

# String-Based Synthesis of Structured Shapes

Javor Kalojanov<sup>1</sup> Isaak Lim<sup>1</sup> Niloy Mitra<sup>2</sup> Leif Kobbelt<sup>1</sup>

<sup>1</sup> Visual Computing Institute, RWTH Aachen University <sup>2</sup> UCL London

## Abstract

We propose a novel method to synthesize geometric models from a given class of context-aware structured shapes such as buildings and other man-made objects. The central idea is to leverage powerful machine learning methods from the area of natural language processing for this task. To this end, we propose a technique that maps shapes to strings and vice versa, through an intermediate shape graph representation. We then convert procedurally generated shape repositories into text databases that, in turn, can be used to train a variational autoencoder. The autoencoder enables higher level shape manipulation and synthesis like, for example, interpolation and sampling via its continuous latent space. We provide project code and pre-trained models.

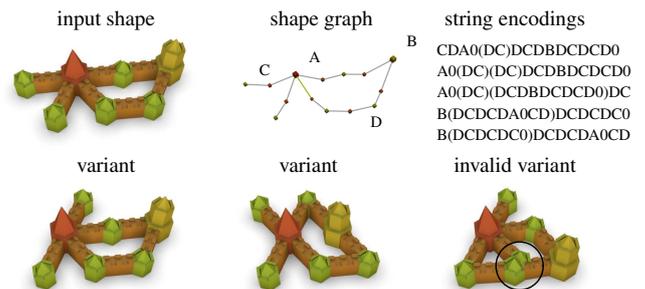
Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

## 1. Introduction

Automatic 3D model synthesis is one of the most challenging problems in computer graphics. The standard approach to creating virtual 3D shapes involves the use of complex modeling software by skilled artists and can be time consuming and expensive. Recent data-driven methods for shape synthesis such as [LXC\*17, SSK\*17] address this problem by training neural networks to perform tasks like shape assembly and interpolation. Similar to how humans approach 3D modeling, these methods consider shapes as being composed out of building blocks. An important limitation of previous approaches, is the limited structural complexity of the produced models. This problem can be attributed to the types of models available in shape repositories, but the more significant hurdle is the inability of neural networks to work on irregularly structured input. In this work we propose a combination of a model-based and a data-driven method that addresses both creating large number of structured shapes of arbitrary complexity and representing the topology of structure graphs via strings.

In order to address the problem of creating structural variations in a principled way, we consider shapes that can be assembled from building blocks according to an initially unknown *tiling* shape grammar: a set of rules describing how to assemble the elementary pieces locally. To this end, we use as an input a small number of shapes segmented into rigid, exchangeable parts (building blocks). These define the set of plausible shape variants: each local configuration of components is considered valid only if the same or a very similar configuration also occurred in the input models.

However, unlike context-free shape grammars, the grammar assembly rules here are not sufficient to derive a recipe for creating shape variants. The difference is that context-free production rules



**Figure 1:** A simple shape, its shape graph, string representations of the graph, and shape variants. Both mappings from graph topology to strings, and back to geometric shapes are not uniquely defined.

are hierarchical and each sequence of applications can be terminated resulting in a globally valid result. Tiling grammar rules describe only local configurations without termination or correctness guarantees. For example, the last shape in Figure 1 contains an error that cannot be resolved by modifying only the pieces of the model that cause the invalid intersection. Hence, with growing amount of components, it quickly becomes impossible for probabilistic procedural modeling methods to generate shapes without violating the tiling grammar. This makes the problem very difficult, but also interesting, since the resulting shapes can contain cycles and regular grid patterns that appear very often in man made objects.

In this work we tackle the problem of 3D modeling using machine learning methods for natural language processing: a domain that also deals with non-context free constructions. To this end, we adapt the simplified molecular-input line-entry system

(SMILES) [Wei88] for encoding the topology of graphs representing shapes constructed out of building blocks (see Figure 1). This string conversion method enables the application of recurrent neural networks for learning the structure of geometric models.

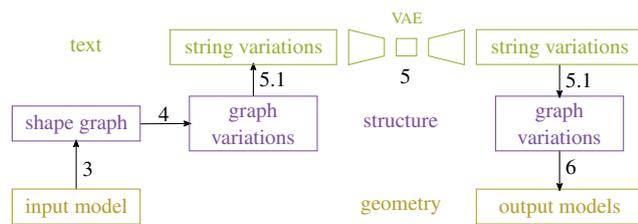
We demonstrate that it is possible to train a variational autoencoder (VAE) [KW13] on strings representing the structure of the randomly generated 3D models similar to [BVV\*15, GDH\*16] and generate geometric shapes constructed out of rigid building blocks. The variational autoencoder is used to map vectors representing shapes to a continuous latent space and decode points in latent space to shape representations. This facilitates high level synthesis operation such as interpolation and sampling of discrete shapes.

Synthesizing geometric shapes from a given graph topology produced in the previous step is also very hard even if there is a unique way to attach pairs of pieces together [SSK\*17], which is not the case here. We address the problem by reducing it to a classification task and train a neural network to estimate how the parts (or graph nodes) are positioned relative to each other in space.

Since existing shape collections usually consist of samples with small or no structural variations, we first have to address the problem of efficiently creating large sets of structured shapes. We extend and automate the procedural modelling method based on partial graph symmetries [BWS10, LVW\*15] and use it for creating sets of samples sufficiently large to enable training machine learning models. As a result, we present a fully automatic shape synthesis framework that can generate large collections of shapes with significant structural variations using just one or two example models as input.

This paper contains three important contributions (see Figure 2):

- A model-based, automatic method for generating structured shape variations given just one or a few examples.
- Adapting a data-driven method from natural language processing, for procedural modeling of 3D geometry.
- A data-driven method for instantiation of shape graph topologies in  $\mathbb{R}^3$  via edge classification.



**Figure 2:** Overview: We convert 3D models into graphs and procedurally create structural variations. These are converted to text databases and used to train a variational autoencoder (VAE). Finally, the training and additional strings representing graph topologies are instantiated back into 3D models. The number next to each arrow indicates the corresponding section in the paper.

## 2. Related Work

**Part-based Modeling** Funkhouser et al. [FKS\*04] propose a data-driven shape synthesis method in which a user can cut out parts of

models in an existing shape database and then use them to compose new shapes. Subsequent works expand on various aspects of the method, e.g., using 2D sketches [LF08] for shape retrieval. Further part-based modeling approaches have been recently surveyed in [MWZ\*13].

