Support Substructures
Support-Induced Part-Level Structural Representation
Huang, Shi-Sheng; FU, Hongbo; Wei, Ling-Yu; Hu, Shi-Min

Published in:
IEEE Transactions on Visualization and Computer Graphics

Published: 01/08/2016

Document Version:
Post-print, also known as Accepted Author Manuscript, Peer-reviewed or Author Final version

License:
Unspecified

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1109/TVCG.2015.2473845

Publication details:

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.

Download date: 16/02/2019
Support Substructures: Support-Induced Part-Level Structural Representation

Shi-Sheng Huang, Hongbo Fu, Lin-Yu Wei, and Shi-Min Hu, Member IEEE

Abstract—In this work we explore a support-induced structural organization of object parts. We introduce the concept of support substructures, which are special subsets of object parts with support and stability. A bottom-up approach is proposed to identify such substructures in a support relation graph. We apply the derived high-level substructures to part-based shape reshuffling between models, resulting in nontrivial functionally plausible model variations that are difficult to achieve with symmetry-induced substructures by the state-of-the-art methods. We also show how to automatically or interactively turn a single input model to new functionally plausible shapes by structure rearrangement and synthesis, enabled by support substructures. To the best of our knowledge no single existing method has been designed for all these applications.

Index Terms—Support Substructure, Shape Synthesis, 3D Modeling, Stability

1 INTRODUCTION

Understanding high-level structure of a 3D model greatly benefits a variety of applications such as structure-preserving editing [1] and upright orientation [2]. High-level structures are often closely related to the functionality of an object and are thus difficult to define and detect. Most existing works (e.g., [3], [4]) towards high-level shape understanding regard symmetry as the main semantic cue for shape analysis. Such approaches are inapplicable to objects or scenes exhibiting little or no symmetry (Figure 1).

Support and stability are two fundamental attributes of objects in the physical world, especially for man-made objects. This motivates us to explore a support-induced high-level structural shape representation. In this work we focus on three types of supporting relationship, namely, “support from below”, “support from above”, and “support from side” (Figure 3). A simplest supporting scenario might be a single object part stably supported by another object part. We extend this supporting scenario by allowing support in multiple hierarchy and/or hierarchical groups of similar parts if any (Section 3.1), which already explains a variety of support substructures (Figure 1 (b)).

Based on this observation we first form an input set of object parts with the three supporting relations as a partially ordered set, represented as a support relation graph (Section 3.2). A bottom-up approach is then presented to first identify a set of basic support substructures and then combine them to form complicated substructures with support in multiple hierarchy.

Our technique results in a new structural representation of object parts. Our support substructures form a basis for deriving structural similarity, which has great potential for various applications. For example, we show that reshuffling two or more support substructures automatically leads to nontrivial, interesting shape variations (Section 4.1) that are difficult to achieve with the existing works. In addition, we apply support substructures to structure rearrangement (Section 4.2) and structure synthesis (Section 4.3), which automatically create many new functionally plausible shape variations from a single model alone. An interactive structure synthesis tool is also presented to allow explicit user control (Figure 1 (e)).

Our work makes three main contributions:

1) For the first time we introduce the concept of support substructure for structural organization of object parts. We show that besides symmetry, support and stability are also important cues for understanding synthesizing functionally plausible new shapes.

2) We present a bottom-up approach to identify support substructures induced by three types of supporting relationship.

3) We show the application of support substructures to shape reshuffling, structure rearrangement, and structure synthesis. No single existing method has been designed for all these applications.

2 RELATED WORK

In recent years support and stability cues are getting popular in the computer vision community for 3D interpretation of a scene, thanks to the availability of consumer-level depth cameras. For example, Jia
et al. [5] proposed to jointly optimize over segmentations, block fitting, supporting relations and object stability for 3D reasoning. The work of Panda et al. [6] first learns the semantics in terms of supporting relationship among different objects and then use the inferred supporting relationship to predict a support order for robotic manipulation in clutter. Silberman et al. [7] present an automatical approach to infer support relationships of an indoor scene from an RGB-D image. Unlike these works, which mainly use support and stability to find a good, physically valid interpretation of a scene, ours takes a stable, self-supporting arrangement of object parts as input and aims to derive semantic substructures for high-level shape editing and manipulation.

Support and/or stability have also been widely used in the field of computer graphics, e.g., for mesh puppetry [8], upright shape orientation determination [2], furniture layout synthesis [9], furniture design [10], self-supporting surface design [11], [12], [13] or freeform architecture design [14], structural analysis for 3D printing [15], balanced shape design [16] etc. Like ours, most of these works determine static stability via geometric validation instead of physical simulation or validation [10]. We believe that our derived support substructures can potentially benefit some of these applications, besides those demonstrated in the paper. Our analysis of part-level relationship is conceptually related to [17], [18], which, however, rely on neither support nor stability.

