GRASS: Generative Recursive Autoencoders for Shape Structures

JUN LI, National University of Defense Technology
KAI XU*, National University of Defense Technology, Shenzhen University, and Shandong University
SIDDHARTHA CHAUDHURI, IIT Bombay
ERSIN YUMER, Adobe Research
HAO ZHANG, Simon Fraser University
LEONIDAS GUIBAS, Stanford University

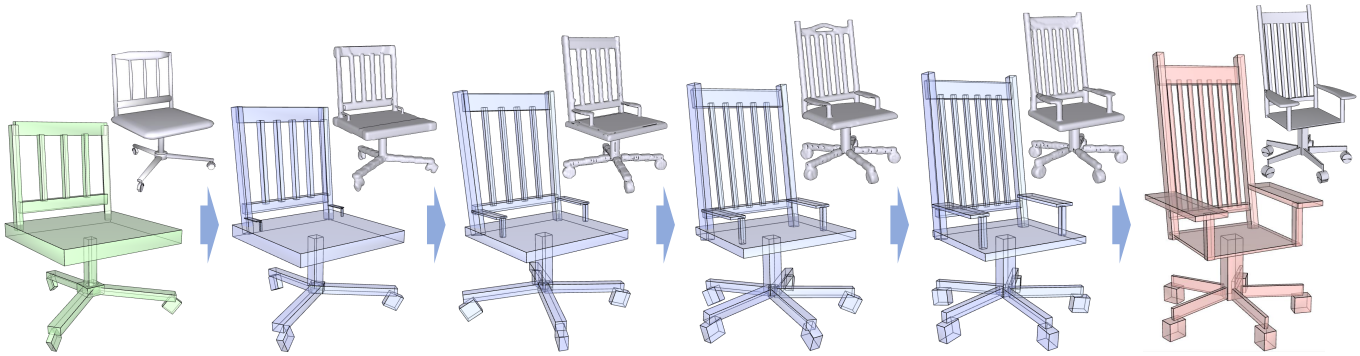


Fig. 1. We develop GRASS, a Generative Recursive Autoencoder for Shape Structures, which enables *structural* blending between two 3D shapes. Note the discrete blending of translational symmetries (slats on the chair backs) and rotational symmetries (the swivel legs). GRASS encodes and synthesizes box structures (bottom) and part geometries (top) separately. The blending is performed on fixed-length codes learned by the unsupervised autoencoder, without any form of part correspondences, given or computed.

We introduce a novel neural network architecture for *encoding* and *synthesis* of 3D shapes, particularly their *structures*. Our key insight is that 3D shapes are effectively characterized by their *hierarchical* organization of parts, which reflects fundamental intra-shape relationships such as adjacency and symmetry. We develop a *recursive* neural net (RvNN) based autoencoder to map a flat, unlabeled, arbitrary part layout to a compact code. The code effectively captures hierarchical structures of man-made 3D objects of varying structural complexities despite being fixed-dimensional: an associated decoder maps a code back to a full hierarchy. The learned bidirectional mapping is further tuned using an adversarial setup to yield a generative model of plausible structures, from which novel structures can be sampled. Finally, our structure synthesis framework is augmented by a second trained module that produces fine-grained part geometry, conditioned on global and local structural context, leading to a full generative pipeline for 3D shapes. We demonstrate that without supervision, our network learns meaningful structural hierarchies adhering to perceptual grouping principles, produces compact codes which enable applications such as shape classification and partial matching, and supports shape synthesis and interpolation with significant variations in topology and geometry.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; *Shape analysis*;

Additional Key Words and Phrases: analysis and synthesis of shape structures, symmetry hierarchy, recursive neural network, autoencoder, generative recursive autoencoder, generative adversarial training

ACM Reference format:

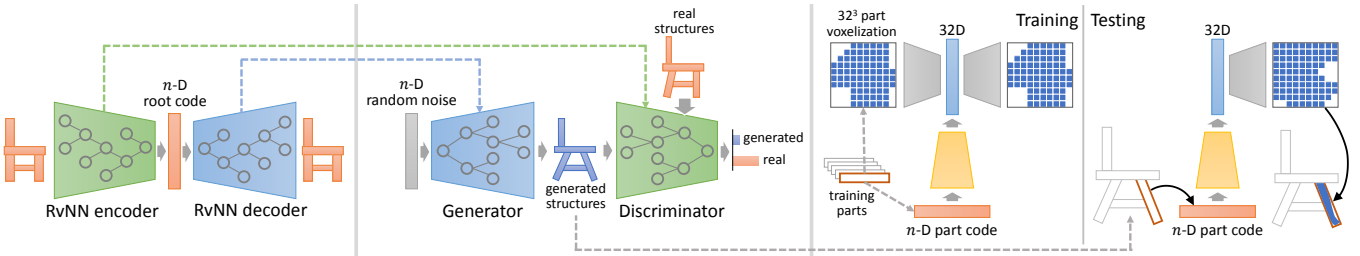
Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. 2017. GRASS: Generative Recursive Autoencoders for Shape Structures. *ACM Trans. Graph.* 36, 4, Article 52 (July 2017), 14 pages.

DOI: http://dx.doi.org/10.1145/3072959.3073613

*Corresponding author: kevin.kai.xu@gmail.com

1 INTRODUCTION

Recent progress on training neural networks for image (van den Oord et al. 2016b) and speech (van den Oord et al. 2016a) synthesis has led many to ask whether a similar success is achievable in learning generative models for 3D shapes. While an image is most naturally viewed as a 2D signal of pixel values and a piece of speech as a sampled 1D audio wave, the question of what is *the* canonical representation for 3D shapes (voxels, surfaces meshes, or multi-view images) may not always yield a consensus answer. Unlike images or sound, a 3D shape does not have a natural parameterization over a regular low-dimensional grid. Further, many 3D shapes, especially of man-made artifacts, are highly structured (e.g. with hierarchical decompositions and nested symmetries), while exhibiting rich structural variations even within the same object class (e.g. consider



(a) RvNN auto-encoder pre-training. (b) RvNN-GAN training. (c) Volumetric part geometry synthesis.

Fig. 2. An overview of our pipeline, including the three key stages: (a) pre-training the RvNN autoencoder to obtain root codes for shapes, (b) using a GAN module to learn the actual shape manifold within the code space, and (c) using a second network to convert synthesized OBBs to detailed geometry.

the variety of chairs). Hence, the stationarity and compositionality assumptions (Henaff et al. 2015) behind the success of most neural nets for *natural* images or speech are no longer applicable.

In this paper, we are interested in learning *generative neural nets* for *structured* shape representations of man-made 3D objects. In general, shape structures are defined by the arrangement of, and relations between, shape parts (Mitra et al. 2013). Developing neural nets for structured shape representations requires a significant departure from existing works on convolutional neural networks (CNNs) for volumetric (Girdhar et al. 2016; Wu et al. 2016, 2015; Yumer and Mitra 2016) or view-based (Qi et al. 2016; Sinha et al. 2016; Su et al. 2015) shape representations. These works primarily adapt classical CNN architectures for image analysis. They do not explicitly encode or synthesize part arrangements or relations such as symmetries.

Our goal is to learn a generative neural net for shape structures characterizing an object class, e.g. chairs or candelabras. The main challenges we face are two-fold. The first is how to properly "mix", or *jointly* encode and synthesize (discrete) structure and (continuous) geometry. The second is due to intra-class structural variations. If we treat shape structures as graphs, the foremost question is how to enable a generic neural network to work with graphs of *different* combinatorial structures and sizes. Both challenges are unique to our problem setting and neither has been addressed by networks which take inputs in the form of unstructured, fixed-size, low-dimensional grid data, e.g. images or volumes.

Our key insight is that most shape structures are naturally *hierarchical* and hierarchies can jointly encode structure and geometry. Most importantly, regardless of the variations across shape structures, a coding scheme that recursively contracts hierarchy or tree nodes into their parents attains *unification* at the top — any finite set of structures eventually collapses to root node codes with a possibly large but *fixed length*. We learn a neural network which can recursively encode hierarchies into root codes and invert the process via decoding. Then, by further learning a distribution over the root codes for a class of shapes, new root codes can be generated and decoded to synthesize new structures and shapes in that class.

Specifically, we represent a 3D shape using a *symmetry hierarchy* (Wang et al. 2011), which defines how parts in the shape are recursively grouped by symmetry and assembled by connectivity. Our neural net architecture, which learns to infer such a hierarchy for a shape in an unsupervised fashion, is inspired by the *recursive neural nets* (RvNN)[1] of Socher et al. (2012; 2011) developed for text and image understanding. By treating text as a set of words and an image as a set of superpixels, an RvNN learns a parse tree which recursively merges text/image segments. There are two key differences and challenges that come with our work:

- First, the RvNNs of Socher et al. (2011) always merge two adjacent elements and this is modeled using the same network at every tree node. However, in a symmetry hierarchy, grouping by symmetry and assembly by connectivity are characteristically different merging operations. As well, the network structures at a tree node must accommodate assembly, reflectional symmetry, and rotational/translational symmetries of varying orders.
- Second, our main goal is to learn a *generative* RvNN, for part-based shape structures that are explicitly represented as discrete structural combinations of geometric entities.

To accomplish these goals, we focus on learning an abstraction of symmetry hierarchies, which are composed of spatial arrangements of oriented bounding boxes (OBBs). Each OBB is defined by a fixed-length code to represent its geometry and these codes sit at the leaves of the hierarchies. Internal nodes of the hierarchies, also characterized by fixed-length codes, encode both the geometry of its child OBBs and their detailed grouping mechanism: whether by connectivity or symmetry.