Several recent related works on procedural modeling [TLL\*11, TYK\*12, RMGH15, RJT18] consider hierarchical segmentations and create further shapes using *context-free* shape grammars. These works address the problem of generating desirable variants via random applications of grammar rules and provide better means for user control compared to our method. In contrast, the non-context-free shape grammars we consider here have higher expressive power and are easier to compute automatically. Our method does not require the shape grammar to be provided as input, which is beneficial for users without 3D modeling skills.

Our method is closely related to inverse procedural modeling techniques for generating variations from a single example like the works by Bokeloh et al. [BWS10, BWKS11] and Liu et al. [LVW\*15]. These methods consider shape decompositions into building blocks that can be assembled into variants. The segmentations are either computed via symmetry detection [BWS10, BWKS11], or given as input [LVW\*15].

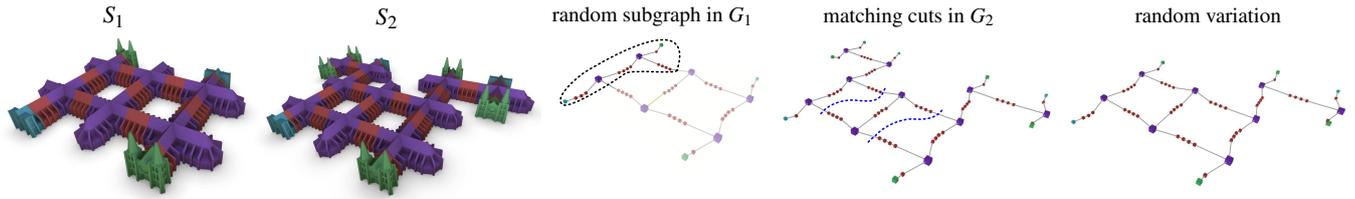
Similar to Liu et al. [LVW\*15] we use partial graph symmetries to generate random variations by splitting and merging example part compositions. However, as detailed in Section 4, we propose an alternative sampling algorithm for faster, automatic and more robust sampling of large amounts of shape variations.

**Data-Driven Shape Processing** Xu et al. [XKHK17] survey recent data-driven methods for shape processing including part based modeling [FKS\*04, KJS07], sketch-based modeling [FWX\*13, XXM\*13] and shape editing [XZZ\*11, ZCOM13]. Similar to us, these methods attempt to simplify 3D modeling by leveraging existing data. The differences are that we generate the example datasets from a single or several examples.

Recently, Nash and Williams [NW17] also proposed the use of a variational autoencoder (VAE) for shape synthesis. Their work however differs significantly from our method since they train a VAE on a set of shapes with the same structure. Other related deep generative models for structure variations consider tree [LXC\*17] or graph [SSK\*17] representations of comparably simpler objects like chairs and vehicles. The main disadvantage of these related methods compared to ours is the constrained structural complexity of the generated shapes which limits them to creating style variations of the training examples. On the other hand, our method enables synthesis of shapes with arbitrary topology thanks to vectorizing shape graphs using sequences.

## 3. Tiling Grammars and Structure Graphs

In the absence of predefined rules for procedural modeling, we have to find a way to characterize the space of valid shape variations. A sound way to address this challenge is to use the input models as examples for a set of valid assembly rules and only use these for shape synthesis. In other words, as a grammar we use the subset of the assembly rules already observed in the input models.



**Figure 3:** Graph-based shape sampling. The input models  $S_1, S_2$  are converted into shape graphs  $G_1, G_2$ . New shape variations are created automatically via random subgraph sampling and subgraph matching.

More specifically, each part of a certain type in the output shape has to have a matching number and types of neighbors. Pieces are considered as neighbors if and only if their surfaces intersect. This assumption is quite general and does not impose restrictions on the shape of the parts or their intersections.

For example, the grammar rules derived from the models in Figure 3 allow each of the green, purple, cyan building blocks to only attach to brown pieces and to have exactly one (green and cyan) or three (purple) neighbors. The brown building blocks have exactly two neighbors and can attach to pieces of every type.

We represent the structure of each shape via a undirected *shape graph*. The nodes of the graph correspond to parts. Every pair of connected pieces is represented with an edge between their corresponding nodes. We transfer the part labels to the graph by assigning a *type* to each graph node. The shape graph abstracts away the geometry information and can be used to reduce the shape synthesis problem to graph sampling and subgraph matching.

We extract these grammar rules regarding graph topology (number and types of neighbors for a given node type) from the shape graphs of the input models and use them to detect variations with invalid graphs. The same set of rules can be used to check for self intersecting assemblies, by verifying the shape graph from the geometric shape after creating it. Later on, the example models are used to derive a set of allowed approximate positions and orientations of connected pairs of pieces of certain types. Therefore, the efficiency of our shape synthesis approach depends on the ability to quickly compute the shape graph of input models of arbitrary complexity. We solve this using collision detection and by assuming that a pair of parts is connected if and only if their corresponding triangle meshes intersect each other.

#### 4. Graph-Based Shape Sampling

Assuming a pair of input models  $S_1, S_2$ , our shape augmentation method consists of the following steps (also see Figure 3). We compute graphs  $G_1, G_2$  for each shape using collision detection. We randomly sample subgraphs in the first shape ( $S_1$ ) and check if a subgraph with matching boundary exists in the second shape ( $S_2$ ). We merge each matching pairs of subgraphs and check the resulting model against the grammar rules extracted from the shape graphs  $G_1, G_2$ . We output valid shape variations with different node type histograms to avoid re-discovering the same shape multiple times.

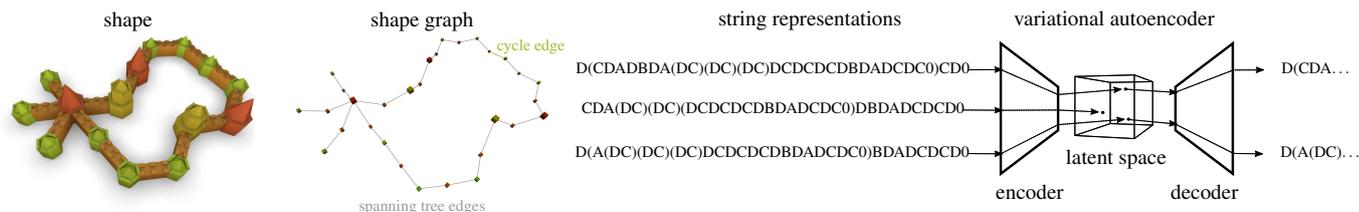
**Subgraph Sampling** In essence, our shape generation approach splits each of the input models into two or more pieces that

are recombined into a new shape. The main difficulty is to efficiently discover compatible sub-graphs and avoid duplicates. Liu et al. [LVW\*15] propose to enumerate all possible graph operations in  $O(n^3)$  worst case complexity and  $O(n^2)$  (quadratic in the number of graph nodes) expected complexity. Instead of exhaustively searching for all possible operations we sample  $p$  random subgraphs in parallel and keep  $p$  linear in the number of available processors (or threads). Doing so, we spend significantly less effort sampling the shape variants created by merging exactly two subgraphs of the input shapes. By recursively applying this procedure on newly created variations we can sample shapes made out of more than two subgraphs from the input.

