

# STRUCTURENET: Hierarchical Graph Networks for 3D Shape Generation

ANONYMOUS AUTHOR(S)  
SUBMISSION ID: 0240

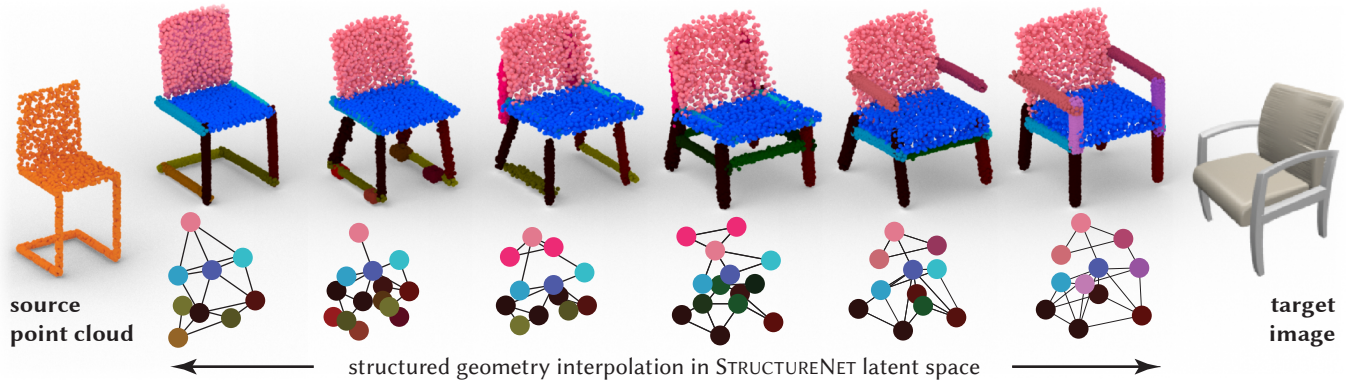


Fig. 1. STRUCTURENET is a hierarchical graph network that produces a unified latent space to encode structured models with both continuous geometric and discrete structural variations. In this example, we projected an un-annotated point cloud (left) and un-annotated image (right) into the learned latent space yielding semantically segmented point clouds structured as a hierarchy of graphs. The shape interpolation in the latent space also produces structured point clouds (top) including their corresponding graphs (bottom). Edges correspond to specific part relationships that are modeled by our approach. For simplicity, here we only show the graphs without the hierarchy. Note how the base of the chair morphs via functionally plausible intermediate configurations, or the chair back transitions from a plain back to a back with arm-rests.

The ability to generate novel, diverse, and realistic 3D shapes along with associated part semantics and structure is central to many applications requiring high-quality 3D assets or large volumes of realistic training data. A key challenge towards this goal is how to accommodate diverse shape variations, including both continuous deformations of parts as well as structural or discrete alterations which add to, remove from, or modify the shape constituents and compositional structure. Such object structure can typically be organized into a hierarchy of constituent object parts and relationships, represented as a hierarchy of  $n$ -ary graphs. We introduce STRUCTURENET, a hierarchical graph network which (i) can directly encode shapes represented as such  $n$ -ary graphs, (ii) can be robustly trained on large and complex shape families, and (iii) be used to generate a great diversity of realistic structured shape geometries. Technically, we accomplish this by drawing inspiration from recent advances in graph neural networks to propose an order-invariant encoding of  $n$ -ary graphs, considering jointly both part geometry and inter-part relations during network training. We extensively evaluate the quality of the learned latent spaces for various shape families and show significant advantages over baseline and competing methods. The learned latent spaces enable several structure-aware geometry processing applications, including shape generation and interpolation, shape editing, or shape structure discovery directly from un-annotated images, point clouds, or partial scans.

CCS Concepts: • **Computing methodologies** → **Spatial and physical reasoning**; **Hierarchical representations**; **Neural networks**; **Learning latent representations**; **Mesh geometry models**; **Shape analysis**.

Additional Key Words and Phrases: shape analysis and synthesis, graph neural networks, object structure, autoencoder, generative models

## ACM Reference Format:

Anonymous Author(s). 2019. STRUCTURENET: Hierarchical Graph Networks for 3D Shape Generation. 1, 1 (May 2019), 18 pages.

## 1 INTRODUCTION

A long-standing problem in shape analysis and synthesis is how to build generative models that support the creation of new, diverse, and realistic shapes. A key challenge is to accommodate diverse shape variations, including both continuous deformations of parts as well as structural or discrete alterations which add, remove, or modify the shape substructures present. We seek a continuous latent space that can incorporate all this diversity [Hinton 1990] and is able to encode, for example, chairs with or without armrests, chairs having four legs or swivel bases, as well as high or low backs, thin or thick legs, etc. Such a latent space, in turn, enables many non-trivial applications including generating shapes with both novel structure and geometry, discovering object structures from raw unannotated point clouds or images by ‘projecting’ them to the learned latent space, manipulating shapes in a structure-aware fashion, etc.

One path to this goal is to represent shapes as *structured objects* [Mitra et al. 2014] comprising of a collection of parts that are organized according to part-level connectivity and inter-part relationships. Further, these parts can naturally be organized into *hierarchies* [Wang et al. 2011a] encoded as  $n$ -ary trees, with objects

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
XXXX-XXXX/2019/5-ART  
<https://doi.org/>

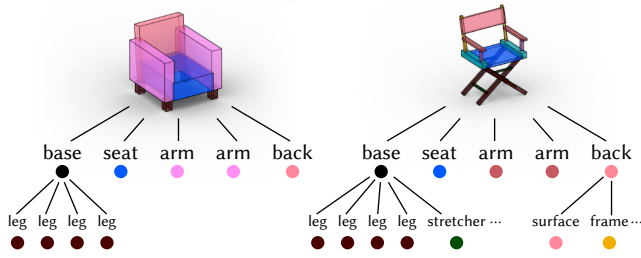


Fig. 2.  **$N$ -ary part hierarchies.** Shape parts can naturally be organized into  $n$ -ary hierarchies. Here we show the part hierarchies of two shapes as defined by PartNet [Mo et al. 2019]. The top row shows oriented bounding boxes of leaf parts and the hierarchy is illustrated below. Hierarchy nodes have the same color as the corresponding part. Note how geometrically dissimilar shapes may have consistent hierarchies. Our shape representation captures this consistency.

coming from the same shape family sharing similar hierarchies (see Figure 2). It is important to note, however, that many semantically significant relationships in the geometry of 3D shapes, such as symmetries, can connect distant nodes in the hierarchy, which may be spatially separated. These ‘horizontal’ relations pose additional constraints on shape encoders. In this paper, we refer to such hierarchical  $n$ -ary trees with horizontal connections as hierarchy of  $n$ -ary graphs, or simply hierarchy of graphs. Access to large volumes of 3D data (e.g., Turbosquid, ShapeNet, etc.) has now opened the possibility of learning class-specific latent spaces directly from data, aided by significant part annotation efforts within these databases. In the case of ShapeNet [Chang et al. 2015], both coarse-grained and fine-grained part annotations are available [Mo et al. 2019; Yi et al. 2016].

A notable work that creates such generative models is the GRASS framework [Li et al. 2017]. Inspired by recursive neural networks introduced in the context of natural language processing for encoding binary trees, GRASS further refines part-level object hierarchies into binary trees, and then recursively utilizes an encoder-decoder network [Socher et al. 2011] to build a latent space from which both a hierarchical structure and geometry at the leaves can be decoded. However, because of the binary constraint, GRASS has to additionally search over possible binarizations of the  $n$ -ary hierarchies found in objects, so that the binarized versions are consistent across objects in the same shape family. While this works nicely on small- to medium-sized datasets, the setup is difficult to train on large to very large shape families (e.g., PartNet [Mo et al. 2019]), as the task of finding a canonical binary tree representation becomes increasingly challenging (see Section 6).

We introduce STRUCTURENET, a hierarchical graph network that directly encodes more general graphs with parents having a variable number of children and horizontal relationships between siblings. STRUCTURENET relies on three main innovations: Firstly, by directly working with  $n$ -ary graphs for object structures, we have fundamentally avoided unnecessary data variation that is introduced with binarization, thus can significantly simplify the learning task. Secondly, we achieve invariance with respect to part-level sibling ordering at both encoding and decoding time, by using symmetric functions (e.g., max-pooling) during encoding, and solving for

a linear assignment problem to establish correspondences during decoding. Finally, we make use of horizontal inter-part *relationship edges* following a novel graph-based message passing protocol. These features enable us to robustly train STRUCTURENET on large to very large shape families so as to effectively capture both structural and geometric variations.

Building such a latent space for structured shapes has several advantages that can be exploited by various applications. First, the structure (i.e., part hierarchy and inter-part relationship) itself is useful for down-stream applications. Editors, for example, can edit, swap, or model parts individually, and make use of structural constraints such as symmetries. Second, structure is often more consistent inside a shape category than geometry. Chairs, for example, usually have a seat, a backrest and a base at a coarse hierarchy level, even if there is large variability in the geometry of these parts (see Figure 2). Finally, the ability to project raw unstructured shapes (e.g., images, point clouds, partial scans) onto such a latent space automatically induces structures on the raw input (i.e., provides a hierarchical part segmentation, capturing part-level contacts and/or symmetry relationships) and subsequently enables a diverse set of structure-aware manipulations.

A broad range of applications is enabled by our hierarchical graph networks, which can be grouped as follows: (i) *abstracting* raw inputs including point clouds, images, partial scans to obtain their structure; (ii) *creating* novel shapes in parameterized form or point cloud form based on a set of training shapes; (iii) allowing *structure-aware interpolation* between source and target shapes while displaying both topological and geometric variations; and finally, (iv) *structure-aware object manipulation* to smartly modify a part or replace a part. For example, in Figure 1, the input source point clouds and target image are first independently abstracted, and then directly interpolated in a latent space that was learned by STRUCTURENET<sup>1</sup> on 5K chairs from the PartNet dataset.

In summary, our contributions include:

- introducing an encoding for shape hierarchies of  $n$ -ary part graphs that is general enough to allow for a consistent hierarchical representations of shapes within a category;
- learning to encode and decode rich geometric relationships (e.g., adjacency, symmetry) between sibling parts, represented as edges in the hierarchical graphs, with graph neural networks to constrain realistic shape generation;
- developing a generative model STRUCTURENET that allows shape synthesis for both box-structures and point clouds with diverse and valid geometrical and structural variations; and
- illustrating the use of STRUCTURENET in both analysis and synthesis tasks, including shape reconstruction, novel shape generation, structure-aware shape interpolation, abstraction of un-annotated point clouds or images, and various shape editing applications.

## 2 RELATED WORK

Our work is primarily related to structure-aware shape representations (see [Mitra et al. 2014]) that go beyond local geometry and depict shapes through the arrangement and relations between shape

<sup>1</sup>Code and data will be released on acceptance.

parts, as well as to 3D shape generative models that aim to model the variations of 3D shapes and to synthesize novel 3D shapes. Finally, our work is inspired by recent developments in neural networks for graph structured data.

*Structure-Aware Shape Representations.* Understanding high-level shape structure such as parts and their relations is a central research topic in shape analysis. Most existing approaches focus on identifying shape parts [Golovinskiy and Funkhouser 2009; Hu et al. 2012; Huang et al. 2011; Kalogerakis et al. 2017, 2010; Makadia and Yumer 2014; Sidi et al. 2011; Xie et al. 2014; Yi et al. 2016], or part parameters and relations [Chaudhuri et al. 2011; Fish et al. 2014, 2016; Ganapathi-Subramanian et al. 2018; Kalogerakis et al. 2012; Kim et al. 2013; Müller et al. 2006; Sung et al. 2017; Yumer et al. 2015]. These approaches are usually restricted in the complexity and variety of part layout they can handle. To be able to process complex structures frequently appearing in the real world, several methods parse object parts in hierarchies [Mo et al. 2019; Van Kaick et al. 2013; Wang et al. 2011a; Yi et al. 2017a; Yu et al. 2019]. We also adapt a hierarchical structure representation for 3D shapes. However, we additionally augment the part hierarchy with ‘horizontal’ relations so as to encode shape structure into a more general  $n$ -ary graph. Our goal is to model and capture the distribution of such graphs in a shape collection. Structured 3D representations are also widely adopted for scene synthesis applications [Li et al. 2019; Liu et al. 2014; Zhao et al. 2016]. These approaches usually target generating a scene hierarchy and rely on object retrieval to complete the scene, while we focus on object-level structure synthesis with corresponding part-level geometric details.

*3D Deep Generative Models.* Recently, deep neural networks have been successfully leveraged to create generative models for 3D shapes. Wu et al. [2016] learn to generate 3D shapes in a volumetric representation through a deep belief network. [Goodfellow et al. 2014] also use volumetric representations for 3D shapes but capture the distribution of objects through a generative adversarial network (GAN). To improve the generation quality, researchers have not only explored novel architecture designs for volumetric representations [Choy et al. 2016; Gwak et al. 2017; Yan et al. 2016], but also studied various 3D representations, such as point clouds [Achlioptas et al. 2017; Fan et al. 2017; Li et al. 2018b], multi-view depth maps [Arsalan Soltani et al. 2017], oct-tree representations [Tatarchenko et al. 2017; Wang et al. 2018b], surface meshes [Groueix et al. 2018; Sinha et al. 2017], string-based shape synthesis [Kalojanov et al. 2019], etc. These approaches, however, focus on low-level geometry without considering the overall object structure in the generation process.