It is well known that symmetry provides a strong cue for high-level understanding of shapes exhibiting rich symmetry. This has motivated a series of symmetry detection algorithms and symmetry-based applications (see an insightful survey in [19]). Symmetry can be used to form a hierarchical organization of object parts [20]. However, it is unclear how such a general symmetry hierarchy could be exploited for applications like ours. Very recently, Zheng et al. [3] introduce a symmetric functional arrangement, called SFARR, which always contains a triplet of shape parts, with one part stably supported by or supporting another two symmetric parts. With such a simple representation a diverse set of plausible model variations can be synthesized from a small set of input models. However their technique is inapplicable to other rich substructures, e.g., with different numbers of symmetric elements or no symmetry at all. In addition how to apply their technique to our other applications is also unclear. In contrast symmetry is not inherent in our support-induced substructures, though our representation does make use of the symmetry information if any and naturally supports not only SFARR but also different types of symmetric or non-symmetric arrangements. The very recent work by [4] presents a topology-varying structural blending algorithm, where symmetry also plays a critical role to produce continuous and plausible in-betweens undergoing topology variations.

Semi-automatically or automatically synthesizing new shape variations from existing models has been of a great interest in recent years. The main goal is to generate hundreds and thousands of models with little or even no user intervention, which otherwise would be a rather tedious process for commercial modeling systems like Autodesk Maya. Below we only review the most relevant works to ours. Bokeloh et al. [21] present an inverse procedural modeling system which examines partial symmetries of a single 3D example model to automatically produce new 3D models that are similar to the input exemplar. Merrell et al. [22] provide a general procedural modeling method to synthesize complex 3D shapes, based on various dimensional, geometric, and algebraic con-
Figure 2. We assume that the direction of a supporting relation between a pair of building blocks is geometrically determined, like those existing in (a)–(c). See a counterexample in (d), the three leg parts support each other, which beyond our assumption requiring that the direction of one supporting relation is unique. The graph in (c) illustrates the support relation graph for the table model.

Figure 3. The supporting relations (“support from below”, “support from above” and “support from side”) between parts of a pre-segmented model (e) is represented as a support relation graph (f). (a-d): a subset of detected support structures, with supporting components shown in green and supported ones in red.

3 SUPPORT SUBSTRUCTURES

In this section we first introduce the definition of support substructures and then present an algorithm to detect such substructures in an input pre-segmented model. We will show three applications of the detected substructures in Section 4.

3.1 Definitions and Assumptions

Supporting relations. Objects, especially man-made ones, can often be decomposed into a set of building blocks (Figure 2 (b) and (c)), which are connected by certain supporting relations to make the objects stable. In this work we explore three types of commonly seen supporting relations, namely “support from below”, “support from above”, and “support from side”, as illustrated in Figure 3. As we will show shortly that while these relations look simple they are surprisingly sufficient to produce rich support substructures and make the state-of-the-art substructures [3] become only a special case of ours.

A supporting relation is essentially a binary relation, defined between a pair of building blocks. We assume that the direction of a supporting relation can be always geometrically determined and is unique. This assumption is applicable to supporting relations existing in many man-made objects (e.g., Figures 2 (b–c) and 3). However, it is not always valid. For instance, we are not interested in a pair of parts that support each other. Figure 2 (d) shows such a counterexample.

Under this assumption a supporting relation, denoted as \( \leq \), brings a partial order to a set of building blocks of an object, denoted as \( M \). Specifically, for all elements \( a, b, \) and \( c \) in \( M \), \( \leq \) satisfies: 1) reflexivity: \( a \leq a \), a building block \( a \) supports itself naturally; 2) antisymmetry: if \( a \) supports \( b \) and \( b \) supports \( a \) at the same time, we have \( a = b \); 3) transitivity: as illustrated in Figure 2 (a), if \( a \) supports \( b \) (i.e., \( a \leq b \)) and \( b \) supports \( c \) (i.e., \( b \leq c \)), we can conclude \( a \) supports \( c \), i.e., \( a \leq c \).

Our three types of supporting relation over \( M \) lead to a partially ordered set \( (M, \leq) \), also called a poset in set theory. Such a poset can be equivalently represented as a support relation graph, which is essentially a directed acyclic graph (DAG), with set elements as nodes and binary relations as directed edges, pointing from supporting elements to supported ones (Figure 3). A similar support relation graph has been explored in [6], but used only for predicting support order.

Support substructures. A support substructure is formed by a subset of our poset or a subgraph of our support relation graph. A desired definition of support substructure should not only allow the ex-
traction of a rich set of substructures from an object but also effectively help form functionally plausible new variations for various applications.

Our definition of support substructure is an extension of a primitive support substructure. A primitive support substructure contains an ordered pair of primitive elements \((a, b)\) with \(a\) stably supporting \(b\), i.e., \(a \leq b\). That is \(b\) can achieve static equilibrium with the support from \(a\), as illustrated in Figure 2 (a). Let \(\{a\} \preceq \{b\}\), with \(\preceq\) meaning a stable supporting relation, denote such a primitive substructure, which is consistently observed across objects (e.g., Figure 3 (c)). Given two primitive substructures \(\{a\} \preceq \{b\}\) and \(\{c\} \preceq \{d\}\), it is easy to perform structural validation. For example, to see if we can form a structurally valid new variation with \(a\) and \(d\), we only need to check stability and the compatibility of support type between \(a\) and \(d\).