**Subgraph Matching** Each subgraph sample  $s$  from the first shape graph  $G_1$ , is matched against the the second shape graph  $G_2$ . We search for a set of nodes in  $G_2$  corresponding to the nodes in  $G_1$  connected via boundary edges of  $s$ . Both the node types as well as *all* pairwise distances between the centroids of the respective parts have to match. The latter is a necessary condition for the existence of a rigid transformation  $T$  that attaches (docks) the geometry represented by  $s$  to  $S_2$  without violating the grammar rules at the boundary [BWS10]. We randomize our search and terminate after finding the first set of corresponding boundary nodes in  $G_2$  in order to speed up the sampling process in the presence of multiple matches.

**Docking Transformation Estimation** Each subgraph with matching boundary needs to be transformed in order to attach it to  $S_2$ . Aligning the centroids of the boundary pieces from  $S_1$  to their matching counterparts in  $S_2$  can be reduced to finding a transformation that minimizes the distances between the two sets of points in the least square sense. The problem can be reduced to computing an SVD (singular value decomposition) of a  $3 \times 3$  covariance matrix (see the supplemental notes to [SA07] for a detailed proof).

**Validation** After creating a variation by replacing a subgraph of  $G_2$  with a subgraph of  $G_1$  we have to verify that the new shape does not violate the grammar rules. This can happen, for example, if some of the nodes in the interiors of the attached subgraphs interfere. Note that we need to test the geometric shape of the variant. The generated graph topology is always correct by construction: we only replace nodes on the boundary of the two subgraphs with nodes of the same type. In order to guarantee that the newly created shape is geometrically feasible, we generate it and compute its shape graph. Both interfering and misaligned parts will pro-



**Figure 4:** Example shape vectorization using SMILES strings. All cycles in the shape graph are broken by removing a random edge (lime green). The remaining spanning tree (gray) is encoded in a string using letters for nodes and brackets to represent branching. The cycle edges are reintroduced by adding a number next to their adjacent nodes in the string. The resulting strings are used to train a variational autoencoder. We train on multiple shape variants composed out of the same building blocks.

duce node configurations that are not allowed by the tiling grammar causing the shape to be discarded as invalid.

We avoid duplicating output models by conservatively discarding shapes with identical node type histograms. Removing duplicates is important if the input shapes have identical subgraphs, in which case multiple subgraph replacement operations yield copies of the original models.

## 5. Variational Autoencoder for Tiling Grammars

The goal of the above shape sampling method is to automatically generate large amounts of models with structural variations, and use them as training sets for machine learning. Particularly interesting is the sequence modeling approach used in [GDH\*16, BVV\*15], which is based on sampling from a continuous structure representation obtained from the latent space of a variational autoencoder (VAE). In this work we generalize the method and apply it for synthesis of arbitrary geometric shapes constructed according to a tiling shape grammar.

The variational autoencoder [KW13] is a generative probabilistic model, which implicitly describes the joint probability distribution over the dataset and corresponding latent variables  $p(x, z)$ . An encoder network is used to approximate  $p(z|x)$  with  $q(z|x)$  by mapping data samples to distributions in latent space (typically gaussian). A decoder network models  $p(x|z)$  by mapping samples from  $q$  in latent space to an output distribution (typically consisting of categorical distributions). Here, the input dataset is a collection of sequences describing the shape graph topology of a set of geometric models (see Figure 4). After training the network with a representative set of sample values one can use the decoding part on points of the latent space not present in the training set. This facilitates operations like sampling, or interpolation of geometric shapes (molecule formulae or sentences in [GDH\*16, BVV\*15]).

### 5.1. Shape Graph Vectorization

In order to use a set of shapes as an input for a variational autoencoder we need to first convert them to vectors. Since we are interested in learning the structure of the shape, i.e., how to construct a model out of building blocks, we want to convert the shape graph to a vector. SMILES strings [Wei88] are a standard method for molecule representation, which is based on enumerating nodes

as they appear in a spanning tree of the molecule graph. The conversion algorithm is general enough to be applied to arbitrary graphs, and we employ an analogous conversion method.

We assign a letter to each node type in the shape graph and compute a string by depth-first traversal of the graph. Multiple subtrees with the same root are enclosed by brackets (except the last subtree at each node). All remaining (cycle) edges in the graph are recorded via number suffixes at both end-nodes. The resulting strings are then trivially converted to one-hot vectors that can be used as input to a sequence autoencoder (Figure 4).

Since we are interested in shape synthesis and do not have access to large bodies of training data, we prefer to compute multiple strings for each shape and use all of them for learning. Intuitively, we try to train the autoencoder to distinguish strings that can represent valid shapes from those that cannot, and accept the slight drawback of representing the same valid example multiple times. After training, the decoding part of the neural network is not able to convert all points in latent space into strings representing valid shapes. Therefore, we have to verify decoded samples against the tiling grammar and discard invalid ones. This can be performed by the same procedure used to verify shape graphs of randomly created variations.

### 5.2. Latent Space Sampling

Even though large regions of points in latent space do not decode to valid strings (according to the grammar), it is possible to create shape variations by sampling and decoding random latent points. Our hypothesis is that valid shapes are mapped to a manifold by the encoding half of the autoencoder. If true, we are likely to find latent points corresponding to valid strings in the neighborhoods of the images of training samples. We experimented with three sampling strategies and observed results that support this assumption.