An alternate approach is to model structure along with geometry, which not only factorizes the complex distribution of 3D objects to facilitate learning but also makes the generation results more useful for downstream applications. Nash and Williams [2017] developed a variational auto-encoder (VAE) [Kingma and Welling 2013] to learn a latent representations for 3D objects, where they could synthesize new shapes in a part-by-part manner. However, they represent shapes as ordered vectors and require one-to-one dense correspondences among training shapes, which is not easy to obtain for shapes with large topological differences. Wang et al. [2018a] first learn to synthesize voxel-based shape structures with parts and

labels using a GAN on a set of segmented shapes, and then refine the geometry of each part through an auto-encoder. Wu et al. [2018] jointly learn and embed the geometry of parts and the pairwise relationship among parts using a VAE, where the geometry and structure features are intertwined in the encoder while disentangled in the decoder, to make the generation process structure-aware. The above approaches do not consider the hierarchical nature of object structures and simply focus on a flat arrangement of parts, making them less applicable to complex structures as defined by Mo et al. [2019]. Tian et al. [2019] generate structured 3D object through executing a series of 3D shape programs, but do not provide a way to sample and generate such shape programs freely.

Most relevant to ours is GRASS [Li et al. 2017], which learns a distribution of binary symmetry hierarchies of shapes. Novel shape structure can be sampled and generated from the distribution. However, the required binary symmetry hierarchy can introduce arbitrary ordering in nodes and make the hierarchy inconsistent across shapes. Instead, STRUCTURENET allows direct handling of  $n$ -ary graphs and further explicitly models relationship between parts in the graph. Hence it can be robustly trained on much larger datasets (see Figure 3 for a comparison).

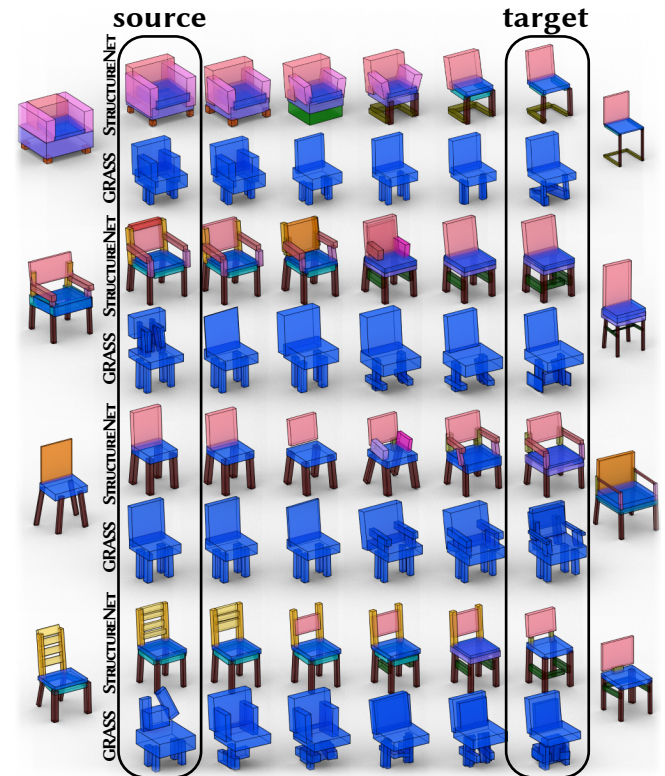


Fig. 3. **Interpolation compared to GRASS.** We compare interpolations between several pairs of chairs using STRUCTURENET (colored boxes), and using GRASS (blue boxes). We interpolate between shapes from our test set shown on the left-most and right-most sides, after being reconstructed by both methods (marked as ‘source’ and ‘target’). Our interpolations use a larger number of smaller structural changes to reach the target shape.



*Neural Networks for Graph Structured Data.* The shape structure we aim to generate, which is essentially the layout of parts and the relationship among them, can be represented as a hierarchy of  $n$ -ary graphs. A wide variety of works have explored how to design deep neural networks that can analyze graph structured data. To encode a tree structure, [Li et al. 2017; Socher et al. 2012, 2011] use recursive neural network (RvNN) to sequentially collapse edges of the graph. In graph convolutional networks [Bruna et al. 2013; Defferrard et al. 2016; Duvenaud et al. 2015; Hamilton et al. 2017; Kipf and Welling 2017; Veličković et al. 2017; Xu et al. 2019a], concepts from mature image CNNs are transferred to generic graphs. This approach has been widely applied in various shape processing methods where shapes are treated as mesh graphs [Boscaini et al. 2015; Masci et al. 2015; Wang et al. 2019; Yi et al. 2017b]. With a similar motivation, we design a novel graph encoding framework that combines RvNNs with graph convolutions to process a shape represented as a hierarchy of graphs.

Our framework is also related to various graph neural networks for synthesis purposes. There is a significant amount of work from the natural language processing and program synthesis communities on tree-like graph generation [Dyer et al. 2016; Maddison and Tarlow 2014; Socher et al. 2011; Vinyals et al. 2015], but most of these works are restricted to trees and not capable of handling more generic graphs. On the other hand, recent works like [Li et al. 2018a; You et al. 2018] aim at general graph generation and do not specialize their method to any domain. In comparison, our graph generation network is not restricted to trees, but is specialized to our shape representation that encodes both shape geometry and structure, and can thus fully leverage domain-specific knowledge.

### 3 OVERVIEW

We represent the hierarchy of shape parts as an  $n$ -ary tree, where each node is a part or part assembly. Geometric relationships of parts, such as symmetries and adjacencies, are captured by additional edges between siblings of the tree, forming a graph among siblings. See Figure 4 for an example. Each node contains information about the geometry of the part, capturing in total the geometry of the shape. In a given category, such as chairs, shapes tend to have consistent part hierarchies, as shown in the PartNet dataset [Mo et al. 2019]. Most chairs, for example, naturally decompose into backrest, seat, and base at the top hierarchy level. The  $n$ -ary tree in our shape representation can directly capture this hierarchy, giving us a shape representation with a high degree of consistency between shapes of a given category. Section 4 describes our shape representation in detail.

This shape representation, including structure and geometry, is mapped into a latent space with a Variational Autoencoder. We introduce *hierarchical graph networks* for the encoder and decoder, which are recursive networks [Socher et al. 2011] that perform graph convolutions [Kipf and Welling 2017; Xu et al. 2019a] at each recursion level. They are described in Section 5.

This gives us a rich latent representation of both the structure and the geometry of shapes in a category, that can be used in several applications. In Section 6 and the Supplementary, we demonstrate shape generation, interpolation, retrieval, editing, as well as the

discovery of structure from unannotated images, point clouds, or raw scans.

### 4 A HIERARCHY OF GRAPHS FOR SHAPE STRUCTURE

We introduce a shape representation that captures both the geometry and structure of a shape, and is suitable for processing by our hierarchical graph network. We assume that the shapes we work with can be decomposed into a meaningful set of parts. A *shape*  $S = (\mathbf{P}, \mathbf{H}, \mathbf{R})$  is then represented by a set of parts  $\mathbf{P} = \{P_1, \dots, P_N\}$ , describing the *geometry* of the shape, and a *structure*  $(\mathbf{H}, \mathbf{R})$  that describes how these parts are organized and related to each other. The structure consists of two superimposed graphs: a *hierarchical decomposition*  $\mathbf{H}$  of the shape into parts, and a set of *geometric relationships*  $\mathbf{R}$  among the parts. Figure 4 illustrates an example.

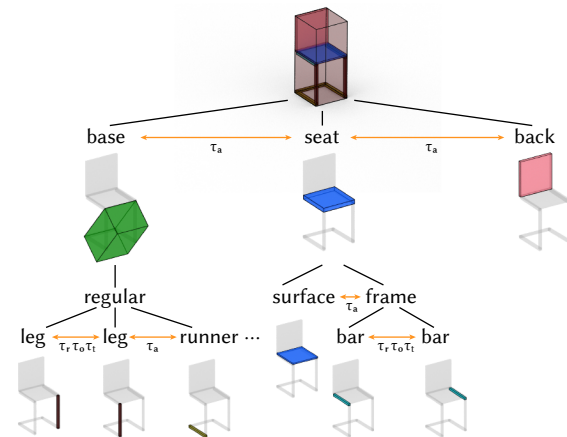


Fig. 4. **Shape Representation.** Shapes are represented by their hierarchical decomposition into parts (black edges), with geometric relationships between siblings (orange arrows): adjacency ( $\tau_a$ ), translational symmetry ( $\tau_t$ ), reflective symmetry ( $\tau_r$ ), and rotational symmetry ( $\tau_o$ ). A pair of parts may have multiple relationships of different types. The part geometry can be represented as point clouds or oriented bounding boxes. Here we show the latter, colored by semantic (see Supplementary for a full list of semantics).

*Part representation.* In our experiments, we support the use of two alternative representations for the geometry of a part  $P_i$ . Either, we represent a part's geometry with its minimum oriented bounding box  $B_i = (c_i, q_i, r_i)$ , where  $c \in \mathbb{R}^3$  are the world coordinates of the box center,  $q \in \mathbb{H}$  is the orientation of the box encoded as quaternion, and  $r \in \mathbb{R}^3$  is the size of the box; or, we represent a part by a corresponding point cloud  $A_i = \{x_1, \dots, x_k\}$ , where  $x \in \mathbb{R}^3$  are the world coordinates of a point. In addition to geometry, each part also has a semantic label  $l_i$ , such as back, seat, or base, that is consistent across shapes of the same category. We refer readers to the supplementary material for the details of the consistent semantic hierarchies we use.

*Hierarchical decomposition.* The hierarchical decomposition starts with the entire shape as root, which is split into a set of constituent parts, such as seat, back, and base for chairs. These parts are then recursively decomposed into their constituent parts, until reaching



the most fine-grained parts at the leaves. We represent this decomposition with a tree  $(\mathbf{P}, \mathbf{H})$ , where  $\mathbf{P}$  are the shape parts and  $\mathbf{H} \subset \mathbf{P}^2$  are directed edges from a parent part to all the children it is composed of. Note that a node may have *any* number of children, and the tree need not be balanced, that is, paths to leaves from a node can differ significantly in length.

*Geometric relationships.* In addition to vertical composition in the hierarchy, parts may also be related ‘horizontally’ by relationships such as *adjacency* and/or *symmetry*. These relationships can be crucial characteristics of shapes. Chairs, for example, often exhibit a reflective symmetry, as well as adjacency between several parts, and failing to take these relationships into account often results in the generation of unrealistic chairs. On the other hand, taking into account potential relationships between all pairs of parts in the hierarchy would require a number of relationships in the order of  $\Theta(N^2)$ , where  $N$  is the total number of parts, and auto-encoding such a large set of relationships accurately poses significant difficulties. In our experiments, we found, however, that encoding all relationships is not necessary, as the most important relationships occur between siblings of the hierarchy (e.g., legs of a chair, drawers in a cabinet, armrests of a couch) and relationships between other parts of the hierarchy are usually less significant or indirectly implied via a chain of relations following the hierarchy tree. Thus, we choose to only capture geometric relationships between siblings in the hierarchy, significantly sparsifying the relationship graph.

We represent these relationships with additional undirected edges  $\mathbf{R}_i$  between siblings  $\{P_j, P_k\}$  among the children  $C_i$  of a parent part  $P_i$ . We denote these edges as  $(\{P_j, P_k\}, \tau)$ . Each edge has an associated relationship type  $\tau$  from a list of possible relationship types  $\mathcal{T}$ . In our experiments, we use four relationship types: adjacency, reflective symmetry, rotational symmetry, and translational symmetry. These edges form a graph  $(C_i, \mathbf{R}_i)$  among siblings, and our hierarchy effectively becomes a *hierarchy of graphs*, where each shape part is expanded into a graph at the next lower level. We call each of these graphs an  $n$ -ary graph, to emphasize the  $n$ -ary nature of the hierarchy over these graphs.

## 5 HIERARCHICAL GRAPH NETWORKS

We propose a novel Variational Autoencoder (VAE) [Kingma and Welling 2013] for our shape representation that can be used for generation, interpolation, and several other applications we will demonstrate in Section 6. Our VAE consists of an encoder  $e$ , that maps a shape  $S$  to a latent feature vector  $z = e(S)$ , and a decoder  $d$ , that maps the feature vector back to a shape, so that approximately  $d(e(S)) \approx S$ . As described later, we measure the quality of this approximation based on both geometric and structural similarity. We introduce STRUCTURENET or *hierarchical graph networks* as a new network architecture for the encoder and decoder that can efficiently encode and decode both the geometry and the structure of our shape representation. Figure 5 shows the network architecture.

### 5.1 Encoder

The encoder maps a shape represented as a hierarchy of  $n$ -ary graphs to a latent feature vector  $z$ . We set the dimensionality of the feature space to 256, i.e.  $z \in \mathbb{R}^{256}$ . Each (leaf or intermediate) node  $i$  in the

tree  $(\mathbf{P}, \mathbf{H})$  is also mapped to a feature vector  $f_i \in \mathbb{R}^{256}$ . The code  $z$  for a complete shape is simply the feature vector describing the root node  $z = f_1$ . The encoder works recursively in a bottom-up manner using two types of encoders (See Fig. 5). First, we compute feature vectors for the leaf nodes using a *geometry encoder*. Then, we encode intermediate nodes using a *graph encoder*. The recursive nature of this approach is similar to previous work on structure encoding [Li et al. 2017, 2019], which in turn was inspired by natural language processing [Socher et al. 2011]. However, since we need to encode  $n$ -ary graphs, that may additionally have a variable number of parts, the architecture of both our encoder and especially our decoder are significantly different.