We pre-train an *unsupervised* RvNN using OBB arrangements endowed with box connectivity and various types of symmetry. Our neural network is an *autoencoder*-based RvNN which recursively assembles or (symmetrically) groups a set of OBBs into a fixed-length root code and then decodes the root to reconstruct the input; see Figure 2(a). The network comprises two types of nodes: one to handle assembly of connected parts, and one to handle symmetry grouping. Each merging operation takes two or more OBBs as input. Our RvNN learns how to best organize a shape structure into a symmetry hierarchy to arrive at a compact and minimal-loss code accounting for both geometry and structure.

To synthesize new 3D shapes, we extend the pre-trained autoencoder RvNN into a generative model. We learn a distribution over root codes constructed from shape structures for 3D objects of the same class, e.g. chairs. This step utilizes a generative adversarial network

[1]Note that we are adding the letter 'v' to the acronym RNN, since by now, the term RNN most frequently refers to *recurrent* neural networks.

(GAN), similar to a VAE-GAN (Larsen et al. 2015), to learn a low-dimensional manifold of root codes; see Figure 2(b). We sample and then project a root code onto the manifold to synthesize an OBB arrangement. In the final stage, the boxes are filled with part geometries by another generative model which learns a mapping between box features and voxel grids; see Figure 2(c). We refer to our overall generative neural network as a *generative recursive autoencoder* for shape structures, or GRASS. Figure 2 provides an overview of the complete architecture.

The main contributions of our work can be summarized as follows.

- The first generative neural network model for structured 3D shape representations — GRASS. This is realized by an autoencoder RvNN which learns to encode and decode shape structures via discovered symmetry hierarchies, followed by two generative models trained to synthesize box-level symmetry hierarchies and volumetric part geometries, respectively.
- A novel RvNN architecture which extends the original RvNN of Socher et al. (2011) by making it generative and capable of encoding a variety of merging operations (i.e. assembly by connectivity and symmetry groupings of different types).
- An *unsupervised* autoencoder RvNN which jointly learns and encodes the structure and geometry of box layouts of varying sizes into *fixed-length* vectors.

We demonstrate that our network learns meaningful structural hierarchies adhering to perceptual grouping principles, produces compact codes which enable applications such as shape classification and partial matching, and support generative models which lead to shape synthesis and interpolation with significant variations in topology and geometry.

2 RELATED WORK

Our work is related to prior works on statistical models of 3D shape structures, including recent works on applying deep neural networks to shape representation. These models can be disciminative or generative, and capture continuous or discrete variations. We review the most relevant works below. Since our focus is shape synthesis, we emphasize generative models in our discussion.

Statistical shape representations. Early works on capturing statistical variations of the human body explored smooth deformations of a fixed template (Allen et al. 2003; Anguelov et al. 2005; Blanz and Vetter 1999). Later papers addressed discrete variations at the part level, employing stochastic shape grammars coded by hand (Müller et al. 2006), learned from a single training example (Bokeloh et al. 2010), or learned from multiple training examples (Talton et al. 2012). Parallel works explored the use of part-based Bayesian networks (Chaudhuri et al. 2011; Kalogerakis et al. 2012) and modular templates (Fish et al. 2014; Kim et al. 2013) to represent both continuous and discrete variations. However, these methods are severely limited in the variety and complexity of part layouts they can generate, and typically only work well for shape families with a few consistently appearing parts and a restricted number of possible layouts. In a different approach, Talton et al. (2009) learn a probability distribution over a shape space generated by a procedure operating on a fixed set of parameters. We are also inspired by some non-statistical shape representations such as the work of Wang et al. (2011) and van Kaick et al. (2013) on extracting hierarchical structure from a shape: our goal in this paper is to learn consistent, probabilistic, hierarchical representations automatically from unlabeled datasets. Mitra et al. (2013) provide an overview of a range of further works on statistical and structure-aware shape representations.

Deep models of 3D shapes. Recently, the success of deep neural networks in computer vision, speech recognition, and natural language processing has inspired researchers to apply such models to 3D shape analysis. While these are of course statistical shape representations, their immediate relevance to this paper merits a separate section from the above. Most of these works have focused on extending computer vision techniques developed for images – 2D grids of pixels – to 3D grids of voxels. Wu et al. (2015) propose a generative model based on a deep belief network trained on a large, unannotated database of voxelized 3D shapes. They show applications of the model to shape synthesis and probabilistic shape completion for next-best view prediction. Girdhar et al. (2016) jointly train a deep convolutional encoder for 2D images and a deep convolutional decoder for voxelized 3D shapes, chained together so that the vector output of the encoder serves as the input code for the decoder, allowing 3D reconstruction from a 2D image. Yan et al. (2016) propose a different encoder-decoder network for a similar application. Yumer and Mitra (2016) present a 3D convolutional network that maps a voxelized shape plus a semantic modification intent to the deformation field required to realize that intent.

In a departure from voxel grids, Su et al. (2015) build a powerful shape classifier based on multiple projected views of the object, by fine-tuning standard image-based CNNs trained on huge 2D datasets and applying a novel pooling mechanism. Masci et al. (2015) build a convolutional network directly on non-Euclidean shape surfaces. Qi et al. (2016) discuss ways to improve the performance of both volumetric and multi-view CNNs for shape classification. In a recent work, Tulsiani et al. (2017) develop a discriminative, CNN-based approach to consistently parse shapes into a bounded number of volumetric primitives.

We are inspired by the work of Huang et al. (2015), who develop a deep Boltzmann machine-based model of 3D shape surfaces. This approach can be considered a spiritual successor of Kalogerakis et al. (2012) and Kim et al. (2013), learning modular templates that incorporate fine-grained part-level deformation models. In addition to being fully generative – the model can be sampled for a point set representing an entirely new shape – the method automatically refines shape correspondences and part boundaries during training. However, like the prior works, this approach is limited in the variety of layouts it can represent.

Wu et al. (2016) exploit the success of generative adversarial nets (GAN) (Goodfellow et al. 2014) to improve upon the model of Wu et al. (2015). At its core, their model is a generative decoder that takes as input a 200-D shape code and produces a voxel grid as output. The decoder is trained adversarially, and may be chained with a prior encoder that maps, say, a 2D image to the corresponding shape code. The method supports simple arithmetic and interpolation on

the codes, enabling, for instance, topology-varying morphs between different shapes. Our work is complementary to this method: we seek to develop a powerful model of part layout variations that can accurately synthesize complex hierarchical structures beyond the representational power of low-resolution grids, can be trained on relatively fewer shapes, and is independent of voxel resolution.

Neural models of graph structure. The layout of parts of a shape inherently induces a non-Euclidean, graph-based topology defined by adjacency and relative placement. Several works, not concerning geometric analysis, have explored neural networks operating on graph domains. The most common such domains are of course linear chains defining text and speech signals. For these domains, recurrent neural networks (RNNs), as well as convolutional neural networks (CNNs) over sliding temporal windows, have proved very successful. Such linear models have even been adapted to generate non-linear output such as images, as in the work of van den Oord et al. (2016b; 2016c), producing the image row by row, pixel by pixel. These models are, however, limited in such adaptations since it is difficult to learn and enforce high-level graph-based organizational structure. Henaff et al. (2015), Duvenaud et al. (2015) and Niepert et al. (2016) propose convolutional networks that operate directly on arbitrary graphs by defining convolution as an operation on the radial neighborhood of a vertex. However, none of these works enable generative models. A different approach to this problem, which directly inspires our work, is the *recursive* neural network (RvNN) proposed by Socher et al. (2012; 2011), which sequentially collapses edges of a graph to yield a hierarchy. We build upon the autoencoder version of this network, adapting it to learn the particular organizational principles that characterize 3D shape structure, and to extend it from a deterministic model to a probabilistic generative one.

3 OVERVIEW

Our method for learning GRASS, a hierarchical, symmetry-aware, generative model for 3D shapes, has three stages, shown in Figure 2. In this section, we summarize the stages and highlight important components and properties of the neural networks we use.

Geometry and structure encoding. We define an abstraction of symmetry hierarchies, which are composed of spatial arrangements of oriented bounding boxes (OBBs). Each OBB is defined by a fixed-length code to represent its geometry. The fixed length code encodes both the geometry of its child OBBs and their detailed grouping mechanism: whether by connectivity or symmetry.

Stage 1: Recursive autoencoder. In the first stage, we train an autoencoder for layouts of OBBs. The autoencoder maps a box layout with an arbitrary number and arrangement of components to a fixed-length root code that implicitly captures its salient features. The encoding is accomplished via a recursive neural network (RvNN) that repeatedly, in a bottom-up fashion, collapses a pair of boxes represented as codes into a merged code. The process also yields a hierarchical organizational structure for the boxes. The final code representing the entire layout is decoded to recover the boxes (plus the entire hierarchy) by an inverse process, and the training loss is measured in terms of a reconstruction error and back-propagated to update the network weights.

Stage 2: Learning manifold of plausible structures. We extend the autoencoder to a generative model of structures by learning a distribution over root codes that describes the shape manifold, or shape space, occupied by codes corresponding to meaningful shapes within the full code space. We train a generative adversarial model (GAN) for a low-dimensional manifold of root codes that can be decoded to structures indistinguishable, to an adversarial classifier, from the training set. Given a randomly selected root code, we project it to the GAN manifold to synthesize a plausible new structure.

