{}'$.

Moreover, with the development of measures for estimating functional similarity, validity, and novelty, we could treat functionality as a relative attribute and learn metric spaces for functionality analysis tasks. The work by Yumer et al. [64] is able to learn semantic attributes to enable continuous shape deformation in the learned

semantic spaces. It would be interesting to explore whether similar approaches are possible for functional attributes and deformation. The key challenge is how to parameterize part exchange operations, which are inherently discrete. Generally, we do not yet know how to construct a continuous "generative functional space" for either a homogeneous or a heterogeneous shape collection. On a positive note, the modeling tool developed in our work can produce a lot of 3D shape data to support this pursuit.

List of References

[1] E. Balashova, V. Singh, J. Wang, B. Teixeira, T. Chen, and T. Funkhouser, "Structure-aware shape synthesis," in *Proceedings of the International Conference on 3D Vision*, 2018, pp. 140–149.

[2] E. Bar-Aviv and E. Rivlin, "Functional 3D object classification using simulation of embodied agent," in *Proceedings of the British Machine Vision Conference*, 2006, pp. 307–316.

[3] S. Bergen and B. J. Ross, "Aesthetic 3D model evolution," *Genetic Programming and Evolvable Machines*, vol. 14, no. 3, pp. 339–367, 2013.

[4] M. Bokeloh, M. Wand, H. Seidel, and V. Koltun, "An algebraic model for parameterized shape editing," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 78:1–78:10, 2012.

[5] H. J. Bremermann, "Optimization through evolution and recombination," in *Self-Organizing Systems*, 1962, pp. 93–106.

[6] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An information-rich 3D model repository," *CoRR*, vol. abs/1512.03012, 2015.

[7] S. Chaudhuri, E. Kalogerakis, L. Guibas, and V. Koltun, "Probabilistic reasoning for assembly-based 3D modeling," *ACM Transactions on Graphics*, vol. 30, no. 4, pp. 35:1–35:10, 2011.

[8] S. Chaudhuri and V. Koltun, "Data-driven suggestions for creativity support in 3D modeling," *ACM Transactions on Graphics*, vol. 29, no. 6, pp. 183:1–183:9, 2010.

[9] D. Chen, X. Tian, Y. Shen, and M. Ouhyoung, "On visual similarity based 3D model retrieval," *Computer Graphics Forum*, vol. 22, no. 3, pp. 223–232, 2003.

[10] C. Coia and B. J. Ross, "Automatic evolution of conceptual building architectures," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2011, pp. 1140–1147.

[11] C. Darwin, *On the Origin of Species*. John Murray, 1859.

[12] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2003.

[13] A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer, 2002.

[14] H. Fu, D. Cohen-Or, G. Dror, and A. Sheffer, "Upright orientation of man-made objects," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 42:1–42:7, 2008.

[15] Q. Fu, X. Chen, X. Su, and H. Fu, "Pose-inspired shape synthesis and functional hybrid," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 12, pp. 2574–2585, 2017.

[16] P. J. Funes and J. B. Pollack, "Computer evolution of buildable objects for evolutionary design by computers," in *Proceedings of the European Conference on Artificial Life*, 1997, pp. 358–367.

[17] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin, "Modeling by example," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 652–663, 2004.

[18] R. Gal, O. Sorkine, N. J. Mitra, and D. Cohen-Or, "iWIRES: An analyze-and-edit approach to shape manipulation," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 33:1–33:10, 2009.

[19] D. Gonzalez and O. van Kaick, "3D synthesis of man-made objects based on fine-grained parts," *Computers & Graphics*, vol. 74, pp. 150–160, 2018.

[20] Y. Guan, H. Liu, K. Liu, K. Yin, R. Hu, O. van Kaick, Y. Zhang, E. Yumer, N. Carr, R. Mech, and H. Zhang, "3D shape generation via functionality-aware model evolution," under review.

[21] M. Hemberg, U. O'Reilly, A. Menges, K. Jonas, M. da Costa Gonçalves, and S. R. Fuchs, "Genr8: Architects' experience with an emergent design tool," in *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, 2008, pp. 167–188.

[22] R. Hu, M. Savva, and O. van Kaick, "Functionality representations and applications for shape analysis," *Computer Graphics Forum*, vol. 37, no. 2, pp. 603–624, 2018.