*Geometry encoder.* The geometry encoder  $f_i = e_{\text{geo}}(P_i)$  encodes the geometric representation of a leaf node  $P_i$  into a feature vector  $f_i$ . The geometric representation of a part can either be the bounding box  $B_i$  of the part or its point cloud  $A_i$ . We employ specialized geometry encoders for each of these representations. The bounding box encoder consists of a single layer perceptron (SLP). Since the part encoder is always followed by one or more applications of the graph encoder, a single layer is sufficient. Point clouds  $A_i$  are first centered at their mean and uniformly scaled to have a unit bounding sphere. The center and scale are then encoded using another SLP, while the normalized point cloud is encoded with a PointNet [Qi et al. 2017a]. Both of these outputs are merged and encoded by another SLP into the feature vector  $f_i$ .

*Graph encoder.* The child graph of a part  $P_i$  is given by  $(C_i, \mathbf{R}_i)$ , where  $C_i$  are the child parts and  $\mathbf{R}_i$  their relationship edges. Each part  $P_j \in C_i$  in the child graph is represented by the concatenation of its feature vector and label  $\hat{f}_j = (f_j, l_j)$ , while relationship edges in  $\mathbf{R}_i$  have as feature only their type  $\tau$ . Both the part label and edge type are encoded as one-hot vectors. Note that we do *not* store the relationship parameters, such as the axis of a rotational symmetry. The graph encoder  $f_i = e_{\text{graph}}(\{\hat{f}_j \mid P_j \in C_i\}, \mathbf{R}_i)$  encodes this child graph into a fixed-length feature vector  $f_i$ .

The architecture of the graph encoder is inspired by the recent Graph Isomorphism Networks (GIN) [Xu et al. 2019b] and Dynamic Graph CNNs [Wang et al. 2019]. To encode the child graph, we perform several iterations of message passing along the edges of the graph. In each iteration, a node aggregates features of its neighbors to compute an updated feature vector. Since we also have features for edges, we include them in each message that is passed over an edge. In each iteration  $t$ , a part’s feature vector is updated with,

$$f_j^{(t)} = \frac{1}{M} \sum_{(\{P_j, P_k\}, \tau) \in \mathbf{R}_i} h^{(t)}(f_j^{(t-1)}, f_k^{(t-1)}, \tau), \quad (1)$$

where  $M$  is the number of neighbors for  $P_j$  and  $h^{(t)}$  are SLPs, one for each iteration;  $h^{(t)}$  encodes the message that is passed over an edge, consisting of the source part’s feature vector  $f_k^{(t-1)}$ , the target part’s feature vector  $f_j^{(t-1)}$  and the edge type  $\tau$ . The messages from all neighboring parts are averaged to get the updated feature  $f_j^{(t)}$ . Iterations start with  $f_j^{(0)} = \hat{f}_j$  and we perform two iterations of message passing for each graph in our experiments. After message passing, the feature vector for the entire graph is computed by

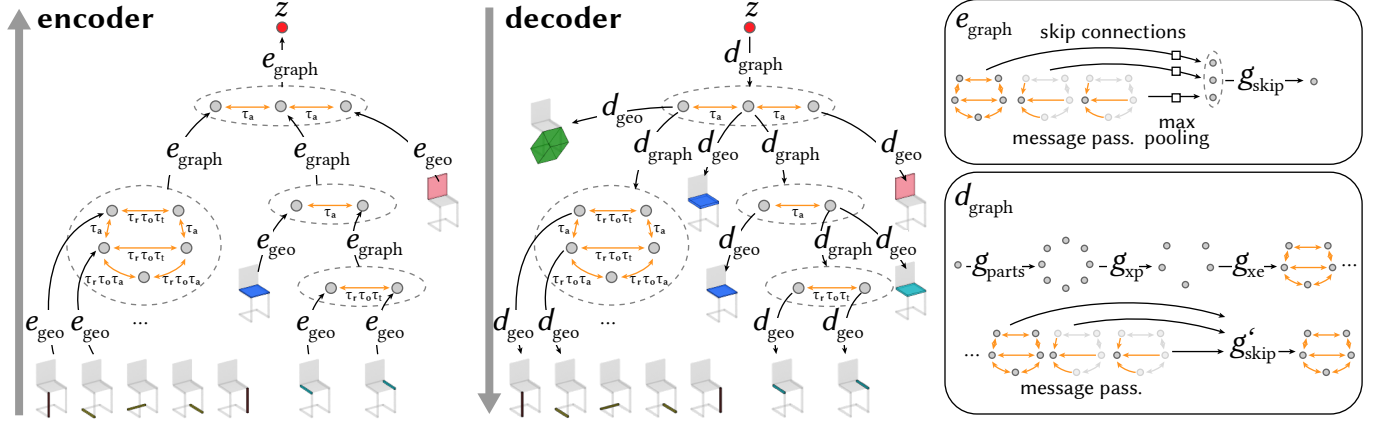


Fig. 5. **Hierarchical Graph Networks.** Our variational autoencoder consists of two encoders and two decoders that both operate on our shape representation. The geometry encoder  $e_{geo}$  encodes the geometry of a part into a fixed-length feature vector  $f$ , illustrated with a gray circle. The graph encoder  $e_{graph}$  encodes the feature vectors of each part in a graph, and the relationships among parts, into a feature vector of the same size using graph convolutions. The graph encoder is applied recursively to obtain a feature vector  $z$  that encodes the entire shape. The reverse process is performed by the graph and geometry decoders  $d_{graph}$  and  $d_{geo}$  to reconstruct the shape. The decoder also recovers the geometry of non-leaf nodes.

max-pooling over all child parts  $P_j \in C_i$ :

$$f_i^{(t)} = \max\{f_j^{(t)}\}. \quad (2)$$

Finally, we concatenate the graph feature vectors computed after each iteration, and pass them through another SLP  $g$ :

$$f_i = g_{skip}(f_i^{(0)}, f_i^{(1)}, f_i^{(2)}). \quad (3)$$

Note that this acts like skip connections for the iterations and allows the network to make use of features from all iterations.

## 5.2 Decoder

The decoder transforms the root feature vector  $z$  back into a shape represented as a hierarchy of graphs. It expands nodes in a top-down fashion. In each step, it first performs the reverse operation of the graph encoder, using the *graph decoder*  $d_{graph}$  to transform a latent code  $f_i$  into its child graph. The decoder then transforms the resulting feature vector of each child back into the geometry representation of the child with the *geometry decoder*  $d_{geo}$ . Unlike in the encoder, we decode the geometry of each part in the hierarchy, not only the leaf parts. This gives additional opportunity for supervision during training in the form of a reconstruction loss on the decoded intermediate geometry, as we will describe in Section 5.3.

*Geometry decoder.* We have two alternative decoders for the bounding box representation  $B_i = d_{geo}(f_i)$  and the point cloud representation  $A_i = d_{geo}(f_i)$  of a part. Both transform the feature vector of a part back to the part's geometry representation. The bounding box decoder is implemented as a multi-layer perceptron (MLP) with two layers, that transforms a feature vector  $f_i$  to a bounding box  $B_i = (c_i, q_i, r_i)$ . The point cloud decoder obtains a normalized point cloud from the feature vector with a three-layer MLP, and the center and scale of the point cloud using an SLP. We pre-train the geometry encoder and decoder for point clouds, as a separate autoencoder for the point cloud geometry of shape parts. This gives us

greatly increased training stability at the cost of a slightly decreased reconstruction accuracy.

*Graph decoder.* The graph decoder transforms a parent feature vector  $f_i$  back into the child graph  $(\{\hat{f}_j \mid P_j \in C_i\}, R_i) = d_{graph}(f_i)$ , where each child part  $P_j \in C_i$  is represented by a feature vector and its label  $\hat{f}_j = (f_j, l_j)$ . Since child graphs have a variable number of parts and edges, we always decode a fixed maximum number  $n_p$  of child parts and all  $n_p^2$  edges between them, together with a binary probability that a predicted part or edge exists in the child graph. Note that parts and their relations are simultaneously decoded. In our experiments, we use a maximum of 10 parts. Parts and edges that are predicted not to exist in the graph are discarded.

We start by decoding initial feature vectors from the parent feature vector using an SLP  $g_{parts}$ :

$$(\tilde{f}_1, \dots, \tilde{f}_{n_p}) = g_{parts}(f_i) \quad (4)$$

for the maximum number of child parts  $n_p$ . To predict the existence of parts, we compute

$$p_j = \sigma(g_{xp}(\tilde{f}_j)), \quad (5)$$

where  $p_j$  is the predicted probability that the child part  $j$  exists,  $\sigma$  is a sigmoid, and  $g_{xp}$  is a single linear layer. Parts with  $p_j < 0.5$  are discarded.

To predict the existence of edges, we can proceed similarly. Recall that the graph encoder accumulates information about the graph neighborhood in the feature of each part. Thus, we can recover the edges between a pair of parts based on their pair of feature vectors:

$$(p_{(j_1, j_2, \tau_1)}, \dots, p_{(j_1, j_2, \tau_{|\mathcal{T}|})}) = \sigma(g_{xe}(\tilde{f}_{j_1}, \tilde{f}_{j_2})), \quad (6)$$

where  $p_{(j_1, j_2, \tau)}$  is the predicted probability that an edge between child parts  $P_{j_1}$  and  $P_{j_2}$  of type  $\tau$  exists, and  $|\mathcal{T}|$  is the number of edge types. As  $g_{xe}$ , we use a two-layer MLP. Edges are discarded if any of the adjacent parts do not exist, or if  $p_{(j_1, j_2, \tau)} < 0.5$ .

We perform two iterations of message passing along the predicted edges, analogous to the message passing described in Section 5.1,

starting with the initial feature vectors  $\tilde{f}_j$  and resulting in the final child feature vectors  $f_j$ . Experimentally, we found the message passing to enable the parts to refine and coordinate their geometry based on the relationships described by the edges, like symmetry or adjacency. As a final step, we decode two additional values from the feature vectors  $f_j$ : the semantic labels  $l_j$  and a probability  $p_j^{\text{leaf}}$  that  $P_j$  is a leaf part. If a part is predicted to be a leaf part, we do not attempt to predict the existence of children for the node, making it easier for the network to stop the recursion. We found that this helps convergence especially in the early stages of training.

### 5.3 Training and Losses

We train our VAE on a dataset  $\mathcal{S}$  of shapes from a given category. We assume models in the dataset not to have parts with more than  $n_p = 10$  children. Each shape is represented as a hierarchy of graphs with known structure.

Our goal is to train the encoder and decoder of our VAE to perform a reversible mapping of each shape  $S$  to a feature vector  $z$  in a latent space where manipulations of the shape such as generation and interpolation are easier. To learn this mapping, we use a loss that is composed of three parts:

$$\mathcal{L}_{\text{total}} = \mathbb{E}_{S \sim \mathcal{S}} [\mathcal{L}_r(S) + \mathcal{L}_{sc}(S) + \beta \mathcal{L}_v(S)], \quad (7)$$

where  $\mathcal{S}$  is the distribution of shapes in a category and  $\mathbb{E}$  denotes the expected value. The *reconstruction loss*  $\mathcal{L}_r$  encourages reversibility of the mapping, the *structure consistency loss*  $\mathcal{L}_{sc}$  encourages consistency between the reconstructed parts and reconstructed relationship edges, and the traditional *variational regularization*  $\mathcal{L}_v$  of VAEs with regularization weight  $\beta$  that encourages the manifold of shapes in latent space to be smooth and simple, see [Kingma and Welling 2013] for a description. We empirically set  $\beta = 0.05$  for our experiments. We now define the reconstruction loss and the consistency loss.

*Reconstruction loss.* The VAE is encouraged to learn a reversible mapping by training it with a reconstruction loss:

$$\mathcal{L}_r(S) = q(S, d(e(S))), \quad (8)$$

where  $q$  is a distance metric between reconstructed shapes and ground truth shapes. The distance needs to be designed to provide good gradients to the encoder and decoder.

To compare two shapes  $S$  and  $S' = d(e(S))$ , we first need to establish a correspondence between parts in the two shapes. Here we need to choose between two strategies: we could either encode and reconstruct the order of parts in  $S$ , or use an order-invariant encoder and establish a correspondence by matching the structure and geometry of the reconstructed shape to the input shape. We choose the second option, as we empirically found the order-invariant network produces superior performance (similar conclusion was reached for point cloud encoding [Qi et al. 2017a]). We compute a linear assignment of the parts in the two shapes separately for each child graph. Starting at the root, the assignment of parent parts determines which child graphs are matched at the next lower level. This gives an assignment  $\mathbf{M} \subset \mathbf{P} \times \mathbf{P}'$  over all parts in the two shapes, where  $\mathbf{P}'$  are the reconstructed parts. To train part and edge existence predictions, we include the reconstructed parts that are

predicted not to exist in this assignment. Parts are matched based on their geometry representations. We define the geometry difference between parts with point cloud geometry as a squared version of the chamfer distance [Barrow et al. 1977] between the point clouds:

$$q_{\text{geo}}(P_i, P_j) = q_{\text{chs}}(A_i, A_j), \quad (9)$$

with the squared version of the chamfer distance [Fan et al. 2017] defined as:

$$q_{\text{chs}}(A_i, A_j) = \frac{1}{|A_i|} \sum_{x_i \in A_i} \min_{x_j \in A_j} \|x_i - x_j\|_2^2 + \frac{1}{|A_j|} \sum_{x_j \in A_j} \min_{x_i \in A_i} \|x_j - x_i\|_2^2. \quad (10)$$

For the bounding box representation, we cannot directly take the difference of the box parameters, since the orientation and scale of the box representation is ambiguous (e.g., a bounding box can be rotated by multiples of 90 degrees about any local axis and re-scaled to give the same bounding box). Instead, we take the chamfer distance between point samples on the boundaries of the two boxes:

$$q_{\text{geo}}(P_i, P_j) = q_{\text{chs}}(T(B_i)\mathbf{U}, T(B_j)\mathbf{U}), \quad (11)$$

where  $\mathbf{U}$  is a pre-computed set of samples on the unit cube, and  $T(B_i)$  is a 4D transformation matrix that transforms the unit cube to the part's bounding box  $B_i$ . Since non-uniformly scaling the unit cube with  $T(B_i)$  results in a non-uniform point density,  $q_{\text{chamfer}}$  weighs the transformed point samples with the area of the face they were sampled from [Tulsiani et al. 2017]. Based on the assignment  $\mathbf{M}$ , the distance  $q$  between two shapes is composed of five loss terms, as described next.

