3D Synthesis of Man-made Objects based on Fine-grained Parts

by

Diego Alexander Gonzalez Becerra, **B.Eng., M.Sc.**

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
The School of Computer Science
Carleton University
Ottawa, Ontario
February, 2018

©Copyright
Diego Alexander Gonzalez Becerra, 2018

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

3D Synthesis of Man-made Objects based on Fine-grained Parts

submitted by **Diego Alexander Gonzalez Becerra**, **B.Eng., M.Sc.**

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 Michel Barbeau, Chair,
School of Computer Science

Ottawa-Carleton Institute for Computer Science
The School of Computer Science
Carleton University
February, 2018

Abstract

In this thesis, we propose a novel approach for the synthesis of new, plausible 3D shapes. While most previous methods that synthesize 3D shapes are based on reusing semantic parts from a set of input models, and combining or blending these parts to create new objects, our method is designed to use more granular parts, thus yielding synthesized shapes with a wider variety of fine details. A key advantage of our method is that, unlike previous approaches, we do not require a semantic segmentation of the input 3D shapes, nor a part correspondence between the shapes of the input set. To enable such a synthesis approach, we extract a set of fine-grained parts from a dataset of 3D shapes, and compute the geometric similarity between these fine-grained segments by using a set of descriptors suitable for this purpose. For the generation of novel 3D shapes, we start by selecting one of the shapes of our input set as a template or reference. Next, for each fine-grained part of that template, we employ our similarity metric to select, from our set of fine-grained segments, a compatible part that we can use for replacing the original part of the template. Moreover, by choosing different compatible segments for each part of the template, and by using different template shapes, our method can synthesize many distinct 3D shapes that have a wide range of fine geometrical variations. In addition, we maintain the plausibility of the novel objects by preserving the topology and general structure of the template. We show with experiments performed on different input datasets that our algorithm can be used for synthesizing a variety of man-made objects.

Acknowledgments

First and foremost, I would like to deeply thank my supervisor Dr. Oliver van Kaick for giving me the opportunity to have worked with him in such an interesting project, for all his teachings, his patience, and his constant support. I feel extremely fortunate to have been able to work with a supervisor who is not only so knowledgeable, but who is also so understanding and kind.

I would also like to thank Dr. David Mould and Dr. Jochen Lang for reviewing my thesis, and providing many valuable observations and suggestions that helped me to improve this work. Moreover, I wish to thank Dr. David and Dr. Oliver for all their teachings and the support that they always have given me as a member of the Graphics, Imaging, and Games (GIGL) Lab.

I also wish to thank all my friends and colleagues from the GIGL Lab and the School of Computer Science at Carleton for their constant support and company. Specially, I thank my friends from the Lab for sharing their insights about so many different topics during our group meetings.

My gratitude also goes to my friends in Colombia and Mexico, and to my family in Colombia, especially my father, Helena, my brother, and my parents in law for their support during the adventure that I decided to undertake when coming to Canada. I must also thank Sandra, Paco, Luca, and Andrea for being my family in Canada.

Finally, there are no enough words to express my gratitude and love to my wife Ines. My love, thank you for your constant support, your love, and for always believing in me. Thank you so much for being the best adventures' partner.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Digital content in the form of 3D models is an ongoing necessity in a variety of fields such as entertainment and product design. However, the manual creation of 3D models is often a difficult and time-consuming task, since it requires both artistic and technical skills, likely involving the use of complex modeling software [2]. Hence, facilitating the creation of 3D models is a fundamental problem in computer graphics. Moreover, the problem of automatically generating compelling 3D shapes has received much attention in the recent literature [3–9].

A popular approach for the automatic creation of novel 3D objects is to reuse parts from existing shapes. In particular, several recent works on shape synthesis propose to extract and recombine parts from existing objects that belong to the same *family* [3, 4, 8]. A family or class is typically defined as a set of objects that share the same functionality and general structure [10]. These synthesis techniques can take advantage of the recent growth of repositories of 3D shapes such as *3D Warehouse* [11] and *ShapeNet* [12], since these repositories provide a large variety of shapes, which can be used to generate large collections of novel objects by part recombination.

Most techniques that recombine parts from existing models make two important assumptions about the input models and their parts. First, the input objects should be segmented into coarse *semantic* parts, that is, parts with a meaningful semantic meaning, and possibly a specific functionality, such as the legs and the top in the case of a table. Second, a *correspondence* should exist among the different parts of the shapes. Such correspondence is usually given as a set of *labels* that indicate the parts that can be exchanged between the models.

One shortcoming of these methods is that the synthesized shapes may have a limited range of variations in their fine geometrical details, since semantic parts are

often large, coarse and exchanged as a whole [7, 13]. Furthermore, although much progress has been made to obtain semantic segmentations of 3D shapes [14, 15], computing highly accurate and consistent segmentations and part correspondences for large collections of shapes is still a challenging problem [16, 17].

In our work, we propose to synthesize shapes by exchanging *fine-grained* parts among a set of input shapes. The use of fine-grained parts allows us to generate shapes with a larger variation in finer geometric details, which is not achievable with methods that exchange semantic parts as a whole. Figure 1.1 shows a comparison between a typical semantic segmentation of a table, and the fine-grained segments that we use for synthesis. Moreover, we require neither a semantic segmentation nor a part correspondence as input, since the fine-grained parts can be obtained by partitioning the shapes with traditional geometry-based segmentation methods [14].

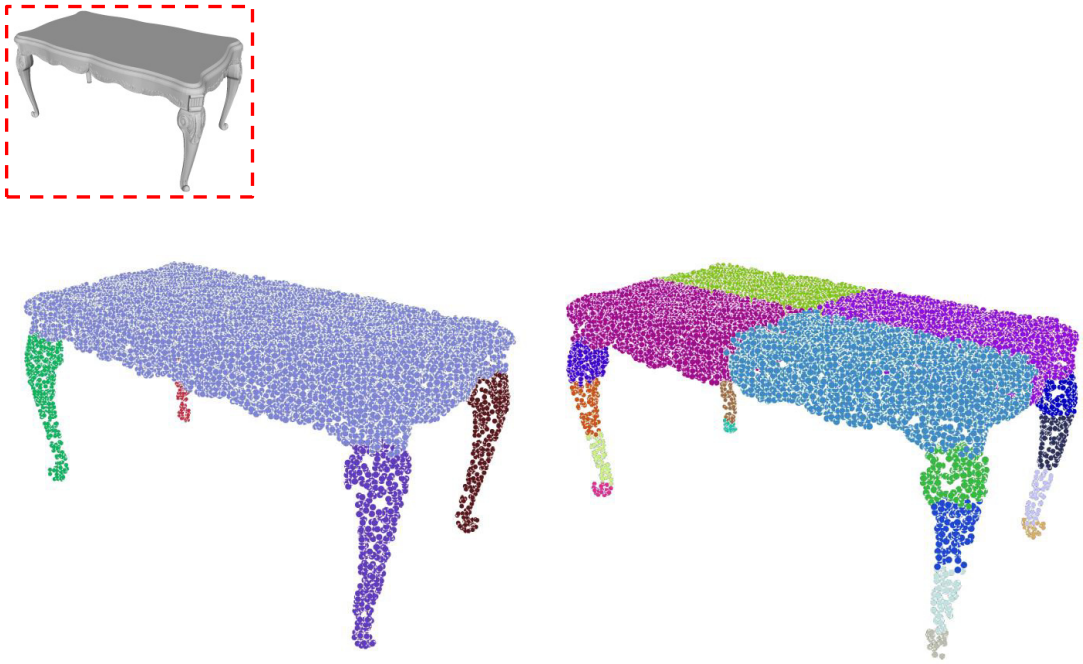


Figure 1.1: Comparison of two segmentations for the table model shown in the inset. Left: typical semantic segmentation; right: segmentation into fine-grained segments used in our work.

One problem arising from the elimination of part semantics is that we require an alternative mechanism to guide and constrain the synthesis of a new shape. Thus, we

propose to use for guidance an input template, which can be a simple configuration of geometric proxies or an example shape. The template constrains the topology of the synthesized shape, which helps to ensure a certain level of plausibility in the generated shapes, and provides a domain for formulating the synthesis of a shape as a graph assignment problem, as we discuss in Section 1.1.

In terms of applications, our method can be applied to generate a collection of novel, plausible shapes showing a broad variety of fine geometric details, which can be used to populate a virtual scene with less repetition and predictability. Figure 1.2 shows a 3D scene created with shapes synthesized by our method. Another potential application of our method is in product design, where, by generating a large collection of shapes with fine geometric details, a user would be able to explore the collection and potentially select a few models to serve as inspiration for new shape designs.

Figure 1.2: Scene created with three shapes synthesized by our method. The shapes exhibit a variety of fine local details and a coherent structure.

In summary, our approach aims at synthesizing 3D shapes which:

- Show a wide range of fine geometric details, which we capture using fine-grained parts acquired from a set of input 3D models by means of a semi-automatic segmentation method; and,
- Possess the topology and general structure of shapes in the input family, since the shapes are synthesized following an input template.

1.1 Overview of the proposed method

In this thesis, we present a method to generate 3D models using a set of fine-grained segments extracted from one or more families of shapes. The input to our synthesis pipeline is a set of triangular meshes partitioned into fine-grained segments, which form a pool of parts that can be used for synthesis. We compute a set of descriptors to represent each segment, and we use these descriptors to define a similarity metric for the comparison of segments.

To synthesize a shape, we take as input a geometric template representing the general structure and topology of the target shape. We define a graph based on the structure and shape of the template, and pose the synthesis as a probabilistic sampling of segments. Our goal is to assign segments from the pool of parts to the nodes of the graph, in order to minimize a *shape energy*. This energy takes into account the similarity between the template and sampled segments, in the form of a unary energy term, as well as the consistency between neighboring segments, captured by a pairwise term. Finally, after sampling the segments that constitute the shape, we perform a series of geometric operations to align the segments and ensure that we obtain a plausible shape as output. At the end of this process, we obtain a shape that respects the topology of the template and possesses a consistent structure, while containing local geometric variations.

Moreover, we analyzed many relevant existing works about shape synthesis and shape analysis, and we performed also several experiments, in order to select descriptors that capture overall geometrical properties of the segments, but are also relatively insensitive to differences in the fine geometrical details of the segments. In this manner, the segments sampled by our method allow us to generate man-made shapes that, in general, can preserve the overall form of the original input template, while, at the same time, the segments can also have differences in their local fine details.

Figure 1.3 shows an example of a shape generated with our method in comparison to the original template model used to guide the synthesis. We can see several differences among local features of each model. To synthesize the new shape shown in Figure 1.3, our approach replaced each fine-grained segment of the input template with a new segment extracted from other models of our input family of tables, producing a shape that exhibits a wide range of local geometrical variations in comparison to the template, and also preserves the structure and topology of the template.

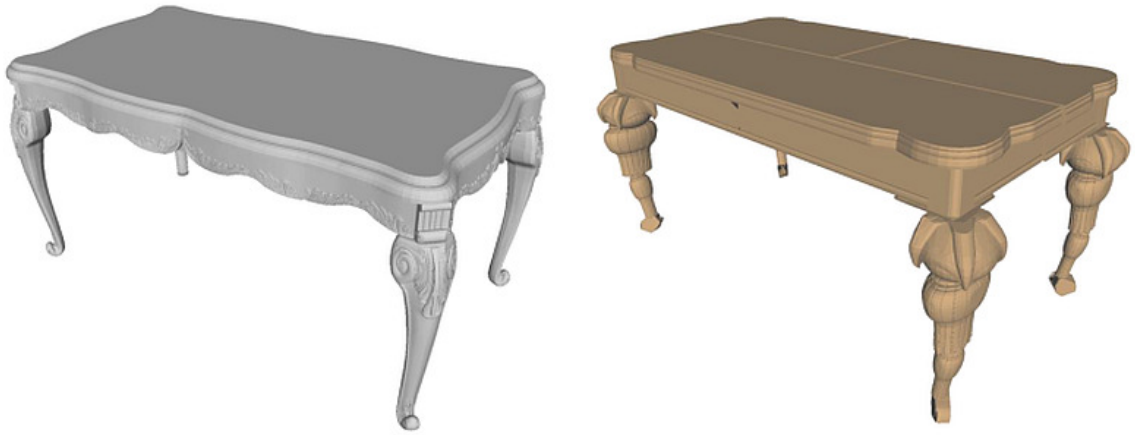


Figure 1.3: Example of a novel 3D model created using our method (right), in comparison to the template shape employed to guide the synthesis process (left).

Since our synthesis approach can consider a large set of fine-grained segments, we can generate different variations for the same input template by sampling multiple shapes with the graph assignment method. In addition, we can further augment the collection of new objects created with our algorithm by selecting new templates from our input dataset, as we discuss in detail in Chapters 4 and 6.

1.2 Contributions

In summary, our research makes the following contributions:

- We present a pipeline for synthesizing 3D shapes using fine-grained segments extracted from an input set of shapes. Our method does not require a segmentation of the input models into semantic parts, nor a part correspondence, but mainly a guiding template and a segmentation where the segments are consistent in size for corresponding parts of the input 3D models.
- We demonstrate the effectiveness of our approach by presenting and analyzing a variety of results synthesized with our method.
- We explore and discuss variations of our pipeline for 3D shape synthesis that can serve to control different aspects of the generated shapes.

1.3 Organization

The remainder of this thesis is organized as follows. In Chapter 2, we discuss the problem of shape synthesis in more depth, and we review previous literature related to this problem. In Chapter 3, we present a general overview of our pipeline for 3D shape synthesis, while we explain the details of each one of the algorithms that compose this pipeline in Chapter 4. In Chapter 5, we discuss the creation of segmented input for our method, and present our interactive tool for the extraction of fine-grained segments. Chapter 6 shows and discusses the results of employing our technique for synthesizing 3D shapes for different families of 3D objects. Finally, Chapter 7 provides our conclusions, and discusses limitations and possibilities for future work.

Chapter 2

Background and related work

In this chapter, we discuss the problem of 3D shape synthesis, and we review some of the methods that have been previously proposed for solving this problem.

2.1 Shape synthesis and modeling

The modification and creation of 3D models is often a difficult and time-consuming task, which frequently requires the use of complex modeling software and is mainly performed by skilled artists [2]. Hence, facilitating the creation and modeling of 3D models is a fundamental problem in computer graphics [18]. Although many studies about shape modeling and synthesis employ a combination of several different methodologies, we loosely classify these works into three main categories: (i) Design of intelligent and more intuitive interfaces for 3D modeling; (ii) Learning, from a collection of 3D objects, a parametric model that allows to guide the editing and synthesis of shapes; and (iii) Automatic and semi-automatic synthesis.

2.1.1 Interfaces for modeling and synthesis

Most 3D modeling and CAD software include different tools for detailed creation and editing of 3D objects [18]. However, many of these tools do not allow a user to easily reuse parts from existing models, and they assume that the user has deep knowledge about the geometrical details of the 3D objects [19]. Next, we describe several studies that propose different types of modeling interfaces aimed at solving these issues.

The seminal work by Funkhouser et al. [20] proposed a system that allows the

user to browse a library of 3D models and to compose new shapes by assembling together parts of the existing 3D objects. Many studies on 3D modeling and synthesis are based on this paradigm, i.e., recombining parts from existing models to generate novel shapes [2]. Following this line of work, Kreavoy et al. [21] propose to perform, in a pre-processing stage, compatible segmentations of a collection of shapes based on a measure of the approximate convexity of the shape's parts. Such compatible segmentations allow to compute and store part to part correspondences between the models of the set. Later, the user can select from the collection of models one shape that will be used as target of the modeling operation, and selects segments or parts from this target shape, as well as corresponding compatible components chosen from the other shapes in the library. Then, the system automatically synthesizes new shapes by exchanging the selected components of the target shape with the corresponding compatible ones selected from the other models.

On the other hand, the so called *SnapPaste* interface of Sharif et al. [22] proposes to adapt the cut-and-paste notion to 3D mesh editing. SnapPaste allows the user to cut a mesh part from a 3D model using a method based on graph cuts [23], and paste it into a target mesh that has a compatible *snapping* region. The meshes are merged by applying a variation of the ICP algorithm [24] that blends the boundaries of the source and target meshes without causing a large deformation of the mesh parts. Following a similar approach, Schmidt et al. [25], and later Takayama et al. [26] allow a user to copy parts from a source 3D model into another region on the same or different model. Specifically, Takayama et al. allow the user to click on a point on the surface of the source model, and to draw a *canvas* region that represents the area of the source model that should be copied. Likewise, a point and a canvas are defined on the target model. The system computes a parameterization for each canvas, as well as a correspondence between the source and target canvases. Finally, the user can draw on either the source or the target canvas, and the source geometry is cloned towards the target area. By computing a parameterization of the 3D surfaces, Takayama et al. can exploit the possibility of performing operations on the 2D domain, allowing their system to provide visual feedback in a fast manner.

The system of Chaudhuri et al. [27] incorporates semantic information into a dataset of pre-segmented 3D shapes, for learning a probabilistic graphical model of a 3D shape, in the form of a Bayesian network [28]. The probabilities learned by the model are used by a modeling interface that presents the user with a list of suggested

parts ranked by relevance, which can be used to compose a 3D object. During the modeling operation, the list is automatically updated, according to the probabilistic model. The 3D objects employed in the system of Chaudhuri et al. are segmented and labeled into semantic parts using the method of Kalogerakis et al. [29], which we will describe in Section 2.2.

Recently, Jaiswal et al. [19] presented an assembly-based interface that also generates suggestions to inspire the user during the modeling process. The system uses a probabilistic graphical model called a factor graph [30] to identify, in a database of parts, the most relevant parts that can be attached to the object being edited. Jaiswal et al. employ two types of factors for the graph: an adjacency factor that represents the likelihood of two components appearing next to each other; and a multiplicity factor that encodes the relationship between similar components that appear multiple times in a 3D model. The 3D objects used in this interface are segmented into coarse parts using the method of Shapira et al. [31], which constructs the segments of the models based on a measure of the local volume of the different regions of the shape.

Similarly to our approach, the system by Jaiswal et al. does not require a labeling of the parts of the input shapes. However, our technique uses fine-grained segments rather than coarse or semantic parts. By using fine-grained segments, we can avoid the requirements of a semantic segmentation and a part correspondence as input to our method.

2.1.2 Synthesis and modeling based on parametric models

Another general approach for 3D shape synthesis and editing consists of learning a parametric model from a collection of 3D shapes of the same class to determine common attributes and relations of shape parts and, on the other hand, to learn a model of the geometrical variations found on the shapes of the class. For human faces, for example, Blanz and Vetter [32] apply Principal Components Analysis (PCA) to a dataset of scans of 3D faces to derive a model that can be used to create plausible modifications of the shape and texture of the faces, and to generate a 3D face model from an input image. Later, Allen et al. [33] obtain a parametric model for human shapes using a set of whole-body range scans. This parametric model can be used to synthesize a plausible 3D model of a person, or to edit existing individuals.

In the context of man-made shapes, Yumer et al. [34] map geometric features of a collection of shapes to a set of semantic attributes. These semantic attributes are

defined according to the class or category of the shapes. For example, for models of a family of chairs, some of the semantic attributes defined are comfortable, elegant, ergonomic, and antique. Using the mapping between the semantic attributes and the geometry of the shapes, the system by Yumer et al. allows a user to deform the shapes using a set of handles that control the *intensity* of each semantic attribute, so that a chair model, for instance, can be deformed to make it more or less antique.

Other methods propose to analyze sets of shapes of the same family for learning a model that captures the variability within the set. For example, Averkiou et al. [35] represent the shapes of a dataset as box-like templates, and generate a 2D embedding of these templates. The user can navigate through this 2D space to explore the collection, or click on empty spaces in the embedding space to create a novel 3D shape. In a similar manner, Fish et al. [36] compute a meta-representation to encode the variability of relations and part-arrangements of a family of pre-segmented and consistently labeled shapes. The meta-representation is computed as a set of Probability Density Functions (PDFs), which are used in an interface where the user explores the collection of shapes by clicking on locations of the distribution. In addition, the user can directly edit the parts of a shape producing novel models whose plausibility can be restored, if necessary, using the meta-representation.

More recently, Fu et al. [37] employ a set of pre-segmented shapes of the same class, including shapes with topological variations, and classify the shapes into different groups based on their constituent parts. From these groups of shapes, a set of group-sensitive priors is learned. Next, each part is represented by its respective Oriented Bounding Box (OBB). These OBBs are employed in an interface where the user can perform several shape editing operations, including addition, translation, rotation, and scaling of parts. The final modified shape is automatically refined according to the previously acquired priors.

Note that all the parametric models described above require a correspondence between the shapes in a family, possibly derived from a semantic segmentation of the shapes. In contrast, our work aims at automatically synthesizing novel shapes without the need of semantic parts or a correspondence.

2.1.3 Automatic and semi-automatic shape synthesis

Even with the use of advanced modeling interfaces, like the ones that we described above, the creation of 3D models can be a time-consuming task. However, many

applications require large collections of shapes to populate virtual scenes. Next, we review several methods that have been proposed in the literature to generate 3D shapes requiring minimal to no user assistance.

Inverse procedural modeling

Bokeloh et al. [38], and later Kalojanov et al. [39], extract the parameters of a shape grammar from a single exemplar shape. The grammar can be used to produce similar shapes. Specifically, these algorithms discover partial symmetries of the input 3D model, and decompose the shape into symmetric regions yielding *docking sites*, where a replacement, insertion, or deletion of a region, can produce novel shapes. As we will see later, the concept of shape grammars has been studied and generalized in other more recent works.

Shape synthesis by style transfer

Recently, several authors have proposed techniques that seek to learn features that characterize diverse *styles* of shapes in a set, with the purpose of generating novel shapes by means of transferring the style of existing models towards the new objects. Xu et al. [40] define a measure or distance that allows them to classify a set of shapes according to their style. Then, their method computes a clustering of a set of shapes according to the style distance and obtains part correspondences between the shapes. Next, these part correspondences are employed for transferring parts from a source towards a target shape, computing an anisotropic scaling of the source parts to match the proportions of the target. Han et al. [41] propose a similar approach for style transfer, employing a different measure of style similarity. The method by Han et al. performs the final placement of the transferred parts on the target shape by aligning the boundaries of the style and content parts using ICP [24].

Other authors have argued that fine-level geometric properties might encode the general style of a shape [42, 43]. Lun et al. [9] use a metric of style similarity for automatically transferring the style of an exemplar shape to a target shape, preserving the functionality and structure of the target. Specifically, the shapes are segmented into meaningful compatible parts using the method of Asafi et al. [44], which we describe in Section 2.2. Lun et al. compute a sequence of operations such as part substitution, part addition, and curve-based part deformation, which seek to minimize the style distance between the target and the exemplar. In a similar manner, Hu et

al. [45] identify style-defining elements over a set of 3D shapes with a learning-based approach. These style-defining elements are patches that capture local regions of the shapes. The elements are then used in a tool for style-guided shape modeling, where, given an input shape and a reference style, the tool suggests style elements that can be added or removed to enhance the desired style of the input shape to produce a new 3D model.

In our work, we aim at capturing fine details from the input 3D models to synthesize shapes with many local geometrical variations. However, the fine details that we capture are in principle independent of the style of the shapes. Our method can take as input shapes from different styles and mix up their geometry, or the user can constrain the style of the generated shapes by providing as input only shapes with a consistent style. We also show in our experimental results that the user can employ our method constraining certain portions of the template, so that our synthesized shapes can maintain parts with specific styles.

Synthesis by blending existing shapes

Other studies have proposed to blend the parts of two or more input shapes, generating thus several novel 3D shapes. Jain et al. [46] propose an approach where a set of shapes is first segmented into its main semantic parts, and then, symmetry relations are detected [47], as well as contact points between the parts. The system then synthesizes new shapes by matching and blending parts with equivalent symmetry and contact structures. In our method, we also make use of the symmetry relations between the fine-grained parts that we obtain for each input shape, although we use a user-assisted method to discover these symmetries, as we discuss in Chapter 5.

Employing a similar approach, Alhashim et al. [13] blend a source and a target 3D shape of the same or similar class, generating several novel shapes by exchanging parts, and creating deformations that gradually change the topology and geometry of the source shape making it more and more similar to the target. Alhashim et al.'s method requires the two input shapes to be consistently pre-segmented and assumes a part correspondence between the parts of the two shapes. Nonetheless, this approach might produce new objects with an unpleasant appearance due to undesirable topological changes, like additional arms or legs in the model of a chair. Thus, the system includes an implausibility filter that aims at detecting and removing synthesized shapes that do not preserve the symmetry and connectivity relations

observed in the two original shapes. In our work, we maintain the plausibility of the generated 3D models by using a template shape that guides the synthesis process, determining the general structure and the topology of the novel shapes.