Stage 3: Part geometry synthesis. In the final stage, the synthesized boxes are converted to actual shape parts. Given a box in a synthesized layout, we compute structure-aware recursive features that represent it in context. Then, we simultaneously learn a compact, invertible encoding of voxel grids representing part geometries as well as a mapping from contextual part features to the encoded voxelized geometry. This yields a procedure that can synthesize detailed geometry for a box in a shape structure.

By chaining together hierarchical structure generation and part geometry synthesis, we obtain the full GRASS pipeline for recursive synthesis of shape structures.

4 RECURSIVE MODEL OF SHAPE STRUCTURE

In this section, we describe a method to encode shape structures into a short, fixed-dimensional code. The learned encoding is fully invertible, allowing the structure to be reconstructed from the code. In Section 5, we present our method to adversarially tune this structure decoder to map random codes to structures likely to come from real shapes. By combining this generator for sampling plausible shape structures with a method for synthesizing the geometry of individual parts (Section 6), we obtain our probabilistic generative model for 3D shapes.

Our key observation is that shape components are commonly arranged, or perceived to be arranged, hierarchically. This is a natural organizational principle in well-accepted theories of human cognition and design, which has been extensively leveraged computationally (Serre 2013). Perceptual and functional hierarchies follow patterns of component proximity and symmetry. Hence, the primary goal of our structural code is to successfully encode the hierarchical organization of the shape in terms of symmetries and adjacencies. An important metric of success is that the hierarchies are *consistent* across different shapes of the same category. We achieve this via a compact model of recursive component aggregation that tries to consistently identify similar substructures.

Our model is based on Recursive Autoencoders (RAE) for unlabeled binary trees, developed by Socher et al. (2014). The RAE framework proposed by Socher et al. consists of an *encoder* neural network that takes two n-dimensional inputs and produces a single n-dimensional output, and a *decoder* network that recovers two n-D vectors from a single n-D vector. In our experiments, $n = 80$.

Given a binary tree with n-D descriptors for the leaves, the RAE is used to recursively compute descriptors for the internal nodes, ending with a root code. The root code can be inverted to recover the original tree using the decoder, and a training loss formulated in terms of a reconstruction error for the leaves.

RAEs were originally intended for parsing natural language sentences in a discriminative setting, trained on unlabeled parse trees. We adapt this framework for the task of learning and synthesizing hierarchical shape structures. This requires several important technical contributions, including extending the framework to accommodate multiple encoder and decoder types, handling non-binary symmetric groups of parts, and probabilistically generating shapes (as described in the subsequent sections).

Criteria for recursive merging. Our model of hierarchical organization of shape parts follows two common perceptual/cognitive criteria for recursive merging: a mergeable subset of parts is either an adjacent pair (the *adjacency* criterion) or part of a symmetry group (the *symmetry* criterion)[2]. An adjacent pair is represented by the bounding boxes of constituent parts. In this stage, we are interested only in representing the gross layout of parts, so we discard fine-grained geometric information and store only oriented part bounding boxes, following earlier work on shape layouts (Ovsjanikov et al. 2011) — fine-grained geometry synthesis is described in Section 6. We recognize three different types of symmetries, each represented by the bounding box of a generator part plus further parameters: (1) pairwise reflectional symmetry, parametrized by the plane of reflection; (2) k-fold rotational symmetry, parametrized by the number of parts k and the axis of rotation; and (3) k-fold translational symmetry, parametrized by k and the translation offset between parts. The different scenarios are illustrated in Figure 3. We generate training hierarchies that respect these criteria, and our autoencoder learns to synthesize hierarchies that follow them.

Synthesizing training data. To train our recursive autoencoder, we synthesize a large number of training hierarchies from a dataset of shapes. These shapes are assumed to be pre-segmented into constituent (unlabeled) parts, but do not have ground truth hierarchies. We adopt an iterative, randomized strategy to generate plausible hierarchies for a shape that satisfy the merging criteria described above. In each iteration, two or more parts are merged into a single one. A mergeable subset of parts is either adjacent or symmetric. We randomly sample a pair that satisfies one of the two criteria until no further merges are possible. In our experiments, we generated 20 training hierarchies for each shape in this fashion. Note that none of these hierarchies is intended to represent "ground truth". Rather, they sample the space of plausible part groupings in a relatively unbiased fashion for training purposes.

[2]Currently, we make the reasonable single-object assumption that all parts are connected by either adjacency or symmetry. For disconnected, asymmetric shapes, we would need further merging criteria.

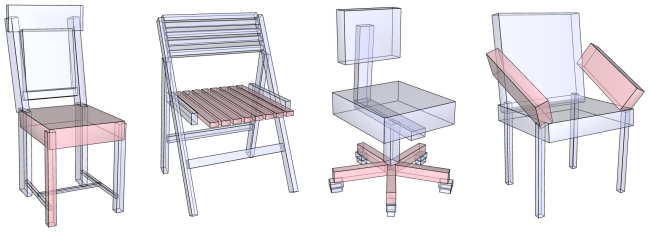


Fig. 3. Merging criteria used by our model demonstrated with 3D shapes represented by part bounding boxes (relevant parts highlighted in red). From left: (a) two adjacent parts, (b) translational symmetry, (c) rotational symmetry, and (d) reflective symmetry.

Autoencoder model. To handle both adjacency and symmetry relations, our recursive autoencoder comprises two distinct types of encoder/decoder pairs. These types are:

Adjacency. The encoder for the adjacency module is a neural network AdjEnc which merges codes for two adjacent parts into the code for a single part. It has two n-D inputs and one n-D output. Its parameters are a weight matrix $W_{ae} \in \mathbb{R}^{n \times 2n}$ and a bias vector $b_{ae} \in \mathbb{R}^n$, which are used to obtain the code of parent (merged) node y from children x_1 and x_2 using the formula

$$y = \tanh(W_{ae} \cdot [x_1\ x_2] + b_{ae})$$

The corresponding decoder AdjDec splits a parent code y back to child codes x_1' and x_2', using the reverse mapping

$$[x_1'\ x_2'] = \tanh(W_{ad} \cdot y + b_{ad})$$

where $W_{ad} \in \mathbb{R}^{2n \times n}$ and $b_{ad} \in \mathbb{R}^{2n}$.

Symmetry. The encoder for the symmetry module is a neural network SymEnc which merges the n-D code for a generator part of a symmetry group, as well as the m-D parameters of the symmetry itself into a single n-D output. The code for a group with generator x and parameters p is computed as

$$y = \tanh(W_{se} \cdot [x\ p] + b_{se})$$

and the corresponding decoder SymDec recovers the generator and symmetry parameters as

$$[x'p'] = \tanh(W_{sd} \cdot y + b_{sd})$$

where $W_{se} \in \mathbb{R}^{n \times (n+m)}$, $W_{sd} \in \mathbb{R}^{(n+m) \times n}$, $b_{se} \in \mathbb{R}^n$, and $b_{sd} \in \mathbb{R}^{m+n}$. In our implementation, we use $m = 8$ to encode symmetry parameters comprising symmetry type (1D); number of repetitions for rotational and translational symmetries (1D); and the mirror plane for reflective symmetry, rotation axis for rotational symmetry, or position and displacement for translational symmetry (6D).

In practice, the encoders/decoders for both adjacency and symmetry are implemented as two-layer networks, where the dimensions of the hidden and output layers are 100D and 80D, respectively.

The input to the recursive merging process is a collection of part bounding boxes. These need to be mapped to n-D vectors before they can be processed by the autoencoder. To this end, we employ additional single-layer neural networks BoxEnc, which maps the 12D parameters of a box (concatenating box center, dimensions and two axes) to an n-D code, and BoxDec, which recovers the 12D parameters from the n-D code. These networks are non-recursive, used simply to translate the input to the internal code representation at the beginning, and back again at the end.

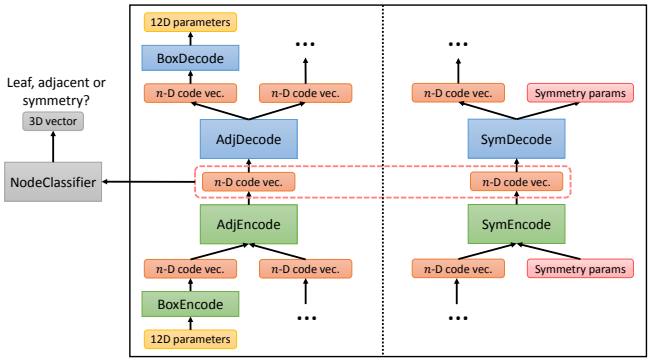


Fig. 4. Autoencoder training setup. Ellipsis dots indicate that the code could be either the output of BoxEnc, AdjEnc or SymEnc, or the input to BoxDec, AdjDec or SymDec.

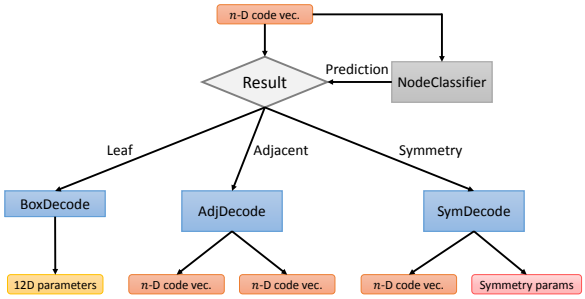


Fig. 5. Autoencoder test decoding setup.

Lastly, we jointly train an auxiliary classifier NodeClsfr to decide which module to apply at each recursive decoding step. This classifier is a neural network with one hidden layer that takes as input the code of a node in the hierarchy, and outputs whether the node represents an adjacent pair of parts, a symmetry group, or a leaf node. Depending on the output of the classifier, either AdjDec, SymDec or BoxDec is invoked.