(i) *Geometry loss.* The *geometry loss* measures the distance between the geometry of two parts:

$$\mathcal{L}_{\text{geo}}(S, S') = \sum_{(P_i, P'_j) \in \mathbf{M}} q_{\text{geo}}(P_i, P'_j). \quad (12)$$

Additionally, the geometry of unmatched parts is trained to be all zeros to make the linear assignment more robust.

(ii) *Normal loss.* The geometry loss works well in general, but it is less sensitive to the orientation of small bounding boxes, especially if they have the same size along some of their dimensions, such as a rotation of thin rods about their longest axis. To make the reconstruction of part geometry represented as bounding boxes more sensitive to the orientation of the boxes, we add the *normal loss* that approximates the distance of the reconstructed box normals to the input box normals:

$$\mathcal{L}_{\text{normal}}(S, S') = \sum_{(P_i, P'_j) \in \mathbf{M}} q_{\text{chs}}(T(q_i)\mathbf{N}, T(q_j)\mathbf{N}), \quad (13)$$

where  $\mathbf{N}$  are the six unique normals of the unit cube. The transformation  $T(q_i)$  rotates these normals to the orientation  $q_i$  of part  $P_i$ . As in Eq. 11, we use the squared chamfer distance between the predicted and ground truth normals.



(iii) *Part existence loss.* The *part existence loss* measures how accurately the existence of parts is reconstructed:

$$\mathcal{L}_{xp}(S, S') = \sum_{P_j \in \mathbf{P}'} H(p_j, \mathbb{1}_{\mathbf{P}'_M}(P_j)), \quad (14)$$

where  $p_j$  is the predicted part existence probability defined in Eq. 5,  $\mathbb{1}$  is the indicator function,  $\mathbf{P}'$  are all parts in  $S'$ , and  $\mathbf{P}'_M$  is the subset that has a match in  $S$ . The cross entropy  $H$  encourages existence for parts that have a match, and non-existence for all other parts.

(iv) *Edge existence loss.* The *edge existence loss* measures how accurately the existence of edges is reconstructed:

$$\mathcal{L}_{xe}(S, S') = \sum_{((P'_{j_1}, P'_{j_2}), \tau) \in \mathbf{R}'} H(p_{(j_1, j_2, \tau)}, \mathbb{1}_{\mathbf{R}'_M}(\{(P'_{j_1}, P'_{j_2}), \tau\})), \quad (15)$$

where  $p_{(j_1, j_2, \tau)}$  is the predicted edge existence probability defined in Eq. 6,  $\mathbf{R}'$  are all edges in  $S'$ , and  $\mathbf{R}'_M$  the subset that has a match of the same type  $\tau$  in  $S$ . This loss encourages existence for edges that have a match, and non-existence otherwise.

(v) *Semantic loss.* The *semantic loss* is the cross entropy between the reconstructed label probabilities and the input labels, given as one-hot vectors:

$$\mathcal{L}_{sem}(S, S') = \sum_{(P_i, P'_j) \in \mathbf{M}} H(l_i, l_j), \quad (16)$$

where  $l_i$  and  $l_j$  are the input and reconstructed labels of the matched parts.

(vi) *Leaf loss.* Finally, the *leaf loss* measures the accuracy of the leaf prediction  $p_i^{\text{leaf}}$ :

$$\mathcal{L}_{leaf}(S, S') = \sum_{(P_i, P'_j) \in \mathbf{M}} H(p_i^{\text{leaf}}, \mathbb{1}_{\mathbf{P}'_{\text{leaf}}}(P_i)), \quad (17)$$

where  $\mathbf{P}'_{\text{leaf}}$  is the subset of parts in  $S$  that are leaves.

Finally, the distance  $q$  between two shapes is the sum of these five losses:

$$q(S, S') = \alpha \mathcal{L}_{\text{geo}} + \gamma \mathcal{L}_{\text{normal}} + \mathcal{L}_{xp} + \mathcal{L}_{xe} + \lambda \mathcal{L}_{sem} + \mathcal{L}_{leaf}. \quad (18)$$

Empirically, we set  $(\alpha, \gamma, \lambda) = (20, 10, 0.1)$  in all our experiments.

*Structure consistency loss.* Some types of errors in the reconstruction of parts are more severe than others. If a part is not in a geometric relationship with other parts, small reconstruction errors in the position, orientation or scale of the part are often less noticeable. However, if these errors break existing relationships, such as symmetry or adjacency relationships, even small errors can be much more apparent. Hence, we add a loss that encourages the reconstructed parts to be structurally consistent with the reconstructed geometric relationships of a shape – a self-consistency constraint between the part relations and the part geometries which is an important aspect of our loss design. This can be understood as a constraint violation loss, where the relationships act as constraints.

Given a relationship edge  $(P'_i, P'_j, \tau)$  of the reconstructed shape, we quantify how much the geometry of the parts  $P'_i$  and  $P'_j$  violates the relationship described by the edge. Additionally, a relationship between two parts should also hold for their subtrees. For example, a mirror symmetry between two parents should also constrain their

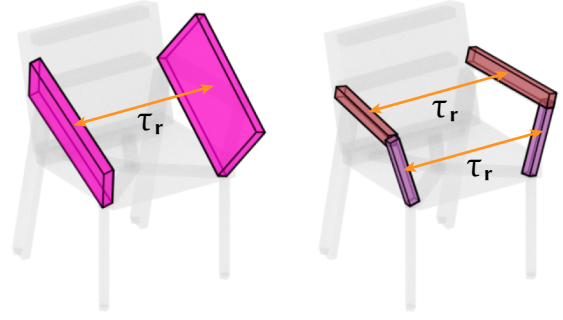


Fig. 6. **Relationships between subtrees.** A relationship between two non-leaf parts also holds for their subtrees. The reflective symmetry  $\tau_r$  of the parent parts on the left also holds for their children on the right.

two subtrees to be mirrored in the same way, see Figure 6 for an illustration. Hence, we also encourage the entire subtrees  $D_i$  and  $D_j$  of  $P'_i$  and  $P'_j$  to follow the same relationship. We first define the point cloud representation of a subtree, including the root, as:

$$D_i = \bigcup_{P_k \in D_i \cup \{P_i\}} T(B_k) \mathbf{U}. \quad (19)$$

As in Eq. 11,  $\mathbf{U}$  is a pre-computed set of samples on the unit cube, and  $T(B_k)$  is a 4D transformation matrix that transforms the unit cube to a part's bounding box  $B_k$ . When representing part geometries with point clouds, we directly use the union of the point clouds.

Next, we introduce a loss for *symmetries*, and a loss for *adjacencies*. For symmetries  $\mathbf{R}'_{\text{sym}}$ , we first compute the closest configuration of  $P'_j$  relative to  $P'_i$  that would not violate the relationship, and vice-versa for  $P'_i$  relative to  $P'_j$ . We use the distance from that configuration as loss:

$$\mathcal{L}_{\text{sym}}(S') = \sum_{((P'_i, P'_j), \tau) \in \mathbf{R}'_{\text{sym}}} q_{\text{chs}}(D_j, \rho_\tau(B_i, B_j)D_i), \quad (20)$$

where  $B_i$  and  $B_j$  are the bounding boxes of the two parts and  $\rho_\tau$  is a function that computes an affine transformation from  $B_i$  to the closest configuration of  $B_j$  that does not violate the relationship type  $\tau$ . When representing part geometries with point clouds, we first compute the oriented bounding box of the corresponding point clouds to obtain  $B_i$  and  $B_j$ , respectively.

For adjacency relations,  $\mathbf{R}'_{\text{adj}}$ , our loss is the minimum distance  $q_{\text{min}}$  between the geometry representations of the leaf parts only:

$$\mathcal{L}_{\text{adj}}(S') = \sum_{((P'_i, P'_j), \tau) \in \mathbf{R}'_{\text{adj}}} q_{\text{min}}(L_i, L_j), \quad (21)$$

where  $L_i$  and  $L_j$  are subsets of  $D_i$  and  $D_j$ , representing only the leaf parts of the subtrees.

Finally, the structure consistency loss is the sum of symmetry and adjacency losses:

$$\mathcal{L}_{sc}(S) = \mathcal{L}_{\text{sym}}(S') + \mathcal{L}_{\text{adj}}(S'), \quad (22)$$

where  $S' = d(e(S))$  is the reconstructed shape.

## 6 EXPERIMENTS

Accurately capturing structure in a smooth latent space gives us a simple and effective approach to many shape understanding and synthesis problems. In this section, we present quantitative and qualitative evaluations to demonstrate the effectiveness of our hierarchical graph networks on 6 tasks: shape reconstruction, generation, interpolation, abstraction, and editing. Additional results are available in the Supplementary.

*Data Preparation.* We use PartNet [Mo et al. 2019] as the main dataset for all the experiments in the paper. PartNet provides fine-grained and hierarchical part annotations with consistent semantic labels for 26,671 3D objects from 24 object categories. We use the three largest categories for our experiments: cabinets, chairs, and tables. In the Supplementary, we show results for three additional categories: vases, trashcans can and beds. Also in the Supplementary is a description of the semantic hierarchy in these categories. Since we have a maximum number of child parts per parent part  $n_p = 10$ , we remove shapes that have more than 10 children in any of their parts. Note that this maximum could also be increased if needed, slightly increasing memory consumption<sup>2</sup>. Additionally, we remove shapes that have unlabeled parts. The remaining 4871 chairs, 5099 tables, and 862 cabinets are divided into training, validation and test sets using the data splits published in the PartNet dataset, which have a ratio of 7 : 1 : 2.

In PartNet, shapes are represented as meshes that are divided into individual parts. Each shape in the dataset is scaled to be contained in the unit sphere. To obtain bounding boxes  $B_i$  for each part, we fit an oriented minimum-volume bounding box to the mesh of each part. Point clouds  $A_i$  are obtained by uniformly sampling the part’s surface with 1000 points. The part hierarchy  $H$  is given explicitly in the dataset. To define geometric relationships  $R$  between parts, we find symmetries using the method described by Wang et al. [Wang et al. 2011b], and define two parts as adjacent if their smallest distance is below  $0.05 * \bar{r}$ , where  $\bar{r}$  is the average bounding sphere radius of the two parts. On average, our shapes have 16.94 parts, arranged in a hierarchy of average depth 3.59, with an average number of 29.93 relationship edges. Each part has a semantic label chosen from a list of labels specific to each category. The number of different labels ranges from 36 for cabinets to 82 for tables.

### 6.1 Shape Reconstruction

As a first experiment, we measure the reconstruction performance of our hierarchical graph networks to find out how accurately our latent space can represent the shapes in the test set. To get an accurate reconstruction performance, just for the experiments in this section, we train a non-variational autoencoder version of our network. We use two groups of errors to measure reconstruction performance. Three reconstruction errors for the geometry, the hierarchy, and the relationship edges, and two structure consistency errors that measure the consistency of the reconstructed geometry with both the reconstructed and the input relationship edges.

The *geometry reconstruction error*  $E_P$  is defined analogously to the geometry loss  $\mathcal{L}_{\text{geo}}$ , except that we use the non-squared chamfer

<sup>2</sup>with our current settings, the memory consumption is  $\sim$  1GB for a batch size of 32.

Table 1. **Reconstruction performance on each shape category.** We compare the reconstruction performance of STRUCTURENET on six shape categories. See the supplementary for a qualitative evaluation of the categories bed, trashcan, and vase. The first three columns show the box geometry, hierarchy, and edge reconstruction errors, respectively. The consistency of the reconstructed shapes with the reconstructed relationship edges (recon) and the ground truth relationship edges (gt) is shown in the last two columns. Bed comes in last due to severe undersampling of the shape category, with only 54 training shapes. Our performance is best for chairs, which have a more balanced variety of training shapes than the other categories.

	reconstruction error			consistency error	
	$E_P$	$E_H$	$E_R$	$E_{rc}$	$E_{gc}$
Bed	0.069	0.609	0.518	0.019	0.032
Cabinet	0.066	0.461	0.386	0.021	0.027
Chair	0.062	0.200	0.246	0.018	0.023
Table	0.073	0.309	0.357	0.021	0.026
Trashcan	0.083	0.073	0.110	0.014	0.015
Vase	0.147	0.214	0.391	0.014	0.060

distance. Since our shapes are contained in the unit sphere, this gives us more easily interpretable distance values in  $[0, 2]$ .