Synthesis by part-recombination

The methods most related to ours extract parts from a collection of related 3D shapes, and recombine these parts to synthesize novel shapes. The method by Kalogerakis et al. [3] learns a probabilistic component-based model from a family of 3D shapes. The input shapes are assumed to be consistently segmented and labeled into semantic parts [29]. The probabilistic model represents relations between the components of the family of shapes, such as adjacency, as well as the relationships of part styles and repetitions for different shape styles. The method creates new shapes from parts of the input shapes which are assembled according to the learned probability distribution. The approach by Kalogerakis et al., as well as other methods based on recombining only coarse parts from existing models, can yield novel shapes with limited geometrical variations [7]. Our method, instead, seeks to synthesize shapes with a wider range of variations, using fine-grained segments that capture local details from the input shapes.

Similarly to the work of Kalogerakis et al., the research by Huang et al. [8] is based on the idea of learning a generative model from a collection of existing models, to create new 3D shapes. However, the technique by Huang et al. requires only one labeled segmented shape per family or class, and performs a joint segmentation of the shapes in the collection. The method uses a deep learning procedure [48] to compute a model that encodes hierarchical relationships of corresponding surface points and parts from the input shapes. This model can be employed for the task of shape correspondence [49], and to generate novel shapes by recombining and deforming parts from the input family of shapes. Nonetheless, some shapes synthesized by the model of Huang et al. can have disconnected parts, due to segmentation artifacts in the 3D model taken as reference for the joint segmentation of the shapes.

On the other hand, Xu et al. [4] developed a method for shape synthesis inspired by the theory of natural evolution. Xu et al.'s system starts from a family of pre-segmented shapes, with defined part correspondences. The method exchanges corresponding coarse parts and performs part deformation operations to produce an initial set of shapes. Afterwards, a subset of the synthesized shapes is presented to

the user, who provides feedback by scoring the shapes in terms of preference. The scores are used to define a fitness function, which produces a new set of evolved shapes prioritizing the variations considered by the user as the most interesting ones. The process is repeated to obtain new sets of shapes.

The recent work by Su et al. [7] is based on recombining parts from diverse classes of 3D models. Su et al. use a reference or template shape that has multi-functional components and a complex structure. From this reference shape, a set of substructures is extracted and matched to corresponding substructures stored in a database, by minimizing an energy function that measures the similarity between the substructures. The method by Su et al. iteratively selects a pair of substructures of the template, and replaces these selected parts using other substructures from the database. In our approach, we also employ a template to guide the synthesis process, combined with a similarity measure for selecting the parts that we employ to create new shapes. However, instead of creating variations by exchanging large substructures of complex shapes with multi-functional parts, we aim at generating variations in the fine details of our synthesized shapes.

The work by Zheng et al. [5], on the other hand, takes as input two pre-segmented shapes, and searches in these two shapes substructures with specific mutual part relations, such as symmetry and number of contacts. Using these relations, the method by Zheng et al. discovers compatible symmetric arrangements on each shape. These arrangements are reshuffled to produce the novel 3D models. Additionally, Zheng et al. improve the placement of the parts of the novel shapes in a post-processing stage by defining regions on each adjacent component as attachment slots or contact points. These contact points are aligned to optimize the placement of neighboring components. As we explain in Section 4.5, we perform a similar alignment process between the adjacent components of our synthesized shapes, to optimize the placement of the parts in our novel shapes.

Huang et al. [50] propose a method that extends the concept of part substructures of Zheng et al. Huang et al. use a pre-segmented object to derive a set of support substructures, discovering relations between the parts of man-made shapes that are in an upright orientation [51]. The support relations between two parts a and b imply a hierarchy in the shape; e.g., a part a provides a *stable support* for part b. The support relations are encoded in a graph that can be used to perform part reshuffling between different shapes, and different part rearrangement on the same shape.

The method of Liu et al. [6] generalizes the shape grammars employed in the work by Bokeloh et al. beyond regular symmetric regions on the 3D shapes. Liu et al. use a prototype interactive tool that allows a user to annotate matching part types on the input models. From the annotated input mesh, the system computes a shape grammar as well as a dual graph that represents each part as a node, and the connections between the parts as edges. The algorithm discovers replaceable substructures in the graphs by matching pairs of nodes if they are compatible with respect to the shape grammar. New 3D models can be synthesized by exchanging and merging compatible substructures. Based on the work of Liu et al., we have developed an interactive tool for segmentation and annotation of segments that can work directly on point clouds, and we use this interactive segmentation program to extract the fine-grained segments that we require in our synthesis method.

2.1.4 Discussion

As we have mentioned earlier, synthesizing shapes by only recombining coarse parts can yield new models with limited variability. One class of methods for 3D synthesis propose to generate more variations by exchanging parts between different classes or families of shapes to obtain more variety in the synthesized objects [7]. However, if the segments extracted from diverse classes are coarse parts with few fine geometrical details, the novel objects will not have the desired variability. Furthermore, other methods have proposed generating variations by deforming parts and changing the topology of the input shapes [13]. Nonetheless, modifying the topology of the shapes of a given family can produce implausible variations. In our synthesis method, on the other hand, we use a template to guide the synthesis process, with the purpose of preserving the plausibility of the new shapes, as we will see in Chapter 4.

2.2 3D shape segmentation and labeling

The segmentation of 3D shapes is a fundamental component in many applications such as shape analysis, 3D shape retrieval, and shape synthesis [15]. We can define shape segmentation as the problem of partitioning an input shape into a set of disjoint sub-shapes [14]. However, for the same shape we can obtain many different segmentations depending on the specific application requirements, and the criteria employed to partition the shape. The curvature, and measures of the convexity of the shape,

among many other properties [14], can be used to segment a 3D object into parts, generating results that can be, in some cases, very different. Many approaches have been proposed to address the problem of segmenting a 3D shape into meaningful parts, especially in the case of shapes represented as meshes, as surveyed in detail by Shamir [14] and more recently by Theologou et al. [15]. Next, we summarize several of these segmentation approaches, according to the general methodology employed to obtain the shape segments.

Region growing

These methods start a region or segment with a given source or seed face, and incrementally add more neighboring faces to the region, following the adjacencies or connectivity of the mesh, until a stopping criterion is satisfied. Moreover, several regions can grow in parallel by defining different source points on the mesh [14]. Many different attributes of the faces of the mesh can be used to guide the growth of segments. For instance, Lavoue et al. [52] use the values of the principal curvatures κ_{min} and κ_{max} [53] of the mesh as the criterion for region growing. Also, Kreavoy et al. [21] propose a region growing approach based on a measure of the approximate convexity of each part or segment, defined as a weighted average of the distances between the faces on the segment and the convex hull of the shape. As we will see in Section 2.2.1, recent approaches have employed measures of convexity of 3D shapes, to obtain segmentations of meshes and point clouds. To extract the fine-grained segments that we require in our shape synthesis approach, we have explored the use of a region growing algorithm that performs the segmentation based on the difference in the dihedral angles between the adjacent faces of the mesh, an attribute employed by different works on mesh segmentation [14]. We will describe the details of our region growing method for fine-grained segmentation in Chapter 5.

Probabilistic and machine learning methods

According to Theologou et al. [15], we can broadly classify as probabilistic methods segmentation techniques that calculate the probability of each face of the mesh being assigned to a given segment or label, according to different properties of the mesh. For instance, Shapira et al. [31] compute a histogram of a feature called the Shape Diameter Function (SDF), which estimates the local volume on different regions of a shape. Then, a Gaussian mixture model (GMM) [54] is used to assign each face

a probability according to the value of the SDF. The SDF is robust to changes in the pose of the object represented by the 3D model [31], and allows to distinguish between thin and thick parts [14]. Thus, the SDF has been used in many works as a discriminative measure of the volume of the parts of 3D shapes [18]. Moreover, the SDF can be adapted to estimate the volume of shapes represented as point clouds [1]. In our work, we also employ the SDF as a descriptor that allows us to capture the volume of our fine-grained segments, as we discuss in Section 4.2.

Kalogerakis et al. [29], on the other hand, employ a supervised learning method that takes as input a set of labeled shapes for training a Conditional Random Field (CRF) [55] that learns the probability of assigning a given label to each face of a mesh. To this end, a set of per-face unary features or descriptors are computed, as well as a set of pairwise descriptors for each adjacent pair of faces. The labeling for each mesh is computed by minimizing an energy function with two terms, one for the unary features, and the other for the pairwise features. This optimization is performed using graph cuts [23]. In our work, we also formulate the synthesis of a 3D shape in terms of optimizing a shape energy function, as we will see in Chapter 4. Furthermore, other recent works for mesh segmentation apply different supervised or unsupervised machine learning strategies, such as Extreme Learning Machines [56], and deep learning networks [16].

On the other hand, Bamba and Ohbuchi [57] present a mesh segmentation strategy that combines a semi-supervised learning method and user guidance. In their system, Bamba and Ohbuchi provide an interface that allows to draw strokes on a 3D model, to interactively produce an initial labeling for an initial set of faces or regions of the mesh. Then, they propagate the labels towards unlabeled regions of the shape. In our work, we also obtain the segmentations of our input models with an interactive tool, although our segmentation program is designed to work with point clouds instead of meshes, as we describe in Chapter 5.

Boundary-based methods

Boundary-based methods seek to locate the part boundaries, instead of the segments themselves [15]. Golovinskiy and Funkhouser [58] propose to use the dual graph of the mesh, assigning as weights to the graph edges a measure of concavity based on the dihedral angle between the faces. The method uses k-means, hierarchical clustering, and graph cuts to produce multiple segmentations. Among these multiple

segmentations, the system selects the most frequent boundaries to produce the final segmentation. More recently, Au et al. [59] present a fully automatic method which also exploits the shape concavity information using a set of concavity-aware scalar fields. Au et al. perform several evaluations of their method employing the measures and datasets of the Princeton Segmentation Benchmark [60], consistently obtaining better results in comparison to various previous approaches. However, Au et al. state that their method can be inappropriate for shapes with many fine surface details. In our synthesis approach, on the other hand, we employ a semi-automatic segmentation method that allows us to extract the fine details and salient features of the shapes, in order to synthesize 3D objects that can exhibit many of these fine details.

Co-segmentation

Some methods for mesh segmentation that we mentioned above employ sets of labeled shapes of the same class for learning a segmentation [29, 56]. Nevertheless, computing consistent segmentations for several 3D models that belong to the same class or family remains a challenging task [15]. Moreover, the parts that are regarded as semantically similar can have different geometric characteristics. Thus, several authors have proposed to perform a co-segmentation of meshes of the same family, that is, to simultaneously segment all the models in a set to obtain a consistent segmentation of all the shapes. Specifically, many studies about co-segmentation use clustering or classification techniques. For instance, Hu et al. [61] propose using a sparse subspace clustering, while Shu et al. [16] recently have used a deep neural network that takes as input several features to generate a high-level feature space, in which a clustering operation is performed, allowing the system to compute a segmentation for a single shape, or a co-segmentation for a family of 3D shapes.

In addition, Sidi et al. [10] obtain a segmentation for each individual 3D model in their input dataset by computing several descriptors per-face, as well as descriptors defined at the segment level. These descriptors are employed to calculate a metric that estimates the similarity between the segments. From the initial set of segments, Sidi et al. derive a statistical model that describes each class of part and use this model to refine the final co-segmentation results, posing the labeling of each shape as an optimization problem that is solved using graph cuts. In a similar manner to Sidi et al., we make use of several descriptors to compute a similarity distance that let us compare each pair of fine-grained segments that we extract from our input shapes.

2.2.1 Point cloud segmentation

Several mesh segmentation methods are based on measures of the approximate convexity of different regions or components of a mesh [21, 62]. In general, a shape is defined as convex if the straight line segment between any two points of the shape does not leave the volume of the shape, i.e., any pair of points of the shape are in a *line-of-sight* [1, 44]. This concept can be used to compute a decomposition of a 3D shape into weakly convex parts. Moreover, it is possible to apply this method on point clouds by representing each point as a small local disk. The disk is obtained by a local approximation involving the nearest neighbors of each point [1]. To compute the segmentation of a point cloud, Asafi et al. first define the *convexity rank* of a set of points as the portion of points that are in line-of-sight out of the total number of points of the set [44]. Next, the segmentation is computed with a clustering method [63] that allows to maximize the convexity rank of each segment.

Similarly, the method by van Kaick et al. [1] also employs the measure of convexity rank to segment meshes and point clouds, although the final segmentation is computed in a different manner. Specifically, the convex components are merged to obtain approximately semantic parts for the 3D shape, by using a measure of the volume of each part computed by adapting the SDF to point clouds. In Section 4.2, we will describe more details about the approach of van Kaick et al. to calculate the SDF for point sets, which we employ to compute a descriptor that captures the volume of our fine-grained segments.

2.2.2 Discussion

On the one hand, much progress has been made to compute 3D shape segmentations, especially for the case of 3D meshes. Furthermore, the maturity of the research on the topic has lead to the development of benchmarks for the evaluation of mesh segmentation techniques [60, 64]. On the other hand, many of the works on mesh segmentation are not directly applicable to point set representations, especially in the case of noisy or partially incomplete point clouds [1]. Moreover, some approaches for mesh segmentation require as input watertight manifold meshes [44]. Considering this, we extract the fine-grained segments required by our synthesis method directly from point clouds that we compute by sampling the input meshes, which allows us to handle non-manifold meshes.

As we mentioned in Section 2.1, many shape synthesis methods require as input a semantic segmentation of the 3D shapes of the input dataset. However, computing consistent semantic segmentations and part correspondences for objects of the same class is still a difficult problem [15, 16].

Although our proposed method does not rely on a semantic segmentation or a part correspondence of the input shapes, we do require fine-grained segments with a consistent size for corresponding parts across the different shapes of our input datasets, so that the segments can be reused for the synthesis of novel shapes. Therefore, we employ a semi-automatic approach that allows to acquire the consistent fine-grained segments that we require in our synthesis algorithm. In Chapter 5, we will describe in detail the methods that we use to compute the fine-grained segmentation of our input shapes.

Chapter 3

Method overview

In this chapter, we present an overview of our approach, while we discuss more details about our pipeline for shape synthesis in subsequent chapters. Figure 3.1 summarizes the complete pipeline of our method.

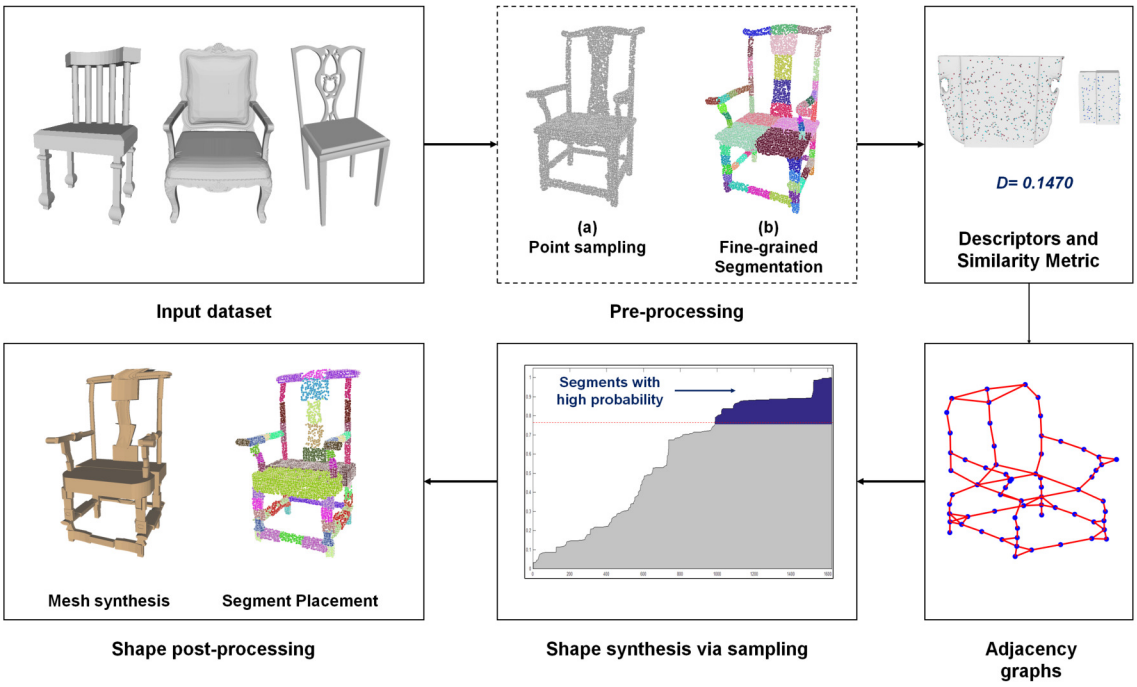


Figure 3.1: Overview of our 3D shape synthesis approach.

Input dataset. The input to our method is a set of triangular meshes from a given family or category. Figure 3.1 shows a few examples of shapes from our dataset of

chairs. Each input dataset contains shapes with diverse styles and geometric details. In addition, the shapes vary in their topology. For instance, our set of chairs contains shapes with and without arms, and shapes with and without support bars between the legs, among other variations. We extract a pool of fine-grained segments from the input dataset, which we use to synthesize new shapes based on a part recombination approach, described as follows.

Pre-processing. We assume that all our input shapes are consistently aligned. Note that this is a common requirement of many shape analysis methods [18], and such an alignment can be obtained manually or automatically with upright orientation methods [51,65]. Next, we normalize the scale of all the input shapes while preserving their aspect ratio. We provide more details on this procedure in Section 4.1.

We sample each mesh to create a point cloud representation, following the method described by Osada et al. [66]. The number of points sampled is given by a user parameter. Similarly to recent shape analysis methods [45, 67–69], we opt to work with point cloud representations to alleviate problems that can arise by the presence of non-uniformly sampled geometry and *non-manifold* shape structures. We explain the details about the algorithm that we employ for point sampling in Section 4.1. Finally, we segment each point cloud into a set of fine-grained segments. In our work, we obtain the segmentation with a semi-automatic tool that considers the geometry of the shapes and user guidance. However, in Chapter 5, we also present preliminary results for a fully automatic method to compute the fine-grained segmentation used in our approach.

Figure 3.1 shows examples of a point cloud from our input family of chairs, before (a) and after (b) our fine-grained segmentation. Each fine-grained segment that we use in our technique is a set of points extracted from one of the input point clouds. Hereafter, we will refer to the fine-grained segments employed in our approach simply as *segments* or *parts*. In addition, we use our segmentation tool to create annotations about the symmetry relations between pairs of fine-grained parts. We employ these symmetry annotations later in our method to constrain the symmetry of the synthesized shapes. We also mark the principal axis of alignment for each part to aid in the alignment of segments in the synthesized shapes.

Descriptors and similarity metric. For each segment created during pre-processing, we compute a set of descriptors that capture different characteristics of

the segments, like their general geometrical form, structure, and volume. We use these descriptors to define a distance metric that allows us to measure the similarity between two segments. Figure 3.1 shows an example of two segments and their distance. The similarity of segments will guide the synthesis process by indicating which segments can be successfully exchanged among shapes. We explain details about how we define the similarity metric in Section 4.2.

Adjacency graphs. For each input shape, we define a graph to capture the adjacencies between the parts of the model. Each graph has one node for each segment of the corresponding shape, and an edge between each pair of segments that are neighbors in the shape. Similarly to Jaiswal et al. [19], we consider that two segments are neighbors if the Euclidean distance between any pair of points, one on each segment, is less than a given threshold. We found experimentally that a threshold of 3.5% of the cubic root of the volume of the 3D model allows us to obtain graphs with correct connections between neighboring segments for the shapes of our input sets. In Section 4.3, we provide more details about the process used for computing the graphs. The adjacency graphs play a key role in our method at two different stages of the synthesis: (i) The graphs allow us to establish what types of segments typically appear adjacent to each other, according to their descriptors, which is valuable information for guiding the selection of segments during the synthesis; (ii) We use the graph of the template shape to guide the placement of the segments in the synthesized shape.

Shape synthesis via sampling. In our method, we provide as input a shape that will act as a *template* or *reference* to guide the synthesis. We process the template in the same manner as we pre-process our dataset of shapes, also obtaining a graph of segments for the template. Then, we pose the synthesis as a sampling of individual segments that are assigned to the nodes of the template's graph. Our goal is to sample segments so that the resulting model minimizes a shape energy.

In more detail, the shape energy is composed of two terms. The first term corresponds to the probability of replacing each segment i of the template with a segment r from our pool of segments. The second term considers the similarity between each given segment i of the template and its neighbors, in comparison to the similarity of the replacement r and its new neighbors in the synthesized shape. Therefore, this term captures the similarity between the original configuration of parts of the template and the configuration of parts of the new shape.

To synthesize a shape, we sample replacement segments r to minimize the shape energy. Since computing an optimal sampling according to the full energy is a difficult problem, we start by sampling segments based on the first energy term. For each part i of the template, we look in our full set of segments for a segment r that can serve as a *good* replacement for the segment i. To find such replacement, we convert the distance metric between the segment i and all the other segments into a probability density function (PDF). Then, we sample, according to the PDF, a segment with a probability higher than a given threshold, i.e., we select a segment that has a high similarity with respect to the original segment of the template. By randomly sampling from the PDF, we obtain different segments having a high replacement probability, which can serve as plausible parts for the creation of diverse new shapes. In addition, we enforce the symmetry of the synthesized models by selecting the same replacement for the segments of the template that were previously marked as symmetric. Finally, we compute the shape energy using both terms of the energy, to assign a probability to the entire shape. By sampling a collection of shapes in this manner, we can retain only the high probability shapes at the end of the synthesis process. Moreover, our method could be employed to design an application that could display to the final user only the shapes with the highest probabilities.

Shape post-processing. Given the segments selected to compose a novel man-made model, we obtain the final shape in a two-stage post-processing: we first define the placement of the segments and then create a triangle mesh from the point cloud geometry.

1. **Segment placement**. We compute a transformation such that each replacement segment r has approximately the same orientation, size, and position of its corresponding segment i of the template. This transformation keeps, nonetheless, the main geometrical features and aspect ratio of the surface represented by the 3D points of each segment r. Next, similarly to previous works [3, 5], we optimize the alignment between the segments of the new shape by finding a set of matching contact points between each pair of adjacent segments, and computing a translation and scaling that aligns the set of contact points.

2. **Mesh synthesis**. To create a triangle mesh from the synthesized point cloud model, one possible approach is to use a surface reconstruction technique, such

as Poisson surface reconstruction [70]. However, such an approach may over-smooth the final object and thus cause the loss of the fine detail that our method aims at transferring between shapes. Hence, we instead employ an approach based on directly transferring pieces of mesh geometry from the original shapes to the synthesized model.

Given a replacement segment r of the new shape and its source triangle mesh, we find the set of faces in the source model that correspond to the segment. Then, we apply the same transformation that we applied to align the point set of the segment r to the vertices of the face set. Since we perform the fine-grained segmentation directly on the point clouds of the input shapes, there are many cases where the boundaries of the fine-grained segments do not match the edges on the original input meshes. Thus, we first perform a subdivision of the faces of the original meshes, to ensure that all the edges are shorter than a given threshold. After this subdivision process, we obtain mesh segments with boundaries that approximately match the boundaries of the fine-grained segments. Finally, we also find a set of matching contact points between the neighboring face sets, and we compute an alignment between these contact points, in a similar way as we do for aligning neighbor segments in the synthesized point cloud.

The output of the synthesis process is thus a shape composed of fine-grained segments transferred from multiple shapes of the input collection. The shape follows the general topology of the template with appropriate connections between segments, but possesses the geometric details existing on the input shapes, as seen in the example in Figure 3.1.

Chapter 4

3D shape synthesis

In this chapter, we describe in detail the steps of our method for generating new 3D objects using a set of fine-grained segments. The details about the extraction of these fine-grained parts are discussed in Chapter 5.

4.1 Point sampling and segmentation

Pre-processing. The input to our method is a set of triangular meshes of the same family, where we consistently orient all the input meshes. After the alignment, the +Y axis of each mesh roughly corresponds to the upward direction of the objects, while the +X axis corresponds to the frontal direction. We perform the alignment manually, although it would be possible to incorporate automatic alignment methods [51, 65] into our pipeline to automate these steps. Next, we normalize the scale of our shapes while preserving their aspect ratio, so that the axis-aligned bounding box of all shapes is a cube centered at the origin whose longest edge is 2 units in length. Thus, all point dimensions are in the range [-1, 1].

Point sampling. Next, we transform each input mesh into a point cloud, using the sampling algorithm described by Osada et al. [66]. Although it would be possible to sample vertices of the 3D mesh, the resulting point set would be biased by the density of vertices sampled on the surface of the shape [66]. Instead, Osada et al. propose to randomly sample points from the surface of the 3D model. Specifically, we compute first the area of each triangle of the mesh, and we transform these areas into a probability distribution by dividing the area of each triangle by the total sum of the areas. Next, we use the distribution to randomly sample a triangle of the mesh,

with a probability proportional to its area. Then, we compute a random point p on the surface of the selected triangle with [66]:

$$p = (1 - \sqrt{r_1})\, v_1 + \sqrt{r_1}(1 - r_2)\, v_2 + \sqrt{r_1}\, r_2\, v_3,$$

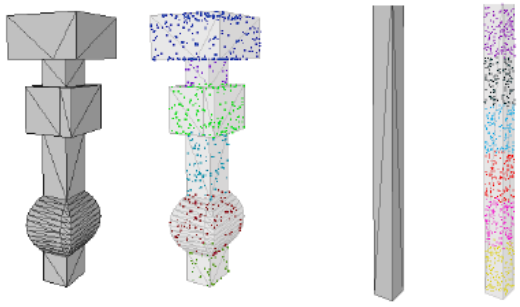
where (v_1, v_2, v_3) are the vertices of the given triangle, and r_1, r_2 are random scalars between 0 and 1. The entire process is repeated until the desired number of points is obtained. The number N of samples is given by a user parameter. For all the experiments that we discuss in this thesis, we sampled $N = 15,000$ points from each mesh. We empirically found that by using this number of points, we can obtain point clouds that capture in an appropriate manner the details of the input meshes. During the sampling, we also compute the normal of each face, and associate this normal with each point that was sampled from the given face. We use the normals associated with each point during the computation of some of our descriptors, as we describe in Section 4.2.