It is very often that a group of elements with very similar or even the same shape, denoted as \(A_s\), together stably support a primitive part \(b\) (e.g., four chair legs supporting a seat in Figure 2 (b)). We thus also consider \(A_s \preceq_T \{b\}\) as a support substructure if the type of supporting relation \(T\) between \(b\) and \(a\), \(\forall a \in A_s\), is the same. See such an example in Figure 3 (b). Since similar elements are perceptually grouped together, no subset of \(A_s\) is allowed to form a substructure with others. Similarly our support substructure also extends to \(\{a\} \preceq_T B_s\) (e.g., an airplane body supporting two wings as shown in Figure 3 (d)), where \(a\) is a primitive part, \(B_s\) is a group of similar parts, and \(a\) stably supports any part in \(B_s\) with the same supporting relation \(T\). In short, our extension of support substructures is applicable to a group of similar elements connected to a single element with the same supporting relation. This also extends to hierarchical grouping of similar elements like the example shown in Figure 3 (a), where we have two wings connected to the body (as a single node) and two turbine engines connected to each of the wings. Note that the sFARR substructures [3] is only a special case of our support substructures.

The primitive support substructure can also be extended to \(A_c \preceq_T \{b\}\), where \(A_c\) denotes the entire set of elements with the same supporting relation to primitive element \(b\) and together stably supporting \(b\). Here individual elements of \(A_c\) are not necessarily of similar shape (Figure 2 (c)). Any subgroup of similar parts in \(A_c\) is still not allowed to form a substructure with others. To keep the stability of an existing substructure, a subset of \(A_c\) is not allowed to form a new substructure with \(b\). However, it is possible that individual elements of \(A_c\) form new substructures with other elements supporting them. For example, we don’t allow any subset \(S_{A_c} \subseteq A_c\) form a support substructure with \(b\), but the support substructure \(S_{A_c} \preceq_T c\) is allowed with \(c \neq b\) if any.

![Figure 4. Four kinds of basic support substructures.](image1)

![Figure 5. Illustration of support substructures combination. Iteratively combining newly created substructures leads to substructures with support in (possibly multiple) hierarchy. S: support from below; H: support from above; A: support from side.](image2)

We call \(\{a\} \preceq \{b\}\), \(A_s \preceq_T \{b\}\), \(\{a\} \preceq_T B_s\), and \(A_c \preceq_T \{b\}\) basic support substructures (Figure 4), since both the supporting and supported components of such a substructure involve only one layer of elements. Many objects exhibit support in hierarchy, i.e., one part supporting the other in single or multiple hierarchy. In case of support in multiple hierarchy, a part is supported by other parts at different hierarchies (Figure 2 (c)). We thus extend the definition of support substructure by support in multiple hierarchy. Specifically, let \(A_1 \preceq_T B_1\) and \(A_2 \preceq_T B_2\) denote two support substructures, with \(A_1 \cap B_2 = \emptyset\) and \(B_1 \cap A_2 = \emptyset\). In other words \(A_1\) and \(B_2\) share certain object parts. Then a new support substructure with an increased hierarchy level can be created by combining the given two substructures, leading to either \(A_3 \cup B_2 \cup A_2 \preceq_T B_1\) or \(A_2 \preceq_T B_2 \cup A_1 \cup B_1\), as illustrated in Figure 5. It can be easily seen that if a group of similar parts connected to a part is considered as a unified element, any support substructure by our definition essentially forms an upper semilattice, which is a subset of our poset with a least upper bound for any nonempty finite subsets.

### 3.2 Detection

We focus on man-made objects that have a known upright orientation [2]. Whether an entire input model is “supported from below” (e.g., by the ground) or “supported from above” (e.g., hung from a ceiling) is also given as input to our algorithm. Below we will explain the algorithm in the context of an input model placed on the ground. Adapting the algorithm to a model supported from above is straightforward.
Pre-segmentation. We require the availability of semantically meaningful segments for the input model. In our implementation, we segment an input model by using an SDF-based mesh partition method [30] and manually adjust the automatically generated segmentation results, if necessary. See such a segmented model in the top-left corner of Figure 7.

Support relation graph. To identify support substructures, we first build a support relation graph, denoted as \( G \), (Figure 7) as follows. Each object part leads to one node in the graph. All the nodes whose corresponding parts touch the ground plane are marked as ground-touching nodes, denoted as \( V_g \). A pair of components \( a \) and \( b \) are determined to be connected if their convex hulls intersect at several points (more than 5 in our experiment) or the minimal distance between them is below a threshold \( \delta \) (Section 5). With Principal Component Analysis (PCA) we approximate a plane \( P \) using the points where \( a \) and \( b \) are in contact.

We first locally classify the type of support between \( a \) and \( b \). If \( a \) and \( b \) are connected, we have a “support from side” relation if the normal of \( P \) is nearly perpendicular to the upright orientation (deviation angle \( \leq 10^\circ \)). Otherwise we temporarily assign a “support from below” relation to \( a \) and \( b \), and add a directed edge from the one below the plane to the other. After all pairs of connected parts are examined, we add a directed edge between each pair of object parts with “support from side”, denoted as \( c \) and \( d \). Specifically, a directed edge from \( c \) to \( d \) (i.e., \( c \leq d \)) is added if there is an undirected path between \( c \) and some of the ground-touching nodes.