Training. To train our recursive autoencoder, we use BFGS with back-propagation, starting with a random initialization of weights sampled from a Gaussian distribution. The loss is formulated as a reconstruction error. Given a training hierarchy, we first encode each leaf-level part bounding box using BoxEnc. Next, we recursively apply the corresponding encoder (AdjEnc or SymEnc) at each internal node until we obtain the code for the root. Finally, we invert the process, starting from the root code, to recover the leaf parameters by recursively applying the decoders AdjDec and SymDec, followed by a final application of BoxDec. The loss is formulated as the sum of squared differences between the input and output parameters for each leaf box.

Note that during training (but not during testing), we use the input hierarchy for decoding, and hence always know which decoder to apply at which unfolded node, and the mapping between input and output boxes. We simultaneously train NodeClsfr, with a three-class softmax classification with cross entropy loss, to recover the tree topology during testing. The training setup is illustrated in Figure 4.

Testing. During testing, we must address two distinct challenges. The first is to infer a plausible encoding hierarchy for a novel segmented shape without hierarchical organization. The second is to decode a given root code to recover the constituent bounding boxes of the shape.

To infer a plausible hierarchy using the trained encoding modules, we resort to *greedy local search*. Specifically, we look at all subsets that are mergeable to a single part, perform *two* levels of recursive encoding and decoding, and measure the reconstruction error. The merge sequence with the lowest reconstruction error is added to the encoding hierarchy. The process repeats until no further merges are possible. Particular cases of interest are *adjacency before symmetry*, and *symmetry before adjacency*, as illustrated in Figure 6. For each such case, we decode the final code back to the input box parameters (using, as for training, the known merging hierarchy) and measure the reconstruction error. This two-step lookahead is employed only for inferring hierarchies in test mode. During training, we minimize reconstruction loss over the hierarchy for the entire shape, as well as over all subtrees. Thus, the encoder/decoder units are tuned for both locally and globally good reconstructions, and at test time a relatively short lookahead suffices.

To decode a root code (e.g. one obtained from an encoding hierarchy inferred in the above fashion), we recursively invoke NodeClsfr to decide whether which decoder should expand the node. The corresponding decoder (AdjDec, SymDec or BoxDec) is used to recover the codes of child nodes until the full hierarchy has been expanded to leaves with corresponding box parameters. The test decoding setup is illustrated in Figure 5.

Several examples of test reconstructions are shown in Figure 7. The above procedures are used to encode a novel shape to a root code, and to reconstruct the shape given just this root code. In Figure 8, we show how our RvNN is able to find a perceptually reasonable symmetry hierarchy for a 3D shape structure, by minimizing the reconstruction error. Given the structure of a swivel chair, the error is much smaller when a wheel and spike are merged before the 5-fold rotational symmetry is applied, than if two separate rotational symmetries (for wheels and spikes respectively) are applied first.

5 LEARNING MANIFOLD OF PLAUSIBLE STRUCTURES

Our recursive autoencoder computes a compact, fixed-dimensional code that represents the inferred hierarchical layout of shape parts, and can recover the layout given just this code associated with the

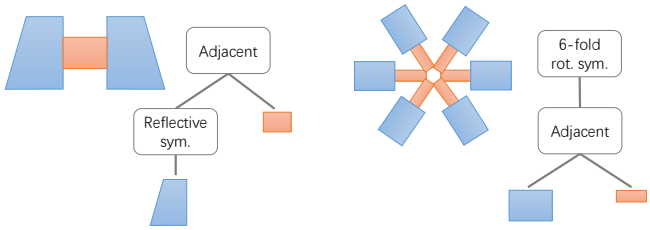


Fig. 6. Different two-step encoding orders for two examples, found by minimizing reconstruction errors during testing. Left: Symmetry (reflective) before adjacency. Right: Adjacency before symmetry (6-fold rotational).

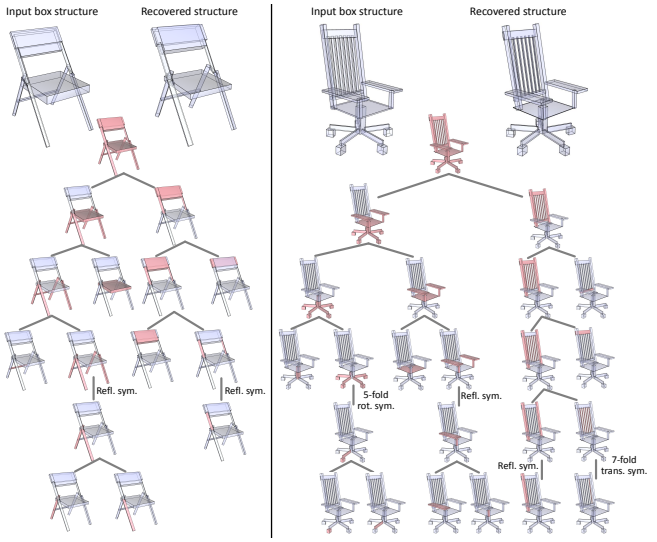


Fig. 7. Examples of reconstructing test shapes, without known hierarchies, by successively encoding them to root codes, and decoding them back. The encoding hierarchies inferred by our RvNN encoder are shown at the bottom.

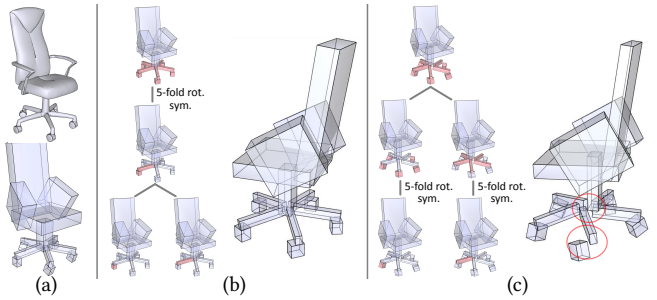


Fig. 8. Our RvNN encoder can find a perceptually reasonable symmetry hierarchy for a 3D shape structure, through minimizing reconstruction error. Given an input structure (a), the reconstruction error is much smaller if parts are grouped by adjacency before symmetry (b), instead of symmetry before adjacency (c).

root of the hierarchy. However, the autoencoder developed so far is not a generative model. It can reconstruct a layout from *any* root code, but an arbitrary, random code is unlikely to produce a plausible layout. A generative model must jointly capture the distribution of statistically plausible shape structures.

In this section, we describe our method for converting the autoencoder-based model to a fully generative one. We fine-tune the autoencoder to learn a (relatively) low-dimensional manifold containing high-probability shape structures. Prior approaches for learning feasible manifolds of parametrized 3D shapes from landmark exemplars include kernel density estimation (Fish et al. 2014; Talton et al. 2009), multidimensional scaling (Averkiou et al. 2014), and piecewise primitive fitting (Schulz et al. 2016). However, these methods essentially reduce to simple interpolation from the landmarks, and hence may assign high probabilities to parameter vectors that correspond to implausible shapes (Goodfellow et al. 2014).

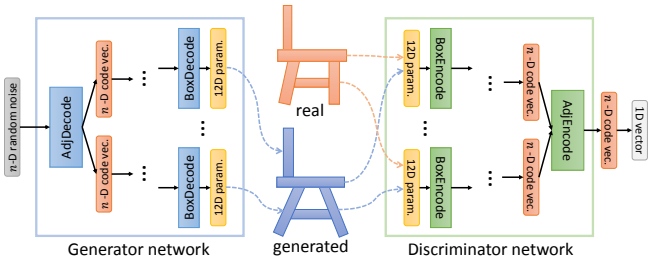


Fig. 9. Architecture of our generative adversarial network, showing reuse of autoencoder modules.

Recently, *generative adversarial networks* (GANs) (Goodfellow et al. 2014) have been introduced to overcome precisely this limitation. Instead of directly interpolating from training exemplars, a GAN trains a synthesis procedure to map arbitrary parameter vectors only to vectors which a classifier deems plausible. The classifier, which can be made arbitrarily sophisticated, is jointly trained to identify objects similar to the exemplars as plausible, and others as fake. This leads to a refined mapping of the latent space since implausible objects are eliminated by construction. Given a completely random set of parameters, the trained GAN "snaps" it to the plausible manifold to generate a meaningful sample.

In addition to enabling the synthesis of novel but statistically plausible shape structures, the learned manifold also supports interpolation between shape codes. The application of this feature to shape morphing is shown in Section 7.

GAN architecture. The architecture of our generative adversarial network comprises a generator (G) network, which transforms a random code to a hierarchical shape structure lying on the estimated manifold, and a discriminator (D) network, which checks whether a generated structure is similar to those of the training shapes or not. Our key observation is that we can *directly reuse and fine-tune the autoencoder modules* learned in the previous section, instead of introducing new components. The decoder component (comprising AdjDec, SymDec, BoxDec and NodeClsfr) is exactly what we need to estimate a structure from a given code: it constitutes the G network. The encoder component (comprising AdjEnc, SymEnc and BoxEnc) is exactly what we need to estimate a code for the generated structure. The final code can be compared to the codes of training structures using an additional fully connected layer and a binary softmax layer producing the probability of the structure being "real". This constitutes the D network. Hence, we initialize the GAN with the trained autoencoder modules and further fine-tune them to minimize the GAN loss. The architecture is illustrated in Figure 9, and the training procedure described below.