Segmentation. From each point cloud that we sampled from the input set of meshes, we then perform a segmentation that has two main goals. The first objective of the segmentation is to provide segments that capture fine geometrical details of the shapes, such as ornaments and salient features, which can then be transferred to the synthesized shapes. As an example, Figure 4.1(a) shows a mesh part representing a chair leg as well as the fine-grained segments computed by our segmentation for the leg. We can see that the fine-grained segments partition the leg into regions that capture distinctive geometrical features.

The second objective of the segmentation is to provide segments that are consistent in size for corresponding parts across the 3D objects of our input datasets, so that they can be easily reused for the synthesis of novel shapes. Thus, although some objects do not display elaborate decorations and salient boundaries, like the chair leg shown in Figure 4.1(b), we still segment such parts of the shapes into finer segments in a regular manner.

In addition, in this step of our method, we mark pairs of segments on the same shape that have reflectional symmetry [71, 72], and annotate the principal axis of orientation of each fine-grained segment. We use these annotations during our synthesis step, to create novel shapes that preserve the symmetry relations of the models in the input dataset, and to properly align neighboring parts in the synthesized shapes.

Following the procedure described above, we obtain a set of segments Ω that is consistent in size for corresponding parts of all the models of our input datasets, and contains also several parts with fine geometrical details that we can employ for synthesizing shapes with a wide range of variations. In the next sections of this chapter, we explain how we employ this set of segments Ω for shape synthesis, while we describe the details about the methods that we employ to extract and create annotations for our fine-grained segments in Chapter 5.



(a) **(b)**

Figure 4.1: Example of our fine-grained segmentation for two different parts: (a) A chair leg with rich local geometrical features; and (b) A chair leg without salient geometrical details or boundaries.

4.2 Descriptors and similarity metric

In this Section, we present the details about the descriptors that we use to determine the similarity between every pair of fine-grained segments of our set Ω.

4.2.1 Segment descriptors

We compute several descriptors that encode different properties of the overall geometry of each segment, such as form and volume. Furthermore, we analyzed many relevant previous works about shape synthesis and shape analysis to select descriptors that: (i) Capture the general similarities in the geometry of the segments extracted from our input shapes; and (ii) Allow us to generate shapes with a wide range of local variations. Additionally, we performed several experiments using different descriptors, and we found that our current set of descriptors can be used to obtain a

similarity metric that represents in an appropriate manner the overall similarities between the segments of our set Ω, as we will describe in Section 4.2.2. We classify our descriptors into two main categories, based on the level at which they are defined:

1. **Point-level descriptors.** We compute geometric descriptors for each point or pair of points of a segment, and then capture the distribution of the descriptor values for all the points with a histogram.

   - **Point feature histogram (PFH)**. As mentioned in Section 4.1, we associate with each point sampled from our input meshes the normal of each face or triangle from where the point was sampled. We use these normals to compute our PFH descriptor. The PFH captures the relative rotation between each pair of normals in a segment [73], which provides an indication of the overall variation in the orientation of the segment's surface. In addition, previous works have shown that the PFH is invariant to the density of the sampling [74, 75], and can be employed to classify the points of a point cloud into different classes such as cylindrical or planar, since the PFH can capture the overall shape of the underlying surface represented by the points [76]. Moreover, the PFH is also invariant to 3D rotations and translations [74]. We use the PFH in our system to obtain a general characterization of the form and the orientation of each one of our segments, independently of the number of points of the given segment.
     To obtain the PFH we follow the method described by Zhao et al. [76]. For each pair of points p and q of a segment, with normals n_p and n_q respectively, we calculate three angles α, ϕ and θ that represent the difference in orientation between n_p and n_q. First, we create a local coordinate frame placed at point p to calculate the angles. We define two unit vectors: $v = n_p \times \frac{q-p}{d}$, and $w = n_p \times v$, where $\times$ denotes the cross product, and d is the Euclidean distance between the points p and q, i.e., $d = ||p - q||$. Next, each angle is computed as follows:

     $$\begin{aligned} \alpha &= v \cdot n_q, \\ \phi &= n_p \cdot \frac{q-p}{d}, \\ \theta &= \arctan(w \cdot n_q, n_p \cdot n_q), \end{aligned}$$

     where $\cdot$ denotes the dot product and $\arctan(y, x)$ computes the arc tangent

of y/x in a robust manner. We then divide the values of each angle into $b = 5$ bins of equal size, to create a histogram of length $b^3 = 125$. Finally, we normalize the histogram dividing by the number of point pairs in the segment to achieve point density invariance [74].

- **Point distribution histogram (PDH)**. We compute a histogram that estimates the mass distribution of each segment, by capturing the distribution in 3D space of the points of the segment [9].
  Similarly to Lun et al. [9], we compute this descriptor by dividing the bounding box of the segment into a grid of size $4 \times 4 \times 4$, and creating a histogram with 64 bins by counting the number of points contained in each cell of the grid. To normalize this histogram, we divide it by the total number of points in the segment.
  The point or mass distribution has been employed as a feature for normalizing the position of 3D models, and to compare 3D shapes in model retrieval applications [77, 78]. We also employ the mass distribution as a feature to perform a comparison between 3D objects, although in our case, such 3D objects are segments of our set Ω instead of complete 3D shapes.

- **Volume (SDF)**. We capture a profile of the volume of each segment by employing the Shape Diameter Function (SDF) [31]. The SDF captures the local volume at a specific point on a surface. Thus, by aggregating the SDF for all the points in a segment, we create a histogram capturing the segment's volumetric profile.
  For computing the SDF for a set of points, we adopt the method by van Kaick et al. [1]. Let i_s represent the segment for which we are going to compute a volumetric profile. We compute the SDF for a given subset of points of the segment i_s. If the number of points of i_s, which we denote by $|i_s|$, is greater than 100, we subsample a set of points from the segment using a Farthest Point Sampling procedure [79], where the number of subsampled points N_{sub} is given by the maximum between 100 points and 10% of the total number of points in the segment, i.e.: $N_{sub} = \max(0.1|i_s|, 100)$. However, if $|i_s| \leq 100$, we keep all the points of the segment in the subset of points used to compute the SDF.
  To obtain the value of the SDF for a given point p of i_s, we create a cone that has its apex at p, is oriented towards the opposite direction of the

normal of p, and has an opening angle of $\frac{1}{3}\pi$. We then trace rays starting at p and falling inside the cone. We find the intersection of these rays with local surface approximations of the points on the other side of the segment. We weight the length of each of these rays by the cosine of the angle between the normal of the point p and the ray, to penalize rays that deviate too much from the normal. Afterwards, we take the median of these lengths as the value of the SDF at point p.

Finally, our volume descriptor is defined as a histogram of 10 bins that represents the distribution of the values of the SDF for the set of points of segment i_s. As in the case of the point distribution histogram, we normalize the SDF histogram by the number of points considered in the histogram.

- **Surface Variation (SV)**. We also calculate a descriptor that captures the *surface variation* represented by the points of each segment. To compute this descriptor, we follow the method of Pauly et al. [80]. We obtain a neighborhood N_p for each point p of each segment, using $k = 10$ nearest neighbors. Then, based on the neighborhood N_p, we compute the 3×3 covariance matrix C_p for each point p as:

$$C_p = \sum_{p_j \in N_p} (p_j - \mu_p)^\intercal (p_j - \mu_p),$$

where μ_p is the centroid of the point set N_p. Next, we compute the eigenvalues $\lambda_0 \leq \lambda_1 \leq \lambda_2$ and their associated eigenvectors v_0, v_1, and v_2 of the matrix C_p. Previous works have demonstrated that the plane $T(x) : (x - \mu_p)\, v_0 = 0$ that passes through μ_p minimizes the sum of squared distances to the neighbors of p, and thus v_0 approximates the surface normal n_p at the point p [80, 81]. Moreover, since the eigenvalues λ_l with $1 \leq l \leq 3$ measure the variation of the points N_p along the direction of the corresponding eigenvectors, then λ_0 measures the variation along the surface normal. Hence, the surface variation at the given point p can be calculated as [80]:

$$\nu(p) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}.$$

The surface variation is a local surface property that is closely related to the curvature of the surface represented by a set of points, but can be

computed more robustly [80]. For instance, a value $\nu(p) = 0$ indicates that all the points of N_p lie on the same plane.

Similarly to our volume descriptor, we compute a histogram of 10 bins to encode the value of the surface variation for all the points of each segment, and normalize the histogram entries by the sum of all bins.

2. **Segment-level descriptors.** The following descriptors are defined at the level of segments, without involving individual properties of points.

   - **Overall geometry (OG)**. This descriptor captures the overall geometric shape of a segment. We encode the descriptor with a three-component vector $[\mu_r, \mu_s, \mu_t]$ [10]:

$$\begin{aligned} \mu_r &= \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3}, \\ \mu_s &= \frac{2(\lambda_2 - \lambda_3)}{\lambda_1 + \lambda_2 + \lambda_3}, \\ \mu_t &= \frac{3\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}, \quad with \quad \lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0, \end{aligned}$$

     where λ_1, λ_2, and λ_3 are the three eigenvalues obtained when applying Principal Components Analysis (PCA) on all the points of the segment. The overall geometry descriptor allows to estimate how linear, planar, or spherical the shape of the given segment is [10, 69].

   - **Relative position (RP)**. This descriptor aims at capturing the relative position of a segment with respect to its containing shape [9, 69], taking into account the consistent pre-alignment of all the input shapes. Specifically, this descriptor consists of two relative position values. To obtain the first value, we project both the centroid of the given segment and the centroid of the shape itself onto the ground plane of the shape, defined in relation to the upright orientation of the shape, and we record the distance between the two projected centroids. To compute the second descriptor value, we compute the *average height* of the segment as the distance between the ground plane of the shape and the projection of the segment's centroid onto the upright vector.

     The use of this descriptor could imply that we can generate some 3D shapes that do not have much variations in comparison to the original input 3D

models, since some segments that could contain several fine local variations in comparison to the original segments of a given input shape can be found in very different positions in other shapes. Nevertheless, after inspecting several results generated in different experiments performed with our method, we found that including this descriptor allows us to preserve in a better manner the general structure of the input shapes in our synthesized man-made objects. Still, as part of the future work for our project, we would like to perform additional experiments with new datasets to analyze in more depth the relevance of the relative position descriptor in our similarity metric.

- **Relative bounding box (RBB)**. Finally, we compute the minimum and maximum coordinate along each dimension for all the points in a segment, which captures the axis-aligned bounding box of the segment relative to its source shape. As we mentioned earlier, we require our segments to be consistent in size for the different shapes of our input set, so that we can find several segments in our set Ω with a high similarity in their general form with respect to each segment of the template that we employ to guide the synthesis process. Thus, by including this descriptor, we seek to consider the relative size of our segments as a feature that allows us to maintain the structure of the input 3D models in our novel shapes. Still, as we will see in Section 4.4, when we select from our set Ω the parts that we will use to synthesize a 3D shape, we discard segments that have dimensions that are very different in comparison to the size of the original segment of the template shape. Thus, like in the case of the relative position, we could consider that our RBB descriptor is probably less relevant in comparison to other descriptors that we employ in our method. However, as we will see in Chapter 6, using our current set of descriptors we can synthesize shapes that preserve in a correct manner the general geometrical form of the input 3D objects.

In addition, we should note that we use a handcrafted set of descriptors following previous shape analysis and synthesis methods, such as [3, 4, 9, 69], instead of using machine learning approaches that seek to directly learn a set of geometric descriptors, since many of these techniques require input data that can provide additional structure information, which can be obtained, for instance, from RGB-D data [82,83].

4.2.2 Similarity measure between segments

Given two segments i and j and their descriptors, we define their distance D in descriptor space as:

$$D(i,j) = \sqrt{D_p(i,j) + D_s(i,j)}, \quad \text{where:} \tag{4.1}$$

$$\begin{aligned} D_p(i,j) &= \sum_{d=1}^{4} \mathrm{EMD}^2(h_i^d, h_j^d), \quad \text{and} \\ D_s(i,j) &= \|\mathrm{OG}_i - \mathrm{OG}_j\|^2 + \|\mathrm{RP}_i - \mathrm{RP}_j\|^2 + \|\mathrm{RBB}_i - \mathrm{RBB}_j\|^2, \end{aligned} \tag{4.2}$$

where $D_p(i,j)$ denotes the distance measure for the point-level descriptors, while h_i^d denotes the histogram for the segment i and point-level descriptor d, and EMD is the Earth-Mover's Distance. The EMD allows to measure the similarity between two probability distributions given in the form of histograms [1, 10]. In addition, $D_s(i,j)$ is the distance for our three segment-level descriptors, where the descriptors are denoted according to the acronyms defined above. After computing the similarity between each pair of segments in our set Ω, we normalize all the distance values to the range $[0,1]$.

Although we performed several experiments using different weights for each descriptor included in our similarity metric, we did not observe significant improvements in our results by using different weights. Nonetheless, we consider that some of our descriptors could be more informative than others. Thus, as future work for our project, we would like to perform additional experiments using different weights for each descriptor of Eq. (4.2). Further, similarly to previous works, e.g., [9, 84], we could explore the option of using Radial Basis Function (RBF) kernels, or other machine learning techniques, to learn automatically the most appropriate weights for our descriptors.

We can define the similarity of two segments by taking the inverse of the distances, or by deriving a probability distribution from the distances. The distance measure allows us to discriminate fine-grained segments with high similarity from segments with significant differences. More specifically, our descriptors encode information of the general geometry of the segments but, at the same time, our descriptors are also relatively insensitive to the fine-detail variations represented by the points of each segment. Therefore, our similarity metric allows us to find segments that have a high similarity in their overall geometry, but can also have many fine-detail variations.

Figure 4.2 shows an example of the distance or similarity between two different pairs of segments. The two segments shown in the left and middle columns of Figure 4.2 have a small distance of 0.05 since this pair of segments have a similar overall geometry: they both were extracted from a part of two different shapes that have a cubic form on one side (right) of the segment attached to a planar decoration on the other (left) side. On the other hand, the two segments shown in the middle and right columns of Figure 4.2 have a much larger distance value of 0.61, denoting the dissimilarity in the general form of the two segments: the segment on the right side is smaller and was extracted from a cubic part that does not have attached any additional decoration. In Section 4.4, we will describe how we use the discriminative power of our distance measure to select the segments that will compose the synthesized 3D shapes.

4.3 Adjacency graphs of shapes

For each shape S in our input dataset, we create a graph G_S that has a node corresponding to each segment of the shape, and an edge between each pair of segments that are neighbors in the 3D object. To determine if a pair of segments i and j of a common shape S are neighbors, we adapt the method employed by Jaiswal et al. [19]. Specifically, we say that the segments i and j are neighbors if we can find a pair of points $p \in i$ and $q \in j$ such that: $||p - q|| \leq 0.035 \sqrt[3]{V}$, where V is the volume of the bounding box of the shape. We obtained the threshold value of 0.035 experimentally. We observed that by using this value, we obtain graphs that connect in a correct manner the segments that are adjacent in the shapes of our experimental datasets. On the other hand, using the volume V of each shape allows us to adapt the computation of the graph to each particular shape, instead of directly using a fixed value to compare the distances between the segments of our input shapes. We use this method to construct graphs that represent the adjacency between the segments of every shape in our input datasets, including the shapes that will be used as templates during the synthesis process. In Figure 3.1, we show an example of the graph for one of the models in our family of chairs. As we discuss in the following sections, the adjacency graphs play a very important role in several steps of our synthesis pipeline.

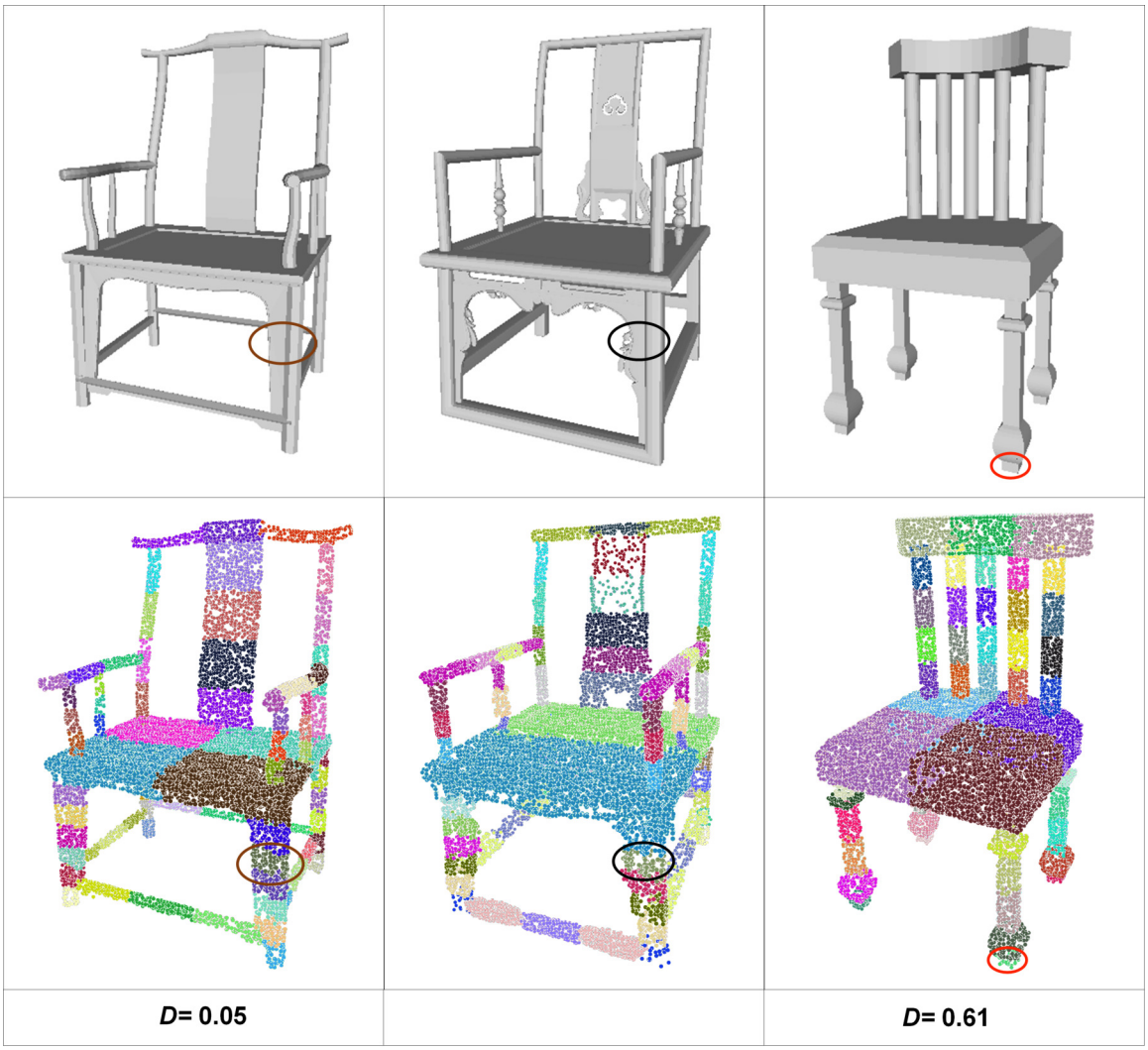


Figure 4.2: The values of the distance or similarity measure for a pair of highly similar segments (left compared to the middle column), and a pair of dissimilar segments (middle compared to the right column).

4.4 Shape synthesis via sampling

In this Section, we discuss the methodology that we employ for selecting, from all the fine-grained segments of our set Ω, the specific subset of segments that we use for generating a novel 3D shape, under the guidance of a template shape.

4.4.1 Similarity score for adjacent segments

As the first step for synthesizing a novel shape, we summarize the neighborhood of each segment in Ω by computing a score involving its adjacent segments. More precisely, for each segment i of a shape S, we use the adjacency graph G_S of the shape S to find the segments that are neighbors of i, and we sum the value of the distance D between i and all the segments that are neighbors of i in S. Thus, for each segment i, we compute this score in the following manner:

$$H_i = \sum_{j \in N_i} D(i, j), \tag{4.3}$$

where D is the similarity or distance measure defined in Eq. 4.1, and N_i is the set of segments that are neighbors of i in the shape S. After calculating all the scores H for every segment of the shape S, we compute the mean $\overline{H_S}$, and standard deviation $\sigma(H_S)$ for the set of scores H_S of the shape S. Next, to obtain a standardized score for each segment i, we normalize the score H_i using the mean and standard deviation of the scores of the shape:

$$\hat{H}_i = \frac{(H_i - \overline{H_S})}{\sigma(H_S)}. \tag{4.4}$$

Since the score given by Eq. (4.4) captures the similarity between each segment and its neighbors in the shape that contains each segment, we employ this score, in a later stage of our process, to perform a comparison between two segments in terms of the corresponding context or neighborhood of each segment.

4.4.2 Template selection

To confer a plausible structure to the 3D objects that we generate, we select one shape from our input dataset which we use as a template or reference for synthesis. This template $\mathcal{L}$ guides the synthesis process, determining the global structure and the topology of the generated shape.

4.4.3 Shape energy

To synthesize a 3D shape with variations in its local geometry while maintaining a plausible structure, we approach the problem as that of sampling segments to compose the synthesized shape. Specifically, we find segments in our pool of segments Ω that

can replace each segment i of the template $\mathcal{L}$ while forming an adequate shape. We formulate this goal as the minimization of a shape energy function given by:

$$E(m) = \sum_{i \in T} E(i, m(i)), \quad \text{with:} \tag{4.5}$$

$$E(i, m(i)) = D(i, m(i)) + \sum_{j \in N_i} |D(i, j) - D(m(i), m(j)|, \tag{4.6}$$

where T is the set of segments that form the template $\mathcal{L}$, D is our similarity measure defined in Eq. 4.1, m is a mapping that encodes the assignment of segments in Ω to the segments of T, and N_i is the set of segments that are neighbors of i, according to the graph of the template $\mathcal{L}$, which we will denote as $G_{\mathcal{L}}$. Thus, the goal of the synthesis is to compute the mapping m. We explain the synthesis process in more detail in the next section.

The energy defined in Eq. (4.5) estimates whether the segments selected to compose a new shape contribute to create a plausible object. The estimation involves the two terms of Eq. (4.6). The first term is a unary energy that measures the overall similarity between segments i of the template and their replacements $m(i)$. The second term of Eq. (4.6) is a pairwise energy that measures the similarity between the context of each segment in the synthesized shape and the context of the original segment, similarly to the score for adjacent segments that we introduced in Section 4.4.1. Specifically, the second term measures the similarity of each pair of adjacent segments i and j of T to the their corresponding replacements $m(i)$ and $m(j)$, thus ensuring that segments that are adjacent in the synthesized shape are as similar as in the template. As we discuss in the following section, we employ Eq. (4.5) to calculate a probability that indicates how plausible a synthesized shape is, which can be used to guide the synthesis process.

4.4.4 Sampling of segments for shape synthesis

To synthesize a plausible shape, we construct the mapping m so that the energy $E(m)$ given by Eq. (4.6) is minimized. Since directly minimizing Eq. (4.6) is only possible in the case of certain convex energies [23], we employ a heuristic sampling approach instead. To perform the sampling of the segments that will compose a new shape, we transform the distances between a segment i in the template and all the segments in the set Ω into a discretized Probability Density Function (PDF) by means of the

following expression:

$$P_i(r) = \frac{\exp(-D(i,r))}{\sum_{j\in\Omega}\exp(-D(i,j))}, \tag{4.7}$$

where $P_i(r)$ is the probability of segment $r \in \Omega$ being an appropriate replacement for the original segment $i \in T$, according to the similarity given by Eq. (4.1). In our heuristic algorithm, we randomly sample a segment with a probability higher than a given threshold τ, to obtain a segment with a high similarity in comparison to the segment $i \in T$. Unless otherwise noted, for the experiments that we performed for this thesis, we sampled segments with a probability higher than $\tau = 0.75$. In addition, we also consider the following filtering conditions to compute the final mapping m:

- We employ the similarity score $\hat{H}_i$ defined by Eq. 4.4, to measure the difference between the score of the original segment $i \in T$ and the corresponding score of the segment r sampled from the PDF. Thus, we compute the following difference: $\hat{h} = |\hat{H}_i - \hat{H}_r|$. Then, we filter out segments where the value of $\hat{h}$ is larger than a given threshold. In practice, we filter out segments where $\hat{h} > 2$. Since the score $\hat{H}$ considers the similarity between the neighbors of each segment, this step allows us to discard segments that belong to a context that is very dissimilar in comparison to the context of the original segment of the template.