[23] R. Hu, O. van Kaick, B. Wu, H. Huang, A. Shamir, and H. Zhang, "Learning how objects function via co-analysis of interactions," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 47:1–47:12, 2016.

[24] R. Hu, Z. Yan, J. Zhang, O. van Kaick, A. Shamir, H. Zhang, and H. Huang, "Predictive and generative neural networks for object functionality," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 151:1–151:13, 2018.

[25] R. Hu, C. Zhu, O. van Kaick, L. Liu, A. Shamir, and H. Zhang, "Interaction context (ICON): Towards a geometric functionality descriptor," *ACM Transactions on Graphics*, vol. 34, no. 4, pp. 83:1–83:12, 2015.

[26] H. Huang, E. Kalogerakis, and B. Marlin, "Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces," *Computer Graphics Forum*, vol. 34, no. 5, pp. 25–38, 2015.

[27] S. Huang, H. Fu, L. Wei, and S. Hu, "Support substructures: Support-induced part-level structural representation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 8, pp. 2024–2036, 2015.

[28] C. Jacob, *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann, 2001.

[29] E. Kalogerakis, S. Chaudhuri, D. Koller, and V. Koltun, "A probabilistic model for component-based shape synthesis," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 55:1–55:11, 2012.

[30] V. G. Kim, S. Chaudhuri, L. Guibas, and T. Funkhouser, "Shape2Pose: Human-centric shape analysis," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 120:1–120:12, 2014.

[31] V. G. Kim, W. Li, N. J. Mitra, S. DiVerdi, and T. Funkhouser, "Exploring collections of 3D models using fuzzy correspondences," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 54:1–54:11, 2012.

[32] V. Kraevoy, D. Julius, and A. Sheffer, "Model composition from interchangeable components," in *Proceedings of the Pacific Conference on Computer Graphics and Applications*, 2007, pp. 129–138.

[33] M. Levoy and P. Hanrahan, "Light field rendering," in *Proceedings of SIGGRAPH*, 1996, pp. 31–42.

[34] J. Lin, D. Cohen-Or, H. Zhang, C. Liang, A. Sharf, O. Deussen, and B. Chen, "Structure-preserving retargeting of irregular 3D architecture," *ACM Transactions on Graphics*, vol. 30, no. 6, pp. 183:1–183:10, 2011.

[35] H. Lipson and J. B. Pollack, "Automatic design and manufacture of robotic lifeforms," *Nature*, vol. 406, no. 6799, pp. 974–978, 2000.

[36] B. T. Lowerre, "The HARPY speech recognition system," Ph.D. dissertation, Carnegie Mellon University, 1976.

[37] Z. Lun, E. Kalogerakis, R. Wang, and A. Sheffer, "Functionality preserving shape style transfer," *ACM Transactions on Graphics*, vol. 35, no. 6, pp. 209:1–209:14, 2016.

[38] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber, "Design galleries: A general approach to setting parameters for computer graphics and animation," in *Proceedings of SIGGRAPH*, 1997, pp. 389–400.

[39] J. McCormack, "Interactive evolution of L-system grammars for computer graphics modelling," in *Complex Systems: From Biology to Computation*, 1993, pp. 118–130.

[40] N. J. Mitra, L. J. Guibas, and M. Pauly, "Partial and approximate symmetry detection for 3D geometry," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 560–568, 2006.

[41] N. J. Mitra, M. Pauly, M. Wand, and D. Ceylan, "Symmetry in 3D geometry: Extraction and applications," *Computer Graphics Forum*, vol. 32, no. 6, pp. 195–204, 2013.

[42] N. J. Mitra, M. Wand, H. Zhang, D. Cohen-Or, and M. Bokeloh, "Structure-aware shape processing," in *Eurographics State of the Art Reports*, 2013, pp. 175–197.

[43] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. J. Mitra, and L. J. Guibas, "StructureNet: Hierarchical graph networks for 3D shape generation," *ACM Transactions on Graphics*, vol. 38, no. 6, pp. 242:1–242:19, 2019.

[44] M. O'Neill, J. McDermott, J. M. Swafford, J. Byrne, E. Hemberg, A. Brabazon, E. Shotton, C. McNally, and M. Hemberg, "Evolutionary design using grammatical evolution and shape grammars: Designing a shelter," *International Journal of Design Engineering*, vol. 3, no. 1, pp. 4–24, 2010.