The *hierarchy reconstruction error*  $E_H$  counts how many missing or unmatched parts are in the reconstructed hierarchy, using the assignment  $M$  between the input and reconstructed shapes described in Section 5.3:

$$E_H = \frac{1}{|P|} (|P \setminus P_M| + |P' \setminus P'_M|), \quad (23)$$

where  $P$  and  $P'$  are the sets of parts in the input and reconstructed shapes, respectively.  $P_M$  and  $P'_M$  denote the corresponding subsets of matched parts.

The *edge reconstruction error*  $E_R$  uses the assignment  $M$  to measure the precision and recall of the reconstructed edges, which we summarize in an error metric defined as one minus the  $F_1$  score:

$$E_R = 1 - \left( 2 \frac{e_p * e_r}{e_p + e_r} \right), \quad (24)$$

where the precision and recall are defined as  $e_p = |R'_M|/|R'|$  and  $e_r = |R'_M|/|R|$ , respectively.

The *reconstructed consistency error*  $E_{rc}$  measures the consistency of the reconstructed geometry with the reconstructed relationship edges. It is defined equivalent to  $\mathcal{L}_{sc}$ , except that we use the non-squared chamfer distance.

Similarly, the *ground truth consistency error*  $E_{gc}$  measures the consistency of the reconstructed geometry with the input relationship edges.

The reconstruction performance of STRUCTURENET on the three shape categories chair, table and cabinet is given in Table 1. In this experiment, we represent part geometry as oriented bounding boxes. The largest cause of reconstruction error we encountered when training our network is the unbalanced variety of shapes in each category. The datasets typically contain some sub-types of a category, such as square tables, more often than other sub-types, such as triangular tables. As a consequence, the network may have too few examples to learn a good representation of the more exotic

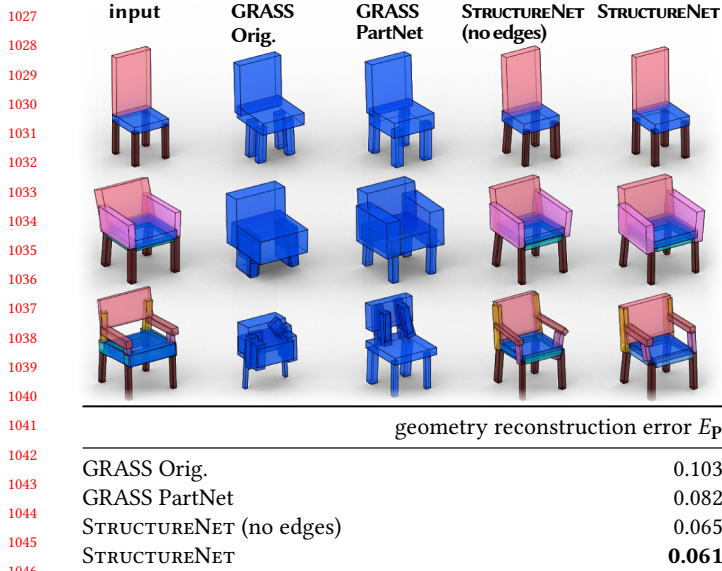


Fig. 7. **Reconstruction compared to GRASS.** In the top three rows, we show reconstructions of the left-most shape using the two variants of GRASS described in Section 6.1, and then reconstructions using a version of STRUCTURENET that does not use edges, and finally using our full method. Due to our more consistent shape representation, our method scales better to datasets with the size of PartNet. Using edges additionally improves part relationships such as the symmetries between the armrests, as seen in the third row. This is confirmed by the reconstruction error over the whole dataset, shown in the table below.

shape varieties. The chair dataset is the most balanced among the categories, giving us lower reconstruction errors than for the other categories. Additionally, the cabinet and bed datasets contain shapes with more complex structure on average, giving us higher hierarchy and edge reconstruction errors.

We compare our reconstruction performance to two baselines: GRASS [Li et al. 2017] as a state-of-the-art structure-aware shape generation method, and an autoencoder based on PointNet++ [Qi et al. 2017b], as a state-of-the-art point cloud autoencoder that holistically encodes the point cloud of a shape without using any explicit structure.

*Comparison to GRASS.* The comparison to GRASS is shown in Figure 7. A qualitative comparison is shown in the top three rows, and a quantitative comparison on the chair dataset in the table below. The oriented bounding box of each leaf part is illustrated as transparent box, colored according to its semantic (see the Supplementary for the full semantic tree of each category). GRASS results are colored uniformly, since semantics are not available. We measure only the geometry reconstruction error, since the structure of GRASS is not directly comparable to our shape representation. GRASS requires a binarization of shape hierarchies that are naturally  $n$ -ary. A binarization that is consistent between all shapes in a category is difficult to find, and this difficulty grows with the number and variety of shapes in a category. On our dataset, which has  $\sim 10$  times the size of the dataset originally used in GRASS, this task becomes too difficult.



Fig. 8. **Reconstruction compared to a holistic approach.** We reconstruct the point cloud representation of the shapes in the top row with a holistic approach (orange) that does not use structure, and compare to STRUCTURENET. The holistic approach suffers from noise that results in a loss of detail, whereas reconstruction errors in our approach take the form of slightly modified chair structures, such as the hole added to the backrest in the third column. However, the functional realism of the shape is usually preserved and the explicit structure allows us to preserve significantly more detail.

For this reason, the authors provided us with two modified version of their method, that each sacrifice some generality for a reduced number of possible binarizations. The first version, which we call *GRASS Original*, reduces generality and possible binarizations by a small amount, resulting in a method very similar to the original GRASS. Results for this method are shown in the second column of Figure 7 and the first row of the table. Due to the large number of possible binarizations, the performance is low. The second version, which we call *GRASS PartNet* uses the semantic hierarchy of PartNet to significantly reduce the number of possible binarizations. This increases the performance of GRASS, as shown in the third column of the figure and the second row of the table. Our approach, on the other hand, can encode and decode  $n$ -ary hierarchies directly, leading to a more consistent representation of the structure that gives us a significant improvement in reconstruction performance, as shown in column four and row 3 of the table. Relationship edges provide and additional boost to the reconstruction performance, by ensuring that symmetries that are present in the input are maintained in the reconstruction. Note that we evaluate this comparison on a reduced subset of 4031 chairs, since the GRASS authors reported that their pipeline failed to produce results for the remaining 840 chairs in our dataset.

*Comparison to a holistic autoencoder.* We train an autoencoder based on PointNet++ [Qi et al. 2017b] and PointSetGen [Fan et al. 2017] to compare the effects of encoding geometry only to our structure-aware latent space. PointNet++ is used as encoder, followed by a point cloud decoder network proposed in PointSetGen. We train both this autoencoder and STRUCTURENET on the chair dataset, with part geometry represented as point clouds. For the PointNet++ autoencoder, we merge the geometry of the leaf parts into a single point cloud for the shape. Results are shown in Figure 8



Table 2. **Shape generation compared to GRASS.** We compare the shape distribution learned by the two version of GRASS described in Section 6.1 to our method without edges and to our full method, using two metrics that measure how close the shapes are to the data distribution (quality) and how much of the data distribution is covered by the generated shapes (coverage). We report the scores relative to our method, higher numbers indicate better performance. Results show that our latent distribution better captures the data distribution.

	rel. quality	rel. coverage
GRASS Orig.	0.714	0.818
GRASS Partnet	0.788	0.818
STRUCTURENET (no edges)	0.984	0.989
STRUCTURENET	<b>1.0000</b>	<b>1.0000</b>

where points are visualized as small spheres, colored by their semantic in the same way as the bounding boxes in Figure 7. Note that the holistic results have significantly more noise, making it harder to recover details. Since we encode structure, errors in our approach instead take the form of slight modifications to the structure and layout of parts, such as added hole in the backrest of the chair in the third column, or the slightly modified arrangement of bars in the backrest of the chair in the last column. The structure, however, tends to remain realistic and since we represent the point cloud of each part separately, individual parts are sharper and details are better preserved.

## 6.2 Shape Generation

A straight-forward application of our hierarchical graph network is shape generation. In the remainder of Section 6, we use a VAE with the variational regularization weight  $\beta = 0.05$ . This gives us a dense and smooth distribution of shapes in latent space that we can draw from to generate new samples of shapes, including geometry and structure. We show both qualitative results and a quantitative comparison to GRASS for this application.

*Qualitative evaluation.* Several examples of generated shapes are presented in Figure 10, using both the bounding box representation and the point cloud representation for part geometry. Our results show a large variety in structure and part geometry, with a layout of individual parts that is functionally plausible. For each shape, we generate our full shape representation, including the geometry of individual parts, the hierarchical decomposition of these parts, symmetry and adjacency relationship edges between siblings, and part semantics. This rich high-level representation of the shapes is useful for several applications, some of which we will present in the following sections.

*Shape novelty and overfitting.* To evaluate the novelty of our generated shapes, we show the top-five closest training samples to several generated chairs in Figure 9, using the chamfer distance as metric. We can see that the generated shapes are quite different in both geometry and structure from the closest matches, suggesting little overfitting to the training set.

*Quantitative evaluation.* Quantitatively, the goal of this application is to cover as much of the data distribution as possible, while



Fig. 9. **Novelty of generated shapes.** In the first column we show three generated shapes, and on the right the five closest matches in the training set, measured with the chamfer distance. Our generated structure and geometry is different from the shapes in the training set.

at the same time, avoiding unrealistic chairs that are distant from the main mass of the data distribution. We quantify this goal with two metrics. The *quality*, of a generated shape set is measured by the average closest distance to any data sample, while the *coverage* is measured as the average closest distance from each data sample to a generated sample.

$$\begin{aligned} \text{quality} &:= \sum_{S' \in \mathcal{S}_G} \min_{S \in \mathcal{S}} d_S(S', S) \text{ and} \\ \text{coverage} &:= \sum_{S \in \mathcal{S}} \min_{S' \in \mathcal{S}_G} d_S(S', S), \end{aligned} \quad (25)$$

where  $\mathcal{S}$  is the training set,  $\mathcal{S}_G$  is a set of generated shapes, and  $d_S$  is the chamfer distance between the point representations of two shapes. To compare with GRASS using these metrics, we compute a set of 1000 shapes using both STRUCTURENET and GRASS, and compute their quality and coverage. We show results relative to the performance of STRUCTURENET in Table 2 (*i.e.* STRUCTURENET scores divided by the method scores). We see an improvement over GRASS in both quality and coverage of the generated shapes.

## 6.3 Shape Interpolation

We further examine the quality of our learned latent space with interpolations between shapes, visualizing samples along line segments in the space. We show several examples of interpolations in Figure 13. The left half of the figure shows interpolations between shapes with bounding box geometry, the right half between shapes with point cloud geometry. Note how the structure changes in small intuitive steps between the source and target of an interpolation. For example, in row 6 on the left side, the backrest of the chair is simplified part by part, while on the base, bars between the legs are added in multiple steps, and armrests are simplified to have fewer parts before disappearing. At the same time, each of the steps represents a valid and functional chair. We find it interesting to see in which way the network learned to arrange shape configurations in latent space, especially since these arrangements often seem to

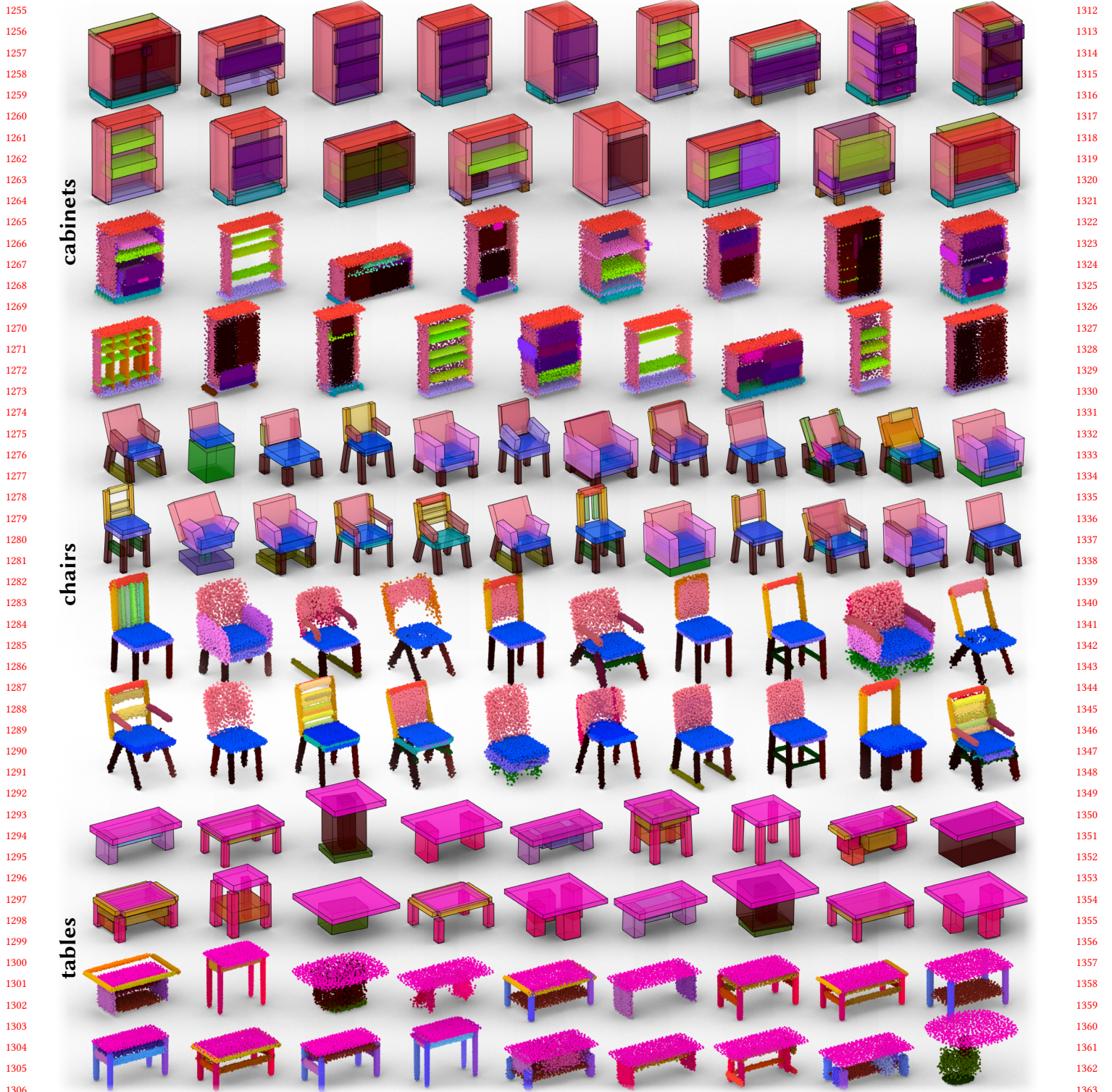


Fig. 10. **Generated shapes.** We show shapes in all categories decoded from random latent vectors, including shapes with bounding box geometry, and shapes with point cloud geometry. Parts are colored according to semantics, see the Supplementary for the full semantic hierarchy for each category. Since we explicitly encode shape structure in our latent representation, the generated shapes have a large variety of different structures.