- In addition, we compute the ratios between the length of each pair of sides of the bounding boxes of each segment, and we find the difference between the maximum of these ratios for each pair of segments in the set Ω. We then discard segments where such difference is larger than a threshold $\eta = 1.8$, which was determined experimentally.

In general, we also filter out any segment coming from the template shape, to obtain synthesized shapes with more variety in its fine details in comparison to the original template shape. Nonetheless, our method also allows to synthesize novel 3D objects preserving some selected segments of the template, which let us produce additional variations in the synthesized shapes, as we will see in Chapter 6.

After the process described above is repeated for each possible segment of the template, we have effectively created a new shape. Thus, we can compute the value of the first term of our energy function in Eq. (4.6) as the probability associated with the segment r sampled from the distribution obtained as stated previously. To compute the second term of our energy function, we use the adjacencies indicated by

the graph $G_{\mathcal{L}}$ to compute a pairwise PDF for the distances or similarities between each pair of segments adjacent to i, and the similarities between the corresponding neighbors selected for the new shape. Thus, this pairwise PDF can be expressed as:

$$P_i^{PW}(m) = \exp\left(-\sum_{j \in N_i} |D(i,j) - D(m(i), m(j)|\right), \tag{4.8}$$

where N_i is the set of segments that are adjacent to i according to the graph $G_{\mathcal{L}}$. After normalizing Eq. (4.8) by the total number of segments of Ω, we compute the cumulative sum for the values of this distribution. We can use this cumulative sum to extract from the distribution the probability that represents the value of the second term of Eq. (4.6). In this manner, we have obtained the value of the energy for each segment $r = m(i)$ of the synthesized shape, as the sum of the two terms in Eq. (4.6).

Furthermore, to obtain a shape that preserves the symmetries found in our input family of shapes, we use the list of symmetric pairs created during the segmentation stage, to determine if each segment i of T has a symmetric segment j on the template. For such symmetric pairs, we compare the total value of the probability computed by means of Eq. (4.6) for each corresponding replacement segment $r = m(i)$ and $s = m(j)$, and we keep in our synthesized shape the segment having the highest probability. During our shape post-processing step, we reflect this segment through the corresponding symmetry plane of the shape [71]. In this manner, we build a 3D shape that maintains the main symmetry relations of the template $\mathcal{L}$.

Finally, we sum the values of the energies for each segment i of T, to obtain the total shape energy value for the new shape, as specified by Eq. (4.5). Using the procedure described above, we can sample different segments from the unary probability distribution computed for each segment i, which let us generate several new shapes from the same template $\mathcal{L}$. Since the energy defined by Eq. (4.6) is the sum of two values sampled from the two probability distributions, the values of the energy computed for each segment are in the range $[0, 2]$. We can employ the value of the shape energy to create a sorted list of the novel shapes, given by its probability, or to present to the final users of our method only the shapes with the highest probability.

4.5 Shape post-processing

Given the segments selected to synthesize a new shape, we finalize the creation of a novel 3D object with a two-stage procedure. First, we synthesize a point cloud by placing the segments defined by the mapping m according to the position, size, and orientation of the segments of the template T. Next, we build a new triangular mesh by finding parts in the source mesh corresponding to each fine-grained segment in the synthesized point cloud. We explain these two stages in more detail in the following subsections.

4.5.1 Segment placement

To generate a point cloud for a novel object from the mapping m, we compute, for each replacement segment, a transformation composed of a rotation, a non-uniform scaling, and a translation that allows us to find its optimal placement in the synthesized shape, with respect to the corresponding size, scale and orientation of the original segment of the template.

First, given a replacement segment r, we compute a rotation so that the orientation of the segment approximately matches the orientation of the original segment i of T. To obtain this rotation, we extract the oriented bounding box (OBB) of the segments r and i, employing the method of Fish et al. [36]. We then find an angle β and axis $\vec{u}$ of rotation, using the *principal axis* of each OBB, which is annotated by the user for each segment. Specifically, we compute $\vec{u} = \vec{v_i} \times \vec{w_r}$ and $\beta = \arccos(\vec{v_i} \cdot \vec{w_r})$, where $\vec{v_i}$ and $\vec{w_r}$ are the principal axes of the OBBs of the segments i and r, respectively. Finally, we obtain the rotation matrix given by the angle β and the axis of rotation $\vec{u}$ [85], and apply the matrix to rotate r.

To obtain the principal axis of orientation of each fine-grained segment, we performed several experiments using the principal component given by PCA, as proposed in previous works [9, 19, 86]. However, we found that this approach did not provide consistent results for all our fine-grained segments. Thus, we resort to the user-given annotations on the principal axis of each segment, which are specified during the segmentation process.

Afterwards, we compute a non-uniform scaling to make the proportions of the bounding box of the segment r match the size of the bounding box of i. In addition, we obtain a translation from the position of the replacement segment r towards the

position of i by calculating the difference between the centroid of the segment i and the centroid of r. In this manner, we compose a transformation that we apply to each fine-grained segment of our novel shape, yielding a complete point cloud that has a structure that closely follows that of the template. At the same time, the replacement segment r can contain fine details that are different from those of the original part i of the template. Thus, our synthesized shape will also exhibit the fine detail variations of the segment r.

It is important to note that, as mentioned in Section 4.4.4, the method that defines the mapping m selects the same replacement r for symmetric segments of the template $\mathcal{L}$. Therefore, in practice, we only compute the transformations described above for one of the symmetric segments of the template T, and reflect the transformed segment through the main symmetry plane of $\mathcal{L}$, generating, in this manner, a new shape that respects the symmetry properties of the template.

Local alignment between adjacent segments

Due to the differences in the geometrical features between the original segments of the template and the segments on the synthesized point cloud, after applying the alignment procedure described above, the novel shape can still have disconnections and misalignments between its neighboring segments. Hence, similarly to previous works [3, 5], we refine the placement of the parts of the new shape by aligning a set of contact points between the neighboring segments.

More precisely, we obtain a set of four matching contacts for each segment boundary that will be aligned. Suppose first that we would like to align two neighboring segments r and s. To acquire the set of contact points, we start by computing the OBBs for the segments r and s. Next, we find the face on the OBB of r that is closest to the points of the segment s, denoted f_r, and, likewise, the face on the OBB of s that is closest to the points of the segment r, denoted f_s. Then, we compute the area of the faces f_r and f_s. We assign as the first set of contacts the four corners of the face that has the smallest area. Let us suppose, w.l.o.g., that f_r is the face with the smallest area. Then, we assign as the four contact points for the segment r the four corners of the face f_r. To find the contact points for the other segment, that is, the contacts for the segment s, we project the four contact points of the segment r towards the plane given by the face f_s and we assign as contacts for the segment s the four points of the segment that are closest to each point that was projected on the

face f_s. Figure 4.3 shows an example of the synthesized point cloud of a chair before the alignment process, where we zoom in two adjacent segments, which we identify as r and s. In addition, Figure 4.3 displays the corresponding contact points of each segment, and illustrates each pair of matching contact points with corresponding matching colors.

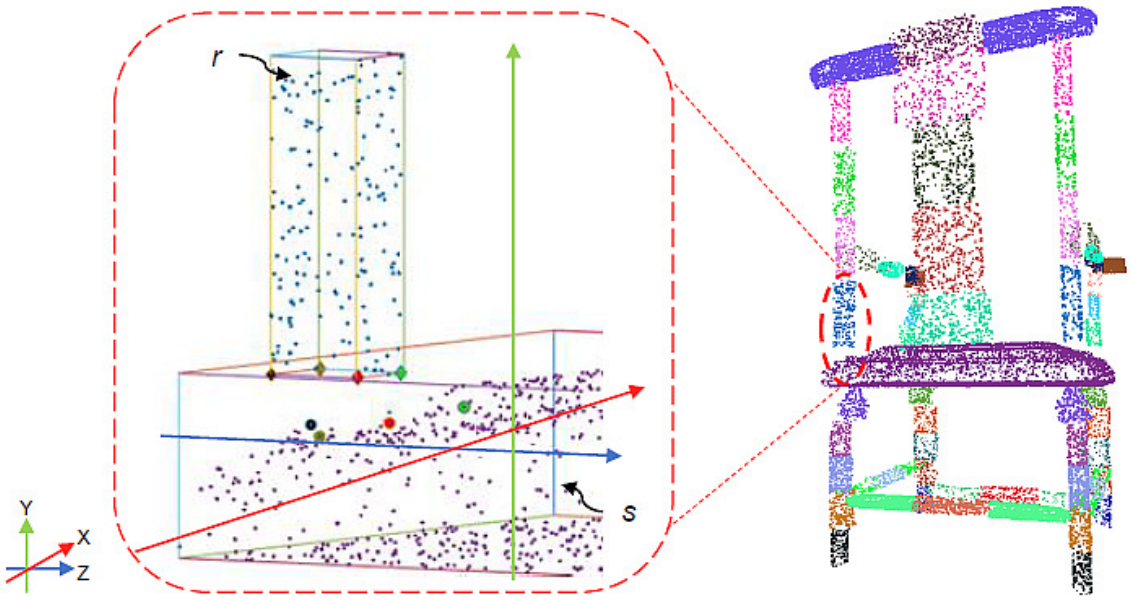


Figure 4.3: A synthesized point cloud before the alignment between adjacent components, showing two neighboring segments and their contact points

Using the set of four pairs of contacts, we compute the alignment between adjacent segments in a similar manner to previous works [3, 5, 6]. Our local alignment between neighbor segments is composed of a translation and a non-uniform scaling that optimizes the alignment between the contact points in a least-squares sense [5]. We find an optimal translation computing the difference between the centroid of the contact points of r, and the centroid of the contact points of its neighbor segment s. To compute the scaling for the segment r, we calculate the average of the scalings required to align each contact point of r to its matching contact points on s.

Furthermore, to perform the alignment for all pairs of segments in the synthesized shape, we traverse the graph of the template $G_{\mathcal{L}}$ by means of a breadth-first search algorithm [87], aligning neighboring segments during the traversal. This procedure allows us to align the segments of the shape in an incremental manner, avoiding the need to realign segments that were already aligned in earlier stages of the process.

Specifically, we select one of the segments of T that has the largest number of neighbors in the template shape $\mathcal{L}$ to start the traversal of $G_{\mathcal{L}}$. Let r be the replacement segment on the synthesized shape corresponding to this initial node or segment of T. We align each neighbor s of r based on the contact points of r and s. Next, we continue the traversal by selecting one of the neighbors of r, say t, and we compute the alignment between t and each one of its respective neighbors in the novel shape, before moving to the next neighbor of r. We continue this process until we have aligned each pair of adjacent neighbors of the synthesized shape. By applying the alignment for each pair of neighboring segments, we can generate a final point cloud where the placement of each segment is optimized relative to each other. Although the procedure described above allows us to compute the alignment in a correct manner for most of our input 3D shapes, there can be, nonetheless, some cases where we can find conflicts while traversing the graph $G_{\mathcal{L}}$. Such conflicts can occur when a pair of adjacent segments that should be aligned have been aligned previously with respect to some other neighbor segment. To avoid disconnecting segments that have been aligned in a previous stage of the process, we mark the segments that we align while we perform the traversal of the graph $G_{\mathcal{L}}$, so that we "freeze" in place each segment that has been aligned.

Finally, after we finish the placement and alignment of all the segments of our synthesized point cloud, we store the full set of transformations that we computed for each one of these segments. We will use this set of transformations to synthesize a new mesh, as we describe in the following section.

4.5.2 Mesh synthesis

To create an output model with a well-defined surface, we directly transfer parts of the input 3D meshes to compose the novel mesh. However, since the fine-grained segments are extracted from the sampled point clouds, there are many cases where the boundaries of the segments do not match the edges of the triangles on the meshes, as we can see, for example, in Figure 4.1b. To obtain mesh segments whose boundaries match the boundaries of the fine-grained segments, we first compute a subdivision of the input meshes.

Mesh subdivision

We compute the subdivision of each mesh from our input dataset using the Loop [88] scheme, which iteratively splits all the faces having edges longer than a given threshold into four triangles. For our subdivision process, we employ a threshold $\xi = 5\%$ of the length of the longest edge of $\mathcal{M}$. Thus, we divide into four triangles all the faces of $\mathcal{M}$ that have edges longer than this threshold, by placing a new vertex in the middle of the edges of each face that will be subdivided, which effectively splits, in each iteration, the edges longer than the threshold ξ by half. The process is repeated until no edge of $\mathcal{M}$ is longer than this threshold. The final result is a subdivided mesh $\hat{\mathcal{M}}$ where all the edges have a length smaller or equal than the threshold ξ. Note that, differently from the original Loop scheme, we do not optimize the position of the newly-created vertices to smooth the meshes during the subdivision, since our goal is mainly to increase the resolution of the meshes.

Extraction of mesh segments

Our next objective is to extract, from the subdivided meshes, a set of mesh segments corresponding to each fine-grained segment in our set Ω. More precisely, let i be a fine-grained segment that was extracted from a point cloud that was computed from an input mesh $\mathcal{M}$, which in turn, was subdivided generating a mesh $\hat{\mathcal{M}}$. Then, we seek to obtain the set of vertices $V_i(\hat{\mathcal{M}})$ and faces $F_i(\hat{\mathcal{M}})$ corresponding to the segment i. Since many of the faces of the mesh $\hat{\mathcal{M}}$ will not contain sampled points, we cannot extract only the faces containing points sampled, since we would obtain mesh segments with many holes and inconsistent boundaries. Instead, to obtain consistent mesh segments corresponding to the fine-grained segments extracted from the point clouds, we employ a heuristic procedure that involves four stages:

1. We start the process by obtaining an initial approximation for the set of vertices and faces of the mesh segment, denoted $V_i^*(\hat{\mathcal{M}})$ and $F_i^*(\hat{\mathcal{M}})$. We define $V_i^*(\hat{\mathcal{M}})$ as the subset of vertices of the subdivided mesh $\hat{\mathcal{M}}$ that lie inside the boundaries of the bounding box of segment i. The initial set of faces $F_i^*(\hat{\mathcal{M}})$ is the subset of faces of $\hat{\mathcal{M}}$ that contain any vertex belonging to the set $V_i^*(\hat{\mathcal{M}})$.

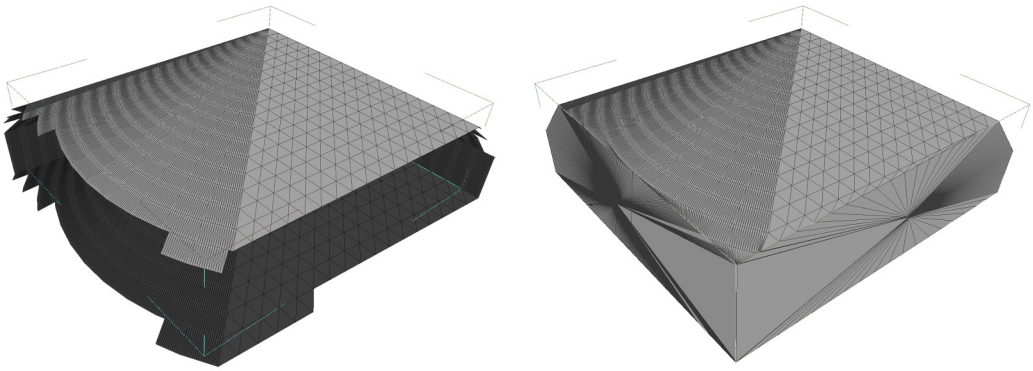
2. As we mentioned in Section 4.1, we associate the normals computed from the faces of the original input mesh $\mathcal{M}$ with each point of our fine-grained segments. In this step, we employ these *point normals* to refine the initial set of faces

$F_i^*(\hat{\mathcal{M}})$. Specifically, we start by computing the normal of each face of the set $F_i^*(\hat{\mathcal{M}})$. Then, we calculate and store the angle between each one of these face normals and the normals of the points of the segment i. Next, we discard from the set $F_i^*(\hat{\mathcal{M}})$ the faces for which all the stored angles are larger than a threshold of $\frac{\pi}{2}$, to obtain a refined set of faces $F_i^r(\hat{\mathcal{M}})$. In our experiments, we observed that in most cases this threshold allows us to filter out additional faces that should not be included in the mesh segment. Next, from the refined set of faces $F_i^r(\hat{\mathcal{M}})$, we obtain the final set of vertices $V_i(\hat{\mathcal{M}})$ of the mesh segment as the set of vertices that belong to all the faces of the refined set $F_i^r(\hat{\mathcal{M}})$. Nevertheless, as we will see in Chapter 6, the heuristic process described above might fail to exclude, in some cases, faces that should not be part of a given segment, producing artifacts in some of our synthesized meshes. In Chapter 7 we will discuss some alternatives that could be employed to generate a mesh directly from our synthesized point clouds, thus avoiding the need of extracting mesh segments from the input meshes.

3. The mesh segments obtained as described above will likely contain faces with edges that lie slightly outside or slightly inside the boundaries of the bounding box of the original fine-grained segment i, yielding mesh segments with jagged boundaries, as shown in the example of Figure 4.4(a). To correct this problem, we find the vertices of the boundaries of the set $V_i(\hat{\mathcal{M}})$ that do not match exactly the boundary of the bounding box of the segment i. Then, we project these vertices towards the plane given by the corresponding closest face of the bounding box of the segment.

4. Finally, to avoid having holes in the synthesized shapes, we add faces to the set $F_i^r(\hat{\mathcal{M}})$ to close the sides of the mesh segments that are open, yielding the final set of faces of the mesh segment $F_i(\hat{\mathcal{M}})$. Figure 4.4 shows an example of a mesh segment during the first stage of the extraction process, in comparison to the same mesh segment after the refinement, i.e., after this fourth stage.

In practice, the processes of mesh subdivision and extraction of mesh segments are performed in an earlier stage of our pipeline. Thus, we can employ the pre-computed mesh segments for synthesizing the new mesh, as we will see in the following sections.



(a) (b)

Figure 4.4: Example of a mesh segment: (a) before the refinement; and (b) after refining the boundaries and closing the sides of the segment.

Placement of mesh segments

As we explained in Section 4.5.1, as part of the process of generation of a novel point cloud, we record the transformations that we applied to each fine-grained segment of the synthesized shape. Hence, in this step we can transform the vertices of each mesh segment employing the transformation that was stored during the point cloud synthesis for the fine-grained segment that corresponds to the mesh segment.

Local alignment between adjacent mesh segments

After extracting and placing each mesh segment, we also optimize the alignment between the adjacent mesh segments on the synthesized 3D object, by means of a local alignment analogous to the process that we employ during the synthesis of point clouds. Hence, we must find a set of vertices on each adjacent mesh segment that we can use as contact points for the alignment process. We experimented different options to obtain the contact points for our mesh segments. In particular, following previous works on mesh synthesis, e.g. [3], we explored the idea of assigning as the contact points of a segment all the vertices close to an adjacent mesh segment, according to a given distance threshold. However, using our subdivided meshes, we observed that in many cases, we obtained a large number of contact points that not only included vertices on the boundary of the mesh segment closer to the neighbor segment, but also non-boundary vertices, which produced intersections and misalignments between the adjacent segments. Hence, to obtain the set of contact vertices of each adjacent

mesh segment, we used the same process that we described previously to obtain the contact points for aligning the adjacent segments of our synthesized point clouds. Nonetheless, as we will discuss in more detail in Chapters 6 and 7, our approach might still yield incorrect results in the alignment process in some cases.

Additionally, we also compute the alignment of the adjacent mesh segments of the new mesh by traversing the nodes of the graph $G_{\mathcal{L}}$ using a breadth-first approach, in the same manner as we did for the generation of a new point cloud. Figure 4.5 shows an example of a synthesized mesh, before and after the alignment process.

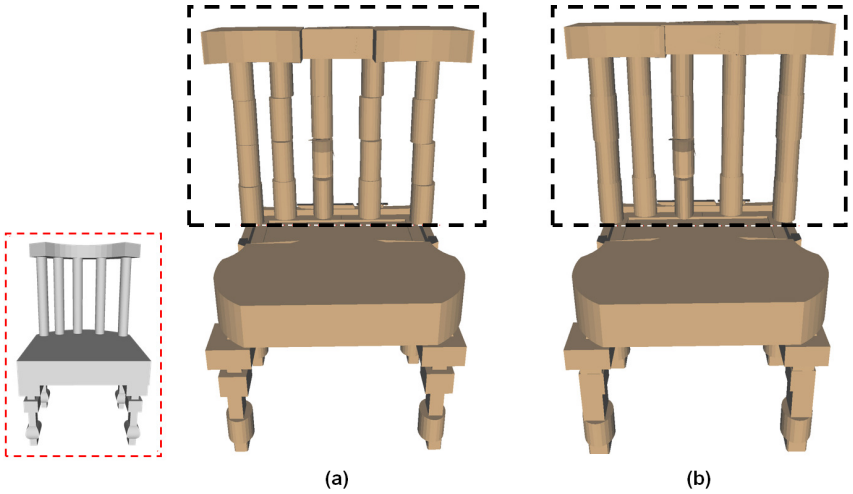


Figure 4.5: Example of a shape synthesized from the input template chair shown in the inset: (a) before the alignment process; and (b) after the alignment.

In summary, our system can align the adjacent segments of the novel 3D objects producing, in general, a final shape whose segments are connected in a correct manner as a result of several factors:

1. The segments that are selected during the sampling process for synthesizing our 3D shapes have a high similarity in their overall geometrical form with respect to the original segments of the template shape, as we can see, for instance, comparing the original template and the synthesized shape shown in Figure 4.5. Moreover, due to the pairwise term included as part of the energy function of Eq. (4.6), the segments selected for the synthesis process are also similar in their neighborhood or context with respect to the segments of the template. For example, the segments that are selected as replacements for the segments of the bars in the back of the template chair model shown in Figure 4.5 are

extracted, most commonly, from other similar contexts of other shapes of our input dataset, and the same occurs with the other parts of the synthesized model of Figure 4.5. In Chapter 6 we will show additional examples that indicate that our energy function allows us to find, in most cases, appropriate segments for the synthesis process according to the overall geometrical form of the segments and also, according to the respective neighborhoods or contexts of the parts. By selecting segments that are similar not only in their overall form but also in their context in comparison to the segments of the template, the processes of placement of each segment and alignment of the adjacent segments of our novel shapes can generate shapes that maintain the general form of the template and where the segments are also well integrated.

2. The process of refinement of the boundaries of the mesh segments that we described previously, help us to connect the boundaries of the adjacent segments in a better way. Moreover, by closing sides of the mesh segments that are originally open, as we show in the example of Figure 4.4, we can avoid having holes in our generated meshes.

3. As mentioned before, by traversing the graph of the template shape using a breadth-first approach, and starting the traversal with one of the segments that has the largest number of neighbors, we can compute, in most cases, the alignment process between pairs of adjacent segments that must be aligned. In this way, we can generate shapes with adjacent segments that are correctly connected and aligned.

4. Finally, the process described in Section 4.5.1 allows us to find pairs of matching contact points that can be used to obtain an optimal scaling and translation that connects and aligns the neighbor segments in a correct manner in most cases, as we can observe, for example, for the segments of the back of the synthesized shape of Figure 4.5.

Nonetheless, there are some cases where the neighboring segments of our synthesized shapes cannot be aligned in an appropriate manner, as we will discuss in more detail when we analyze the issues and limitations of our method in Chapter 6. However, we will also describe in Chapter 7 alternatives to avoid some of the limitations of the process of alignment between adjacent segments of our generated shapes.

4.6 Synthesizing collections of 3D shapes

As mentioned earlier, our method can be employed to synthesize several distinct 3D man-made objects from a single template, by sampling different replacements from our pool of segments Ω. Moreover, by selecting different templates, we can create a comprehensive collection of 3D shapes of a given class.

In the following chapter, we discuss the techniques that we employ to extract the fine-grained segments from an input family of 3D objects, while in Chapter 6 we will present the results of using our method to synthesize shapes of different classes.

Chapter 5

Extraction of fine-grained segments

As discussed in Chapter 2, several works have studied the problem of computing segmentations for triangular meshes [14,15]. In addition, recent techniques have been proposed to obtain segmentations of point cloud representations of shapes, even for the case of incomplete or noisy point clouds [1,44]. However, computing a meaningful segmentation of a 3D shape is still a challenging task [15,16], which can be even more complex when a consistent segmentation of different 3D objects that belong to the same class is required. Moreover, many of the works on 3D shape segmentation that have been presented in the literature are tailored to the specific purpose of computing coarse semantic segments, while our method requires fine-grained segments instead.