**Point Perturbations** We sample the neighborhood around a latent point  $p_v$  corresponding to a valid training example with string  $s_v$ . We add a random offset vector  $r_1$  to  $p_v$  and decode the resulting point  $p_r := p_v + r_1$ . Let  $p_r$  decode to a string  $s_r$ . If  $s_r = s_v$  or  $s_r$  is not valid according to the grammar rules, we re-sample  $p_r$  using a new offset vector  $r_2$  with increased magnitude in each dimension. We repeat this procedure until we reach a maximum magnitude (we used 0.25) or find a valid string not equal to the known  $s_v$ .

**Linear Interpolation** We interpolate points on the line segment between the images of pairs of training samples. We sample points on the line segment on equal intervals and sample their neighborhoods with the previous strategy, using smaller offset vector magnitudes. Usually this sampling strategy yielded valid strings close to the endpoints of the line segments, which is in line with the hypothesis of valid latent point living close to a manifold in latent space.

**Latent Path Sampling** The results of the previous sampling strategies motivated us to further restrict sampling locations near “valid” points in latent space. Instead of sampling point on the line segment between pairs of encoded training samples, we construct a path (of length 32) by repeatedly splitting the current segments with the training samples closest to both endpoints of each segment. We then sample each path segment equidistantly and sample in the resulting neighborhoods analogous to the linear interpolation case. Path sampling delivered more valid samples compared to linear interpolation, which in turn was slightly more productive than simply perturbing points.

### 5.3. Latent Space Structuring

The main motivation behind our learning method is the continuous shape representation provided by the variational autoencoder, which facilitates high-level shape synthesis operations such as sampling and interpolation. In this Section, we describe how to equip the latent space with a metric that assigns a semantic or intuitive meaning to geodesic distances on that (unknown) manifold of samples decoding into valid strings. This can be achieved either by a *continuous* deformation that brings samples corresponding to similar shapes closer together or by a *discrete* structure (graph) that connects samples of similar shapes.

We demonstrate an example by introducing a pairwise histogram similarity metric

$$E(x, y) = \frac{1}{N_x + N_y} \sum_{T \in Types} |T_x - T_y|,$$

where  $N_x$  is the number of building blocks in shape  $x$ , and  $T_x$  is the number of building blocks of type  $T$  in the shape.

To improve the efficiency of shape sampling, we implemented an alternative method for latent space exploration based on topological graph search. A conceptually similar approach has been proposed in [BYMW13]. Here, we compute a set of (random and/or training) points that decode to valid strings and organized them as nodes in a search graph. We connect each node with a user defined number of closest neighbors (w.r.t. Euclidean distance in latent space) and every other node in the graph that decodes to an *equivalent* string, i.e., a string representing the same graph topology. Because each shape is represented via multiple strings, we kept the additional degree of the nodes low (between 3 and 5) in order to avoid close to fully connected graphs. Using topological search operations on this graph is more efficient than blindly sampling in latent space, and allows incorporating application dependent objectives as edge weights. We then assign edge weights using the similarity metric, denoted as  $E$ .

This allows formulating interpolation between pairs of shapes as

the shortest path in the above graph. The resulting operation is more efficient, because it does not depend on finding regions of the latent space that decode to valid words. Furthermore, because of the similarity metric and the graph topology, the intermediate shapes on paths in the graph are subjectively more intuitive interpolation results than the shapes we encountered by sampling on line segments in latent space.

We also experimented with a triplet contrastive loss [CSSB10, SSK\*17], which can improve sampling results by grouping the embedding of similar shapes in latent space. However, in our experiments, the additional loss slowed down training without noticeably improving the point distribution in latent space. We therefore report results only with a standard VAE loss function.

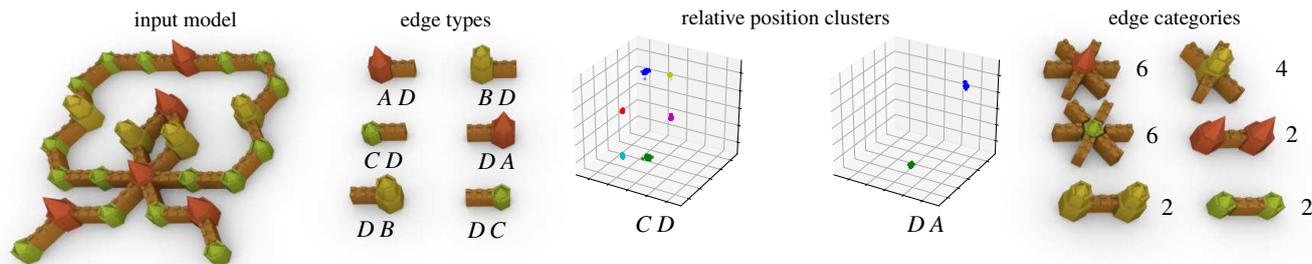
## 6. Instantiation of Structure Graphs

The shape synthesis approach we discussed so far consists of two parts. We first use an inverse procedural modeling method to generate a set of models using partial graph symmetries. Then, we vectorize the shape graphs of the sample models using SMILES strings, which are subsequently used as a training set for a variational autoencoder. Sampling from the latent space of the autoencoder generates string representation of further shape graphs with unknown instantiation in Euclidean space.

In the following, we describe how we reduce instantiation of structured shape graphs to a classification problem. The main motivation behind our approach is to re-use the procedurally generated shape database, and train a recurrent neural network to embed topological graph representation back into 3D. The method consists of two main steps. We first estimate a set of possible *edge categories* for each edge type. In other words, for each pair of parts  $a, b$  and of types  $A, B$ , we compute all possible ways to assemble them together and discretize them into a finite set of categories. (see Figure 5). Then, we train a recurrent neural network to estimate these discrete edge configurations from SMILES strings. The resulting sequence define a spatial embedding of the shape graph represented by the string.

**Local Coordinate Frame Estimation** In order to compute allowed configurations of pairs of parts, we first need to estimate a standard local coordinate frame for each individual piece. A common approach to address this problem is to perform PCA (Principal Component Analysis [Pea01]) on the set of vertices and use the principal axes as a coordinate frame. However, this method is not reliable for parts with global symmetries because they do not have a canonical ordering of their principle components.

Instead of PCA, we use a similar approach that considers only extremal points which makes it robust to deformations on the interior of parts. We compute a shape “diagonal” by finding the vertex  $v$  farthest away from the center of gravity and the vertex farthest away from  $v$ . We then find the vertex with maximal distance to the diagonal. We average multiple vertices with the maximal distances in any of the three searches if the result is not unique. The diagonal and the last vertex together with its projection onto the diagonal define a local coordinate frame, which worked better than PCA axes in our experiments.