[45] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. J. Guibas, "Discovering structural regularity in 3D geometry," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 43:1–43:11, 2008.

[46] M. L. Pilat and C. Jacob, "Creature academy: A system for virtual creature evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2008, pp. 3289–3297.

[47] S. Pirk, V. Krs, K. Hu, S. D. Rajasekaran, H. Kang, Y. Yoshiyasu, B. Benes, and L. J. Guibas, "Understanding and exploiting object interaction landscapes," *ACM Transactions on Graphics*, vol. 36, no. 3, pp. 31:1–31:14, 2017.

[48] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser, "A planar-reflective symmetry transform for 3D shapes," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 549–559, 2006.

[49] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 77–85.

[50] R. Salakhutdinov and G. Hinton, "Deep Boltzmann machines," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2009, pp. 448–455.

[51] M. Savva, A. X. Chang, P. Hanrahan, M. Fisher, and M. Nießner, "SceneGrok: Inferring action maps in 3D environments," *ACM Transactions on Graphics*, vol. 33, no. 6, pp. 212:1–212:10, 2014.

[52] ——, "Pigraphs: Learning interaction snapshots from observations," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 139:1–139:12, 2016.

[53] P. Simari, E. Kalogerakis, and K. Singh, "Folding meshes: Hierarchical mesh segmentation based on planar symmetry," in *Proceedings of the Eurographics Symposium on Geometry Processing*, 2006, pp. 111–119.

[54] K. Sims, "Artificial evolution for computer graphics," in *Proceedings of SIGGRAPH*, 1991, pp. 319–328.

[55] ——, "Evolving virtual creatures," in *Proceedings of SIGGRAPH*, 1994, pp. 15–22.

[56] O. Sorkine, "Laplacian mesh processing," in *Eurographics State of the Art Reports*, 2005, pp. 53–70.

[57] X. Su, X. Chen, Q. Fu, and H. Fu, "Cross-class 3D object synthesis guided by reference examples," *Computers & Graphics*, vol. 54, pp. 145–153, 2016.

[58] R. Unger and J. Moult, "Genetic algorithms for protein folding simulations," *Journal of Molecular Biology*, vol. 231, no. 1, pp. 75–81, 1993.

[59] O. van Kaick, H. Zhang, G. Hamarneh, and D. Cohen-Or, "A survey on shape correspondence," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1681–1707, 2011.

[60] Z. Wu, X. Wang, D. Lin, D. Lischinski, D. Cohen-Or, and H. Huang, "SAGNet: Structure-aware generative network for 3D-shape modeling," *ACM Transactions on Graphics*, vol. 38, no. 4, pp. 91:1–91:14, 2019.

[61] K. Xu, V. G. Kim, Q. Huang, and E. Kalogerakis, "Data-driven shape analysis and processing," *Computer Graphics Forum*, vol. 36, no. 1, pp. 101–132, 2017.

[62] K. Xu, H. Li, H. Zhang, D. Cohen-Or, Y. Xiong, and Z. Cheng, "Style-content separation by anisotropic part scales," *ACM Transactions on Graphics*, vol. 29, no. 6, pp. 184:1–184:10, 2010.

[63] K. Xu, H. Zhang, D. Cohen-Or, and B. Chen, "Fit and diverse: Set evolution for inspiring 3D shape galleries," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 57:1–57:10, 2012.

[64] M. E. Yumer, S. Chaudhuri, J. K. Hodgins, and L. B. Kara, "Semantic shape editing using deformation handles," *ACM Transactions on Graphics*, vol. 34, no. 4, pp. 86:1–86:12, 2015.

[65] Y. Zheng, D. Cohen-Or, and N. J. Mitra, "Smart variations: Functional substructures for part compatibility," *Computer Graphics Forum*, vol. 32, no. 2, pp. 195–204, 2013.

[66] Y. Zheng, H. Fu, D. Cohen-Or, O. K. Au, and C. Tai, "Component-wise controllers for structure-preserving shape manipulation," *Computer Graphics Forum*, vol. 30, no. 2, pp. 563–572, 2011.