Hence, for the extraction of the fine-grained segments required in our technique, we employ a semi-automatic approach that incorporates user assistance. In the following section, we introduce an interactive tool that we developed for the segmentation of point clouds, which allows us to obtain the fine-grained segments that we require for our synthesis algorithm. Nonetheless, in Section 5.2, we discuss our advances towards the implementation of a fully-automatic process for the extraction of the fine-grained segments that we use in our approach.

5.1 Interactive segmentation of point clouds

We implemented a program for segmentation of point clouds based on the work of Liu et al., which focuses on the visualization and annotation of mesh components for part-based modeling [6]. Our program provides a set of high-level tools that the user can employ to easily extract fine-grained segments from any given point cloud representing a 3D shape, such as the one shown in the example of Figure 5.1.

Furthermore, our system can take as input a point cloud that has been pre-segmented into coarse parts, whether by an interactive external tool or an automatic method, and allows the user to segment these large parts into smaller segments of regular size.

Next, we describe the main components of our segmentation program, which are organized on the interface into five areas that we identify with numbers in Figure 5.1.

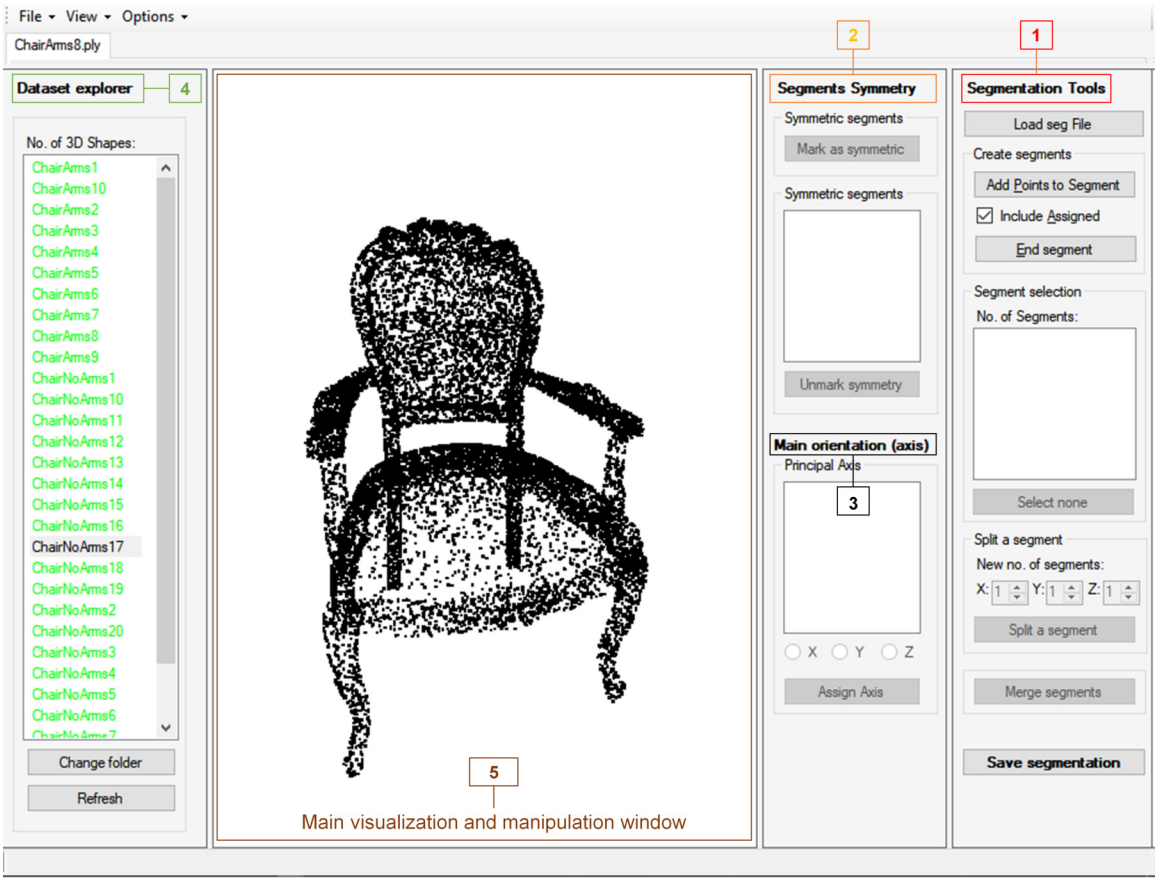


Figure 5.1: Interface of our interactive tool for point cloud segmentation, showing the main elements of the program.

1. **Segmentation tools**. The segmentation tools in our program, shown as part of the area or panel 1 in Figure 5.1, allow the user to create a new segmentation from scratch, or to modify an existent segmentation for a point cloud. The point cloud being processed by the tool appears in the main visualization and manipulation area of the interface, which is identified in Figure 5.1 with the number 5.

Selection of points to create new segments. The segmentation tool allows the user to select a set of points from the point cloud, using a simple selection box, which the user can draw directly on the screen. All the points enclosed by the box, which are all the points projected onto the area selected by the user according to the current view, are assigned to the segment. After selecting all the points that are going to be assigned to a given segment, the user indicates that the construction of the current segment has ended, and the system creates a new segment by assigning a numeric label to all the selected points. To facilitate the visualization and later selection of the segments, the system also assigns a random color to the new segment. All the points of the new segment are displayed with this color. In addition, the label of the new segment is added to a list in the middle of the panel of segmentation tools. On this list, the label of the segment is presented using the same color assigned to the corresponding segment's points. Figure 5.2 presents the interface of our program displaying the fine-grained segments obtained with our segmentation tool, for the same point cloud shown in Figure 5.1.

Modification of existing segments. Our program also allows the user to select and modify segments previously created. To select existing segments, the user can employ the list of segments on the panel of segmentation tools, or can select the segment directly on the main visualization window, by selecting any point in the point cloud that belongs to the given segment. After selecting a segment, the user can perform changes to the segment in several ways. First, the user can select a group of points that might have been incorrectly assigned to the segment, and can assign them to a different or new segment. Additionally, the user can also divide a segment that is currently selected into a smaller number of parts of a regular size.

Regular partitioning of segments. Our segmentation interface provides an option for merging two or more segments, as well as an option for splitting an existing segment into smaller segments of a given size.

For merging two or more segments, the user selects the segments and clicks on the button *Merge segments* that appears on the bottom of the panel 1 shown

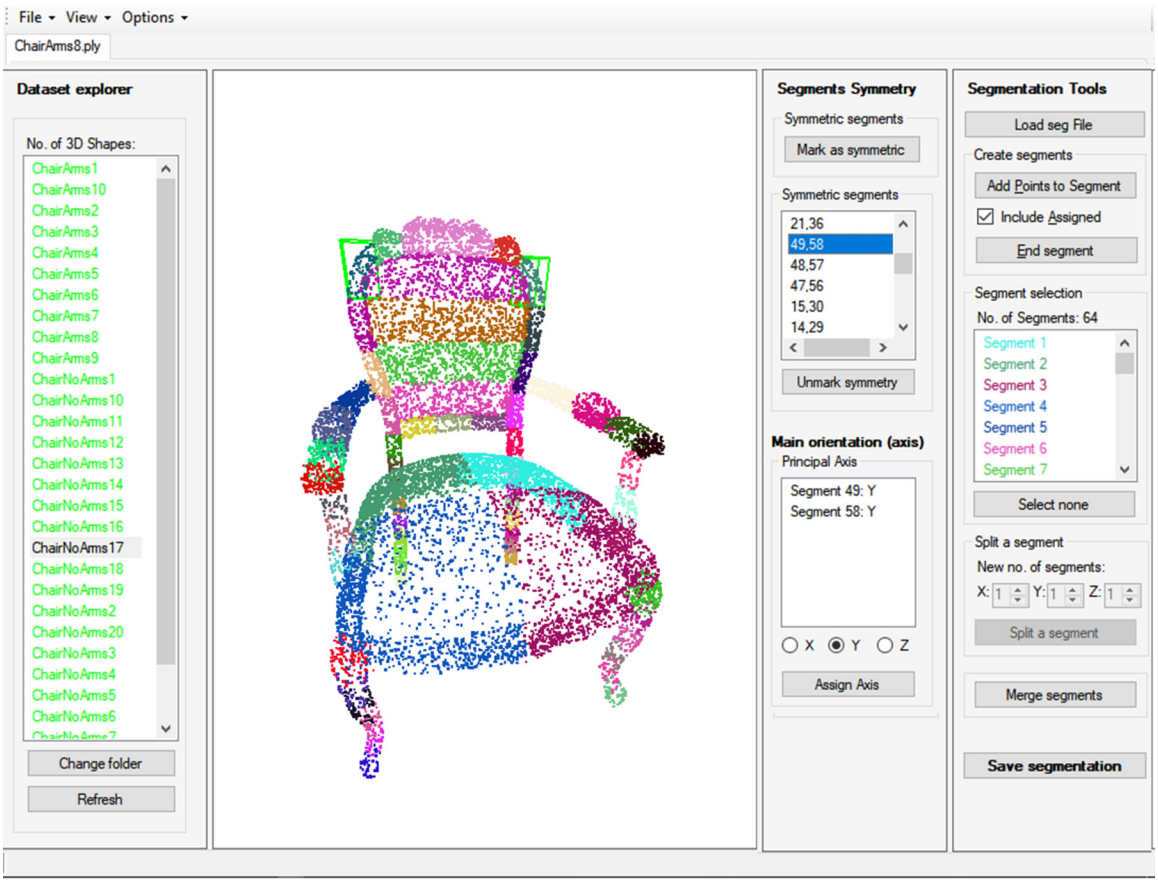


Figure 5.2: Example of a fine-grained segmentation obtained using our interactive program for point cloud segmentation.

in Figure 5.1. The system will find, among the merged segments, the segment with the smallest label number, and will assign the same number to all the segments that have been merged. Additionally, the program will update on the screen the points of the merged segments to show all of them with the same color, indicating that all these points now belong to the same segment. Moreover, the program automatically updates all the numeric labels assigned to each segment, to guarantee a consistent sequential numbering of all the segments. To conclude the merging operation, the system also updates all the annotations about symmetry and principal axis, according to the new labels. We explain below how to create these annotations for each segment using our system.

On the other hand, our program also allows to divide a segment into a determined number of smaller segments. Figure 5.3 illustrates an example of the sequence of steps of a splitting operation with our system. To perform a

splitting, the user selects the segment to be divided, and enters the number of desired new segments per coordinate or dimension in the respective text box on the interface. Finally, when the user clicks on the button *Split segments*, our segmentation program: (i) Calculates the size of each new smaller segment dividing the size of the original segment by the number of new segments indicated by the user per coordinate; (ii) Updates all the labels of the segments of the shape, and assigns a new random color to each new segment; and (iii) Eliminates the annotations that might have been associated with the segment that was divided.

2. **Annotation of symmetric segments**. Panel 2 displayed in Figure 5.1 contains controls that can be employed by the user to annotate the symmetry relations between pairs of segments. As we explained in Chapter 4, in our approach, we specifically mark pairs of segments having reflectional symmetry [71]. These annotations let us determine the symmetry relations of our input shapes, and generate novel shapes where these relations are preserved. For creating these annotations, the user selects a pair of segments and marks them as symmetric. Then, this pair of segments are added to a list of symmetric segments. The user can later select each pair of symmetric segments from this list, whether for visualization of these segments, or for adjusting the symmetry annotations. Note that such symmetric relations can also be detected automatically with approaches such as the one of Mitra et al. [71].

3. **Annotation of principal axis per segment**. Our segmentation interface also allows the user to create and manage annotations about the principal axis of orientation of each fine-grained segment. As we can observe in Figure 5.1, this functionality is included in the panel 3 in our interface. To create these annotations, the user can select one or several segments, choose the principal axis from the list of canonical axes, and assign this axis to the segment or segments that have been selected. These annotations are also displayed in a list that contains all the segments of the shape that are currently selected, as we show in the example of Figure 5.2. As we explained in detail in Section 4.5, we employ the annotations about the principal axis of each segment to compute a rotation between each segment of the template and the corresponding segment selected to generate a new 3D shape. Although identifying one particular axis

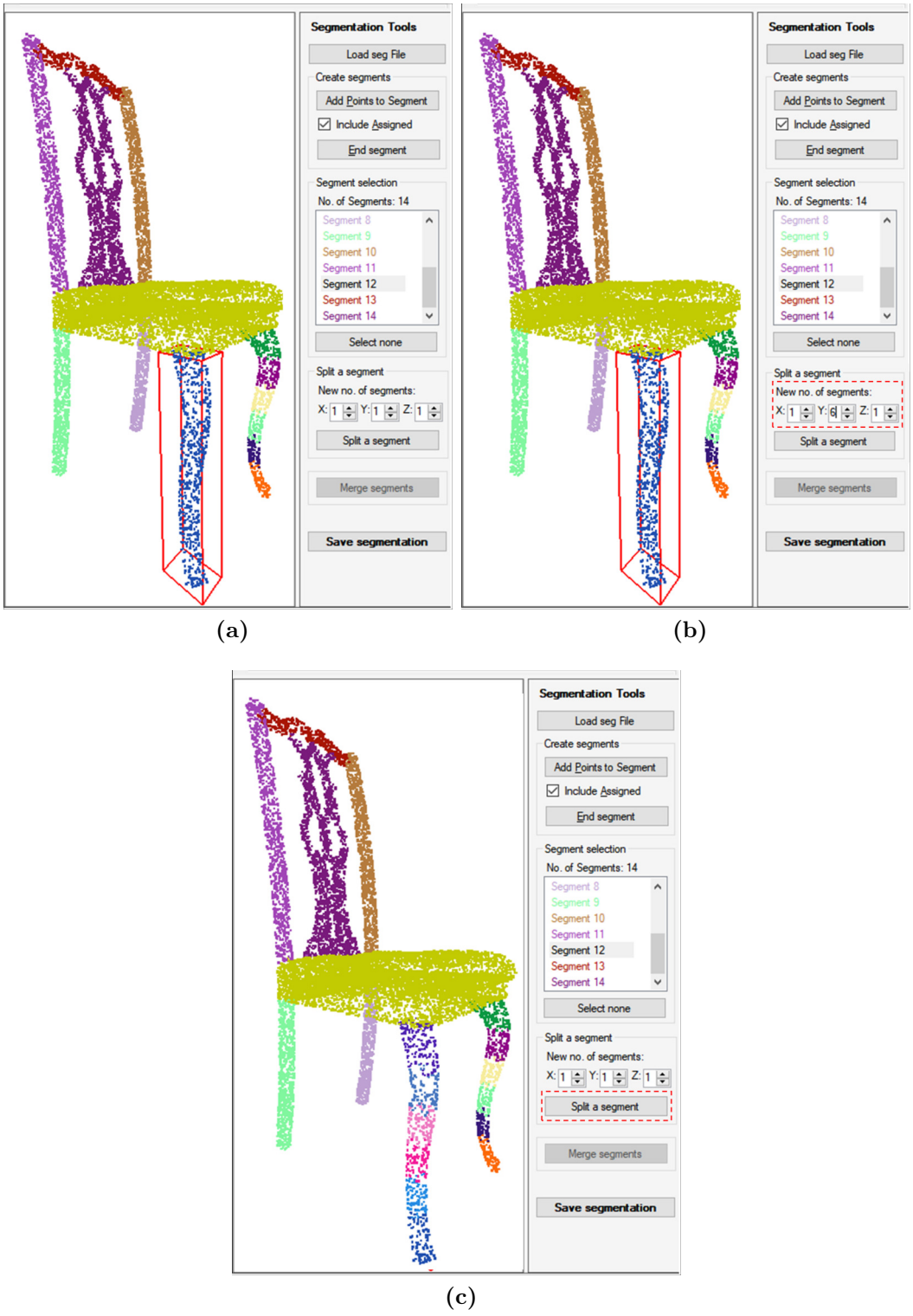


Figure 5.3: Illustration of a splitting operation in our program for point cloud segmentation. The user selects the segment that will be split (a), and enters the number of desired new segments per coordinate (b). Then, after clicking the button *Split a segment*, the program splits the segment that was selected and displays the new segments (c).

as the principal axis of orientation, and using the canonical axes as the only option for the orientation, might not apply for some parts, (like a spherical part, for instance), we observed that, for the experimental datasets that we employ in this thesis, marking a given axis as the principal axis of orientation for all the segments using our interactive program, allows our method to obtain consistent results when computing the rotation that matches the orientation of the segments selected for synthesis with the orientation of the corresponding segments of the template.

4. **Dataset explorer**. Our segmentation system also provides a dataset explorer, shown in the panel or area 4 of Figure 5.1. This explorer shows a list of all the file names of the point clouds contained in a folder that acts as the repository for our input 3D models. The user can employ this list for selecting each shape of our input dataset in a fast manner.

Processing of input dataset. We use the segmentation tool to semi-automatically segment all the shapes given as input to our method. For some shapes in our input datasets, which have more complex geometry and would require significant work for manual segmentation, we used the technique of van Kaick et al. [1] for computing an initial segmentation of the point clouds. Figure 5.4 presents an example of such a segmentation. Then, we loaded these pre-segmented point clouds in our segmentation interface, and we used the regular splitting tool, as well as the options for selection and modification of existing segments in our interactive program, for extracting the fine-grained segments required by our synthesis method. For shapes with simpler geometry, we observed that it is faster to manually create the correct segmentation, rather than to edit a pre-segmentation.

Table 5.1 summarizes the time required for segmenting the input shapes, and shows the proportion of shapes that were pre-segmented with the method of van Kaick et al. [1].

The time required to perform the fine-grained segmentations using our interactive software largely depends on the complexity of the geometrical form of the input shape. For instance, in the case of the fine-grained segmentation of the point cloud that we present in Figure 5.1, for which we employed a pre-segmented point cloud as the input for our interactive program, the total time required to complete the extraction

Figure 5.4: Example of a segmentation into coarse semantic parts for one of our input shapes, computed with the method of van Kaick et al. [1]. For some shapes in our input datasets, we used segmentations like the one shown in this example as input to be further refined with our interactive program.

Class	Sub-class	No. of shapes	No. of pre-segmented shapes used	Avg. segmentation time per shape (h)
Tables	–	15	9	1.3
Chairs	Chairs with arms	10	5	3.2
	Chairs without arms	20	17	1.2
Total for chairs	–	30	22	1.8
Total for all	–	45	31	1.6

Table 5.1: Summary of the fine-grained segmentation process for each input class of shapes.

and annotations of the fine-grained segments was approximately 3 hours. In the case of a 3D shape with similar geometrical characteristics for which we were not able to start the fine-grained segmentation process from a pre-segmented point cloud, we only required approximately 0.5 additional hours to complete the segmentation and annotation for the 3D model. Table 5.1 shows, in the last column, the approximate average time, in hours, required to obtain the fine-grained segmentation for each

shape of our input sets. The average times required to segment the shapes of our input set of tables are much lower, since many table models have a simpler topology in comparison to the shapes of our family of chairs. In particular, we can see that the higher times were required to extract the fine-grained segments for the shapes in the sub-class *chairs with arms*, which have a more complex geometry and more local fine details, in comparison to the shapes in the sub-class *chairs without arms*.

Using our segmentation program, we extracted a total of 415 fine-grained segments for the 15 models that are part of our set of tables. For the dataset of chairs, we acquired a total of 1,627 fine-grained segments: 656 for the sub-class *chairs with arms*, and 971 for the sub-class *chairs without arms*.

5.2 Automatic fine-grained segmentation

We have also explored the development of an approach for computing the fine-grained segments that we require for our synthesis algorithm in a fully-automatic manner. Our prototype algorithm for automatic fine-grained segmentation uses information of both the mesh and the point cloud representations of each input shape.

Our automatic segmentation algorithm works in two stages. The first part of the process is based on a *region growing* approach that has been employed in previous works for segmentation of 3D shapes [14, 15]. We start with a region composed of a single seed face, and greedily add faces to the region according to the adjacency graph of the mesh. As we mentioned in Chapter 2, different criteria can be used to determine whether or not to add an adjacent face to an existing segment. Our segmentation method uses a criterion based on the dihedral angle between neighboring faces for joining faces to a region.

While traversing the graph, we create a preliminary set of mesh segments by keeping together the faces of the mesh where the difference in the dihedral angle of the faces is below a predefined threshold γ. We measure the dihedral angle between the faces by simply computing the angle between their normal vectors. In a similar manner to previous works on mesh segmentation [58], we employ the dihedral angle as a measure of concavity that allows to find several of the regions of the shape with salient geometrical features, like the ones that we seek to capture in our fine-grained segments. However, as shown in the example of Figure 4.1b, some of our input meshes have regions that have no salient boundaries, or where there is a very small

or no difference in the dihedral angle between the faces of the mesh. In such cases, our criterion based on the dihedral angle will maintain these parts of the mesh in the same segment. Thus, our preliminary set of mesh segments can contain several coarse segments, which we partition in the next stage of the process.

In the second step of our algorithm, to extract fine-grained segments for coarse parts coming from regions of the shapes without salient features, we divide these large parts into a smaller number of segments of approximately equal size. Since some of the triangles in the mesh can have triangles with large edges, we first translate the segments extracted from the mesh into corresponding segments on the point cloud, and then perform the regular segmentation on the point cloud.

Nonetheless, there are several issues that must be addressed to obtain consistent fine-grained segments using our automatic method. The first key challenge consists in finding a value for the threshold γ that measures the difference in the dihedral angle between the faces. Our goal is to find a value that lets us extract the most important fine details from all the shapes in our input dataset, while also keeping together in a single segment, parts of the shape where we can probably find relatively large differences in the dihedral angle between the faces, but which should not be assigned to different segments. Figure 5.5 shows an example of the fine-grained segmentations for three chairs, which we obtained in an early experiment with our automatic algorithm, using a threshold of $\gamma = \frac{3}{4}\pi$. As we can see, using this threshold, our method cannot capture all the fine geometrical details on the legs of the chairs in Figures 5.5(a) and (b). Other experiments performed using different values for γ, allowed us to extract some of the fine details that we want to capture in our input 3D shapes only in some cases, while, for other shapes in our input dataset, we found some important inconsistencies in the segments, such as triangles on two sides of a single bar on the back of the model of a chair that were assigned to different segments as we can see in Figure 5.5(c).

In addition, during the second step of our algorithm we use the same size threshold to divide all the coarse segments that might be extracted during the first part of the method, which produces in some cases small segments that are not meaningful, such as the example shown in Figure 5.5(a), where some small segments of the bars on the back of the chair are not consistent with respect to the segments obtained for similar parts in other models of the same family. Moreover, we would like to segment the large or coarse parts using an adaptive approach that considers the number of

(a) (b) (c)

Figure 5.5: Examples of fine-grained segmentations computed with our automatic algorithm.

parts that we most commonly find for similar parts in all the shapes of the same family, as well as different geometrical properties of the coarse segments, like their volume and their curvature [14], for example. Further, using an adaptive approach for the segmentation of the large parts in our 3D shapes, we could also reduce the number of fine-grained segments for some of these parts. Specifically, using our current automatic segmentation implementation, we obtained a large number of fine-grained segments for the seats and the tops of the backs in some of the shapes of our chairs family, as we can observe in Figure 5.5, while, using our interactive segmentation program, we can consistently extract a smaller number of fine-grained segments for parts such as the seats of the chairs, as we can see in the example in Figure 5.2.

In summary, our current implementation for the automatic extraction of the fine-grained segments could benefit from the addition of measures like the volume of the parts, which we could obtain using the SDF [31], or measures about the convexity of the parts, like the ones employed by Kreavoy et al. on their system for 3D model compositions from interchangeable components [21]. On the other hand, another alternative to be explored for the acquisition of our fine-grained segments is to adapt segmentation techniques that can take as input a point cloud representation [1, 44]. Employing methods that can use a point cloud, we could avoid problems in our automatic implementation arising from the selection of a given value for the threshold γ used to compare the dihedral angles between the faces of the input meshes.

Chapter 6

Experimental results

In this chapter, we present the results of diverse experiments performed with our synthesis approach. We applied our algorithm on three different sets of man-made shapes: a family of chairs, a family of tables, and a hybrid set that combines all the table models with a subset of shapes selected from our chairs dataset. First, we present examples of shapes synthesized with our approach using different templates chosen from our input datasets of tables and chairs. Next, we show the result of experiments synthesizing various models from a single template. Additionally, we present results of shapes synthesized with our hybrid dataset, demonstrating that our approach can be employed to create a shape of a given class employing segments extracted from shapes of diverse categories. Finally, we show that we can synthesize models while preserving selected segments of the template.

Input datasets. We collected the input models to test our approach from three different sources: the dataset of furniture models of diverse styles of Hu et al. [45], and two public online repositories, *ShapeNet* [12] and *3D Warehouse* [11]. Our input family of chairs contains 30 shapes, including chairs with and without arms. For our input set of tables, we have selected 15 models of diverse characteristics, including round and rectangular tables. In addition, we created a hybrid set that includes a total of 30 shapes, comprising all the table models, and a subset of 15 chair models. We present all the shapes of our input datasets in Appendix A.