**Figure 5:** Data-driven estimation of edge configurations. From left to right: the input model, the allowed edge types according to the extracted grammar, the part centroids for edges of type  $CD$  and  $DA$ , and all valid pairwise configurations for each edge type (the second piece is replicated to illustrate possible local assemblies). We cluster relative positions (and orientations) for each valid pair of types to estimate possible configurations. Then we train a recurrent neural network to assign configurations to graph edges and thereby embed shape graphs in 3D.

**Edge Category Estimation** Using the canonical coordinate frames for each part type, we can compute a set of valid configurations for pairs of adjacent parts. Each configuration represents a different way to attach (or dock) the two pieces to each other. We solve this by clustering relative part positions (and, if necessary orientations) for each edge type (see Figure 5) across the shape database. We use Mean Shift Clustering [Che95], because the number of clusters is not known in advance. Means Shift Clustering is a good fit for our data also because the sample points form well-separated clusters thanks to the precision we enforce when generating the set of training shapes. Note that we estimate 3D edge configurations, meaning that the method is not restricted to planar shape graphs.

**Edge Category Mapping** A problem we need to work around is that the characters of SMILES strings represent node sequences instead of edges. However, the graph nodes appear in the string in the same order they would appear in spanning tree of the graph. Therefore, it is possible to map a graph edge to each character in the string representing a graph node: the edge between the node and its parent in the spanning tree. Additionally, numbers that represent cycles also correspond directly to graph edges. We assign a dummy category to each string character that does not represent a graph node. These include brackets and special characters used to separate numbers superseding nodes adjacent to multiple cycles.

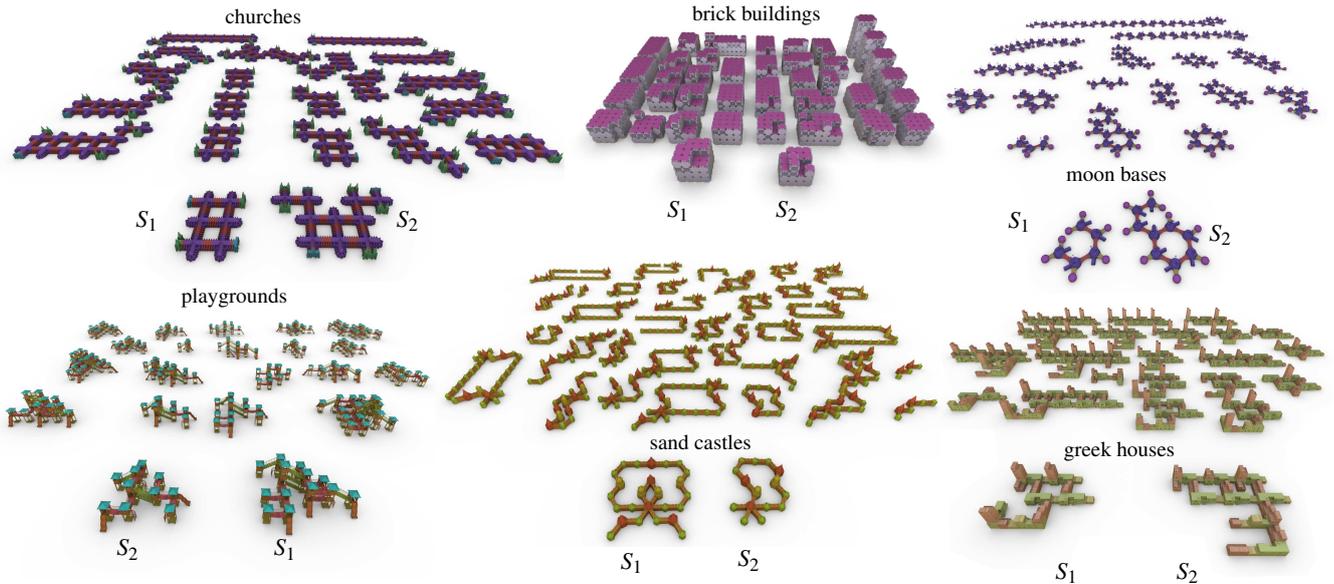
**Edge Category Learning** After constructing a mapping between each SMILES string and a sequence of graph edge categories, we can train a sequence to sequence model [SVL14] on the resulting pairs. Since our target sequences have the same length as the input strings, we can use a standard deep recurrent neural network consisting of an LSTM and a fully connected layer. We extended the model with a masking layer that restricts prediction to only plausible edge categories for the particular edge type. The possible categories for each edge in a string depend only on the node types connected by the edge and can be computed for any shape graph (see Figure 5). Therefore we multiply each output vector with a mask that zeroes the predicted probabilities for types different than the type of the current edge. In our experiments, introducing masking lowered the initial prediction accuracy, but slightly slowed conver-

gence to the optimal set of parameters during training. On the other hand, masking eliminates significant number of possible prediction errors and improves the prediction accuracy on the test samples.

The availability of multiple SMILES-like representations of the same graph plays to our advantage here, since it provides multiple training samples from each generated shape. Even though the general problem of instantiating structure graphs is extremely hard, learning on the shape collections we generate via graph-based sampling resulted in very accurate predictions: the accuracy on the validation set of samples reached 90% and the model consistently predict test sequences with only few mistakes (e.g. 6 wrong categories for a sequence of length 85). We reduced the amount of errors by estimating categories for multiple SMILES strings representing the same graph and selecting the most likely category for each edge across all of its occurrences in the sequences. As a result the trained neural network can be used to provide an initial guess for the instantiation of strings generated by the VAE in Section 5.

**Graph Instantiation** Using the estimated edge categories, we convert the graph topology to a geometric shape as follows. We start at a random edge of the target graph and incrementally add nodes by transplanting random edges with matching categories from the (two) input shapes. Note that we need to consider the edge categories in both edge directions for a correct instantiation. We improve the success rate of the algorithm by computing the current shape subgraph and checking it for grammar violations or poorly attached building blocks. If we detect a violation that cannot be repaired by selecting a different edge with matching categories, we assume that the requested pair of edge categories is wrong and resort to a Monte-Carlo edge insertion step: We select a random edge from the input shapes with matching node types and resume the previous construction algorithm if we find one that can be transplanted without violating the grammar.

The success rate of our instantiation method depends on the quality of estimated edge categories. We could not embed very large cycles without estimating correctly all (or all but one) configurations of participating edges. However, our method almost always succeeds for tree-like shape graphs and shapes with smaller cycles, as discussed in the following section.