Appendix A

Computation of Functionality Distance

The descriptor $\mathbb{D}_{u_k}(W_i)$ in (4.2) is formulated by decoupling the point-level property values from the bins of the histogram that represents u_k, written as:

$$\mathbb{D}_{u_k}(W_i) = \mathbb{B}_k W_i, \tag{A.1}$$

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 (4.4), the unary distance is computed as:

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

Similarly, based on (4.5), the binary distance is computed as:

$$\begin{aligned} d_b(W, \mathcal{M}) &= \sum_{i,j} \sum_{b_{i,j,k}} \alpha_k^b \sum_{l=1}^{n_k^b} \left(W_i^T \mathbb{B}_{k,l}^b W_j - \mathcal{N}(b_{i,j,k})_l \right)^2 \\ &= \sum_{b_k} \sum_{l=1}^{n_k^b} \alpha_k^b \sum_{i,j} \left(W_i^T \mathbb{B}_{k,l}^b W_j - \mathcal{N}(b_{i,j,k})_l \right)^2 \\ &= \sum_{b_k} \sum_{l=1}^{n_k^b} \alpha_k^b \left\| W^T \mathbb{B}_{k,l}^b W - N_{k,l}^b \right\|_F^2 , \end{aligned} \tag{A.3}$$

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 $N^b_{k,l} = [\mathcal{N}(b_{i,j,k})_l; \forall i, j] \in \mathbb{R}^{m \times m}$, where m is the number of proto-patches in $\mathcal{M}$ and $\mathcal{N}(b_{i,j,k})_l$ is the l^{th} bin of the histogram $\mathcal{N}(b_{i,j,k})$. Note that both $\mathbb{B}^b_{k,l}$ and $N^b_{k,l}$ are symmetric.

Appendix B

Examples of Beam Search

Given a hybrid shape possessing multiple functionalities, Figure B.1, Figure B.2, and Figure B.3 show the complete reverse 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*.

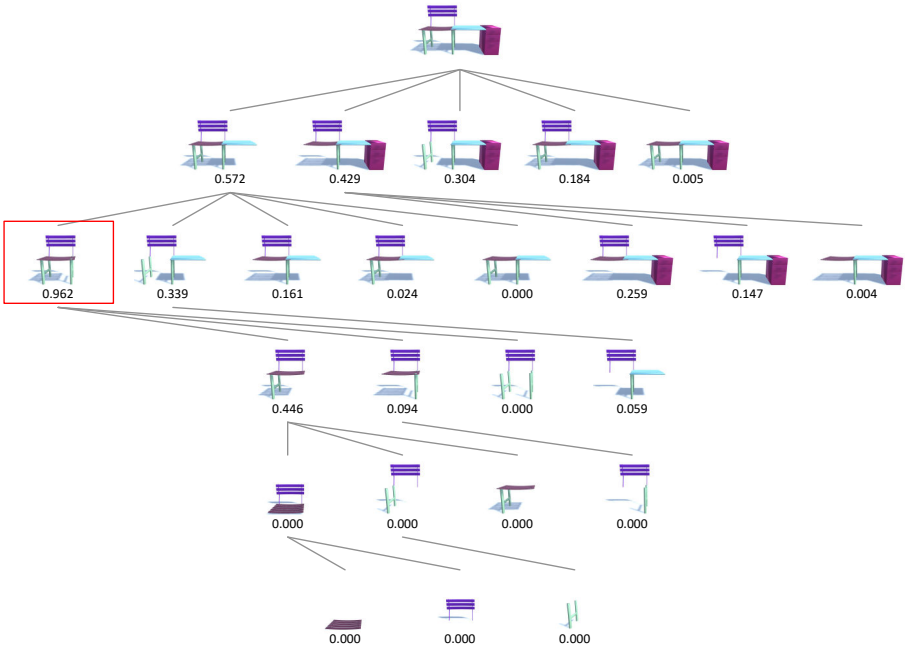


Figure B.1: Beam search for functionality partial matching, where we search for the subset of parts that provides the highest score for the *chair* category.