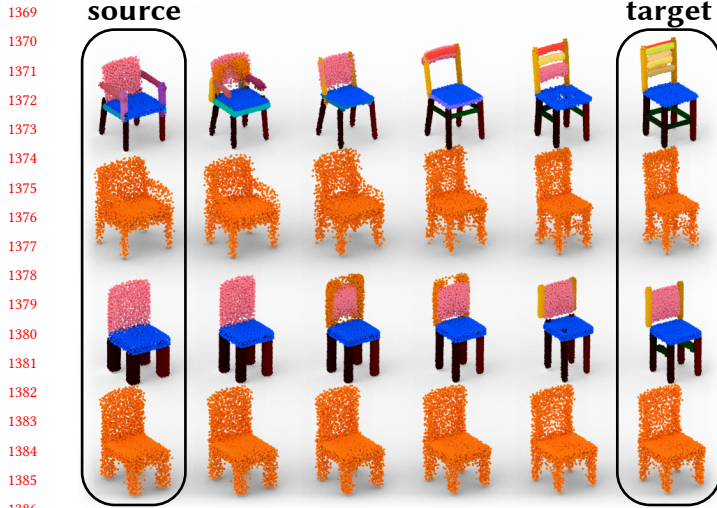


Fig. 11. **Interpolation compared to a holistic approach.** We compare our interpolation (colored) to a holistic approach (orange) that encodes shapes as point clouds without any structure. Explicitly encoding structure gives us a sequence of small structural changes in the intermediate steps, whereas the holistic approach produces no significant structural changes. Additionally, our per-part geometry is cleaner than the per-part geometry in the holistic approach, where it is hard to identify detailed parts.

correspond to our own intuition. For example, the pedestal base of the chair on the top right is first made smaller, before completely disappearing, and reappearing again as 4 separate legs that increase in size. Similarly, the transition from shelves with few boards to shelves with many boards near the bottom right of the figure, transitions by increasing or decreasing the number of boards step by step.

*Comparison to GRASS.* We provide a qualitative comparison of these interpolations to GRASS PartNet in Figure 3. First, we see issues with the reconstruction accuracy of grass, but looking at the interpolations only, we see that some interpolation of the structure, such as the reduction of the number of legs of the chairs happens much less gradual than in our interpolations, with fewer, larger steps.

*Comparison to holistic interpolation.* Figure 11 compares STRUCTURENET for point cloud geometry to interpolating without structure using the same holistic point cloud network described in Section 6.1, but using a VAE instead of an autoencoder. The VAE is necessary to obtain a smooth latent space suitable for interpolation, but makes the shapes much noisier than for the autoencoder, whereas our point clouds do not suffer as much from this switch to a VAE. Interpolations are smooth for the holistic VAE, but lack interesting transitions between structural details. Parts such as the armrests in the second row gradually disappear, whereas our armrests get replaced by simpler, but still functionally valid variants before being removed.

*Partial interpolation.* Since shape structures in a category are consistent, we can do interpolations between corresponding parts of a

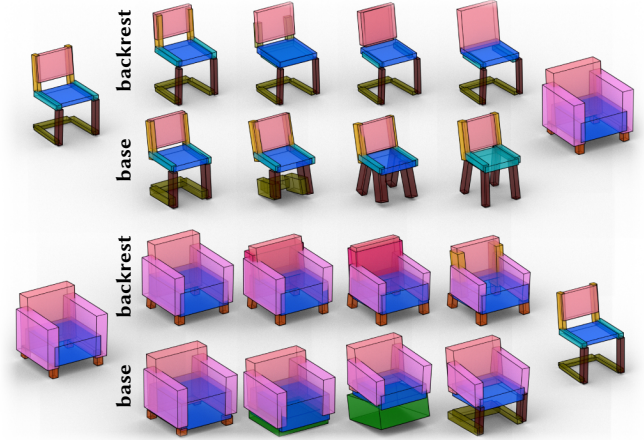


Fig. 12. **Part Interpolation.** We interpolate either only the backrest (first row) or only the base (second row) between the chairs on the left and right side. Intermediate shapes preserve structural plausibility of the interpolated result mainly through geometric differences to the target part, but faithfully interpolate the structure. We observe that these interpolations are not necessarily symmetric: the base interpolations follow different paths to be compatible with the different back styles.

shape. We perform the partial interpolation by taking the encoded part latent vector  $f_i$  of a shape, interpolating it with the corresponding part latent vector of another shape, and then re-encoding the shape with the interpolated part latent vector. This ensures plausibility of the resulting shape by effectively projecting the shape to the learned manifold of shapes. In Figure 12, we interpolate either only the backrest or only the base for each of two chairs. The structure of the interpolated part changes to resemble the target part, while the other parts of the shape remain largely unchanged. We can also see that the final step of the interpolation does not fully reach the target part, because the network ensures the plausibility of the resulting chair. For example, in the second interpolation, the short legs of the sofa would result in an implausible shape when attached to the chair. Thus, the interpolation depends on the context of the part. Even though the geometry is not the same, the *structure* of the chair bases is interpolated correctly and resembles the target structure at the final interpolation step.

## 6.4 Shape Abstraction

Discovering higher-level structure in un-annotated point clouds and images are long-standing vision and graphics problems. Our rich latent representation provides us with an approach to tackle these problems. We can encode images, point clouds, and shapes into a common latent space, using separately trained encoders for images and un-annotated point clouds. Given a trained STRUCTURENET autoencoder  $d(e(S))$  for the box representation of our shapes, we train additional encoders to take images  $I$  and un-annotated point clouds  $O$  to the same latent space. From the latent space, we can use our pre-trained decoder  $d$  to recover a shape  $S'$  that is similar to the shape represented with the input image or point cloud, but has all the information of our shape representation.



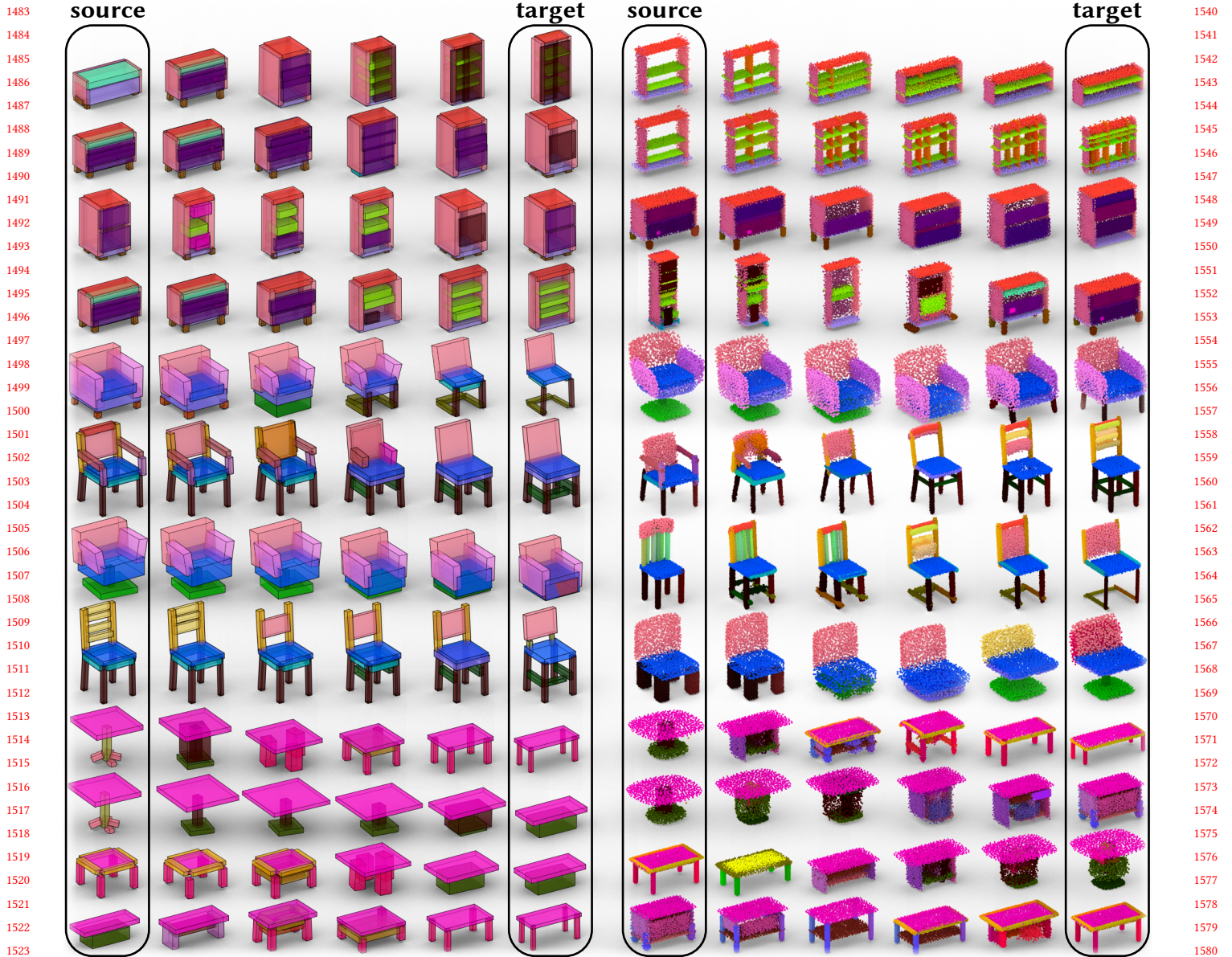


Fig. 13. **Interpolation of shapes.** We show interpolations between a source shape and a target shape from all categories. Interpolations are symmetric, so source and target are interchangeable. Interpolations between shapes with box geometry is shown on the left side, and point cloud geometry on the right side. Note how each interpolation is a smooth transition between two different structures that preserves functional plausibility in each step. In the interpolations between shelves with different numbers of boards, for example (bottom right), the number of boards is gradually increased/decreased in each step, and each step is a functional shelf.

*Image abstraction.* The image encoder  $e_I$  is a ResNet18 [He et al. 2016] that was pre-trained on ImageNet [Deng et al. 2009]. Inspired by the joint embedding approach of [Li et al. 2015], we refine this encoder on a dataset of images rendered from the shapes in our training set. We render the shapes with textures obtained from ShapeNet [Chang et al. 2015], from 24 random angles around their up vector, from a random elevation between 25 and 30 degrees, and a random distance between 1.2 and 2.0 times the

bounding sphere radius. For each image, we additionally have the corresponding latent vector in the latent space of the trained autoencoder  $d(e(S))$ . We train the image encoder to map each image to the latent representation of the shape it was rendered from  $\mathcal{L}_I = e(S) - e_I(\text{render}(S, \theta))$ , where  $\theta$  are the random camera parameters. We test on images generated from shapes in our validation and test sets, examples are shown in Figure 14, top. While the proportions of objects are not completely accurate, the overall shape,

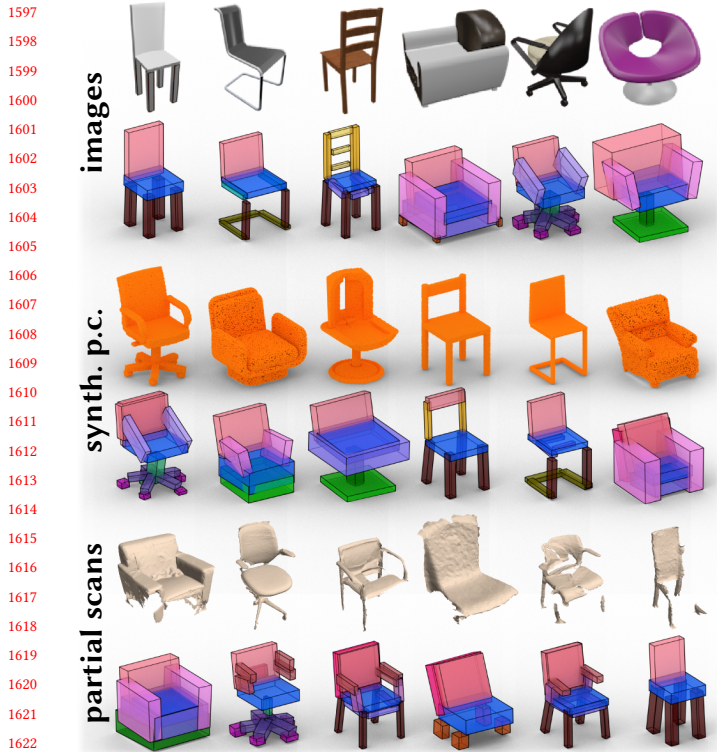


Fig. 14. **Image and point cloud abstraction.** Images, synthetic point clouds, and real-world scans from ScanNet [Dai et al. 2017] are embedded into our learned latent space, allowing us to effectively recover a full shape description that matches the raw input.

and even many of the details are represented accurately in the recovered shape, suggesting that the joint embedding successfully aligns structurally similar images and shapes in the latent space.