**Figure 6:** Input shapes  $S_1, S_2$  and some (not hand-picked) sampled variants for each test model. Note that we are able to synthesize shapes containing cycles of building blocks, which is not possible when using context-free shape grammars. We maintain high quality by automatically repairing shapes with poorly connected pieces and enforce diversity by discarding variants with repeated node histograms.

## 7. Evaluation

**Implementation** We implemented the shape synthesis algorithms in the paper using C++ with Thrust [BH12], and Python. Please note that we provide the code as supplemental material and will make the implementation available as open source with the MIT license.

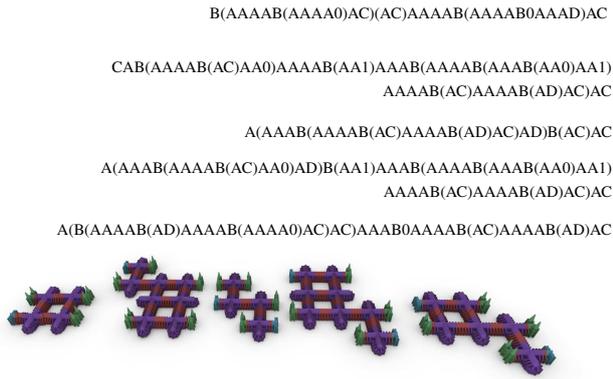
**Test Models** We tested our implementation on 3D models downloaded from online repositories or used in related work (the playground model) with node counts ranging from 23 to 70 and triangle counts between 20000 and 400000. Where necessary, we split the model into parts and assigned node types by coloring them before creating two example shapes by applying copy-paste operations, translations, and rotations of building blocks. We then iteratively select pairs of shapes, sample random subgraphs and attempt to combine them into new, valid variations. At the end of each iteration we attempt to correct local part orientations of newly created shapes and discard the model if the attempt fails, thereby maintaining reasonably high quality of sampled variations. We repeat the procedure until we reach a sufficiently large number of strings that we use for training. Note, that with the exception of the greek houses examples, we evaluate on examples with very restrictive grammar rules, which makes it difficult to sample new variations both procedurally and later from the latent space of the autoencoder.

**Shape Interpolation** See Figure 7 for an example shape interpolation result using the latent space of the trained autoencoder. While useful for discovering additional shapes, this method is not guaranteed to deliver desirable intermediate shapes, because distances the latent space of the autoencoder do not necessarily correspond

scene	size $S_1$ size $S_2$	iteration: 1	2	3	4	final
		strings shapes	strings shapes	strings shapes	strings shapes	strings shapes
church	27K	169	864	4389	28596	138835
	45K	4	13	37	145	529
sand castle	23K	88	374	1333	3832	146594
	13K	5	17	42	91	484
moon base	243K	124	227	346	451	19299
	413K	5	7	7	10	68
play- ground	129K	220	629	1751	3257	557895
	87K	6	12	17	18	137
brick buildings	113K	233	699	1265	2693	181969
	103K	6	9	15	18	104
greek houses	28K	501	448639	–	–	448639
	37K	19	1084	–	–	1084

**Table 1:** Input model sizes in the number of triangles and shape collection growth with each iteration of sampling. The larger amount of strings compared to shapes is due to the multiple possible string representations of the same graph. We did not generate all possible strings for each shape.

to shape similarity. This can be addressed easily via constraints on latent space such as the construction detailed in Section 5.3. The graph topological search (see Figure 10) performed very well on the majority of datasets and in particular on the playgrounds, greek houses and moon bases. We validated the embedding provided by our variational autoencoder by constructing the search



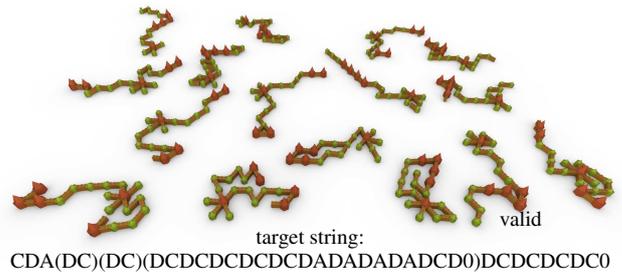
**Figure 7:** Latent path example: The models (left to right) and their string encodings (top to bottom) are discovered on a random path in latent space with the leftmost and rightmost shapes as endpoints.

graph naïvely from strings (see Table 2). Each point (string) is connected to its 3 nearest points (strings) and all equivalent strings. However, the simplified method produced graphs with multiple disconnected components, and edge augmentation made interpolation paths very long both topologically and according to  $E$ .

scene	graph type	graph size	connected components	avg weight	max weight
sand	naïve	1000	102	4.71	12.17
castle	ours	1000	1	0.50	1.09
moon	naïve	1000	30	2.29	5.50
base	ours	1000	1	0.39	0.55
greek	naïve	1000	67	2.55	11.36
houses	ours	1000	1	0.31	0.49

**Table 2:** Ablation study comparing a naïve search graph construction using string similarity directly instead of distance in the autoencoder latent space (see Section 5.3). We sample shortest paths between nodes using the string similarity metric  $E$  as edge weights in both cases and report average and maximum (accumulated) edge weights along these random path samples (lower is better).

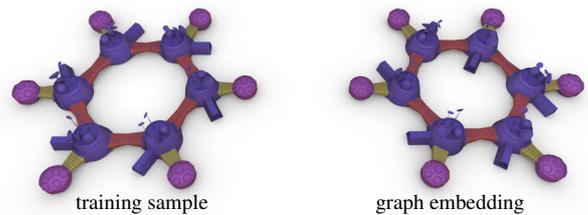
**Probabilistic Methods** Shape modeling methods based on subsequent random applications of grammar rules to generate shapes are not well suited for the topological variations we consider here. We can demonstrate this by ignoring the suggested edge categories in the last step of our instantiation implementation, converting it to a naïve random shape assembly algorithm. In Figure 8 we can see several failed attempts to compute a valid shape containing a long cycle. The probability of reproducing this particular cycle is approximately  $\frac{1}{5^{14}}$  because of the 6 possible edge configurations for each of the 15 participating green (C) and red (A) tower pieces. Even a seemingly simple example such as the shape in Figure 9 is very unlikely to be constructed via random applications of production rules, while we were able to embed the string



**Figure 8:** Invalid MCMC attempts to embed a graph with a large cycle in 3D. The only “valid” (according to the extracted grammar) shape does not have the graph topology represented by the string. The remaining shapes contain a wall piece (D) and/or a red tower piece (A) with less than two neighbors.

from the first attempt. It should be noted, that related works (such as [RJT18, RMGH15]) using context-free grammars, where this problem is not present, can provide improved probability at each step, however the exponential nature of the problem remains.