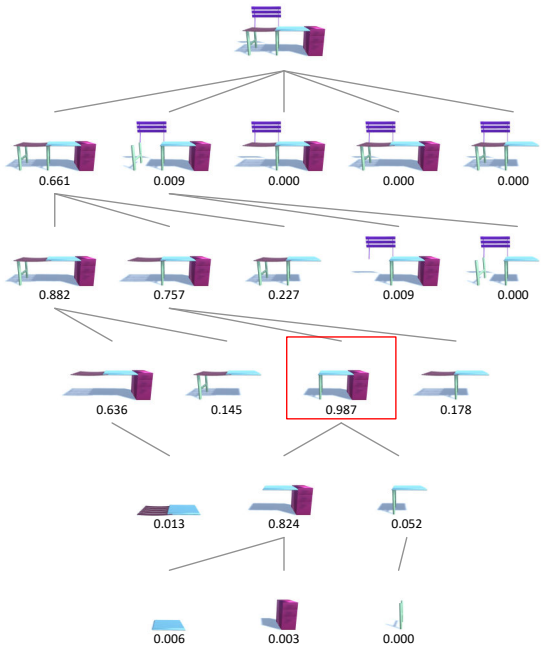


Figure B.2: Beam search for functionality partial matching, where we search for the subset of parts that provides the highest score for the *desk* category.

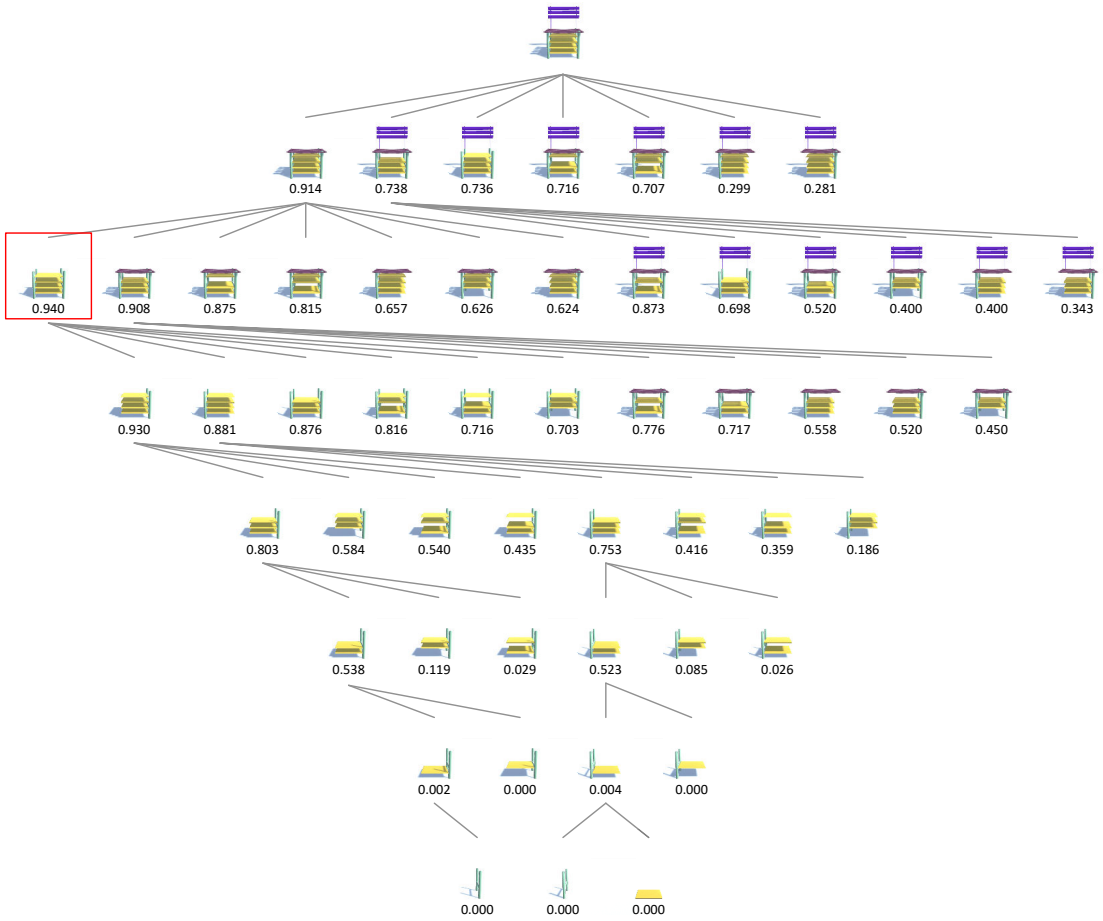


Figure B.3: Beam search for functionality partial matching, where we search for the subset of parts that provides the highest score for the *shelf* category.