Locally it is difficult to distinguish between “support from below” and “support from above”. Since the input model is placed on the ground, “support from below” was initially used. Now we perform region growing starting from the ground-touching nodes \( V_g \) to iteratively rectify “support from above”. Specifically, we first identify all the nodes \( V_{c, r} \) to each of them there is a directed path from a node in \( V_g \). Let \( V_r \) denote the rest of the nodes. We change the relation from “support from below” to “support from above” and correspondingly reverse the direction for such edges between \( V_r \) and \( V_{c, r} \) and among \( V_r \). Then in \( V_r \) we label nodes which touch \( V_{c, r} \) as \( V_g \) and repeat the above steps until all the nodes are visited. We shown an illustration example with 8 object parts and rectify the “support from below”/“support from above” relations after two times rectification in Figure 6.

Support substructure. As discussed previously, a support substructure is essentially a subgraph of the support relation graph \( G \). For each group of similar parts \( A \), connected to a single part, we first reduce the graph \( G \) by contracting the nodes corresponding to \( A \) and replacing them with a single node, followed by necessary updates on the set of edge connectivity (Figure 7 (top)). This reflects the requirement that no subset of \( A \) is allowed to form a substructure with others. This process is repeated until each level of hierarchial grouping of similar parts leads to a single node in the graph.

Like the previous works (e.g., [5], [3]), we determine static stability via geometric validation. Specifically, given a group of elements \( B \) with multiple supports from another group of elements \( A \), we say \( B \) is stably supported by \( A \) if the projection of \( B \)’s center of mass to the ground falls inside the projection of the convex hull of the multi-supporting areas to the ground. In \( G \) we also contract a pair of nodes \( a \) and \( b \) if \( b \) is supported by \( a \) only but the support is unstable.

The static stability analysis often does not work for “support from side”, where stable support is typically achieved by other means like nail joints. Since stability provides a strong cue for the extraction of meaningful substructures by analyzing the geometry alone, we first search for support substructures among the parts connected by “support from below” and/or “support from above”. To achieve this, we temporarily break the edges with “support from side” in \( G \), leading to a set of weakly connected subgraphs \( \{ G_i \} \).

We take a bottom-up approach to search for all basic support substructures in each subgraph \( G_i \). First, we determine the support order starting from the ground-touching nodes, using a level order traversal approach similar to [6]. The order prediction is performed on a transitive reduction of a copy of \( G_i \). Otherwise the predicted order would be undesired in case of support by multiple hierarchy. Second, from the lowest order, for each node \( a \) we search for a basic support substructure that contains \( a \). Specifically, let \( b \) be the node directly supported by \( a \). If \( b \) is supported by multiple nodes including \( a \), we check the stability of \( b \) against the set of multiple nodes. If they are stable, \( b \) and the set of multiple supporting nodes form a support substructure. The above steps are repeated until all the nodes are visited.

To get the basic support substructures purely formed by “support from side”, we break the edges in \( G \) with “support from below” and “support from above”. We then use a similar bottom-up approach but without stability validation to identify those support substructures in the resulting subgraphs.

Finally we combine the basic support substructures (Section 3.1), which possibly involve different support types, to form new substructures in multiple hierarchy (Figure 7). The more rounds we combine, the more complicated substructures we get. In general, the number of rounds for substructure combination is application dependent. However, we found 1-2 rounds are already sufficient for our applications to synthesize many nontrivial shape variations.

4 Applications

In this section we introduce three applications, where support substructures play a major role and make the
### 4.1 Shape Reshuffling

This application is motivated by the recent work of Zheng et al. [3], which creates new in-class or across-class shape variations by reshuffling compatible symmetry-induced substructures, called SFARR-s. We will show that our support-induced substructure representation is able to create many more variations, which received higher scores by the user study participants than those by SFARR.

Given \( n \) \((n \geq 2)\) input objects, which are possibly from different model families, with pre-computed segmentation but not necessarily having explicit part correspondence, we first detect a set of support substructures \( S_i = \{s_1^i, s_2^i, ..., s_{k_i}^i\} \) for each object (Section 3.2). Let \( S = \{S_1, S_2, ..., S_n\} \). We then cluster all support substructures in \( S \) by the type of support, leading to 3 clusters, corresponding to 3 support types. In each cluster, we sort the support substructures by their bounding box size. For a pair of support substructures \( s_i \) and \( s_j \) in each cluster, we measure structure compatibility \( \gamma(s_i, s_j) \) based on the difference in terms of scale, contact, and context. Specifically,

\[
\gamma(s_i, s_j) = \lambda \cdot \theta \cdot \beta, 
\]