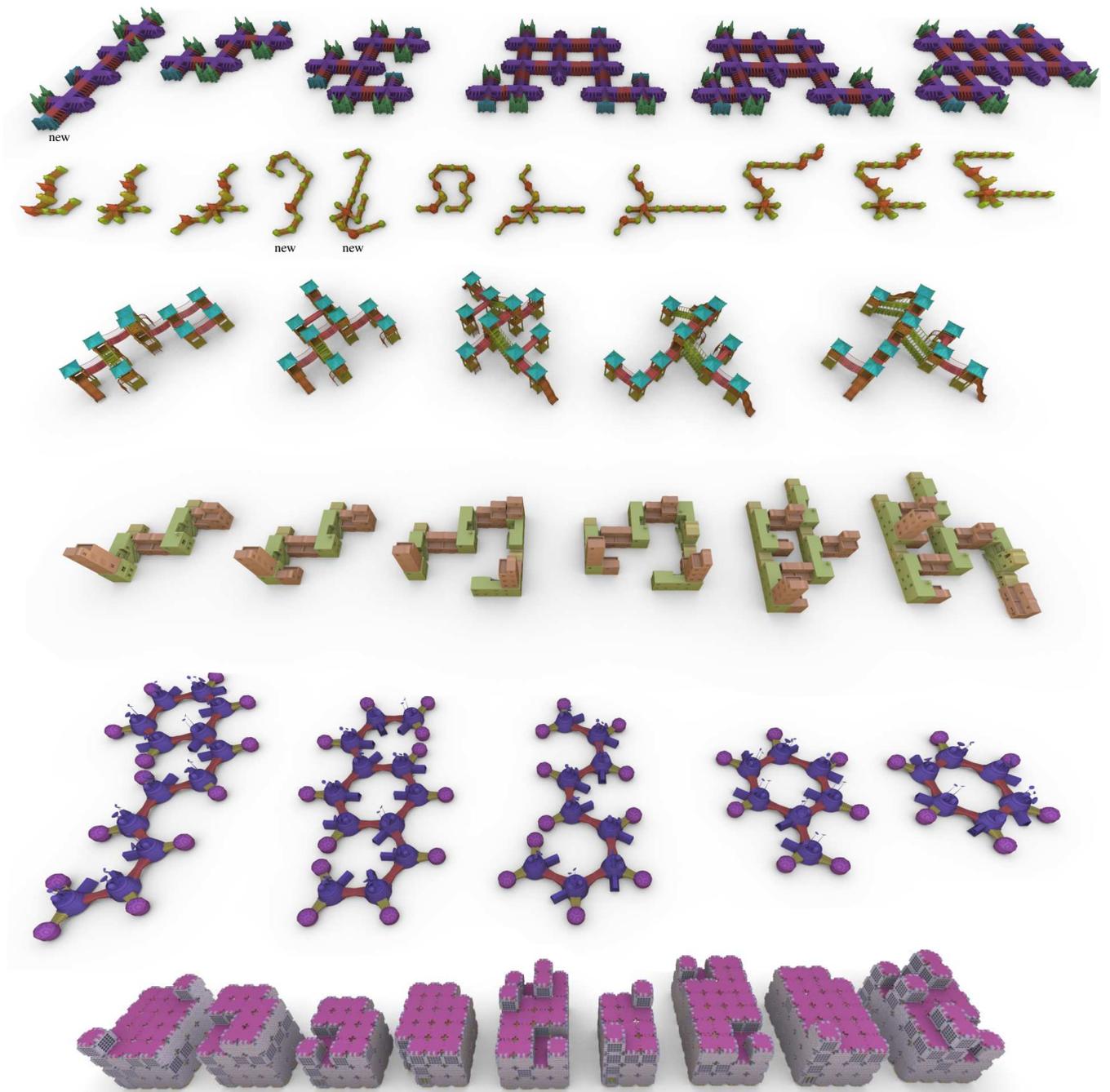
string: B(D)C(AC(BD)AC(AC0BD)BD)AC(AC(BD)A0)BD



**Figure 9:** Example model, its string decoded from the VAE’s latent space, and embedded in 3D via estimated edge configurations.

**Compression** The trained VAE model and instantiation RNN can be used as a compressed representation of the shape database. Each of the training shapes as well as additional samples generated via latent space sampling can be stored as two sequences: the string representing the graph topology and the corresponding sequence of edge categories suffice to reconstruct the shape exactly by “borrowing” geometry from the pair of input models.

**Limitations** Even though, this work makes a step towards solving the problem, both discovering and instantiation of strings representing shapes with cycles, remains harder compared to context-free variations. This is most likely caused by the string conversion method, which is better suited for encoding sequences and works around branching and cycles via additional characters. An extreme example presents itself in the brick buildings dataset, where we had to discard the cycle edges and only used strings representing the spanning trees of the shape graphs for training. We believe that these limitation can be lifted by using a better variational autoencoder, better suited for sampling vectorized graph topologies, e.g., methods similar to Liu et al. [LABG18]. It should also be noted that the amount of automation (an important aspect of our work) can become a limitation in application scenarios that require a higher degree of user control during modeling.



**Figure 10:** Shape interpolation examples. We display random shortest paths from the graph in latent space with edge weights according to a similarity metric. The resulting intermediate shapes are more intuitive compared to using Euclidean distances in latent space. Shapes marked as “new” were not present in the (automatically generated) training sets and were discovered by sampling the latent space of the variational autoencoder.

## 8. Conclusion

We introduce a combination of a model-based and a data-driven shape synthesis method able to create shapes from non-context-free shape grammars, a problem that is undecidable in general. We achieve this by learning vectorized representations of shape graphs with a variational autoencoder and perform high-level modeling operations via sampling in the resulting continuous latent space. We also demonstrate how the resulting sequences can be mapped back to geometric shapes by solving a classification problem. In addition, we deduce the assembly rules and automatically create sufficient training shapes from a single (or a few) example shape decomposed into building blocks. Altogether, our method provides a proof of concept for automatic shape synthesis of structured shape variations with arbitrary number of parts.