6.1 Shapes synthesized from different templates

Figure 6.1 presents a set of shapes created with our method from the input dataset of tables. In a similar manner, Figure 6.2 shows several shapes synthesized using our set of chairs. Each shape presented in Figures 6.1 and 6.2 was created using a different input template, which guides the synthesis process providing the topology and general structure for the synthesized shape. However, note that all the synthesized shapes that we present in this, and the following two sections, do not contain any of the original fine-grained segments of the input template.

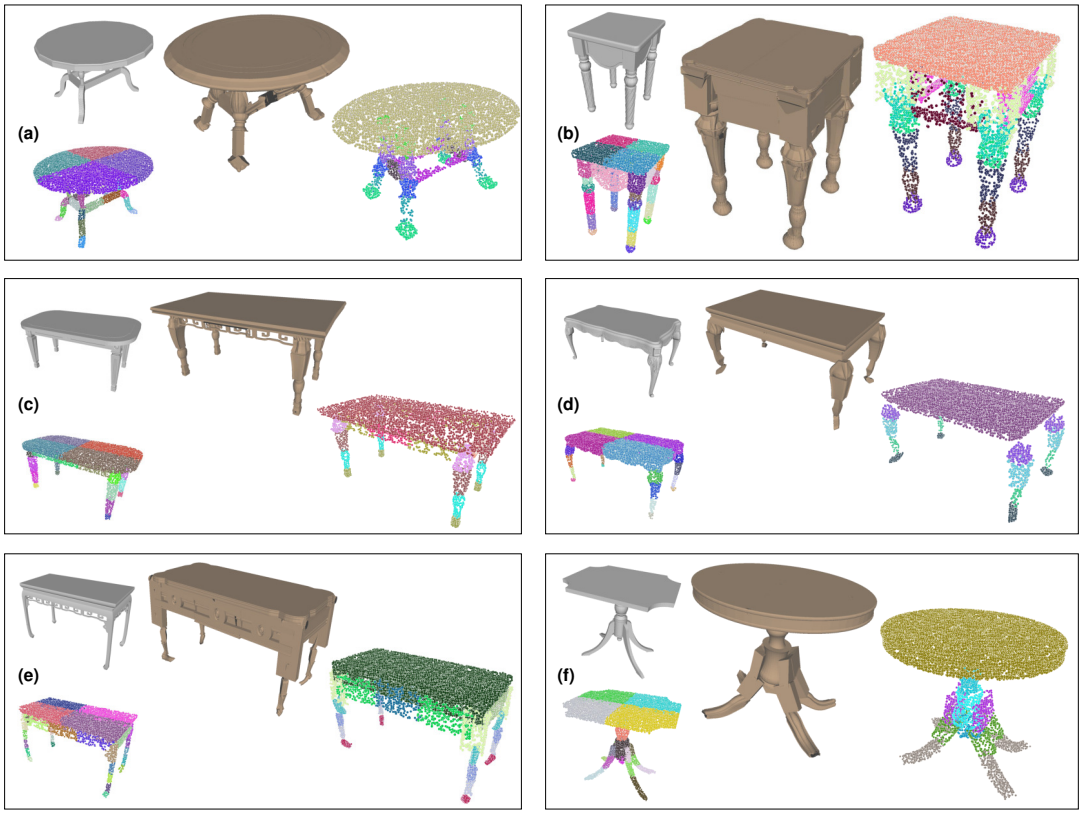


Figure 6.1: Shapes synthesized using different templates from our input set of tables. For each group of results, we show the template shape (top-left, in gray), the fine-grained segmentation of the template (bottom-left), the synthesized mesh (center), and the synthesized point cloud (right).

As we mentioned in Chapter 4, following the symmetry annotations that were made for each shape during the segmentation, our method employs the same segment

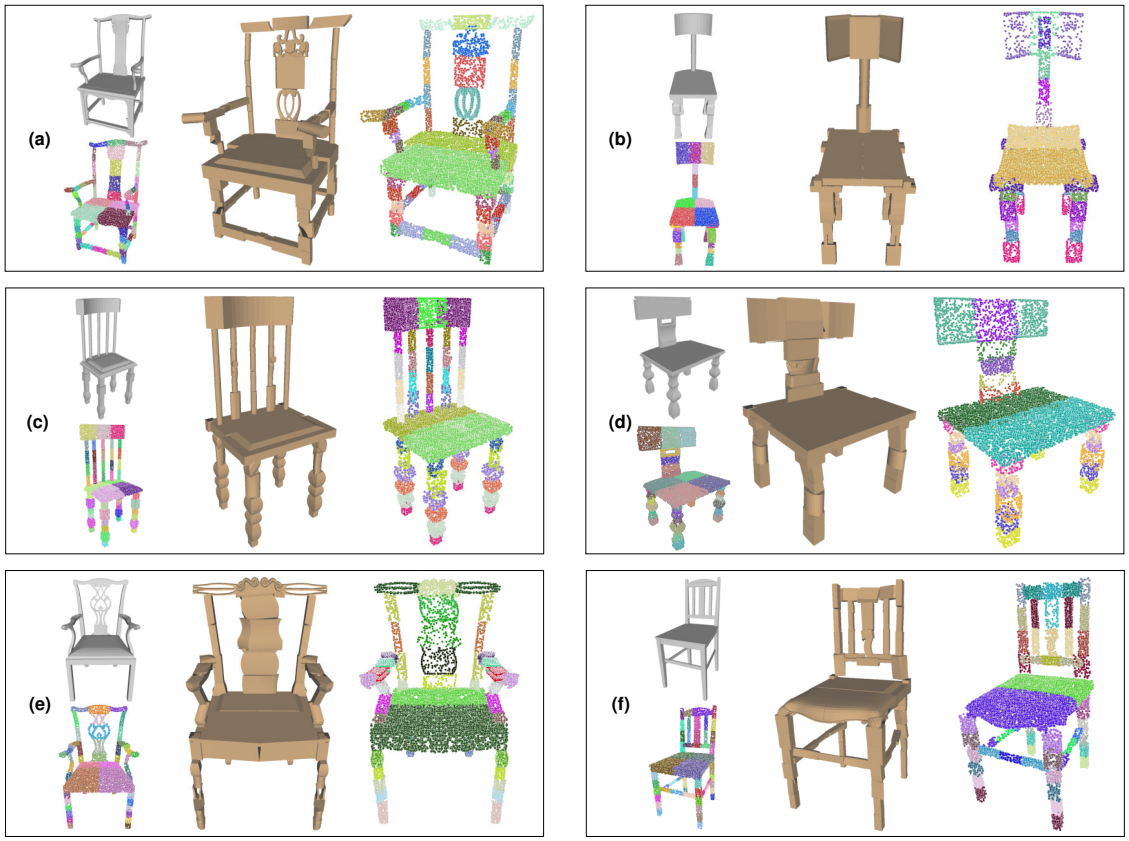


Figure 6.2: Shapes synthesized using different templates from our input set of chairs. For each group of results, we show the template shape (top-left, in gray), the fine-grained segmentation of the template (bottom-left), the synthesized mesh (center), and the synthesized point cloud (right).

to replace each part that was identified as symmetric in the corresponding input template. Furthermore, all the synthesized point clouds that we present throughout this chapter show the same color for all the symmetric parts, which indicates that these symmetric parts correspond to the same fine-grained segment, although they have a different orientation according to the form of the template. As we can see in the shapes shown in Figures 6.1 and 6.2, using the same segment for all the symmetric parts allows our method to better preserve the structure of the template in the synthesized models.

Moreover, the sampling process used to synthesize a shape finds segments that have a high similarity, in their general form, with respect to the original segments of the template. In this manner, our algorithm can generate shapes that keep a plausible

structure while using templates that have diverse forms and part arrangements. For example, in the results of Figure 6.1(a), (b), and (f) for the tables and, similarly, in Figure 6.2(a), (b), and (c) for the set of chairs, we can see that the synthesized shapes maintain the general form of the corresponding input templates, which exhibit different structures and topologies. We analyze in more detail the process of sampling the segments for synthesizing man-made shapes with our approach in Section 6.5.

On the other hand, our method can generate shapes with many different local variations in comparison to the original templates, as we can see in the examples of Figures 6.1 and 6.2. For instance, the synthesized shape shown in Figure 6.1(e) displays ornaments on its side, next to the top of the table, that are different from the original decorations of the input template. By using fine-grained segments, our technique can generate models containing diverse fine details, as we can see, for example, in the synthesized table in (c), which precisely employs the decorations extracted from the sides of the input model shown in (e). Additionally, the legs of the synthesized tables in the results of Figure 6.1(b), (c), and (d) show significant differences in their local details with respect to the same parts in the original templates.

Likewise, the synthesized chair models in Figure 6.2(b), (c), and (d), show a wide range of variations in local details of the legs with respect to the same parts of the corresponding input templates. In addition, when comparing the input template and the synthesized shape in Figure 6.2(a) and (e), we can also see many local variations in the back of each synthesized chair.

However, in some cases the shapes generated with our method can also contain parts that do not exhibit significant variations in their geometrical details in comparison to the original parts of the template, as we can see in particular for the back of the chairs in the results of Figure 6.2(b), (c), and (f). Yet, the synthesized chairs in (b), (c), and (f) exhibit, in general, a cohesive structure, and variations in some other parts, such as the seat of the chairs in (c) and (f).

6.2 Shapes generated from the same template

Figure 6.3 presents three shapes synthesized from a single template selected from our set of chairs. We also show in Figure 6.3 the value of the shape energy computed using Eq. 4.5 for each synthesized shape, and for the input template. We can see in Figure 6.3 that our approach has generated three shapes that are distinct of each other

and the template, possessing many local variations. This can be seen especially in the backs of each synthesized chair. Additionally, we can see that all the shapes maintain the general form of the template, although the synthesized shape in Figure 6.3(a), which has the highest shape energy, presents a form that is more similar to the structure of the template in comparison to the shapes that we show in the results (b) and (c). As we explained in Section 4.4.3, the value of the shape energy given by Eq. 4.5 captures, firstly, the general similarity of the individual segments by means of a unary term and, secondly, the similarity between the contexts or neighborhoods of each segment of the synthesized shape in comparison to the corresponding original segments of the template, by means of a second, pairwise term. Considering our definition of Eq. 4.5, we conjecture that the value of the shape energy could be used as a measure for comparing different shapes generated from the same template, in terms of their overall similarity with respect to the template.

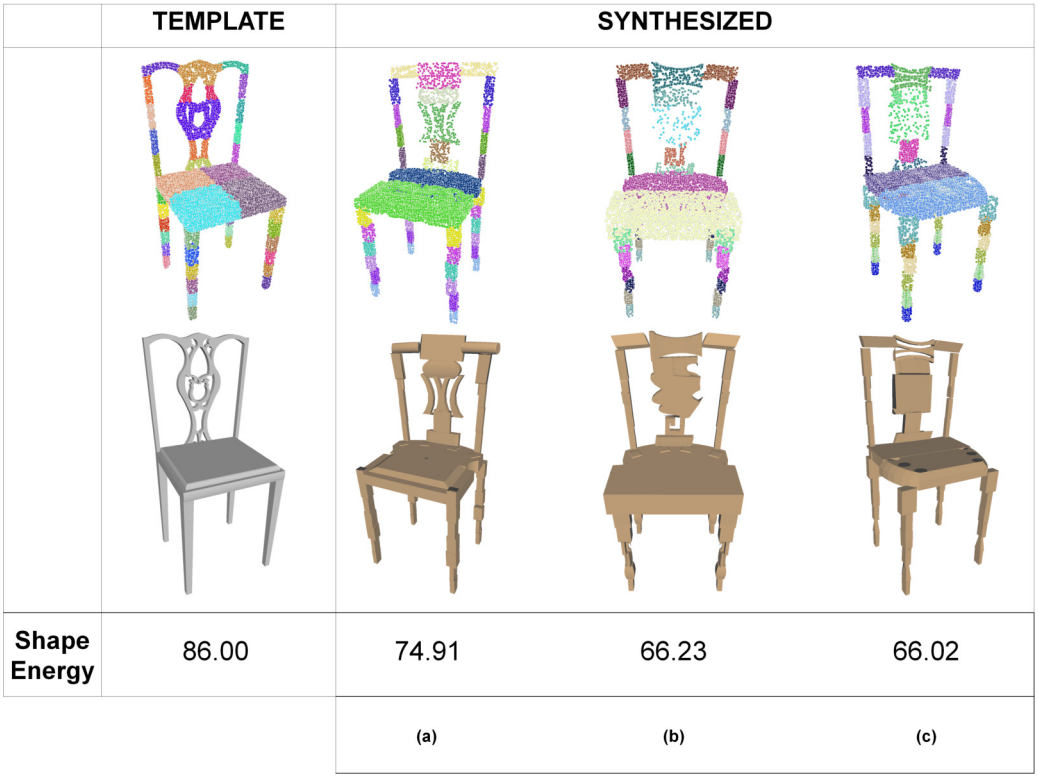


Figure 6.3: Models generated from a single template chair. Top row: segmented point cloud for the input template, and synthesized point clouds; middle row: input and synthesized meshes; bottom row: shape energy for each shape.

Furthermore, all the table models that we present in Figures 6.4 and 6.5 exhibit many fine detail variations while, at the same time, they all maintain a general form that is similar to the structure of the respective template. Correspondingly, the values of the shape energy for all the synthesized shapes are similar. In addition, we can observe that some of the segments of the legs of the synthesized table of Figure 6.4(a) present small differences in their overall shape with respect to the form of the segments of the legs of the template shape and, as expected, the value of the shape energy of the shape in Figure 6.4(a) is slightly lower in comparison to the energy value of the other two synthesized shapes presented in Figure 6.4. The results displayed in Figures 6.4 and 6.5 provide additional evidence supporting our hypothesis that the shape energy defined by Eq. 4.5 captures in an appropriate manner the similarity between the segments of the template and the fine-grained parts sampled for the creation of shapes.

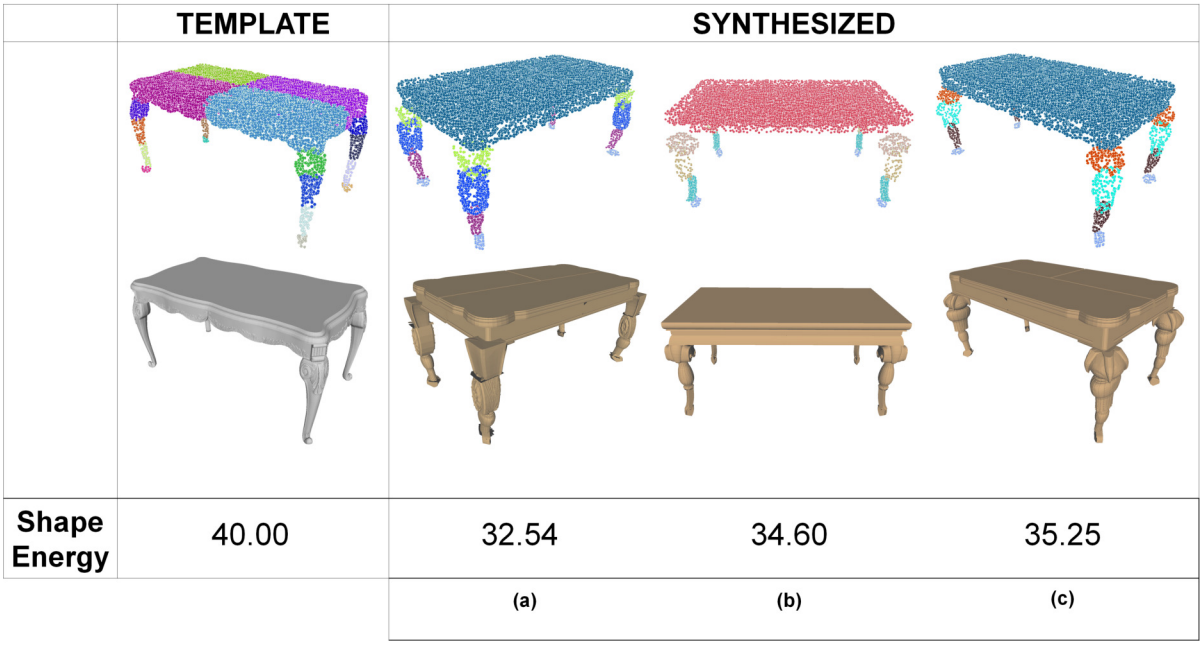


Figure 6.4: Shapes synthesized from a single template table. Top row: segmented point cloud for the input template, and synthesized point clouds; middle row: input and synthesized meshes; and bottom row: shape energy for each shape.

Nonetheless, to collect additional evidence to support our hypothesis about how the shape energy correctly captures the similarity between the overall form of the synthesized shapes and the form of the template, we would like to conduct, as future

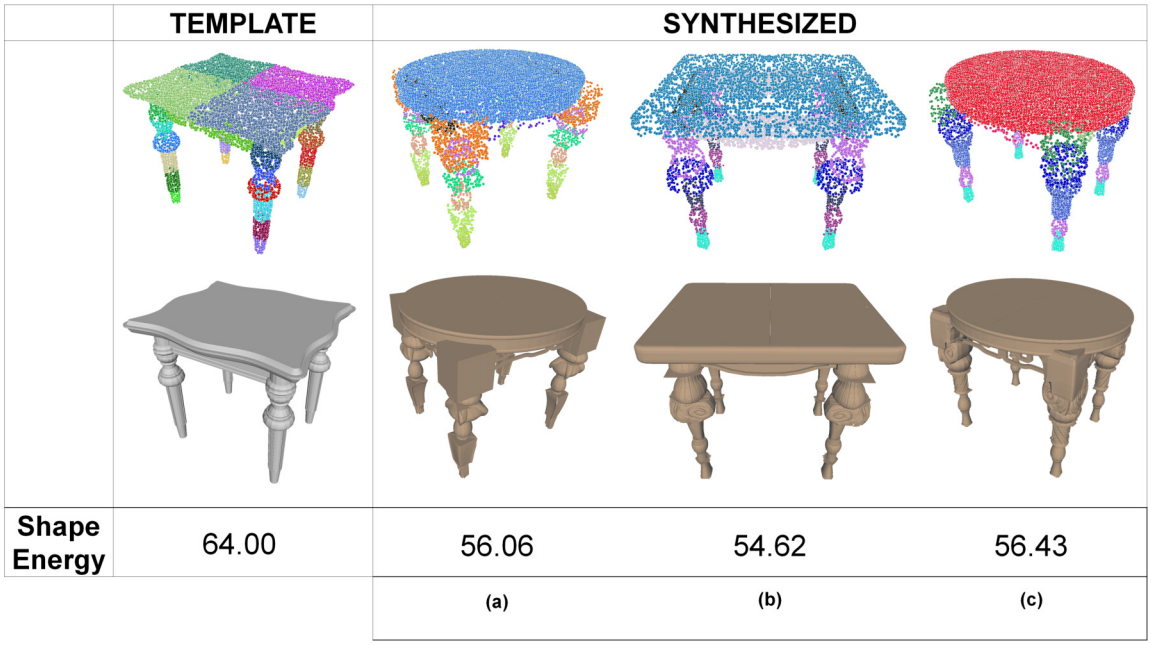


Figure 6.5: Another example of shapes synthesized from a single template table. Top row: segmented point cloud for the input template, and synthesized point clouds; middle row: input and synthesized meshes; and bottom row: shape energy for each shape.

work for our project, a user study where we could ask the users to select, among several shapes generated from a single template, the 3D model that resembles more closely the general form of the template. In this manner, we could compare the values of the shape energy with the users' answers. We will further discuss in this and the following chapter the idea of conducting a user study to evaluate our shape energy, as well as other aspects of our method.

On the other hand, although the shape energy for the shape in Figure 6.5(a) is slightly higher than the energy of the shape in Figure 6.5(b), we can observe that the segments on the legs of the table model of Figure 6.5(a) that are closer to the top of the table have been enlarged in an exaggerated manner. This issue can occur in cases where our alignment process fails to find appropriate contact points between neighbor segments that have a dissimilar geometrical shape, such as the round top and the box-like segments adjacent to the top in the synthesized table of Figure 6.5(a). We discuss this issue of the alignment process in more detail in Section 6.5.

Figure 6.6 shows another example of a set of chair models generated from the same input template. Similarly to the example of Figure 6.3, if we compare the

general structure of the three synthesized shapes displayed in Figure 6.6, we can observe that the structure of the shape of Figure 6.6(a) resembles more closely the general form of the input template. Correspondingly, the shape of Figure 6.6(a) has the highest shape energy. In Section 6.5, we will present a more detailed analysis of the similarities between individual segments of different synthesized shapes and the corresponding original segments of a template shape, and the relation of those similarities with the energy value computed with Eq. 4.6.

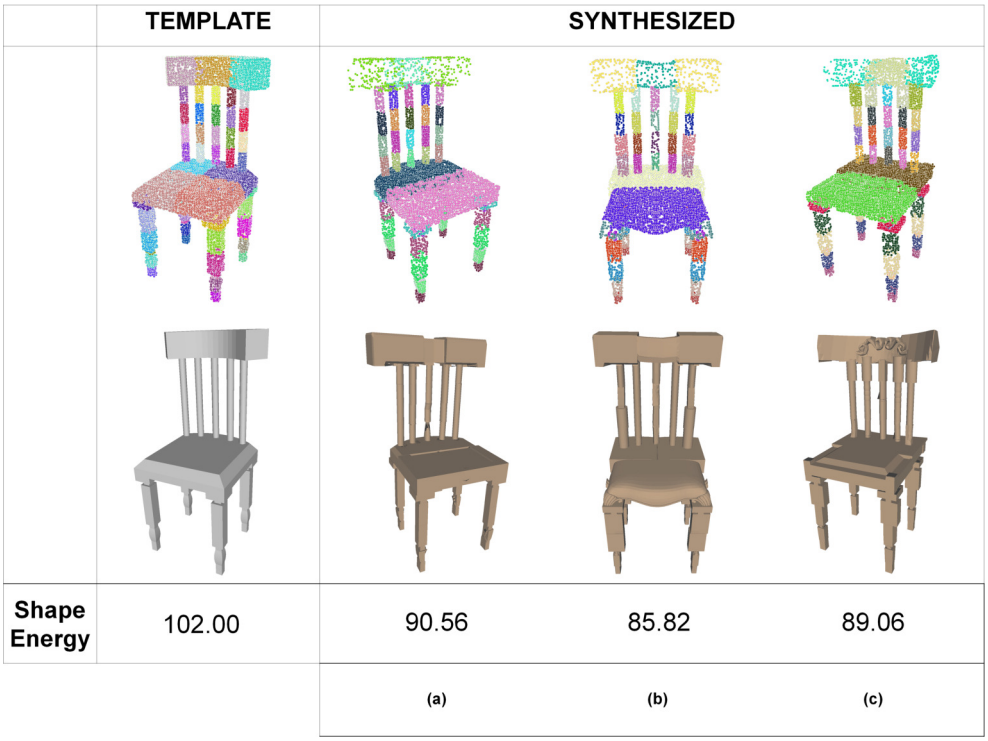


Figure 6.6: Shapes generated from a single template chair. Top row: segmented point cloud for the input template, and synthesized point clouds; middle row: input and synthesized meshes; bottom row: shape energy for each shape.

Figure 6.7 further suggests that our shape energy captures in a correct manner the differences between the overall geometry of the parts of the synthesized shapes and the parts of the template. Specifically, the synthesized shapes of Figure 6.7 show the result of an experiment where we sampled the segments for the synthesis process using a low probability threshold $\tau = 0.5$, unlike all the other experiments that we have discussed in this thesis, for which we have performed the sampling of segments with a threshold $\tau = 0.75$. Comparing the synthesized shape of Figure 6.7(a) with

the results of Figure 6.3, as well as the generated shape of Figure 6.7(b) with the results of Figure 6.5, and the synthesized shape of Figure 6.7(c) with the results of Figure 6.6, we can see that the novel shapes shown in Figure 6.7 have a significantly lower energy value, which reflects the dissimilarity of some of the segments of the synthesized shapes in comparison to the segments of template, which is especially noticeable for the segments of the seat of the two synthesized chairs, and the top of the synthesized table.

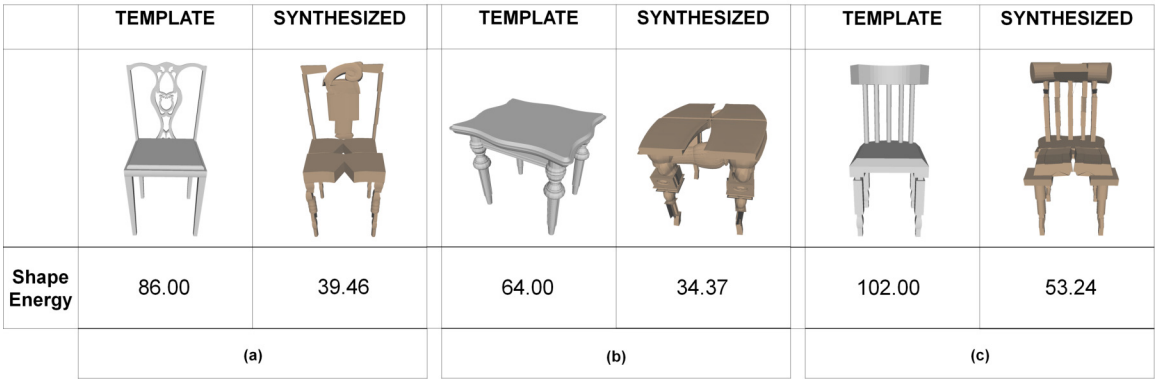


Figure 6.7: Synthesized 3D models and its corresponding shape energy for an experiment where we sampled the segments for the synthesis process using a low probability threshold of $\tau = 0.5$.