where \( \lambda \) measures the scale compatibility between the corresponding supporting and supported components in \( s_i \) and \( s_j \) at the bounding box level (see a similar definition in [3]). Let \( \theta_i \) denote the volume of the convex hull formed by the contact slots (Figure 8 (a)) between \( SS_i^{ring} \) and \( SS_i^{cd} \). We then have \( \theta = g(\theta_i, \theta_j) \) to compare the difference of component contact between \( s_i \) and \( s_j \), where \( g(x,y) = 1 + |x - y|/|x + y| \). Finally, \( \beta \) measures context compatibility, i.e., the difference of the context of \( s_i \) and \( s_j \) in their original models. Specifically, let \( f_{SS}^{T_i} \) denote the number of supporting relations of type \( T_i \) between \( SS_i^{cd} \) and the other object parts which are not in the substructure but connected to \( SS_i^{cd} \). For example, the supported component (in red) in Figure 8 (b) has \( f_{SS}^{T_2} = 2 \), with \( T_2 \) = “support from below”. Note we count 1 for each group of similar parts (e.g., the two chair arms). \( \beta \) is then computed as \( \prod_{k=1}^{6} g(D_i^k, D_j^k) \), where

![Diagram](image-url)

Figure 6. An illustration example with 8 parts to show how we rectify the "support from below" and "support from above" relations. (a) we temporarily assign "support from below" relations, (b) the first rectification, (c) after reverse the direction, we perform the second rectification, (d) the final result after two times rectification.

![Diagram](image-url)

Figure 7. Support substructure detection starts with basic support substructures \( (1, 2, 3, 4) \). Substructures with support in hierarchy are obtained by combining basic substructures: after 1st round \( \rightarrow 5 \) and 7; after 2nd round \( \rightarrow 6 \) and 8. The combination order shows the hierarchy attribution of support substructures.
Figure 8. Illustration for contact slots (dots in orange) and context compatibility.

\[ D_i = (f_{SS_i^1}, f_{SS_i^2}, f_{SS_i^3}, f_{SS_i^4}, f_{SS_i^{10}}, f_{SS_i^{11}}, f_{SS_i^{12}}, f_{SS_i^{13}}) \] is a 6D context compatibility descriptor.

Figure 9 (a) and (b) show four variations by performing reshuffling twice. It is shown that when the structure compatibility \( \gamma \) is of small values, the corresponding reshuffling results are generally of high quality. In contrast, high structure compatibility costs often lead to unpleasing results, as shown in Figure 9 (c) and (d). We thus perform reshuffling greedily, starting from a pair of support substructures (from different input objects) with the minimum value of \( \gamma(s_i, s_j) \). In this way, many functionally implausible results can be effectively avoided.

**User Study.** We conducted a user study to evaluate the quality of new shape variations achieved by reshuffling. We applied our reshuffling technique to the main inputs tested by [3]. There were in total 5 different sets of input models (see Figures 1, 12 and 14 of [3], or the thumbnails in Figure 10 (left)), varying from 3 to 6 models. For each set of input models, we automatically synthesized 100 models, each of which came with an increasing value of \( \gamma(s_i, s_j) \). Please refer to our supplemental materials for the detailed reshuffling results. The 500 results by our technique, together with the unique results by SFARR (more details later), were presented in a random order to in total 60 participants (all of them were university students), who were asked to rate every synthesized shape on a discrete scale from 1 (worst) to 5 (best). They were suggested to give a score for each model based on their own answers to the following questions: “are they coherent with your understanding of man-made objects?” and “how likely a similar object would appear in reality?”.

To check whether a reshuffling result with a lower structure compatibility value would receive a higher score by the participants, we calculated the average score given the participants for the first \( k \) synthesized models (with increasing values of \( \gamma(s_i, s_j) \), \( 1 \leq k \leq 100 \), in each set of models. As seen from Figure 10 (left), the average score for each set overall decreased with \( k \), the number of synthesized models, indicating an effective quality control by \( \gamma(s_i, s_j) \). This figure also suggested that the top 40 results were often reasonably good (with average score \( \geq 3.5 \)) and the top 80 results were still acceptable (with average score \( \geq 3.0 \)).

**Comparison with SFARR.** When a triplet of parts with the special arrangements required by SFARR appears in the object, it will also be detected by our algorithm as a support substructure. SFARR is thus only a special support substructure. Our support substructure representation is much more general and exists beyond symmetric arrangements. It supports multiple hierarchy and does not restrict the number of elements. Theoretically our technique is able to reproduce all the results shown in [3]. However, due to the adoption of different compatibility metrics, 39 out of their 106 results (Figures 1, 12 and 14 of [3]) were not in the set of the 500 results by our technique. The quality of these 39 results was rated by our participants during the user study.

Our technique produced many more reshuffling results. It is more important to verify whether our results are comparable to or even better than those by SFARR. To this end, from the top 80 results in each set we randomly sampled the same number of results as the corresponding set in [3], and calculated the average scores for the sampled results and the results by SFARR, respectively. Each of their result sets contained 11 to 35 models. For fairer comparisons, the above process was repeated for 10 times for each set of input models. It was found that our results were rated consistently and significantly higher than those by SFARR, as confirmed by t-tests (\( p \)-value < 0.05 in all cases). This is possibly because the unique results by SFARR were rated relatively poorly, as shown in Figure 10 (f). For each set of input models, Figure 10 (a–e) shows a box-and-whisker plot of the scores of the 10 sets of our randomly picked results and the corresponding sets of results by SFARR.