Training. The GAN is trained by stochastic gradient descent using different loss functions for the discriminator D and the generator G. In each iteration, we sample two mini-batches: training box structures x with their associated hierarchies, and random codes $z \in \mathbb{R}^n$. The x samples, with known hierarchies, are passed only through the discriminator, yielding $D(x)$, whereas the z samples are passed through both networks in sequence, yielding $D(G(z))$. The

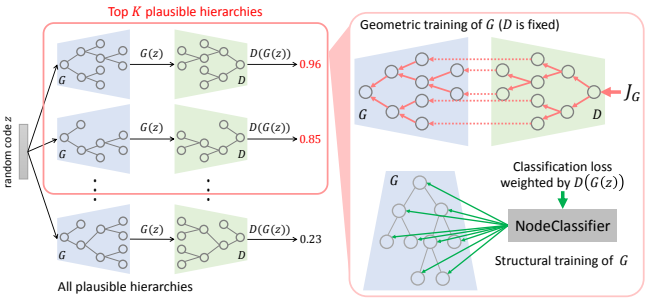


Fig. 10. The training of our GAN model. Left: Given a random code, we select the top K "plausible" hierarchies from which G can decode a box structure to best fool D. Right: For each selected hierarchy, the training of G is split into geometric (top) and structural (bottom) tuning, based on different loss functions.

loss function for the discriminator is

$$J_D = -\frac{1}{2}E_x\left[\log D(x)\right] - \frac{1}{2}E_z\left[\log(1 - D(G(z)))\right],$$

while the loss function for the generator is

$$J_G = -\frac{1}{2}E_z\left[\log D(G(z))\right].$$

By minimizing the first loss function w.r.t. the weights of the network D, we encourage the discriminator to output 1 for each training sample, and 0 for each random sample. By minimizing the second loss function w.r.t. the weights of the network G, we encourage the generator to fool D into thinking a random sample is actually a real one observed during training. This is a standard adversarial training setup; see Goodfellow et al. (2014) for more details.

With this straightforward training, however, it is still hard to converge to a suitable balance between the G and D networks, despite the good initialization provided by our autoencoder. This is due to the following reasons. *First*, when mapping a random code z to the manifold, the G network (which is just the recursive decoder) may infer a grossly incorrect hierarchy. The D network finds it easy to reject these implausible hierarchies, and hence does not generate a useful training signal for G. *Second*, the implausible hierarchies generated from random codes may not provide reasonable pathways to back-propagate the loss from D so that G can be tuned properly. *Third*, since the decoding networks in G are split into geometric (e.g. AdjDec) and topological (NodeClsfr) types, they should be tuned separately with different losses deduced from D. To these ends, we devise the following training strategies and priors, to better constrain the training process:

- *Structure prior for G.* In an initial stage, we need to prevent G from mapping a random code z to a severely implausible hierarchy. This is achieved by introducing a strong structure prior to G. We constrain the hierarchies inferred by G to lie in a plausible set. This set includes all hierarchies used to train the autoencoder in Section 4. It also includes all hierarchies inferred by the autoencoder, in test mode, for the training shapes. For each z, we search the plausible hierarchies for the top $K = 10$ ones that best fool the discriminator, minimizing J_G (Figure 10, left). These hierarchies are then used to back-propagate the loss J_G.

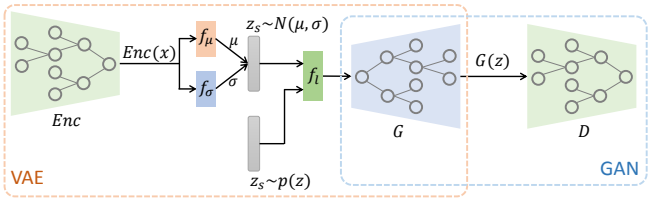


Fig. 11. Confining random codes by sampling from a learned Gaussian distributions based on learned root codes $Enc(x)$. Jointly learning the distribution and training the GAN leads to a VAE-GAN network.

- *Separate geometric and structural training.* Given a selected hierarchy, we first tune the geometric decoders of G via back-propagating the corresponding loss J_G through the hierarchy. This tuning is expected to further fool the discriminator, leading to a higher estimate $D(G(z))$ that $G(z)$ is real (Figure 10, top-right). For each selected hierarchy, with its the newly updated $D(G(z))$, we then tune the structural component, NodeClsfr, of G. This is done by minimizing the classification loss of NodeClsfr at each node in the given hierarchy, using the node type as ground-truth (Figure 10, bottom-right). To favor those hierarchies that better fool D, we weight the loss by $D(G(z))$.
- *Constrained random code sampling.* Given the priors and constraints above, it is still difficult to train G to reconstruct a plausible hierarchy from arbitrarily random codes. Therefore, instead of sampling random codes from an uniform distribution, we sample them from Gaussian distributions around the training samples x. Specifically, G takes samples from a multivariate Gaussian distribution: $z_s(x) \sim N(\mu, \sigma)$ with $\mu = f_\mu(Enc(x))$ and $\sigma = f_\sigma(Enc(x))$. Here, Enc is the recursive encoder originally trained with the autoencoder (before adversarial tuning), running in test mode. f_μ and f_σ can be approximated by two neural networks. We optimize it to minimize the reconstruction loss on x, in addition to the generator loss in the GAN. In fact, the networks Enc and G constitute a variational autoencoder (VAE) if we also tune Enc when learning the parameters of the Gaussian distribution. This leads to an architecture similar to the VAE-GAN proposed by Larsen et al. (2015); see Figure 11.

  Consequently, we also impose the loss function for VAE that pushes this variational distribution $p(z_s(x))$ towards to the prior distribution of the standard normal distribution $p(z)$. In summary, we minimize the following loss function:

$$L = L_{\text{GAN}}(z_p) + \alpha_1 L_{\text{recon}} + \alpha_2 L_{\text{KL}} \quad (1)$$

  The GAN loss is $L_{\text{GAN}} = \log D(x) + \log(1 - D(G(f_l(z_p))))$, with $z_p \sim p(z)$. This loss is minimized/maximized by G/D, respectively. The reconstruction loss is defined as $L_{\text{recon}} = \|G(f_l(z_s(x))) - x\|_2$. f_l is a network used to map a latent code to a root code, before passing the latent code through G. The KL divergence loss is $L_{\text{KL}} = D_{\text{KL}}\left(p(z_s(x)) \,\|\, p(z)\right)$. We set $\alpha_1 = 10^{-2}$ and $\alpha_2 = 10$ in our experiments.

The results of the GAN training process are fine-tuned RvNN decoder modules. The new decoders map any random n-D vector to a structure lying on the plausible manifold. Together with a module to

generate fine-grained part geometry, described in the next section, this constitutes our recursive, generative model of 3D shapes.

6 PART GEOMETRY SYNTHESIS

In the previous sections, we described our generative model of part layouts in shapes. The final component of our framework is a generative model for fine-grained part geometry, conditioned on the part bounding box and layout. Our solution has two components. First, we develop a fixed-dimensional part feature vector that captures both the part's gross dimensions and its context within the layout. Second, we learn a low-dimensional manifold of plausible part geometries while simultaneously also learning a mapping from part feature vectors to the manifold. This mapping is used to obtain the synthesized geometry for a given part in a generated layout. Below, we describe these steps in detail.

Structure-aware recursive feature (SARF). The recursive generator network produces a hierarchy of shape parts, with each internal node in the hierarchy represented by an n-D code. We exploit this structure to define a feature vector for a single part. A natural contextual feature would be to concatenate the RvNN codes of all nodes on the path from the part's leaf node to the root. However, since paths lengths are variable, this would not yield a fixed-dimensional vector. Instead, we approximate the context by concatenating just the code of the leaf node, that of its immediate parent, and that of the root into a $3n$-D feature vector (Figure 12). The first code captures the dimensions of the part's bounding box, and the latter two codes capture local and global contexts, respectively.

SARF to part geometry. In the second stage, we would like to map a SARF feature vector to the synthesized geometry for the part, represented in our prototype as a $32 \times 32 \times 32$ voxel grid. Such a mapping function is difficult to train directly, since the output is very high (8000) dimensional yet the set of *plausible* parts spans only a low-dimensional manifold within the space of all outputs. Instead, we adapt a strategy inspired by Girdhar et al. (2016). We set up a deep, convolutional autoencoder, consisting of an encoder GeoEnc to map the voxel grid to a compact, 32D part code, and a decoder GeoDec to map it back to a reconstructed grid. The learned codes efficiently map out the low-dimensional manifold of plausible part geometries. We use the architecture of Girdhar et al., and measure the reconstruction error as a sigmoid cross-entropy loss. Simultaneously, we use a second deep network GeoMap to map an

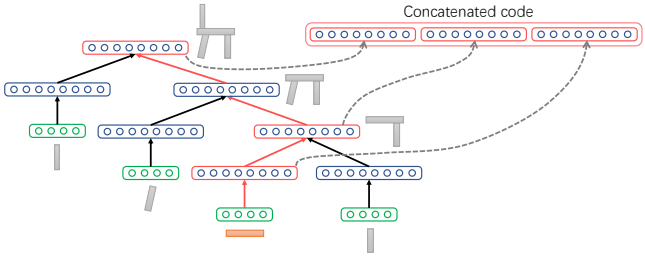


Fig. 12. Construction of structure-aware recursive feature (SARF) for a part in a hierarchy. We concatenate the RvNN codes of the part, its immediate parent, and the root into a fixed-dimensional vector.