6.3 Shapes synthesized from a hybrid set

Since our approach is based on the geometrical properties of the segments used for synthesis and not on their semantic labeling, our method can be employed in a straightforward manner on a hybrid set containing shapes of different classes. In this way, we can introduce additional variations into our synthesized shapes. In Figure 6.8, we show various shapes created with our method using a hybrid dataset of tables and chairs. For each synthesized chair model shown in Figure 6.8, we highlight a part that was extracted from a table input model. Likewise, for each table in Figure 6.8, we point out a part that was extracted from one of the input chairs in the hybrid set.

For comparison, Figure 6.8(a) and (e) show examples of chair models generated from the same template that we used previously to synthesize the shapes in Figure 6.2(a) and (e). Our approach has combined the fine-grained segments of the

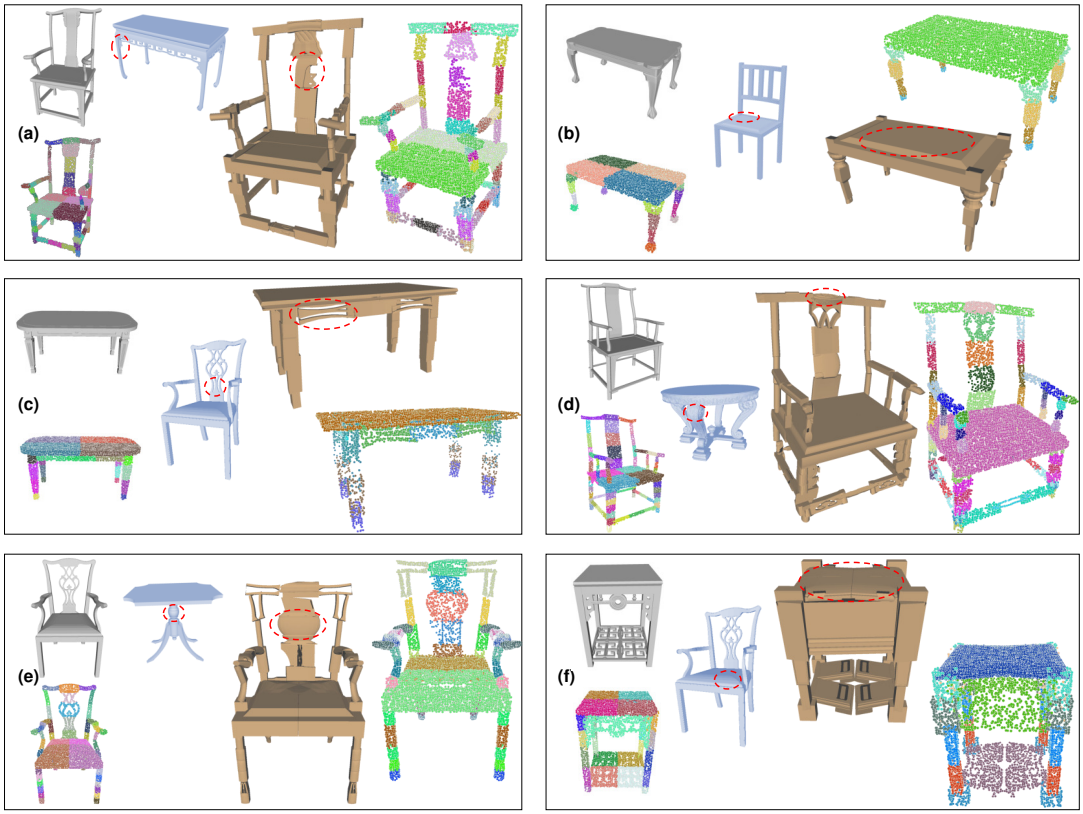


Figure 6.8: Shapes synthesized using different templates chosen from our hybrid dataset. For each group of results we include the original template shape and the fine-grained segmentation of the template. Additionally, we highlight a segment in the synthesized shape, which was extracted from a shape (blue) that belongs to a family different from the template.

two families of our hybrid set to generate chair models that are plausible variations of the original template models, and show also diverse local details that are different from the synthesized chairs previously generated only with the dataset of chairs. In particular, the results in Figure 6.8(a) and (e) contain in their backs several segments that have been extracted from shapes of the family of tables. Meanwhile, for the table model shown in Figure 6.8(c), only the segments of the top of the table were taken from another input table model, and all the other segments, including the fine details on the sides of the table, were sampled from segments extracted from shapes of the chair family. Thus, using our hybrid set, our method has generated a table model with several differences in comparison to the synthesized shape presented in Figure 6.1(c).

Figure 6.8(b), (d), and (f), present additional examples of shapes generated with our hybrid set using different templates. Specifically, the synthesized table models that we show in (b) and (f) employ several segments from shapes of the chairs family. For instance, we can observe that the seat of a chair has been used in a feasible manner to replace the top of the input table template in both (b) and (f). Additionally, we can see in the result of Figure 6.8(d) an example of a novel chair model created from a template with a complex structure, which also contains various details in the legs and the back.

In general, the examples in Figure 6.8 show that our algorithm can produce more variety in the synthesized objects combining segments of different families of shapes.

6.4 Shapes synthesized while preserving parts of the template

Using our approach, we can also select specific segments of the template shape that must be preserved in the synthesized shapes. To achieve this goal, we perform the process of sampling segments for the synthesis in a constrained manner, to find replacements only for the set of segments of the template that are not going to be maintained in the novel object. Then, we combine the sampled segments with the original segments of the template, placing and aligning all the parts to produce the shape. Since, in general, we do not employ segments from the original template to generate novel shapes, we can use this alternative to add more diversity in our results.

Figure 6.9 shows several shapes that we have synthesized using our three input datasets, and maintaining different parts of a given template shape. For each synthesized shape, we highlight the segments of the template that were preserved in the input template. We generated the results shown in Figure 6.9(a) and (b) using our dataset of tables, while (c) and (d) present shapes synthesized from our input family of chairs, and (e) and (f) show two results created using our hybrid dataset.

To generate the result of Figure 6.9(a), we selected the segments of the legs of the template as the parts to be preserved in the synthesized shape. As we can observe, our method can place and align in an appropriate manner the segments of the original shape together with the segments that were sampled from other tables, producing a cohesive and plausible shape. Additionally, we can also preserve in the

Figure 6.9: Shapes synthesized using different input datasets, and preserving specific parts of the template shape. For each case, we show in red circles the parts of the template that were maintained in the synthesized shape.

synthesized shape smaller portions of the template. For instance, to generate the table in Figure 6.9(b), we chose to maintain only the two fine-grained segments from the lowest part of each leg of the template. Similarly, for the synthesized shape in Figure 6.9(c), we marked the segments of the seat of the template as the parts to be maintained. In this case, our method has generated a shape that combines the seat of the input chair model with fine-grained segments that contain rich details, as we can see especially in the top part of the back of the synthesized chair.

Moreover, since our segments capture salient features and fine details of the input shapes, we can preserve in the generated objects some of the segments that capture such details, even if the segments to be preserved belong to parts that are not adjacent in the template. For instance, to generate the chair in Figure 6.9(d), we marked a segment on the top and the segments of the seat of the template as the parts to be maintained, which allowed us to generate a chair model with a plausible structure.

In general, we have seen in the examples shown in this and the previous sections of this chapter that our approach for shape synthesis can generate 3D shapes that have a wide range of fine geometrical details, and preserve the topology and general form of the input template.

6.5 Analysis of the method

By means of our shape energy function, which incorporates the similarities of each individual segment and the similarities between the segments that are adjacent in the 3D shapes, our method can find suitable parts to replace the segments of the input template for synthesizing a shape. We illustrate in Figure 6.10 examples of some of the original fine-grained segments that have been used by our method to generate the shape that we show in Figure 6.2(b). As shown in Figure 6.10, the fine-grained parts that are selected for the synthesized shapes, in general, have a high similarity with respect to the corresponding original segments of the template, not only in their overall geometrical properties, but also in their context with respect to the original shapes. For example, as we can see in Figure 6.10, the segments that belong to the legs of other shapes are the segments most commonly selected by our approach for the legs of the generated chair model, and the same occurs with the other parts of the shapes. By selecting segments with a high similarity in comparison to the segments of the input template, we can preserve in a better way the general form of the template in our synthesized shapes.

Furthermore, the example of Figure 6.11 also indicates that the segments selected by our method for generating new shapes are, generally, segments with a high similarity in their overall form and in their context in comparison to the segments of the template. As we can observe in Figure 6.11, the segments selected as replacement of the seat of the template chair, for instance, are extracted, in all cases, from the seat of other chairs of our input dataset.

In addition, we illustrate in Figure 6.11 an example of the energy values computed by means of Eq. 4.6 for some of the fine-grained segments of the shapes shown in Figure 6.6. In particular, the segments that we denote as $A1$ and $B1$ in Figure 6.11 have a high similarity with respect to the corresponding segments $T1$ and $T2$ of the template shape. Since the energy value of Eq. 4.6 takes into account the similarities between the adjacent segments of the synthesized shape in comparison to the neighbor

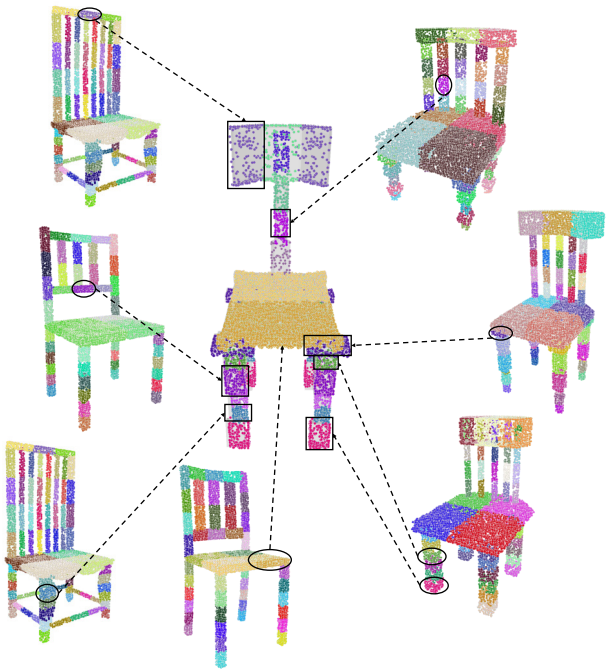
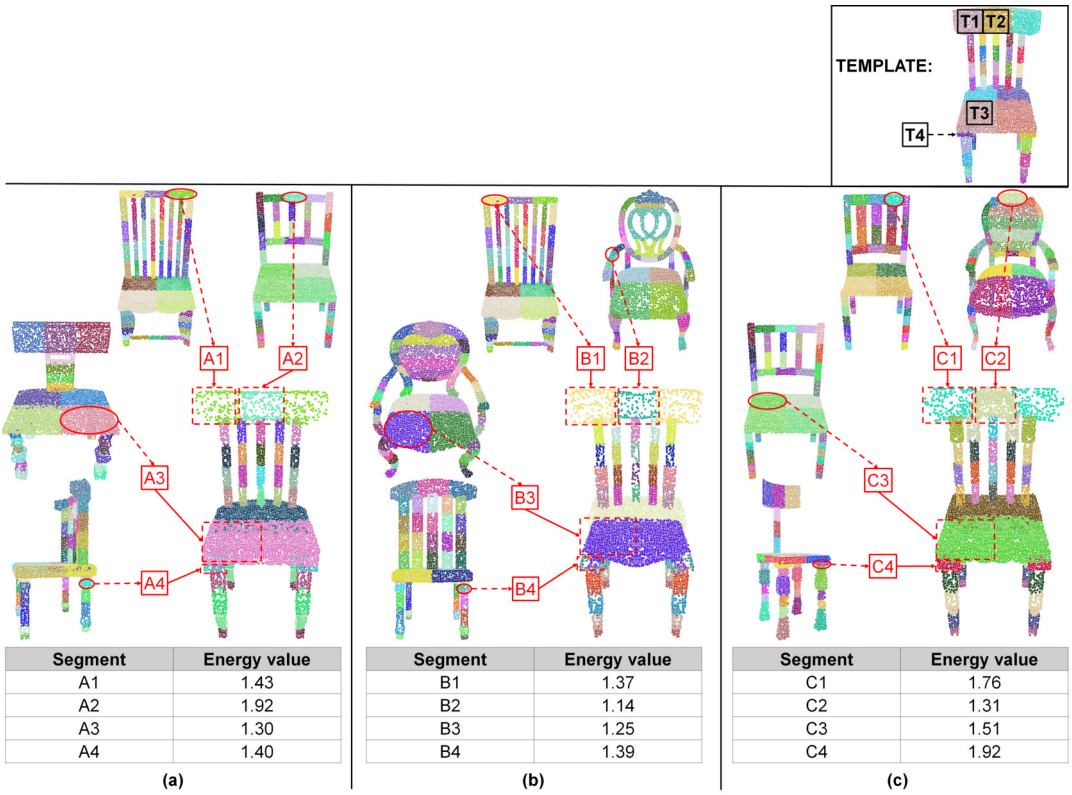


Figure 6.10: Example of some of the fine-grained segments selected by our method for synthesizing the chair shown in the center. The mesh synthesized from this point cloud is shown in Figure 6.2(b).

segments in the original template, the final energy value for segment $A1$ is higher than that of segment $B1$ because the similarity between the pair of segments $T2$ and $A2$ is higher than the similarity between $T2$ and $B2$. Likewise, the high energy value for the segment $C1$, sampled for the shape of Figure 6.11 (c), indicates the high similarity between the segment $C1$ and the segment $T1$ of the template, while the lower energy value obtained for the segment $C2$ denotes the differences in geometry and context between segment $C2$ and the corresponding segment $T2$.

Moreover, analyzing the energy values for segments $A3$, $B3$, and $C3$ in Figure 6.11, we can observe that the segment with the lowest energy value, i.e. segment $B3$, has also a general geometry that is more different than the geometry of the corresponding segment $T3$ of the template, in comparison to the geometry of segments $A3$ and $C3$. In addition, the segments $A4$ and $B4$ have a similar energy value, and these two segments were extracted from parts with an overall geometry that is similar to the one of segment $T4$ of the template. Further, segments $A4$ and $B4$ are symmetric parts in



Segment	Energy value
A1	1.43
A2	1.92
A3	1.30
A4	1.40

(a)

Segment	Energy value
B1	1.37
B2	1.14
B3	1.25
B4	1.39

(b)

Segment	Energy value
C1	1.76
C2	1.31
C3	1.51
C4	1.92

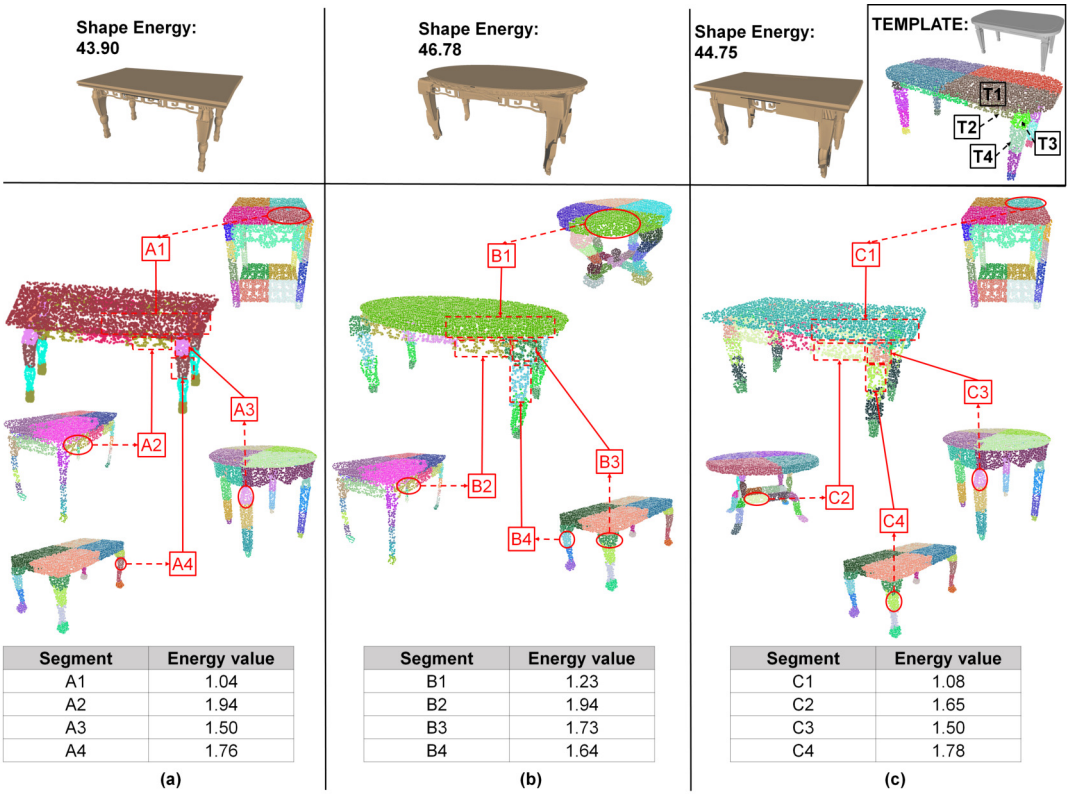
(c)

Figure 6.11: Example of some of the fine-grained segments sampled by our method for synthesizing the chairs of Figure 6.6. For each segment, we show the energy values computed by Eq. 4.6

one of the input shapes, which implies that their respective context or neighborhood is the same, which is captured by the energy function yielding a very similar value for segments $A4$ and $B4$. Meanwhile, segment $C4$ also has a form that, in general, is similar to the form of segment $T4$, but the segment $C4$ was extracted from a context that is much more similar with respect to the context of the original segment $T4$ of the template: both segments, $C4$ and $T4$, are parts of the legs in their corresponding input chair model that are the closest to the seat of the chair. The similarity in the context of segments $C4$ and $T4$ also is represented appropriately by the higher energy value for segment $C4$.

Similarly, the example in Figure 6.12 presents three synthesized table models generated from a single template table, together with the energy values of some of the segments sampled by our method as replacements for the corresponding segments of the input template. The example of Figure 6.12 also suggests that the energy function given by Eq. 4.6 enables us to find segments that are, in overall, very similar

in comparison to the segments of the template. Specifically, we can see that the segments $A1$, $B1$, and $C1$, corresponding to a segment from the top of the original template, were extracted, in all cases, from other tops of different input tables. Yet, unlike segments $A1$ and $C1$, segment $B1$ has an overall rounded form, which is more similar to the form of the original segment $T1$, and this is correctly captured by our energy function: the energy for the segment $B1$ is higher in comparison to the energy for $A1$ and $C1$. Additionally, we can see that the energy for segment $C2$ is lower in comparison to the energy of segments $A2$ and $B2$, denoting a highest dissimilarity between the context or neighborhood of segments $C2$ and segment $T2$, in comparison to the context of the parts $A2$ and $B2$ with respect to the context of $T2$.



Segment	Energy value
A1	1.04
A2	1.94
A3	1.50
A4	1.76

(a)

Segment	Energy value
B1	1.23
B2	1.94
B3	1.73
B4	1.64

(b)

Segment	Energy value
C1	1.08
C2	1.65
C3	1.50
C4	1.78

(c)

Figure 6.12: Example of some fine-grained segments sampled by our method for synthesizing three table models from a single template (shown in the inset on the top-right). For each segment, we show the energy values computed by Eq. 4.6. The value of the shape energy for the template table in this Figure is 56.00.

Since the value of the shape energy given by Eq. 4.5 is defined as the sum of the energies of each individual segment, the examples of Figures 6.11 and 6.12, as well

as the results that we present in Section 6.2, support our hypothesis that the shape energy captures in an appropriate manner the overall similarity between the segments of the template and the segments of the synthesized shapes.

On the other hand, comparing the three synthesized shapes shown in Figure 6.12, we can see that the three tables preserve the general structure of the template but, in this case, we could consider that the shape of Figure 6.12(a) exhibits *interesting* local variations with respect to the original template and, nonetheless, the value of the shape energy computed with Eq. 4.5 for the shape of Figure 6.12(a) is slightly lower than the shape energy of the other two synthesized shapes. This example could indicate that the shape energy computed by our method cannot be employed to evaluate our synthesized shapes in terms of the variations obtained with respect to the template, but only as a measure of general similarity between our generated shapes and the input template. Additionally, for this thesis we have evaluated the shapes generated by our method by means of a visual inspection. As we will discuss in the following Chapter, a user study could be an option to perform a more thorough assessment of our synthesized man-made 3D shapes.

Issues and limitations.

Our results, in general, suggest that the similarity metric defined in Eq. 4.1, based on the set of descriptors that we have chosen, enables our synthesis system to select segments that allow us to preserve in a better manner the structure of the template in our synthesized shapes. However, we also observed in some shapes generated from a single template, e.g. the results presented in Figure 6.12, that our algorithm finds as replacements, for some cases, parts that are very similar to each other, as we can see especially for the segments used to replace the sides (next to the top) of the template table of Figure 6.12. Even though it is possible that our pool of fine grained parts might not contain enough segments with a high probability of being a suitable replacement for some particular parts of a given template, it is also possible that our method cannot find different replacements for some segments due to issues in our similarity metric. Specifically, although some of our descriptors are, very probably, more informative than others, we currently employ the same weights for each descriptor in our distance metric. Nonetheless, we should perform additional experiments, possibly involving new input datasets, to determine if, by using different weights in our distance metric, we can generate more local variations in our shapes

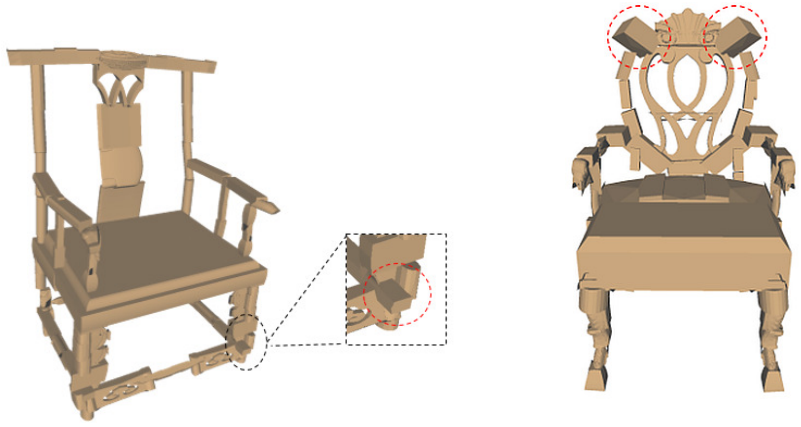
without affecting the capacity of our system to generate plausible shapes that preserve the general structure of the input template.

On the other hand, as we discussed in Section 4.5, our algorithm for the alignment of adjacent segments depends on finding matching contact points. In the case of a synthesized shape such as the one shown in Figure 6.1 (a), the round top of the table model must be aligned with the box-like segment that is adjacent to it. However, in this case, the process for aligning the segments initially places the segments of the legs close to the sides of the rounded top of the table. Hence, our algorithm obtains contact points that are misaligned, due to the significant differences in the surface of the adjacent boundaries of the segments. Using such contact points, the alignment process produces an incorrect scaling for the leg of this synthesized table. We can observe a similar issue in the case of the table presented in Figure 6.1 (a). As we mentioned earlier, we explored the use of previous approaches for mesh synthesis, e.g. [3], which assign as contact points all the vertices or points close to an adjacent segment. However, we found that using this approach, we obtain too many contact points that cannot be correctly matched.

Additionally, as observed by Kalogerakis et al. [3], the alignment process between adjacent segments that we use in our system does not control intersections between the neighboring segments. In some of our synthesized shapes, we also noticed that the alignment can produce intersections in some parts of our synthesized shapes, as we show in Figure 6.13(a).

Moreover, in some other cases, two adjacent segments can be part of a region of the template that has a curved surface, while the corresponding replacement segments do not exhibit these curvatures. In such cases, the alignment can fail to bring into contact the segments of the synthesized shape, or can align only a small portion of the boundaries of each adjacent segment, as we show in the inset of the left side of Figure 6.14. As we will discuss in Chapter 7, applying a deformation on one of the adjacent segments could be an alternative to solve this issue.

Similarly, in some cases we find conflicts in aligning adjacent segments while traversing the adjacency graph of the template shape, when a pair of adjacent segments have been aligned previously with respect to some other neighbor segment. For example, the inset on the right side of Figure 6.14 shows a case where the segment that we mark as number 2 has been aligned with respect to segment 1, and segment 3 has been aligned with respect to segment 4. In this case, we should also align the segments



(a) **(b)**

Figure 6.13: (a) Example of a case where the alignment process produced intersections between segments; (b) Wrong orientation of a segment caused by a suboptimal OBB computed for the segment.