Figure 11 shows around top-30 results by applying our reshuffling technique to new sets of input models. Many of the results (e.g., those highlighted in yellow) would be difficult to produce by SFARR due to the lack of necessary symmetry parts in the input objects. However, we also admit that the flexibility of our representation is at the cost of slightly more complicated operations towards functionally plausible reshuffling results.

**4.2 Structure Rearrangement**

Our support substructures give a structural decomposition of an input object. Each support substructure is structurally valid and thus can be used as a whole for shape editing. Based on this observation, we introduce a shape rearrangement application, which automatically rearranges substructures to create nontrivial variations from a single input model. This application is more like an in-model reshuffling. For simplicity, we perform rearrangement operations in 2D (ground plane) only.
Figure 9. Shape reshuffling enabled by support substructures (a), with structure compatibility $\gamma(S_1, S_2) = 1.105$. Repeated reshuffling of substructures from intermediate results lead to further variations (b), with structure compatibility $\gamma(S_1, S_2) = 1.214$. Note that such reshuffling results are difficult to synthesize with sFARR due to the lack of proper triplets of shape parts between models. (c) and (d) show two examples rejected by our technique due to their high structure compatibility values ($\gamma(S_1, S_2) = 5.672$ and $4.824$ for (c) and (d), respectively).

Again let $S = \{s_1, s_2, ..., s_n\}$ denote a set of support substructures for an input model, which is often a scene model for creating more variations. We cluster the substructures by support type, leading to 3 clusters. We then align the support substructures in each cluster by registering their supporting components $SS_i^{s_{i}}$ via ICP. The substructures with small alignment errors are grouped together. Denote the resulting groups as $G = \{g_1, g_2, ..., g_m\}$. To create interesting rearrangement results we drive rearrangement mainly by a principal group $\hat{g} \in G$, which has the biggest bounding box size. For example the support substructures 2, 5, and 7 in Figure 1 form such a principal group.

Every pair of substructures in $\hat{g}$ have their supporting components well aligned. We thus can switch between two substructures in $\hat{g}$ for structure rearrangement, without destroying the contact conditions in the 2D plane. Specifically structure rearrangement is achieved by a two-step approach, as illustrated in Figure 12:

1. Iteratively switch between pairs of support substructures in $\hat{g}$. We randomly select an unvisited pair $s_i, s_j \in \hat{g}$ and then replace with each other. After $s_i$ is replaced with $s_j$, $s_j$ might need rotated to avoid severely intersecting with the existing substructures originally adjacent to $s_i$. We use mesh collision detection to check the availability of severe intersections, specifically by checking whether the ratio between the intersection part and the original model is below a threshold $\varepsilon$ (Section 5) or not. The height of the existing substructures which are originally connected with $SS_i^{s_{i}}$ is adjusted to get well connected to $s_j$. This process is repeated till no unvisited pair is found or the number of iterations exceeds a user-specified number (e.g., 20).

2. Iteratively relocate support substructures from non-principal groups. For each support substructure $s_{k_i}$ in a non-principal group $g_{k_i}$, we examine whether

Figure 10. Left: the average score rated by the participants overall decreased with the number of reshuffling results, indicating the effectiveness of our structure compatibility metric. (a)–(e): box-and-whisker plots of the scores of our results and sFARR’s results for each set of input models. In each plot, the first 10 boxes are for 10 sets of our randomly-picked results and the 11th for sFARR’s results. (f): the user-rated score of each of the unique models by sFARR, please refer to the supplemental materials for the detailed results.
Figure 11. Four sets of shape reshuffling results by our technique. Note that each set contains around top-30 results, without cherry picking. The input models are those with colored parts. The results highlighted in red are less visually appealing, and those highlighted in yellow are difficult to synthesize by SFANN.
it is possible to relocate \( s_k \) to a new position. We first find a support substructure \( s_i \) from another non-principal group \( g_i \), which shares the largest number of supporting components with \( s_k \). We can then relocate \( s_k \) to get connected to \( s_j \), since \( s_j \) and \( s_i \) are well aligned and thus \( s_j \) is very likely to well support the relocated version of \( s_k \). The new location of \( s_k \) is determined by first aligning \( s_k \) with \( s_j \) using their shared components and then transforming \( s_k \) by the optimal transformation between \( s_k \) and \( s_j \). \( s_k \) might need rotated to avoid intersecting with the existing substructures. The relocation step is repeated till we reach a user-specified number (e.g., 20).

**Results.** Different from the application of shape reshuffling, which needs a relatively large set of substructures to create many variations, structure rearrangement is already able to create many new functionally plausible shapes with a relatively small number of support substructures. Hence for this application we only combine basic substructures for one round (Section 3.2). Figure 1 shows the rearrangement results after three iterations. Please refer to Figure 13 and the supplemental materials for more arrangement results.

### 4.3 Structure Synthesis

The application of shape rearrangement essentially changes only the locations of support substructures. Now we show another application which turns a single input model to new nontrivial variations by duplicating substructures and connecting them together, in a spirit of procedural modeling. We follow the notations introduced in Section 4.2. We will first present the main idea using a 2D example and then discuss its extension to 3D synthesis.