*Point cloud abstraction.* The point cloud encoder  $e_O$  is implemented as PointNet++ [Qi et al. 2017b] and similar to the image encoder, we train the network to encode 10k points obtained from each shape in our training set into the latent representation of the corresponding shape with  $\mathcal{L}_O = e(S) - e_O(\text{sample}(S))$ . Examples of abstractions computed for point clouds in our test set are shown in Figure 14, rows 3 and 4. We also tested our approach on real-world scans obtained from ScanNet [Dai et al. 2017]. Even though the statistics of the point distribution on real-world scans is likely to be different from our synthetic point sets, and the scans are missing large regions, our shape abstraction can recover good matches for each of the point clouds (see Figure 14, last two rows).

## 6.5 Shape Editing

Edits performed on a shape in a traditional 3D editor do not take into account the plausibility of the resulting shape. Our learned latent space gives us a definition of shape plausibility. Edits of a shape that preserve plausibility can thus be performed by finding the shape in our latent space that best satisfies the given edit. Below we present two simple shape editing applications based on this approach.

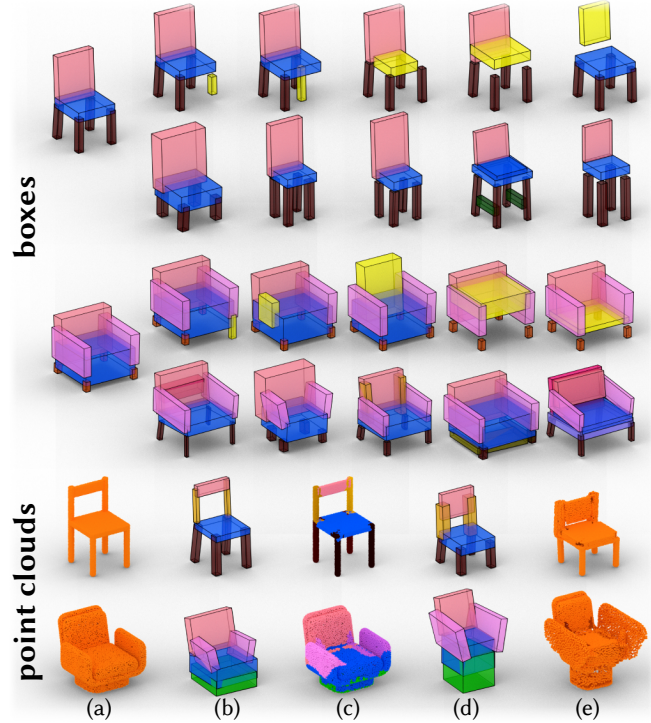


Fig. 15. **Structure-aware part editing.** We show editing results on two shapes with box geometry (first four rows) and two shapes with point cloud geometry (two bottom rows). For the two shapes with box geometry, we perform five different edits each, one edit per column. The edited box is highlighted in yellow, and the result is shown below. We see that the other boxes in the shape are adjusted to maintain shape plausibility. For the two shapes with point cloud geometry, we show intermediate results for one edit each. From left to right, these are (a) the original point cloud; (b) the predicted box abstraction; (c) the induced segmentation; (d) edited boxes; and (e) the induced edit of the point-cloud.

*Structure-preserving part edits.* In our latent space, shapes with similar structure are located close to each other. We can preserve the structure of a shape during editing by working with shapes that are close to the original shape in latent space. Starting from a shape with bounding box geometry, we edit one of its boxes, by translating, non-uniformly scaling, or rotating it. We then optimize for a shape in our latent space that is as close as possible to the original shape, while also satisfying the box edit:

$$\arg \min_z \left( \|z - z^*\|_2^2 + q_{\text{chs}}(T(B_e(z))\mathbf{U}, T(B_e^t)\mathbf{U}) + \mathcal{L}_{sc}(d(z)) \right). \quad (26)$$

The first term is the squared distance between the latent vector of the edited shape  $z$  and the latent vector of the original shape  $z^*$ . The second term is minimized by shapes that satisfy the edit, using the squared chamfer distance between the configuration of the edited box  $B_e$  in  $z$  and its target configuration  $B_e^t$ . Since the box edit is likely to break existing symmetries, we also specifically optimize for a shape  $S = d(z)$  that is consistent with its relationships using the loss  $\mathcal{L}_{sc}$ , as defined in Eq. 22.



Figure 15 shows 5 edits on each of the two shapes shown on the top left-hand side. Edited boxes are marked in yellow in the top row of each shape, and the result of the edit is shown in the bottom row. Since results only use shapes that can be found in our latent space, the results maintain plausibility of the shapes, adjusting the other parts in the shape as needed. Since we optimize for proximity to the original shape in latent space, the resulting shapes have similar structure, with a few minor exceptions, such as the added bars between the chair legs in the fourth column. This experiment suggests that structural differences are more distant in the latent space than geometric differences. This can also be verified by examining the visualization of the latent space provided in the Supplementary.

*Point Cloud Editing.* We can extend this editing approach to unannotated point clouds using the abstraction approach described in Section 6.4. The abstraction of the point cloud induces an instance segmentation, where each point is assigned to the closest bounding box. After editing the boxes with the method described above, we can update the subset of points corresponding to each box with the same transformation applied to the box, giving us an edited point cloud. The bottom two rows of Figure 15 show these steps in two examples edits. The edited point clouds show some artifacts due to the hard boundaries between different segments, but closely resemble the edited boxes. In the future, we could augment this method with either soft assignments of points to boxes (where boxes act similar to bones in character animation), or a segment refinement step, where the segment boundaries are optimized to coincide with surface discontinuities of the shape.

## 6.6 Limitations and Failure Cases

We discuss several limitations and failure cases: (i) STRUCTURENET, being a data-driven method, naturally inherits any data sampling biases in the datasets (e.g., shape families with very few examples such as pingpong tables). (ii) Even though our empirical experiments demonstrate good performance on adjacency recovery and symmetry enforcement, the inferred latent space may contain models with detached parts or asymmetric parts, especially for datasets that contain exotic, poorly represented shape variants. (iii) We restrict the maximum sibling count to  $n_p = 10$ , and hence cannot encode shapes with more than 10 childs in any given part (the full shape can have a much larger number of parts). The memory cost is quadratic in  $n_p$ , although, at our current setting, this is still far from being the most memory-consuming component (our current consumption is approx. 1 – 2 GB). (iv) Noise that would make a point cloud more blurry in holistic generation methods instead affects the structure in our method. Strong noise may result in missing parts, duplicate parts, detached parts, etc., although the structure, being discrete, is quite robust to this type of noise. See the bed category in the Supplementary for an example of structural noise. (v) Structure-aware point cloud generation is still a new topic requiring further research. In our experiments, to stabilize network training, we pretrain and freeze the part point cloud networks, which increases training robustness at the cost of failing to recover fine-grained geometry details (e.g., details on chair legs and chair backs). Figure 16 shows different failure cases of STRUCTURENET.

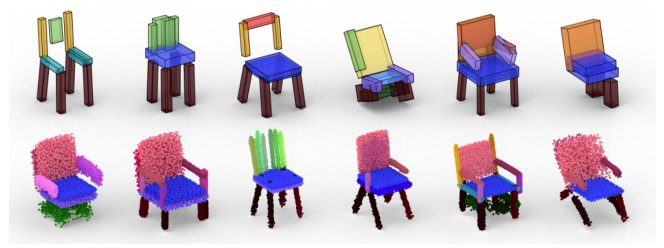


Fig. 16. **Failure cases analysis.** We present several failure cases we observed for box-shape and point cloud generation. We see discrete errors such as missing parts (e.g. first row, first column), duplicate parts (e.g. second row, second column), detached parts (e.g. first row, third column), asymmetric parts (e.g. second row, first column) and fuzzy point cloud generation (e.g. second row, fifth column).

## 7 CONCLUSION

We have presented STRUCTURENET as a VAE that directly encodes shape structure and geometry represented as a hierarchy of  $n$ -ary graphs. We have achieved this by proposing a recursive/hierarchical encoder-decoder architecture that simultaneously considers both geometry of parts, either as oriented bounding boxes or point clouds, and inter-part structures capturing adjacency and symmetry relations. Our key technical novelty is the handling of  $n$ -ary graphs by (i) explicitly predicting the presence or absence of parts or relationship edges, (ii) designing the encoder and the decoder to be invariant of the ordering of siblings across instances of the  $n$ -ary graphs, and (iii) introducing novel losses that enforce consistency between geometry and structure at all levels of the hierarchy. The learned  $n$ -ary structural graph latent space, by jointly capturing geometric and structure, greatly simplifies several applications. For example, we can ‘enter’ the latent space by *projecting* un-annotated data (e.g., partial scans, point clouds, or images) onto the latent space; perform structure-aware edits on individual shapes; do hierarchy-preserving interpolation between multiple shapes, or generate novel and diverse variations by directly sampling the encoded latent shape.

In future work we plan to (a) obtain user feedback to improve the estimated structure of un-annotated data and use this feedback to improve the latent embedding; (b) look at more nuanced representations of fine-grained part geometry and its correlations with structural motifs; (c) estimate point-level correspondences between raw shape data from images or point clouds and the learned models so as to transfer textures and other appearance information to the model and more generally study style-content factorizations; and (d) extend these ideas from individual objects to entire scenes, where the objects now become the parts and we now focus on object relationships in scenes.

## REFERENCES

- Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2017. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392* (2017).
- Amir Arsalan Soltani, Haibin Huang, Jiajun Wu, Tejas D Kulkarni, and Joshua B Tenenbaum. 2017. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1511–1519.
- H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. 1977. Parametric Correspondence and Chamfer Matching: Two New Techniques for Image Matching. In