2 and 3, but performing this alignment would disconnect the two pairs of segments that were already aligned. As we explained in Section 4.5.1, to avoid computing the alignment in cases that could cause a disconnection of adjacent segments previously aligned, we mark the segments that we align while we traverse the adjacency graph of the template shape, so that we can avoid computing the alignment for segments that were aligned in a previous stage. Nevertheless, we have found that performing the traversal of the adjacency graph using a breadth-first approach, and starting this traversal from one of the segments that has the largest number of neighbors in the template, allows us to avoid this type of conflicts in most cases.

In addition, a general limitation of our approach is that we require our fine-grained segments to be consistent in size for corresponding parts of the shapes of our input datasets. Therefore, we must segment some parts of the input shapes that do not contain salient features or fine geometrical details. For input datasets containing a large number of 3D models, achieving a consistent fine-grained segmentation using our interactive segmentation program would require significant manual effort. However, in Chapter 7, we will describe some options that can be employed to compute our fine-grained segments automatically.

Moreover, our synthesized shapes contain some segments that exhibit certain artifacts, as we can see, for example, in one of segments of the legs of the shape in

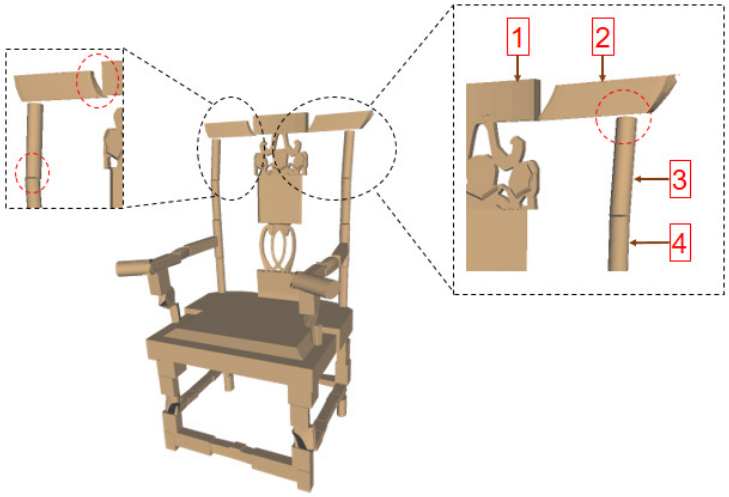


Figure 6.14: Shape synthesized with our method where we highlight some of the issues and limitations of our process of alignment between adjacent segments.

Figure 6.14. In some cases, these artifacts originate in issues in the fine-grained segments extracted with our interactive tool for point cloud segmentation, i.e., the segment captured using our interactive program might include points that we did not want to include in a given segment, or exclude some points that should have been included in a segment. However, some other artifacts in our segments are present only in our synthesized meshes, since such artifacts are due to cases where our heuristic method for filtering out faces that should not be included in a mesh segment fails to exclude some faces or, on the contrary, excludes faces that should be part of the mesh segment. In the following chapter, we discuss possible alternatives that can be explored to fix or avoid these issues caused by the segmentation.

Finally, one additional limitation of our method occurs in some cases where the algorithm to compute the OBBs of the segments provides an incorrect orientation for the box. This problem, which has also been observed in previous works [36], can cause inconsistencies in the orientation of some of the segments of our synthesized shapes, as we show in the example of Figure 6.13(b). Furthermore, as we mentioned in Chapter 5, in some cases, identifying a specific axis as the principal axis of orientation might not be applicable for some segments, e.g., in the case of parts with a spherical form. Nevertheless, we observed that, by marking a given axis of orientation for all our fine-grained parts using our interactive segmentation program, we can obtain a consistent orientation for most of our segments.

6.6 Implementation details and execution times

We implemented the main components of our pipeline using Matlab R2016a, while we employed C++ for the extraction of the Oriented Bounding Boxes (OBBs) of the segments, using the method by Fish et al. [36]. We executed our experiments on a PC with a 3.4Ghz Intel Core (TM) i7 6700 and 16Gb of RAM.

After pre-processing all the input shapes to extract our set of fine-grained parts, we obtained the set of descriptors for each segment. With our current implementation, computing all the descriptors for the input set of chairs, which contains 1,627 fine-grained segments, requires 251.4 secs. (4.19 mins). For our set of tables, which has 415 segments, computing all the descriptors takes 131.4 secs (2.19 mins).

Next, we obtain the value of our similarity metric, given by Eq. 4.1, between each pair of segments. For the set of chairs, calculating this distance requires approximately 49.8 secs (0.83 mins). For the input set of tables, computing our similarity metric only takes 24 secs (0.4 mins). In the case of our hybrid set, we can use the value of the descriptors computed for the segments of the datasets of tables and chairs. Thus, for the hybrid set, which contains a total of 1,229 segments, we only require to calculate our similarity metric, which requires approximately 48.6 secs. (0.81 mins).

In practice, we only compute the descriptors and the distance similarity between each pair of segments once, and we store the distance values for later use. Moreover, we also perform the processes of mesh subdivision and extraction of mesh segments as part of our pre-processing. Thus, to generate a synthesized shape for any given input template, we only require executing the sampling of segments for the shape, which takes only a fraction of a second, and the shape post-processing, which includes generating a point cloud and 3D mesh. We summarize in Table 6.1 the average times required for our shape post-processing, with our unoptimized Matlab implementation.

Dataset	Point cloud processing	Writing of point cloud file	Mesh processing	Writing of mesh file
Tables	14.98	0.81	0.86	10.06
Chairs	13.92	0.77	0.69	8.21
Hybrid	16.21	0.73	0.78	9.98

Table 6.1: Average times, in seconds, required to synthesize a shape.

As we can see in Table 6.1, after aligning the segments of the synthesized point cloud, creating the final mesh requires a very short time. However, to output the file containing the synthesized 3D mesh requires a considerable time, since our subdivided meshes contain a relatively large number of faces.

Chapter 7

Conclusions and future work

In this thesis, we presented a pipeline for 3D synthesis of man-made shapes based on recombining fine-grained segments extracted from an input set of shapes. The results of experiments using our approach with different shape datasets have shown that, by using a set of fine-grained segments, our method can synthesize shapes that have a wide range of local fine details, unlike previous approaches for shape synthesis which use large semantic segments [3, 5, 6]. Additionally, we showed that our method can generate shapes using a hybrid set that contains shapes of different classes.

Moreover, our method does not require a semantic segmentation or part correspondences between the shapes of the input dataset, unlike most previous works. In contrast, our method uses a template shape to guide the synthesis process, generating plausible shapes that maintain the structure and topology of the template.

Furthermore, we formulated the synthesis problem as a sampling of individual segments to compose a new shape, replacing the segments of the template with other fine-grained segments extracted from the input dataset. To generate a plausible shape, we find each segment for composing the shape by minimizing an energy function that encodes a similarity metric based on a set of descriptors. The results of our experiments suggest that our energy function captures in an appropriate manner the overall similarities between the parts of the original template and the parts sampled by our process to synthesize a new shape. In addition, we also explored an alternative to our pipeline which generates shapes that preserve certain portions of the original template, allowing us to constrain the synthesis process.

For the extraction of the fine-grained segments used in our work, we presented a semi-automatic method that employs user assistance, which allows us to obtain the segments directly from the point cloud representations that we extract from the 3D

shapes of our input dataset. Additionally, we discussed our advances to compute a fine-grained segmentation automatically.

Since our fine-grained segments are extracted from the point cloud representations of the input shapes, we also proposed a method to obtain a triangle mesh corresponding to each segment defined on the point clouds, allowing us to generate our synthesized shapes using both representations: point clouds and 3D meshes.

Future work. As mentioned in Chapter 6, we would like to perform a user study to make a more comprehensive assessment of different aspects of our synthesis approach. Firstly, we could measure the effectiveness of our method to produce compelling and plausible fine local variations in comparison to the input template by asking the user to identify the synthesized shapes that have interesting or unexpected fine details that do not change the general structure of the template. Secondly, we could also ask the users to choose, among various 3D shapes generated from a single template, the model that has an overall form that is more similar to the form of the template, which could allow us to compare the values of the shape energy computed with our approach with the answers provided by the users. Thirdly, we could evaluate the plausibility of the shapes synthesized by our approach with respect to the 3D objects of the input dataset, in comparison to the plausibility achieved by previous works, e.g., [4, 29]. Similarly, assuming that we can employ the same input dataset to generate 3D shapes with our method and other existing approaches, we could ask the users to select, between the shapes generated by our algorithm and the shapes generated by those previous works, the models that exhibit the most interesting fine local variations with respect to the original input shapes.

In addition, there are many other directions for future work that could allow us to improve or to extend different components of our pipeline.

First, we would like to extract our fine-grained segments in a fully automatic manner. To this end, we could improve our implementation by combining the tools for regular partitioning of segments, which are part of our semi-automatic segmentation tool, with the automatic approach that segments meshes based on their geometry. In this manner, we could obtain fine-grained segments that are more consistent in size using an automatic process. Additionally, we could also improve the consistency of the segments by training a model to perform the fine-grained segmentation, by means of traditional learning methods [29], or recent deep-learning techniques [16].

Another possibility for future work consists in the use of additional options to constrain the synthesis process. For instance, by using not just a single input template, but by combining parts coming from different templates, would allow us to generate shapes with additional variations, producing, for example, topological changes.

As we mentioned in Section 6.5, there are cases where the segments that are adjacent in our synthesized shapes cannot be aligned in a correct manner, due to significant differences in the geometry of the adjacent boundaries of each segment. In such cases, instead of computing an alignment between the segments, we could deform the boundaries of the segments so that they match to each other. This could be achieved, for instance, with a deformation with transformation propagation [89]. In addition, our energy function could also consider the matching energy only between the boundaries of adjacent segments.

Another possibility for future work consists in obtaining a mesh from the synthesized point cloud with a surface reconstruction approach, instead of extracting the mesh segments from the original input shapes. As we mentioned in Chapter 3, methods like Poisson surface reconstruction [70] may produce an over-smoothed mesh, causing the loss of the fine details that we seek to maintain. Nonetheless, we could constrain the reconstruction process to preserve the internal parts of the segments while smoothing the boundary regions, thus avoiding the loss of the salient features.

Appendix A

Input datasets

In this appendix, we present figures containing all the shapes of our input datasets. Figure A.1 shows our input set of tables, which includes 15 models, while Figure A.2 shows our input dataset of chairs, which consists of 30 shapes. Additionally, we also have employed a hybrid dataset to further validate our approach. Out hybrid set contains all the table models together with 15 shapes chosen from our chairs dataset. The chair models that are part of our hybrid set are shown with blue color in the two bottom rows of Figure A.2.

Figure A.1: Shapes of our dataset of tables.

Figure A.2: Shapes of our dataset of chairs. The models displayed in blue are also part of our hybrid dataset.

List of References

[1] O. van Kaick, N. Fish, Y. Kleiman, S. Asafi, and D. Cohen-Or, "Shape segmentation by approximate convexity analysis," *ACM Trans. on Graphics*, vol. 34, no. 1, pp. 4:1–4:11, 2014.

[2] N. Mitra, M. Wand, H. R. Zhang, D. Cohen-Or, V. Kim, and Q.-X. Huang, "Structure-aware shape processing," in *SIGGRAPH Asia 2013 Courses*, pp. 1:1–1:20, ACM, 2013.

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

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

[5] 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.

[6] H. Liu, U. Vimont, M. Wand, M.-P. Cani, S. Hahmann, D. Rohmer, and N. J. Mitra, "Replaceable substructures for efficient part-based modeling," *Computer Graphics Forum*, vol. 34, no. 2, pp. 503–513, 2015.

[7] 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, 2015.

[8] 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.

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

[10] O. Sidi, O. van Kaick, Y. Kleiman, H. Zhang, and D. Cohen-Or, "Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering," *ACM Trans. on Graphics*, vol. 30, no. 6, pp. 126:1–126:10, 2011.

[11] "Trimble 3D Warehouse." `https://3dwarehouse.sketchup.com/index.html`, 2017. Accessed: 2017-10-02.

[12] 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," Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[13] I. Alhashim, H. Li, K. Xu, J. Cao, R. Ma, and H. Zhang, "Topology-varying 3D shape creation via structural blending," *ACM Trans. on Graphics*, vol. 33, no. 4, pp. 158:1–158:10, 2014.

[14] A. Shamir, "A survey on mesh segmentation techniques," *Computer Graphics Forum*, vol. 27, no. 6, pp. 1539–1556, 2008.

[15] P. Theologou, I. Pratikakis, and T. Theoharis, "A comprehensive overview of methodologies and performance evaluation frameworks in 3D mesh segmentation," *Comp. Vision and Image Understanding*, vol. 135, pp. 49–82, 2015.

[16] Z. Shu, C. Qi, S. Xin, C. Hu, L. Wang, Y. Zhang, and L. Liu, "Unsupervised 3D shape segmentation and co-segmentation via deep learning," *Computer Aided Geometric Design*, vol. 43, pp. 39–52, 2016.

[17] L. Yi, L. Guibas, A. Hertzmann, V. G. Kim, H. Su, and E. Yumer, "Learning hierarchical shape segmentation and labeling from online repositories," *ACM Trans. on Graphics*, vol. 36, no. 4, pp. 70:1–70:12, 2017.

[18] K. Xu, V. G. Kim, Q. Huang, N. Mitra, and E. Kalogerakis, "Data-driven shape analysis and processing," in *SIGGRAPH ASIA 2016 Courses*, p. 4, ACM, 2016.

[19] P. Jaiswal, J. Huang, and R. Rai, "Assembly-based conceptual 3D modeling with unlabeled components using probabilistic factor graph," *Computer-Aided Design*, vol. 74, pp. 45–54, 2016.

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

[21] V. Kraevoy, D. Julius, and A. Sheffer, "Model composition from interchangeable components," in *Proc. Pacific Graphics*, pp. 129–138, IEEE, 2007.

[22] A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or, "Snappaste: an interactive technique for easy mesh composition," *The Visual Computer*, vol. 22, no. 9, pp. 835–844, 2006.

[23] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Pattern Analysis & Machine Intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[24] P. J. Besl, N. D. McKay, *et al.*, "A method for registration of 3-d shapes," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[25] R. Schmidt and K. Singh, "Meshmixer: an interface for rapid mesh composition," in *ACM SIGGRAPH 2010 Talks*, p. 6, ACM, 2010.

[26] K. Takayama, R. Schmidt, K. Singh, T. Igarashi, T. Boubekeur, and O. Sorkine, "Geobrush: Interactive mesh geometry cloning," *Computer Graphics Forum*, vol. 30, no. 2, pp. 613–622, 2011.

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

[28] D. Koller and N. Friedman, *Probabilistic Graphical Models*. MIT Press, 2009.

[29] E. Kalogerakis, A. Hertzmann, and K. Singh, "Learning 3D mesh segmentation and labeling," *ACM Trans. on Graphics*, vol. 29, no. 4, pp. 102:1–102:12, 2010.

[30] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[31] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonisation using the shape diameter function," *The Visual Computer*, vol. 24, no. 4, p. 249, 2008.

[32] V. Blanz and T. Vetter, "A morphable model for the synthesis of 3D faces," in *Proc. SIGGRAPH*, pp. 187–194, ACM Press, 1999.

[33] B. Allen, B. Curless, and Z. Popović, "The space of human body shapes: reconstruction and parameterization from range scans," *ACM Trans. on Graphics*, vol. 22, no. 3, pp. 587–594, 2003.

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

[35] M. Averkiou, V. G. Kim, Y. Zheng, and N. J. Mitra, "ShapeSynth: parameterizing model collections for coupled shape exploration and synthesis," *Computer Graphics Forum*, vol. 33, no. 2, pp. 125–134, 2014.

[36] N. Fish, M. Averkiou, O. Van Kaick, O. Sorkine-Hornung, D. Cohen-Or, and N. J. Mitra, "Meta-representation of shape families," *ACM Trans. on Graphics*, vol. 33, no. 4, pp. 34:1–34:11, 2014.

[37] Q. Fu, X. Chen, X. Su, J. Li, and H. Fu, "Structure-adaptive shape editing for man-made objects," *Computer Graphics Forum*, vol. 35, no. 2, pp. 27–36, 2016.

[38] M. Bokeloh, M. Wand, and H.-P. Seidel, "A connection between partial symmetry and inverse procedural modeling," *ACM Trans. on Graphics*, vol. 29, no. 4, pp. 104:1–104:10, 2010.

[39] J. Kalojanov, M. Bokeloh, M. Wand, L. Guibas, H.-P. Seidel, and P. Slusallek, "Microtiles: Extracting building blocks from correspondences," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1597–1606, 2012.

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

[41] Z. Han, Z. Liu, J. Han, and S. Bu, "3d shape creation by style transfer," *The Visual Computer*, vol. 31, no. 9, pp. 1147–1161, 2015.

[42] T. Liu, A. Hertzmann, W. Li, and T. Funkhouser, "Style Compatibility for 3D Furniture Models," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, pp. 1–9, 2015.

[43] Z. Lun, E. Kalogerakis, and A. Sheffer, "Elements of style: learning perceptual shape style similarity," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, pp. 84:1–84:14, 2015.

[44] S. Asafi, A. Goren, and D. Cohen-Or, "Weak convex decomposition by lines-of-sight," *Computer Graphics Forum*, vol. 32, no. 5, pp. 23–31, 2013.

[45] R. Hu, W. Li, O. V. Kaick, H. Huang, M. Averkiou, D. Cohen-Or, and H. Zhang, "Co-Locating Style-Defining Elements on 3D Shapes," *ACM Trans. on Graphics*, vol. 36, no. 3, pp. 33:1–33:15, 2017.

[46] A. Jain, T. Thormählen, T. Ritschel, and H.-P. Seidel, "Exploring shape variations by 3D-model decomposition and part-based recombination," *Computer Graphics Forum*, vol. 31, no. 2pt3, pp. 631–640, 2012.

[47] A. Berner, M. Bokeloh, M. Wand, A. Schilling, and H.-P. Seidel, "A graph-based approach to symmetry detection.," in *Volume Graphics*, vol. 40, pp. 1–8, 2008.

[48] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," in *Artificial Intelligence and Statistics*, pp. 448–455, 2009.

[49] O. Van Kaick, H. Zhang, G. Hamarneh, and D. Cohen-Or, "A Survey on Shape Correspondence," *Computer Graphics Forum*, pp. 1–24, 2011.

[50] S.-S. Huang, H. Fu, L.-Y. Wei, and S.-M. Hu, "Support substructures: support-induced part-level structural representation," *IEEE Trans. Visualization & Computer Graphics*, vol. 22, no. 8, pp. 2024–2036, 2016.

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

[52] G. Lavoué, F. Dupont, and A. Baskurt, "A new cad mesh segmentation method, based on curvature tensor analysis," *Computer-Aided Design*, vol. 37, pp. 975–987, 2005.

[53] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, *Polygon mesh processing*. CRC press, 2010.

[54] N. Shental, A. Bar-Hillel, T. Hertz, and D. Weinshall, "Computing gaussian mixture models with em using equivalence constraints," in *Advances in neural information processing systems*, pp. 465–472, 2004.

[55] C. Sutton, A. McCallum, *et al.*, "An introduction to conditional random fields," *Foundations and Trends® in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2012.

[56] Z. Xie, K. Xu, L. Liu, and Y. Xiong, "3D shape segmentation and labeling via extreme learning machine," *Computer Graphics Forum*, vol. 33, no. 5, pp. 85–95, 2014.

[57] K. Bamba and R. Ohbuchi, "Supervised, geometry-aware segmentation of 3d mesh models," in *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on*, pp. 49–54, IEEE, 2012.

[58] A. Golovinskiy and T. Funkhouser, "Randomized cuts for 3d mesh analysis," *ACM transactions on graphics (TOG)*, vol. 27, no. 5, pp. 145:1–145:12, 2008.

[59] O. K.-C. Au, Y. Zheng, M. Chen, P. Xu, and C.-L. Tai, "Mesh segmentation with concavity-aware fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 7, pp. 1125–1134, 2012.

[60] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3d mesh segmentation," *ACM transactions on graphics (TOG)*, vol. 28, no. 3, p. 73, 2009.

[61] R. Hu, L. Fan, and L. Liu, "Co-segmentation of 3d shapes via subspace clustering," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1703–1713, 2012.

[62] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polyhedra and its applications," *Computer Aided Geometric Design*, vol. 25, no. 7, pp. 503–522, 2008.

[63] H. Zhang, O. Van Kaick, and R. Dyer, "Spectral mesh processing," *Computer graphics forum*, vol. 29, no. 6, pp. 1865–1894, 2010.

[64] H. Benhabiles, J.-P. Vandeborre, G. Lavoué, and M. Daoudi, "A comparative study of existing metrics for 3d-mesh segmentation evaluation," *The Visual Computer*, vol. 26, no. 12, pp. 1451–1466, 2010.

[65] Z. Liu, J. Zhang, and L. Liu, "Upright orientation of 3D shapes with convolutional networks," *Graphical Models*, vol. 85, pp. 22–29, 2016.

[66] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, "Shape distributions," *ACM Trans. on Graphics*, vol. 21, no. 4, pp. 807–832, 2002.

[67] V. G. Kim, W. Li, N. J. Mitra, S. Chaudhuri, S. DiVerdi, and T. Funkhouser, "Learning part-based templates from large collections of 3D shapes," *ACM Trans. on Graphics*, vol. 32, no. 4, pp. 70:1–70:12, 2013.

[68] C. Ma, H. Huang, A. Sheffer, E. Kalogerakis, and R. Wang, "Analogy-driven 3d style transfer," *Computer Graphics Forum*, vol. 33, no. 2, pp. 175–184, 2014.

[69] R. Hu, O. van Kaick, B. Wu, H. Huang, A. Shamir, and H. Zhang, "Learning how objects function via co-analysis of interactions," *ACM Trans. on Graphics*, vol. 35, no. 4, pp. 47:1–47:13, 2016.

[70] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 3, pp. 29:1–29:13, 2013.

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

[72] 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. 1–23, 2013.

[73] R. Hu, C. Zhu, O. V. Kaick, L. Liu, A. Shamir, and H. Zhang, "Interaction Context (ICON): Towards a Geometric Functionality Descriptor," *Siggraph*, 2015.

[74] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Learning informative point classes for the acquisition of object model maps," in *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pp. 643–650, IEEE, 2008.

[75] R. B. Rusu, *Semantic 3D Object Maps for Everyday Robot Manipulation*. Springer, 2013.

[76] X. Zhao, H. Wang, and T. Komura, "Indexing 3D scenes using the interaction bisector surface," *ACM Trans. on Graphics*, vol. 33, no. 3, pp. 22:1–22:14, 2014.

[77] R. Ohbuchi, K. Osada, T. Furuya, and T. Banno, "Salient local visual features for shape-based 3d model retrieval," in *2008 IEEE International Conference on Shape Modeling and Applications*, pp. 93–102, 2008.

[78] J. W. H. Tangelder and R. C. Veltkamp, "A survey of content based 3d shape retrieval methods," in *Proceedings Shape Modeling Applications, 2004.*, pp. 145–156, 2004.

[79] F. Mémoli and G. Sapiro, "Comparing point clouds," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pp. 32–40, ACM, 2004.

[80] M. Pauly, M. Gross, and L. P. Kobbelt, "Efficient simplification of point-sampled surfaces," in *Proc. Visualization*, pp. 163–170, IEEE Computer Society, 2002.

[81] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *SIGGRAPH Comput. Graph.*, vol. 26, July 1992.

[82] A. Zeng, S. Song, M. Niener, M. Fisher, J. Xiao, and T. Funkhouser, "3dmatch: Learning local geometric descriptors from rgb-d reconstructions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 199–208, 2017.

[83] L. Bo, X. Ren, and D. Fox, "Unsupervised feature learning for rgb-d based object recognition," in *Experimental Robotics: The 13th International Symposium on Experimental Robotics* (J. P. Desai, G. Dudek, O. Khatib, and V. Kumar, eds.), pp. 387–402, Springer International Publishing, 2013.

[84] H. Laga, M. Mortara, and M. Spagnuolo, "Geometry and context for semantic correspondences and functionality recognition in man-made 3d shapes," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 5, pp. 150:1–150:16, 2013.

[85] R. Parent, *Computer animation: algorithms and techniques*. Newnes, 2012.

[86] A. Tevs, Q. Huang, M. Wand, H.-P. Seidel, and L. Guibas, "Relating shapes via geometric symmetries and regularities," *ACM Trans. on Graphics*, vol. 33, no. 4, pp. 119:1–119:12, 2014.

[87] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[88] C. Loop, "Smooth subdivision surfaces based on triangles," Master's thesis, University of Utah, 1987.

[89] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rssl, and H.-P. Seidel, "Laplacian surface editing," in *Symposium on Geometry Processing*, Eurographics, 2004.