**Structure Synthesis.** As illustrated in Figure 14, it is operated among the substructures in the principal group \( \hat{g} \). Each time a pair of support substructures \( s_i, s_j \in \hat{g} \) are randomly selected. If there exists a 2D transformation \( T_{ij} \) which can link \( s_i \) with \( s_j \), and the transformed \( s_i \) does not seriously intersect with the other support substructures, \( s_i \) is then copied and transformed to link with \( s_j \). \( T_{ij}(s_i) \) is added to \( \mathcal{G} \) (Figure 14 (right column)). This process is repeated until there exist no such a pair of substructures or it reaches the prescribed number of iterations. Below we give the details on the definition of \( T_{ij} \).

**Definition of \( T_{ij} \).** To define \( T_{ij} \) we first introduce a contact descriptor between \( s_i \in \hat{g} \) and any other substructure \( s_j \in S \) (Figure 14 (left column)). Specifically the contact information between \( s_i \) and \( s_j \) is analyzed in three aspects: (1) a set of sharing parts between the supporting components of \( s_i \) and \( s_j \), i.e., \( A_{ij} = S_{i_s} \cap S_{j_s} \); (2) the inner direction \( d_{in}^{ij} \) from the centroid of \( A_{ij} \) to the centroid of \( S_{i_s} \); (3) the outer direction \( d_{out}^{ij} \) from the centroid of \( A_{ij} \) to the centroid of \( S_{j_s} \). This leads to a contact descriptor \( D_{ij} = \{ A_{ij}, d_{in}^{ij}, d_{out}^{ij} \} \) for \( s_i \) and \( s_j \). It is easy to see that \( A_{ij} = A_{ji}, d_{in}^{ij} = d_{out}^{ij} \). Note that the directions \( d_{in}^{ij} \) and \( d_{out}^{ij} \) are both 2D vectors. Finally we get a contact descriptor set \( D_i = \{ D_{ik}|A_{ik} \neq \emptyset \} \) for each \( s_i \in \hat{g} \).

To encourage forming more links between a pair of substructures in the principal group and thus creating more variations, we try to add more potential contact information from each of other substructures in \( \hat{g} \), denoted as \( s_i \) to \( s_j \). Specifically, we compute a 2D transformation \( T^{li} : s_i \rightarrow s_j \) that best aligns \( S_{i_s} \) to \( S_{j_s} \). The contact descriptor \( D_i \) of \( s_i \) is transformed by \( T^{li} \) as \( T^{li}(D_i) = \{ T^{li}(A_{ik}), T^{li}(d_{in}^{ik}), T^{li}(d_{out}^{ik}) \} \). \( s_i \)'s contact descriptor set is then updated as \( D_i = D_i \cup T^{li}(D_i) \).

Now for each \( s_i \in \hat{g} \) we have \( D_i = \{ D_{ik}|A_{ik} = (A_{ik}, d_{in}^{ik}, d_{out}^{ik}) \} \). Given \( s_i, s_j \in \hat{g} \), they can get linked together if there exists a 2D transformation \( T \) such that (1) \( T(A_{ik}) = A_{jl} \), i.e., the sharing supporting components get well aligned; (2) \( T(d_{in}^{ik}) = d_{in}^{jl} \), the inner direction \( d_{in}^{ik} \) of \( s_i \) is aligned with the outer direction \( d_{out}^{jl} \) of \( s_j \); (3) \( T(d_{out}^{ik}) = d_{out}^{jl} \), the outer direction \( d_{out}^{ik} \) of \( s_i \) is consistent with the inner direction \( d_{in}^{jl} \) of \( s_j \). We also use mesh collision detection to check the availability of severe intersections. This simple rule forms the basis for our structure synthesis procedure.

**Results.** Figure 15 shows several results created by our structure synthesis enabled by support substructures. Our work bears some resemblance to inverse procedural modeling, in particular the work by Bokeloh et al. [21]. However unlike [21], which requires the detection of partial symmetry regions, our technique relies on support and stability. Both of the techniques are able to produce unique results. See one more example in Figure 1.

**Interactive synthesis.** We also present a simple interface to permit explicit user control over the design. The user is allowed to interactively specify a growing direction (arrow in red in Figure 17 (a)) out of possible growing directions (arrows in white in Figure 17 (b)). The user may also specify how many times substructure duplication should be performed. Figure 17 shows interactive structure synthesis in action and Figure 1 (e) gives another interactive modeling result. Please refer to the supplemental material to see more automatically and interactively synthesis results.

**Extension to 3D Synthesis.** We experimented a simple way to perform structure synthesis in 3D, based

![Figure 12. Illustration of two-step structure arrangement.](image-url)
on the following observation: given two support substructures $s_1$ and $s_2$, we can conclude that $s_2$ can stably support $s_1$ if $s_2^{ed}$ (the supported component of $s_2$) stably supports $s_1^{ing}$ (the supporting component of $s_1$). This motivates us to perform synthesis along the SUPPORT direction. Specifically, we first randomly select two support substructures $s_i$ and $s_j \in \hat{g}$. We then find a 2D rotation and 2D translation to align the projected centers of $s_i$ and $s_j$. Finally we translate the transformed $s_i$ along the SUPPORT direction until the slots of $s_i^{ing}$ well touch $s_j^{ed}$. This process is repeated until there exist no such a pair of substructures or it reaches the prescribed number of iterations. Figure 16 shows a synthesized result.