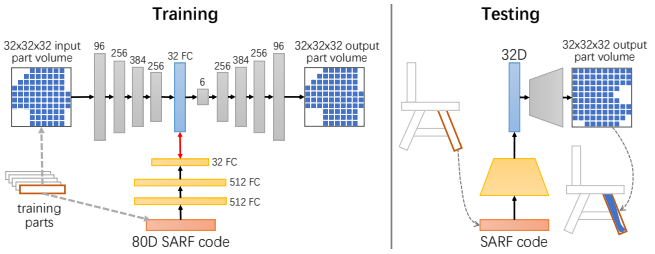


Fig. 13. Training and testing setup for part geometry synthesis.

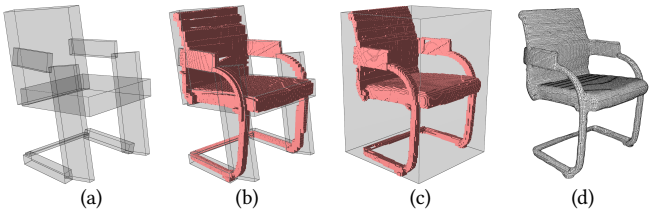


Fig. 14. Geometry synthesis from part structure. Given a generated part structure (a), we synthesize the geometry inside each part box in volumetric representation (b). The per-box volumes are then embedded into a global volume (c) from which we reconstruct the final meshed model (d).

input SARF code to the 32D part code, with both networks accessing the same code neurons. The mapping network employs a Euclidean loss function. We train both networks jointly, using both losses, with stochastic gradient descent and backpropagation. At test time, we chain together the mapping network GeoMap and the decoder GeoDec to obtain a function mapping SARF codes to synthesized part geometry. The training and test setups are illustrated in Figure 13. The synthesis of the overall shape geometry is done by predicting part-wise 3D volumes, which are then embedded into a global volume, from which we reconstruct the final meshed model. See Figure 14 for an example.

7 RESULTS AND EVALUATION

We evaluate our generative recursive model of shape structures through several experiments. First, we focus on validating that our autoencoder-based RvNN learns the "correct" symmetry hierarchies, where correctness could be qualified in different ways, and the resulting codes are useful in applications such as classification and partial matching. Then we test the generative capability of our VAE-GAN network built on top of the RvNN.

Dataset: We collected a dataset containing 1000 3D models from five shape categories: chairs (500), bikes (200), aeroplanes (100), excavators (100), and candelabra (100). These models are collected from the ShapeNet and the Princeton ModelNet. Each model is pre-segmented according to their mesh components or based on the symmetry-aware segmentation utilized in (Wang et al. 2011). The average number of segments per shape is 12 for chairs, 10 for bikes, 7 for aeroplanes, 6 for excavators, and 8 for candelabra. Symmetric parts are counted as distinct. We do not utilize any segment labels.

Our RvNN autoencoder is trained with all shapes in the dataset. The generative VAE-GAN is trained per category, since its training involves structure learning which works best within the same shape

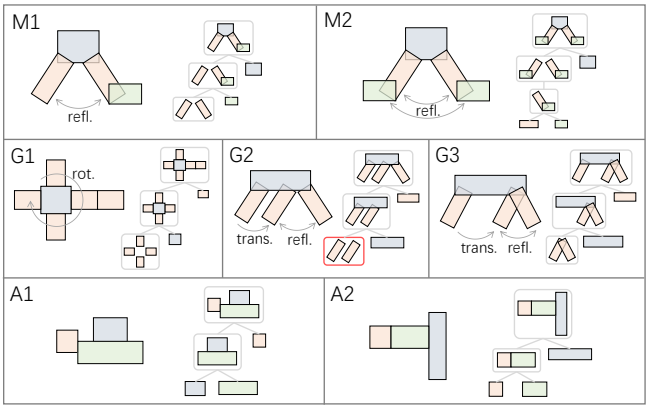


Fig. 15. Our RvNN encoder correctly parses six out of seven 2D box arrangements designed to test handcrafted, perceptually-based grouping rules from (Wang et al. 2011). The G2 rule is violated in our example, with 2-fold translational symmetry (highlighted in the red box) taking precedence over the reflectional one.

category. Part geometry synthesis is trained on all parts from all categories.

Learning recursive grouping rules. In the original work on symmetry hierarchies by Wang et al. (2011), a total of seven precedence rules (labeled M1, M2, G1, G2, G3, A1, and A2; see the Appendix for a reproduction of these rules) were handcrafted to determine orders between and among assembly and symmetry grouping operations. For example, rule A1 stipulates that symmetry-preserving assembly should take precedence over symmetry-breaking assembly and rule M2 states that assembly should be before grouping (by symmetry) if and only if the assembled elements belong to symmetry groups which possess equivalent grouping symmetries. These rules were inspired by Gestalt laws of perceptual grouping (Köhler 1929) and Occam's Razor which seeks the simplest explanation. One may say that they are perceptual and represent a certain level of human cognition.

The intriguing question is whether our RvNN, which is unsupervised, could "replicate" such cognitive capability. To test the rules, we designed seven box arrangements in 2D, one per rule; these patterns are quite similar to those illustrated in Wang et al. (2011). For rule A2, which involves a connectivity strength measure, we simply used geometric proximity. In Figure 15, we show the seven box arrangements and the grouping learned by our RvNN. As can be observed, our encoder correctly parses all expected patterns except in the case of G2, where 2-fold translational symmetry takes precedence over the reflectional one in our example.

Consistency of inferred hierarchies. Our RvNN framework infers hierarchies consistently across different shapes. To demonstrate this, we augment two categories of our segmented dataset – *chair* and *candelabra* – with semantic labels (e.g., for chairs: "seat", "back", "leg", and "arm"). Note that these labels occur at relatively higher levels of the hierarchies, since legs, backs, etc., may be subdivided into smaller parts. If the hierarchies are consistent across shapes, these high-level labels should follow a consistent merging order.

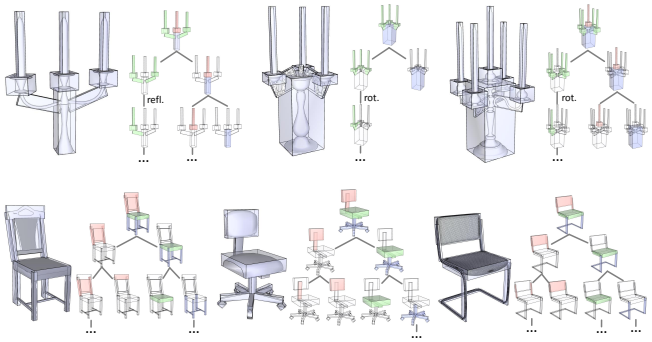


Fig. 16. Inferred hierarchies are consistent across sets of shapes, shown for two shape classes (candelabra and chairs).

For example, the seat and legs should be merged before the seat and back are merged. Let ℓ_p denote the label of part p. Given another label ℓ, let $h(p, \ell)$ denote the shortest distance from p to an ancestor that it shares with a part with label ℓ. Note that $h(p, \ell_p) = 0$ by definition. Let S_ℓ be the set of parts with label ℓ. For labels ℓ_1, ℓ_2, we measure the probability $P_\ell(\ell_1 \prec \ell_2)$ that ℓ_1 is more regularly grouped with ℓ than ℓ_2 as $\sum_{p \in S_\ell} \mathbb{I}(h(p, \ell_1) < h(p, \ell_2))/|S_\ell|$, where $\mathbb{I}$ is the indicator function and the sum is additionally restricted over shapes in which all three labels appear. The overall consistency is estimated as one minus the average entropy over all label triplets:

$$C = 1 + \binom{|L|}{3}^{-1} \sum_{\ell, \ell_1, \ell_2 \in L,\ \ell \neq \ell_1 \neq \ell_2} P_\ell(\ell_1 \prec \ell_2) \log_2 P_\ell(\ell_1 \prec \ell_2)$$

The average consistency over the two categories of training shapes was measured as 0.81, and over the two categories of test shapes as 0.72. The high values show that our RvNN infers hierarchies consistently across different shapes. Figure 16 shows several pairs of shapes with consistent inferred hierarchies.

Classification of shape structures. Our autoencoder generates compact encodings for shapes segmented into arbitrary numbers of parts, via a recursively inferred hierarchy. To test whether these codes effectively characterize shapes and shape similarities, we conducted a *fine-grained* shape classification experiment for each of four classes: airplane, chair, bike, and candle. The sub-classes were: airplane – 5 classes including jet, straight-wing, fighter, delta-wing, swept-wing; chair – 5 classes including armchair, folding, swivel, four-leg, sofa; bike – 4 classes including motorcycle, casual bicycle, tricycle, mountain bike; and candelabra – 3 classes including with arms, w/o arm, with two-level arms. To represent each shape, we used the average of all codes in the shape's hierarchy, which, as in Socher et al. (2011), we found to work better than just the root.

Following the standard protocol for each category of shapes, we hold out one shape in turn, and sort the remaining shapes by increasing the L_2 distance between average codes, terminating the results by a variable upper limit on the distance. The number of results from the class of the query shape are considered as true positives.

We show precision-recall plots for four classes of interest in Figure 17. The average accuracy of (subclass) classification over all four classes is 96.1%.