- 1825 *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume*  
1826 *2 (IJCAI'77)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 659–663.  
1827 <http://dl.acm.org/citation.cfm?id=1622943.1622971>
- 1828 Davide Boscaini, Jonathan Masci, Simone Melzi, Michael M Bronstein, Umberto Castel-  
1829 lani, and Pierre Vanderghelynst. 2015. Learning class-specific descriptors for de-  
1830 formable shapes using localized spectral convolutional networks. In *Computer*  
1831 *Graphics Forum*, Vol. 34. Wiley Online Library, 13–23.
- 1832 Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks  
1833 and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).
- 1834 Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang,  
1835 Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. 2015. Shapenet:  
1836 An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012* (2015).
- 1837 Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun.  
1838 2011. Probabilistic reasoning for assembly-based 3D modeling. In *ACM Transactions*  
1839 *on Graphics (TOG)*, Vol. 30. ACM, 35.
- 1840 Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 2016.  
1841 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In  
1842 *European conference on computer vision*. Springer, 628–644.
- 1843 Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and  
1844 Matthias Nießner. 2017. ScanNet: Richly-annotated 3D Reconstructions of Indoor  
1845 Scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, IEEE.
- 1846 Michaël Defferrard, Xavier Bresson, and Pierre Vanderghelynst. 2016. Convolutional  
1847 neural networks on graphs with fast localized spectral filtering. In *Advances in*  
1848 *neural information processing systems*. 3844–3852.
- 1849 Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A  
1850 large-scale hierarchical image database. In *2009 IEEE conference on computer vision*  
1851 *and pattern recognition*. Ieee, 248–255.
- 1852 David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy  
1853 Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks  
1854 on graphs for learning molecular fingerprints. In *Advances in neural information*  
1855 *processing systems*. 2224–2232.
- 1856 Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent  
1857 neural network grammars. *arXiv preprint arXiv:1602.07776* (2016).
- 1858 Haoqiang Fan, Hao Su, and Leonidas J Guibas. 2017. A point set generation network for  
1859 3d object reconstruction from a single image. In *Proceedings of the IEEE conference*  
1860 *on computer vision and pattern recognition*. 605–613.
- 1861 Noa Fish, Melinos Averkiou, Oliver Van Kaick, Olga Sorkine-Hornung, Daniel Cohen-Or,  
1862 and Niloy J Mitra. 2014. Meta-representation of shape families. *ACM Transactions*  
1863 *on Graphics (TOG)* 33, 4 (2014), 34.
- 1864 Noa Fish, Oliver van Kaick, Amit Bermanto, and Daniel Cohen-Or. 2016. Structure-  
1865 oriented networks of shape collections. *ACM Transactions on Graphics (TOG)* 35, 6  
1866 (2016), 171.
- 1867 Vignesh Ganapathi-Subramanian, Olga Diamanti, Soeren Pirk, Chengcheng Tang,  
1868 Matthias Niessner, and Leonidas Guibas. 2018. Parsing geometry using structure-  
1869 aware shape templates. In *2018 International Conference on 3D Vision (3DV)*. IEEE,  
1870 672–681.
- 1871 Aleksey Golovinskiy and Thomas Funkhouser. 2009. Consistent segmentation of 3D  
1872 models. *Computers & Graphics* 33, 3 (2009), 262–269.
- 1873 Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil  
1874 Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In  
1875 *Advances in neural information processing systems*. 2672–2680.
- 1876 Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry.  
1877 2018. A papier-mâché approach to learning 3d surface generation. In *Proceedings of*  
1878 *the IEEE conference on computer vision and pattern recognition*. 216–224.
- 1879 J Gwak, Christopher B Choy, Animesh Garg, Manmohan Chandraker, and Silvio  
1880 Savarese. 2017. Weakly supervised generative adversarial networks for 3d recon-  
1881 struction. *arXiv preprint arXiv:1705.10904* (2017).
- 1882 Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning  
1883 on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.
- 1884 Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning  
1885 for image recognition. In *Proceedings of the IEEE conference on computer vision and*  
1886 *pattern recognition*. 770–778.
- 1887 Geoffrey E. Hinton. 1990. Mapping Part-whole Hierarchies into Connectionist Networks.  
1888 *Artif. Intell.* 46, 1-2 (Nov. 1990), 47–75. [https://doi.org/10.1016/0004-3702\(90\)90004-J](https://doi.org/10.1016/0004-3702(90)90004-J)
- 1889 Ruizhen Hu, Lubin Fan, and Ligang Liu. 2012. Co-segmentation of 3d shapes via  
1890 subspace clustering. In *Computer graphics forum*, Vol. 31. Wiley Online Library,  
1891 1703–1713.
- 1892 Qixing Huang, Vladlen Koltun, and Leonidas Guibas. 2011. Joint shape segmentation  
1893 with linear programming. In *ACM transactions on graphics (TOG)*, Vol. 30. ACM,  
1894 125.
- 1895 Evangelos Kalogerakis, Melinos Averkiou, Subhansu Maji, and Siddhartha Chaudhuri.  
1896 2017. 3D shape segmentation with projective convolutional networks. In *Proceedings*  
1897 *of the IEEE Conference on Computer Vision and Pattern Recognition*. 3779–3788.
- 1898 Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. 2012.  
1899 A probabilistic model for component-based shape synthesis. *ACM Transactions on*  
1900 *Graphics (TOG)* 31, 4 (2012), 55.
- 1901 Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. 2010. Learning 3D mesh  
1902 segmentation and labeling. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 102.
- 1903 Javor Kalojanov, Isaak Lim, Niloy Mitra, and Leif Kobbelt. 2019. String-Based Synthesis  
1904 of Structured Shapes. *Computer Graphics Forum* 38, 2 (2019), 027–036.
- 1905 Vladimir G Kim, Wilmt Li, Niloy J Mitra, Siddhartha Chaudhuri, Stephen DiVerdi, and  
1906 Thomas Funkhouser. 2013. Learning part-based templates from large collections of  
1907 3D shapes. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 70.
- 1908 Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv*  
1909 *preprint arXiv:1312.6114* (2013).
- 1910 Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph  
1911 Convolutional Networks. In *International Conference on Learning Representations*  
1912 (ICLR).
- 1913 Chun-Liang Li, Manzil Zaheer, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov.  
1914 2018b. Point cloud gan. *arXiv preprint arXiv:1810.05795* (2018).
- 1915 Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas.  
1916 2017. GRASS: Generative Recursive Autoencoders for Shape Structures. *ACM*  
1917 *Transactions on Graphics* 36, 4 (2017).
- 1918 Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir,  
1919 Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. 2019. Grains:  
1920 Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics*  
1921 (TOG) 38, 2 (2019), 12.
- 1922 Yangyan Li, Hao Su, Charles Ruizhongtai Qi, Noa Fish, Daniel Cohen-Or, and Leonidas J.  
1923 Guibas. 2015. Joint Embeddings of Shapes and Images via CNN Image Purification.  
1924 *ACM Trans. Graph.* 34, 6, Article 234 (Oct. 2015), 12 pages. <https://doi.org/10.1145/2816795.2818071>
- 1925 Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018a. Learning  
1926 deep generative models of graphs. *arXiv preprint arXiv:1803.03324* (2018).
- 1927 Tianqiang Liu, Siddhartha Chaudhuri, Vladimir G Kim, Qixing Huang, Niloy J Mitra, and  
1928 Thomas Funkhouser. 2014. Creating consistent scene graphs using a probabilistic  
1929 grammar. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 211.
- 1930 Chris Maddison and Daniel Tarlow. 2014. Structured generative models of natural  
1931 scene code. In *International Conference on Machine Learning*. 649–657.
- 1932 Ameesh Makadia and Mehmet Ersin Yumer. 2014. Learning 3d part detection from  
1933 sparsely labeled data. In *2014 2nd International Conference on 3D Vision*, Vol. 1. IEEE,  
1934 311–318.
- 1935 Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vanderghelynst. 2015.  
1936 Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings*  
1937 *of the IEEE international conference on computer vision workshops*. 37–45.
- 1938 Niloy J. Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, Vladimir Kim, and Qi-  
1939 Xing Huang. 2014. Structure-aware Shape Processing. In *ACM SIGGRAPH 2014*  
1940 *Courses (SIGGRAPH '14)*. ACM, New York, NY, USA, Article 13, 21 pages. <https://doi.org/10.1145/2614028.2615401>
- 1941 Kaichun Mo, Shilin Zhu, Angel Chang, Li Yi, Subarna Tripathi, Leonidas Guibas, and Hao  
1942 Su. 2019. PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical  
1943 Part-level 3D Object Understanding. In *Proceedings of the IEEE Conference on Computer*  
1944 *Vision and Pattern Recognition (CVPR)*.
- 1945 Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006.  
1946 Procedural modeling of buildings. *Acm Transactions On Graphics (Tog)* 25, 3 (2006),  
1947 614–623.
- 1948 Charlie Mendel and Christopher KI Williams. 2017. The shape variational autoencoder: A  
1949 deep generative model of part-segmented 3D objects. In *Computer Graphics Forum*,  
1950 Vol. 36. Wiley Online Library, 1–12.
- 1951 Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017a. Pointnet: Deep  
1952 learning on point sets for 3d classification and segmentation. In *Proceedings of the*  
1953 *IEEE Conference on Computer Vision and Pattern Recognition*. 652–660.
- 1954 Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. Pointnet++: Deep  
1955 hierarchical feature learning on point sets in a metric space. In *Advances in Neural*  
1956 *Information Processing Systems*. 5099–5108.
- 1957 Oana Sidi, Oliver van Kaick, Yanir Kleiman, Hao Zhang, and Daniel Cohen-Or. 2011.  
1958 *Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering*.  
1959 Vol. 30. ACM.
- 1960 Ayan Sinha, Asim Unmesh, Qixing Huang, and Karthik Ramani. 2017. Surfnet: Gen-  
1961 erating 3d shape surfaces using deep residual networks. In *Proceedings of the IEEE*  
1962 *conference on computer vision and pattern recognition*. 6040–6049.
- 1963 Richard Socher, Brody Huval, Bharath Bath, Christopher D Manning, and Andrew Y Ng.  
1964 2012. Convolutional-recursive deep learning for 3d object classification. In *Advances*  
1965 *in neural information processing systems*. 656–664.
- 1966 Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. 2011. Parsing natural  
1967 scenes and natural language with recursive neural networks. In *Proceedings of the*  
1968 *28th international conference on machine learning (ICML-11)*. 129–136.
- 1969 Minhyuk Sung, Hao Su, Vladimir G Kim, Siddhartha Chaudhuri, and Leonidas Guibas.  
1970 2017. Complementme: weakly-supervised deep convolution suggestions for 3D modeling.  
1971 *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 226.
- 1972 Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. 2017. Octree generating  
1973 networks: Efficient convolutional architectures for high-resolution 3d outputs. In  
1974 *Proceedings of the IEEE International Conference on Computer Vision*. 2088–2096.
- 1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038

- 1939 Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B.  
1940 Tenenbaum, and Jiajun Wu. 2019. Learning to Infer and Execute 3D Shape Programs.  
1941 In *International Conference on Learning Representations*.
- 1942 Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. 2017.  
1943 Learning shape abstractions by assembling volumetric primitives. In *Proceedings of  
1944 the IEEE Conference on Computer Vision and Pattern Recognition*. 2635–2643.
- 1945 Oliver Van Kaick, Kai Xu, Hao Zhang, Yanzhen Wang, Shuyang Sun, Ariel Shamir,  
1946 and Daniel Cohen-Or. 2013. Co-hierarchical analysis of shape structures. *ACM  
1947 Transactions on Graphics (TOG)* 32, 4 (2013), 69.
- 1948 Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and  
1949 Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*  
1950 (2017).
- 1951 Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey  
1952 Hinton. 2015. Grammar as a foreign language. In *Advances in neural information  
1953 processing systems*. 2773–2781.
- 1954 Hao Wang, Nadav Schor, Ruizhen Hu, Haibin Huang, Daniel Cohen-Or, and Hui Huang.  
1955 2018a. Global-to-local generative model for 3d shapes. In *SIGGRAPH Asia 2018  
1956 Technical Papers*. ACM, 214.
- 1957 Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. 2018b. Adaptive O-CNN: a  
1958 patch-based deep representation of 3D shapes. In *SIGGRAPH Asia 2018 Technical  
1959 Papers*. ACM, 217.
- 1960 Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M.  
1961 Solomon. 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transac-  
1962 tions on Graphics (TOG)* (2019).
- 1963 Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiqian Cheng,  
1964 and Yueshan Xiong. 2011a. Symmetry hierarchy of man-made objects. In *Computer  
1965 Graphics Forum*, Vol. 30. Wiley Online Library, 287–296.
- 1966 Y. Wang, K. Xu, J. Li, H. Zhang, A. Shamir, L. Liu, Z. Cheng, and Y. Xiong. 2011b.  
1967 Symmetry Hierarchy of Man-Made Objects. *Computer Graphics Forum* 30, 2 (2011),  
1968 287–296. <https://doi.org/10.1111/j.1467-8659.2011.01885.x>
- 1969 Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. 2016.  
1970 Learning a probabilistic latent space of object shapes via 3d generative-adversarial  
1971 modeling. In *Advances in neural information processing systems*. 82–90.
- 1972 Zhijie Wu, Xiang Wang, Di Lin, Dani Lischinski, Daniel Cohen-Or, and Hui Huang.  
1973 2018. Structure-aware generative network for 3d-shape modeling. *arXiv preprint  
1974 arXiv:1808.03981* (2018).
- 1975 Zhige Xie, Kai Xu, Ligang Liu, and Yueshan Xiong. 2014. 3d shape segmentation and  
1976 labeling via extreme learning machine. In *Computer graphics forum*, Vol. 33. Wiley  
1977 Online Library, 85–95.
- 1978 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019a. How Powerful are  
1979 Graph Neural Networks?. In *International Conference on Learning Representations*.  
1980 <https://openreview.net/forum?id=ryGs6iA5Km>
- 1981 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019b. How Powerful are  
1982 Graph Neural Networks?. In *ICLR*.
- 1983 Xinchun Yan, Jimie Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. 2016. Perspec-  
1984 tive transformer nets: Learning single-view 3d object reconstruction without 3d  
1985 supervision. In *Advances in Neural Information Processing Systems*. 1696–1704.
- 1986 Li Yi, Leonidas Guibas, Aaron Hertzmann, Vladimir G Kim, Hao Su, and Ersin Yumer.  
1987 2017a. Learning hierarchical shape segmentation and labeling from online reposi-  
1988 tories. *arXiv preprint arXiv:1705.01661* (2017).
- 1989 Li Yi, Vladimir G Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing  
1990 Huang, Alla Sheffer, and Leonidas Guibas Guibas. 2016. A scalable active framework  
1991 for region annotation in 3D shape collections. *ACM Transactions on Graphics (TOG)*  
1992 35, 6 (2016), 210.
- 1993 Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. 2017b. Syncspecnn: Synchronized  
1994 spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on  
1995 Computer Vision and Pattern Recognition*. 2282–2290.
- 1996 Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018.  
1997 Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv  
1998 preprint arXiv:1802.08773* (2018).
- 1999 Fenggen Yu, Kun Liu, Yan Zhang, Chenyang Zhu, and Kai Xu. 2019. PartNet: A Recursive  
2000 Part Decomposition Network for Fine-grained and Hierarchical Shape Segmentation.  
2001 In *CVPR*.
- 2002 Mehmet Ersin Yumer, Siddhartha Chaudhuri, Jessica K Hodgins, and Levent Burak  
2003 Kara. 2015. Semantic shape editing using deformation handles. *ACM Transactions  
2004 on Graphics (TOG)* 34, 4 (2015), 86.
- 2005 Xi Zhao, Ruizhen Hu, Paul Guerrero, Niloy Mitra, and Taku Komura. 2016. Relationship  
2006 templates for creating scene variations. *ACM Transactions on Graphics (TOG)* 35, 6  
2007 (2016), 207.
- 1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052