5 DISCUSSION
Parameters. In our experiments we always set $\delta = 0.05d$, where $d$ is the diagonal length of the input model’s bounding box, and intersection error $\varepsilon = 0.10$. The default values for the other parameters were already given in the previous text. Figure 18 illustrates that an improper value of $\varepsilon$ would lead to artifacts for structure rearrangement and synthesis. For example, the part highlighted in red (Figure 18 (b)) blocks the way to a sliding board. In Figure 18 (d), severe intersection is not removed due to the improper value of $\varepsilon$. We also analyze the roles of the three supporting types in the synthesized results. Specifically we collect the frequency of the three supporting types used in the results produced by our applications. We find that “supporting-from-below” (normalized frequency: 47.05%) and “support-from-side” (normalized frequency: 35.30%) are more popular than “supporting-from-above” (normalized frequency: 17.65%). This is mainly because most of our tested input models represent objects that are supposed to stably stand on some flat surface (e.g., ground, floor, table, etc.).

Limitations. First, similar as other recent high-level shape synthesis methods (e.g., [27], [28], [3], [4]) our method relies on pre-segmentation of good quality. Second, we assume that the direction of support relation graph can be geometrically determined, which is not always possible for example for structurally in-determined structures [11]. The properties (i.e.,...
Figure 15. Automatic structure synthesis enabled by support substructures. Please refer to Figure 13 for the input models.

Figure 16. After detecting the support substructures of the model, we select 3 support substructures (a). Support substructure 2 is copied and transformed in the SUPPORT direction (supported by support substructure 1) to get support substructure 4(b). Support substructure 3 is copied and transformed in the SUPPORT direction (supported by support substructure 4) to get support substructure 5(b).

reflexive, antisymmetric, transitive) of our adopted three types of support do not apply to all kinds of supporting relationships. Lastly, supporting relationships themselves might not be sufficient to capture semantic relationships between parts. Therefore, like many other shape understanding systems, our technique may produce interesting but functionally not very plausible results. See such reshuffling results in Figure 19.

Like most shape synthesis work, our rearrangement and synthesis technique is designed on shapes with enough self-symmetry in structures. That is to say, the shape need have enough amount of similar support substructures such to build a principal group with an appropriate size. If not, we can’t perform such technique leading to shape variations. Another limitation is that we need enough linkage information between structures to compute enough grow directions leading to shape variations.

**Time Complexity.** Since our technique is part-based, the running time for each algorithm is very fast. The most time consuming part is sub-structure detection. For a shape with n part, the complexity to build structure graph is $O(n^2)$ and the structure detection is $O(n)$. So the total sub-structure detection complexity is $O(n^2) + O(n)$. Once the structure detected, the shape reshuffling algorithm is fast. It takes a few minutes to obtain about 100 synthesis result shown in the cases of Figure 11. For rearrangement and synthesis, it is also very fast. As can be seen in the supplementary video, both rearrangement and synthesis can be real time.

6 Conclusion and Future Work

This work presented the concept of support substructures, a high-level structural representation of object parts based on support and stability, and defined them as special semilattices induced by the supporting relations as partial order over a set of object parts. Although our definition of support substructure is simple, it enables various applications, including shape reshuffling, structure rearrangement, and structure synthesis, as demonstrated in the paper. None of the previous works is able to handle all these
applications in a single framework. The current structure rearrangement and synthesis are operated in 2D only. Since our support substructures already encode vertical hierarchies, it would be interesting to extend these applications to the 3D domain. In the future we are also interested in refining or generalizing the definition of support substructure, aiming at more high-level shape editing applications. We also would like to carefully study the linkage of the synthesis structures and a more careful and systemic stability analysis, e.g., a more careful and systematic treatment of force flow and structural stability with more realistic physical assumptions, and non-trivial structural optimization based on reassembling and varying parts etc.

ACKNOWLEDGMENTS

This work was supported by the National Basic Research Project of China (Project Number 2011CB302203), the Natural Science Foundation of China (Project Number 61120106007), Research Grant of Beijing Higher Institution Engineering Research Center, and Tsinghua University Initiative Scientific Research Program. Hongbo Fu was partially supported by grants from the Research Grants Council of HKSAR, China (Project Nos. 113513, 11204014 and 11300615).

REFERENCES


Shi-Min Hu is currently a professor in the Department of Computer Science and Technology at Tsinghua University, Beijing. He received the PhD degree from Zhejiang University in 1996. His research interests include digital geometry processing, video processing, rendering, computer animation, and computer aided geometric design. He is associate Editor-in-Chief of The Visual Computer, and on the editorial boards of IEEE TVCG, Computer-Aided Design and Computer and Graphics.