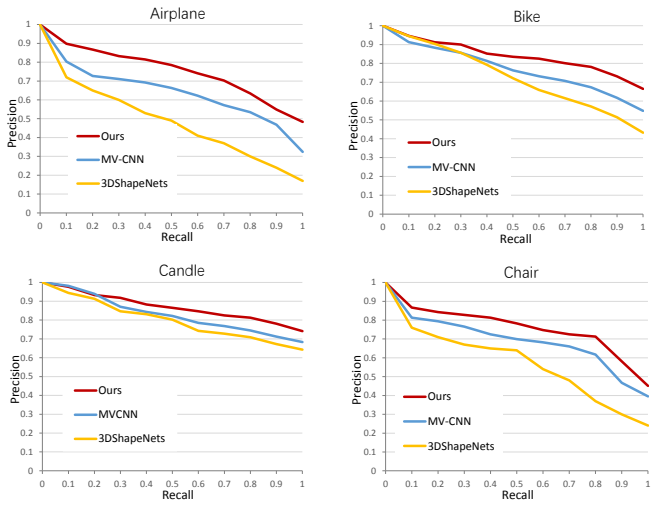


Fig. 17. Precision-recall plots for classification tasks.

As baselines, we show the performance of two state-of-the-art descriptors on this task (Su et al. 2015; Wu et al. 2015). This is not an entirely equal-grounded comparison: our method leverages a prior segmentation of each shape into (unlabeled) parts, whereas the baseline methods do not. However, our method does not consider any fine-grained part geometry, only oriented bounding box parameters. The considerable improvement of our method over the baselines demonstrates that gross structure can be significantly more important for shape recognition than fine-grained geometry, and accurate and consistent identification of part layouts can be the foundation of powerful retrieval and classification methods.

Partial structure matching. While the previous experiment tested full shape retrieval, it is also interesting to explore whether subtree codes are sufficiently descriptive for part-in-whole matching. As before, we use the average of codes in a subtree as the feature for the subtree. Figure 18 contains some partial retrieval results, showing that our method correctly retrieves subparts matching the query.

Shape synthesis and interpolation. Our framework is generative, and can be used to synthesize shapes from the learned manifold in a two-step process. First, the VAE-GAN network is sampled using a random seed for a hierarchical bounding box layout. Second, the leaf nodes of the hierarchy are mapped to fine-grained voxelized geometry, which is subsequently meshed. Several examples of synthesized shapes are shown in Figure 19.

Our model can also be used to interpolate between two topologically and geometrically different shapes. For this task, we compute the root codes of two shapes via inferred hierarchies. Then, we linearly interpolate between the codes, reconstructing the shape at each intermediate position using the synthesis procedure above. Although intermediate codes may not themselves correspond to root codes of plausible shapes, the synthesis procedure projects them onto the valid manifold by virtue of the VAE-GAN training. We demonstrate example interpolations in Figure 20. Note that our model successfully handles topological changes both in the part layout and within parts, while maintaining symmetry constraints.

Unlike Jain et al. (2012), we do not require prior knowledge of part hierarchies. Unlike both Jain et al. and Alhashim et al. (2014), we do not require part correspondences either, and we can handle smooth topological changes in individual parts.

Implementation and Timing. Our RvNN and VAE-GAN are implemented in MATLAB. The geometry synthesis model is implemented using the MatConvNet neural network library. Pre-training the autoencoder (Section 4) took 14 hours. Adversarial fine-tuning (Section 5) took about 20 hours for each shape class. Training the part geometry synthesis network (Section 6) took 25 hours. Mapping a random code vector to the manifold of plausible structures to synthesize a hierarchy takes 0.5 seconds, and augmenting it with synthesized fine-grained part geometry takes an additional 0.2 seconds per part.

8 DISCUSSION, LIMITATION, AND FUTURE WORK

With the work presented, we have only made a first step towards developing a structure-aware, generative neural network for 3D shapes. What separates our method apart from previous attempts at using neural nets for 3D shape synthesis is its ability to learn, without supervision, and synthesize *shape structures*. It is satisfying to see that the generated 3D shapes possess cleaner part structures, such as symmetries, and more regularized part geometries, when compared to voxel fields generated by previous works (Girdhar et al. 2016; Wu et al. 2016). What is unsatisfying however is that we decoupled the syntheses of structure and fine geometry. This hints at an obvious next step to integrate the two syntheses.

The codes learned by our RvNN do combine structural and geometric information into a single vector. Through experiments, we have demonstrated that the hierachical grouping learned by the RvNN appears to conform to perceptual principles as reflected by the precedence rules handcrafted by Wang et al. (2011). The codes also enable applications such as fine-grained classification and partial shape retrieval, producing reasonable results. However, the internal

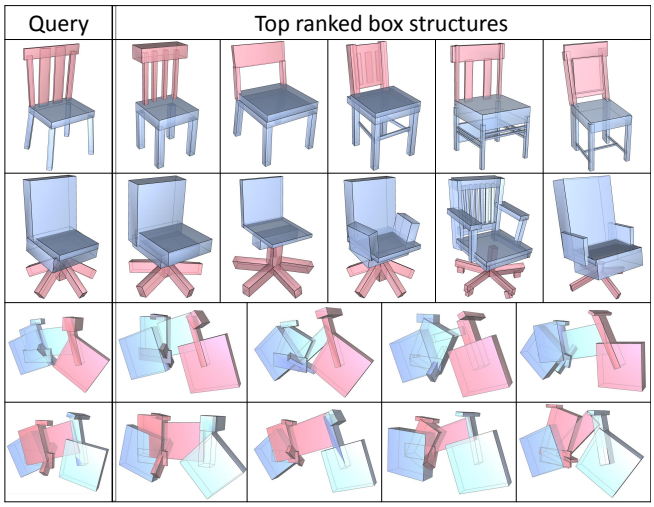


Fig. 18. Partial structure retrieval results for two shape classes (chair and bicycle). The query and matching parts are highlighted in red.

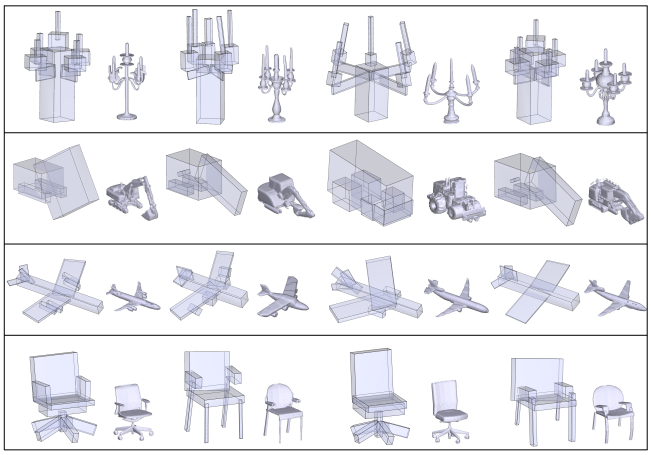


Fig. 19. Examples of shapes synthesized from different classes.

mechanisms of the code and precisely how it is mixing the structural and geometric information is unclear. The fact that it appears to be able to encode hiearachies of arbitrary depth with a fixed-length vector is even somewhat mysterious. An interesting future work would be to "visualize" the code to gain an insight on all of these questions. Only with that insight would we be able to steer the code towards a better separation between the parts reflecting the structure and the parts reflecting low-level geometry.

Our current network still has a long way to go in fully mapping the *generative structure manifold*. We cannot extrapolate arbitrarily – we are limited to a VAE-GAN setup which samples codes similar to, or in between, the exemplars. Hence, our synthesis and interpolation are confined to a local patch of that elusive "manifold". In fact, it is not completely clear whether the generative structure space for a 3D shape collection with sufficiently rich structural variations is a low-dimensional manifold. Along similar lines, we have not discovered flexible mechanisms to generate valid codes, e.g., by applying algebraic or crossover operations, from available codes. All of these questions and directions await future investigations. It would be interesting to thoroughly investigate the effect of code length on structure encoding. Finally, it is worth exploring recent developments in GANs, e.g. Wasserstein GAN (Arjovsky et al. 2017), in our problem setting. It would also be interesting to compare with plain VAE and other generative adaptations.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable comments and suggestions. We are grateful to Yifei Shi, Min Liu and Yizhi Wang for their generous help on data preparation and result production. Jun Li is a visiting PhD student of University of Bonn, supported by the China Scholarship Council. This work was supported in part by NSFC (61572507, 61532003, 61622212), an NSERC grant (611370), NSF Grants IIS-1528025 and DMS-1546206, a Google Focused Research Award, and awards from the Adobe, Qualcomm and Vicarious corporations.

REFERENCES

Ibraheem Alhashim, Honghua Li, Kai Xu, Junjie Cao, Rui Ma, and Hao Zhang. 2014. Topology-Varying 3D Shape Creation via Structural Blending. In *Proc. SIGGRAPH.*

Brett Allen, Brian Curless, and Zoran Popović. 2003. The Space of Human Body Shapes: Reconstruction and Parameterization from Range Scans. In *Proc. SIGGRAPH.*

Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. In *Proc. SIGGRAPH.*

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. *arXiv preprint arXiv:1701.07875* (2017).

Melinos Averkiou, Vladimir Kim, Youyi Zheng, and Niloy J. Mitra. 2014. ShapeSynth: Parameterizing Model Collections for Coupled Shape Exploration and Synthesis. *EUROGRAPHICS* (2014).

Volker Blanz and Thomas Vetter. 1999. A Morphable Model for the Synthesis of 3D Faces. In *Proc. SIGGRAPH.* 187–194.

Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. 2010. A Connection Between Partial Symmetry and Inverse Procedural Modeling. In *Proc. SIGGRAPH.*

Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. 2011. Probabilistic Reasoning for Assembly-Based 3D Modeling. In *Proc. SIGGRAPH.*

David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Al'an Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Proc. NIPS.*

Noa Fish, Melinos Averkiou, Oliver van Kaick, Olga Sorkine-Hornung, Daniel Cohen-Or, and Niloy J. Mitra. 2014. Meta-representation of Shape Families. In *Proc. SIGGRAPH.*

Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. 2016. Learning a predictable and generative vector representation for objects. In *Proc. ECCV.*

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proc. NIPS.*

Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep Convolutional Networks on Graph-Structured Data. *CoRR* abs/1506.05163 (2015). http://arxiv.org/abs/1506.05163

Haibin Huang, Evangelos Kalogerakis, and Benjamin Marlin. 2015. Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces. In *Proc. SGP.*

Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. 2012. Exploring Shape Variations by 3D-Model Decomposition and Part-based Recombination. In *EUROGRAPHICS.*

Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH)* 31, 4 (2012).

Vladimir G. Kim, Wilmot Li, Niloy J. Mitra, Siddhartha Chaudhuri, Stephen DiVerdi, and Thomas Funkhouser. 2013. Learning Part-based Templates from Large Collections of 3D Shapes. In *Proc. SIGGRAPH.*

W. Köhler. 1929. *Gestalt Psychology.* Liveright.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. 2015. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300* (2015).

Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. 2015. Geodesic convolutional neural networks on Riemannian manifolds. In *ICCV Workshops.*

Niloy Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, and Martin Bokeloh. 2013. Structure-aware shape processing. In *Eurographics State-of-the-art Report (STAR).*

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural Modeling of Buildings. In *Proc. SIGGRAPH.*

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *Proc. ICML.*

Maks Ovsjanikov, Wilmot Li, Leonidas Guibas, and Niloy J. Mitra. 2011. Exploration of Continuous Variability in Collections of 3D Shapes. In *Proc. SIGGRAPH.*

Charles R. Qi, Hao Su, Matthias Niessner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. 2016. Volumetric and multi-view CNNs for object classification on 3D data. In *Proc. CVPR.*

Adriana Schulz, Ariel Shamir, Ilya Baran, David Isaac William Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. 2016. Retrieval on Parametric Shape Collections. *ACM Trans. Graph. (to appear)* (2016).

Thomas Serre. 2013. Hierarchical Models of the Visual System. In *Encyclopedia of Computational Neuroscience*, Dieter Jaeger and Ranu Jung (Eds.). Springer NY.

Ayan Sinha, Jing Bai, and Karthik Ramani. 2016. Deep Learning 3D Shape Surfaces using Geometry Images. In *Proc. ECCV.*

Richard Socher. 2014. *Recursive Deep Learning for Natural Language Processing and Computer Vision.* Ph.D. Dissertation. Stanford University.

Richard Socher, Brody Huval, Bharath Bhat, Christopher D. Manning, and Andrew Y. Ng. 2012. Convolutional-Recursive Deep Learning for 3D Object Classification. In *Proc. NIPS.*

Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proc. ICML.*

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. 2015. Multi-view convolutional neural networks for 3D shape recognition. In *Proc. ICCV*.

Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. 2012. Learning Design Patterns with Bayesian Grammar Induction. In *Proc. UIST*. 63–74.

Jerry O. Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. 2009. Exploratory Modeling with Collaborative Design Spaces. In *Proc. SIGGRAPH Asia*.

Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. 2017. Learning Shape Abstractions by Assembling Volumetric Primitives. In *Proc. CVPR*.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. 2016a. WaveNet: A Generative Model for Raw Audio. *CoRR* abs/1609.03499 (2016). http://arxiv.org/abs/1609.03499

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016b. Pixel Recurrent Neural Networks. *CoRR* abs/1601.06759 (2016). http://arxiv.org/abs/1601.06759

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. 2016c. Conditional Image Generation with PixelCNN Decoders. *CoRR* abs/1606.05328 (2016). http://arxiv.org/abs/1606.05328

Oliver van Kaick, Kai Xu, Hao Zhang, Yanzhen Wang, Shuyang Sun, Ariel Shamir, and Daniel Cohen-Or. 2013. Co-Hierarchical Analysis of Shape Structures. In *Proc. SIGGRAPH*.

Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiquan Cheng, and Yueshan Xiong. 2011. Symmetry Hierarchy of Man-Made Objects. In *EUROGRAPHICS*.

Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. 2016. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *Proc. NIPS*.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets: A deep representation for volumetric shapes. In *Proc. CVPR*.

Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. 2016. Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision. In *Proc. NIPS*.

M. E. Yumer and N. J. Mitra. 2016. Learning Semantic Deformation Flows with 3D Convolutional Networks. In *Proc. ECCV*.

APPENDIX: PRECEDENCE RULES FOR SYMMETRY HIERARCHY

We reproduce the precedence rules stipulated in Wang et al. (2011) for sorting symmetry grouping and assembly operations:

M1 (Grouping before assembly): Grouping by symmetry takes precedence over assembly operations, with an exception given by the next rule (**M2**).

M2 (Assembly before grouping): Assemble before grouping if and only if the assembled nodes belong to symmetry cliques which possess equivalent grouping symmetries.

G1 (Clique order): If there are still symmetry cliques of order greater than two in the contraction graph, then higher-order cliques are grouped before lower-order ones.

G2 (Reflectional symmetry): If there are only order-2 cliques in the graph, then group by reflectional symmetry before rotational symmetry and translational symmetries.

G3 (Proximity in symmetry clique): If **G1** and **G2** cannot set a precedence, e.g., between rotational and translational symmetries of the same order, then grouping of part ensembles closer in proximity takes precedence.

A1 (Symmetry preservation): Symmetry-preserving assembly takes precedence over symmetry-breaking assembly.

A2 (Connectivity strength): If **A1** cannot set a precedence, then order assembly operations according to a geometric *connectivity strength* measure.

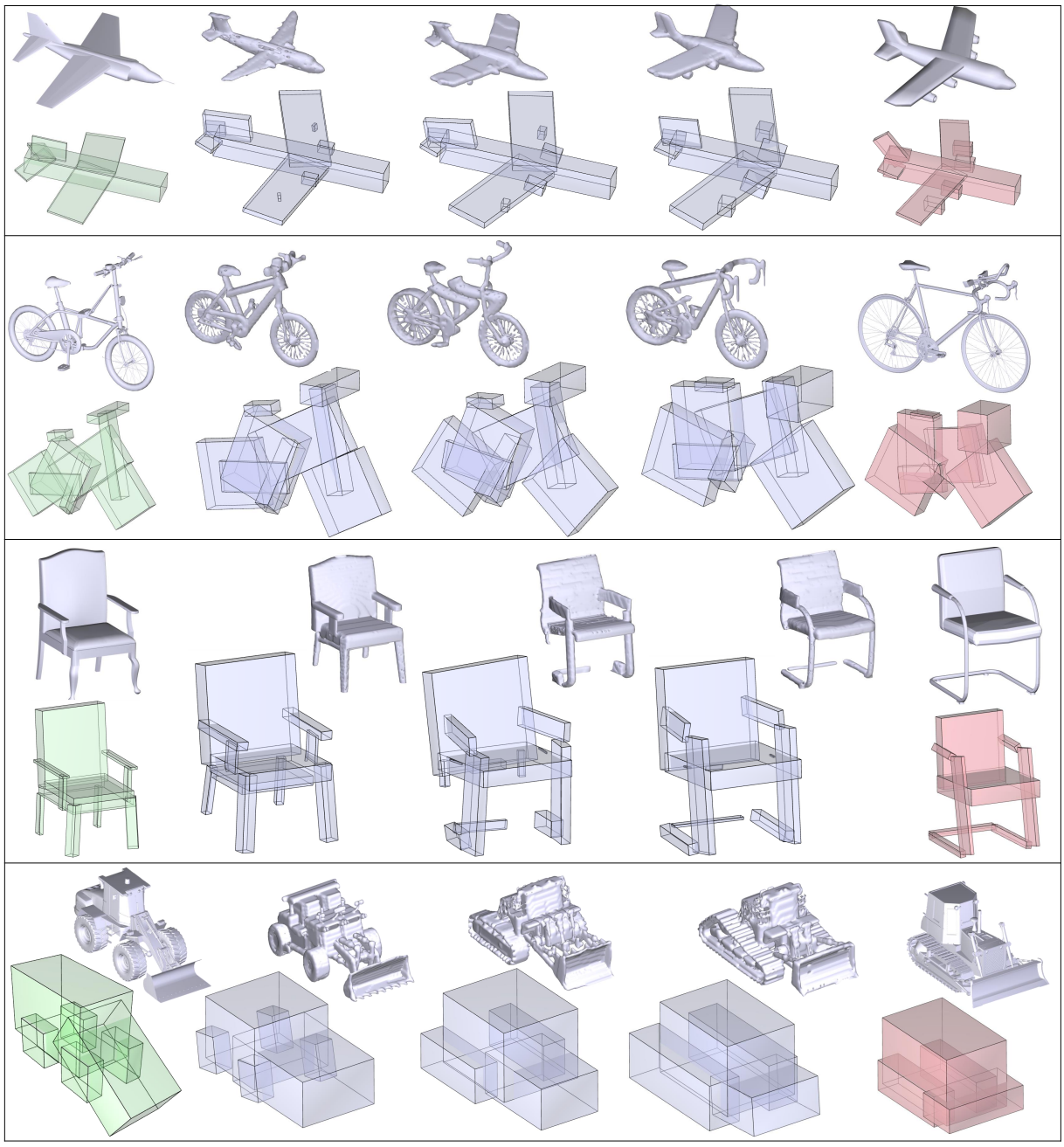


Fig. 20. Linear interpolation between root codes, and subsequent synthesis, can result in plausible morphs between shapes with significantly different topologies.