**Acknowledgements** We would like to thank the Thingiverse users for sharing the initial church, sand castle, and moon base models. This work was supported by the European Research Council under the European Union's Seventh Framework Programme, ERC grant agreement 340884 (ACROSS) as well as the Gottfried-Wilhelm-Leibniz Programme of the Deutsche Forschungsgemeinschaft DFG.

## References

- [BH12] BELL N., HOBEROCK J.: Thrust: Productivity-oriented library for cuda. [7](#)
- [BVV\*15] BOWMAN S. R., VILNIS L., VINYALS O., DAI A. M., JÓZEFOWICZ R., BENGIO S.: Generating sentences from a continuous space. *CoRR abs/1511.06349* (2015). [2, 4](#)
- [BWKS11] BOKELOH M., WAND M., KOLTUN V., SEIDEL H.-P.: Pattern-aware shape deformation using sliding dockers. *ACM Trans. Graph.* **30** (Dec. 2011), 123:1–123:10. [2](#)
- [BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph.* **29** (July 2010), 104:1–104:10. [2, 3](#)
- [BYMW13] BAO F., YAN D.-M., MITRA N. J., WONKA P.: Generating and exploring good building layouts. *ACM Transactions on Graphics* **32**, 4 (2013). [5](#)
- [Che95] CHENG Y.: Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* **17**, 8 (Aug. 1995), 790–799. [6](#)
- [CSSB10] CHECHIK G., SHARMA V., SHALIT U., BENGIO S.: Large scale online learning of image similarity through ranking. *J. Mach. Learn. Res.* **11** (Mar. 2010), 1109–1135. [5](#)
- [FKS\*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. *ACM Trans. Graph.* **23**, 3 (Aug. 2004), 652–663. [2](#)
- [FWX\*13] FAN L., WANG R., XU L., DENG J., LIU L.: Modeling by drawing with shadow guidance. *Computer Graphics Forum* **32**, 7 (2013), 157–166. [2](#)
- [GDH\*16] GÓMEZ-BOMBARELLI R., DUVENAUD D. K., HERNÁNDEZ-LOBATO J. M., AGUILERA-IPARRAGUIRRE J., HIRZEL T. D., ADAMS R. P., ASPURU-GUZIK A.: Automatic chemical design using a data-driven continuous representation of molecules. *CoRR abs/1610.02415* (2016). [2, 4](#)
- [KJS07] KREAVOY V., JULIUS D., SHEFFER A.: Model composition from interchangeable components. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (2007), PG '07, IEEE Computer Society, pp. 129–138. [2](#)
- [KW13] KINGMA D. P., WELLING M.: Auto-Encoding Variational Bayes. *ArXiv e-prints* (Dec. 2013). [2, 4](#)
- [LABG18] LIU Q., ALLAMANIS M., BROCKSCHMIDT M., GAUNT A. L.: Constrained graph variational autoencoders for molecule design. *CoRR abs/1805.09076* (2018). [8](#)
- [LF08] LEE J., FUNKHOUSER T.: Sketch-Based Search and Composition of 3D Models. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2008), Alvarado C., Cani M.-P., (Eds.), The Eurographics Association. [2](#)
- [LVW\*15] LIU H., VIMONT U., WAND M., CANI M.-P., HAHMANN S., ROHMER D., MITRA N. J.: Replaceable substructures for efficient part-based modeling. *Computer Graphics Forum* **34**, 2 (2015), 503–513. [2, 3](#)
- [LXC\*17] LI J., XU K., CHAUDHURI S., YUMER E., ZHANG H., GUIBAS L.: Grass: Generative recursive autoencoders for shape structures. *ACM Trans. Graph.* **36**, 4 (July 2017), 52:1–52:14. [1, 2](#)
- [MWZ\*13] MITRA N. J., WAND M., ZHANG H., COHEN-OR D., BOKELOH M.: Structure-Aware Shape Processing. In *Eurographics 2013 - State of the Art Reports* (2013), The Eurographics Association. [2](#)
- [NW17] NASH C., WILLIAMS C. K. I.: The Shape Variational Autoencoder: A Deep Generative Model of Part-segmented 3D Objects. *Computer Graphics Forum* (2017). [2](#)
- [Pea01] PEARSON K.: On lines and planes of closest fit to systems of points in space. *Philosophical Magazine* **2**, 11 (1901), 559–572. [5](#)
- [RJT18] RITCHIE D., JOBALIA S., THOMAS A.: Example-based Authoring of Procedural Modeling Programs with Structural and Continuous Variability. *Computer Graphics Forum* **37**, 2 (2018). [2, 8](#)
- [RMGH15] RITCHIE D., MILDENHALL B., GOODMAN N. D., HANRAHAN P.: Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Trans. Graph.* **34**, 4 (July 2015), 105:1–105:11. [2, 8](#)
- [SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Proceedings of Symposium on Geometry Processing* (2007), pp. 109–116. [3](#)
- [SSK\*17] SUNG M., SU H., KIM V. G., CHAUDHURI S., GUIBAS L.: Complementme: Weakly-supervised component suggestions for 3d modeling. *ACM Trans. Graph.* **36**, 6 (Nov. 2017), 226:1–226:12. [1, 2, 5](#)
- [SVL14] SUTSKEVER I., VINYALS O., LE Q. V.: Sequence to sequence learning with neural networks. *CoRR abs/1409.3215* (2014). [6](#)
- [TLL\*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM Trans. Graph.* **30**, 2 (Apr. 2011), 11:1–11:14. [2](#)
- [TYK\*12] TALTON J., YANG L., KUMAR R., LIM M., GOODMAN N., MĚCH R.: Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (2012), UIST '12, ACM, pp. 63–74. [2](#)
- [Wei88] WEININGER D.: Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences* **28**, 1 (1988), 31–36. [2, 4](#)
- [XKHK17] XU K., KIM V. G., HUANG Q., KALOGERAKIS E.: Data-Driven Shape Analysis and Processing. *Computer Graphics Forum* **36**, 1 (2017), 101–132. [2](#)
- [XXM\*13] XIE X., XU K., MITRA N. J., COHEN-OR D., GONG W., SU Q., CHEN B.: Sketch-to-Design: Context-Based Part Assembly. *Computer Graphics Forum* **32**, 8 (2013), 233–245. [2](#)
- [XZZ\*11] XU K., ZHENG H., ZHANG H., COHEN-OR D., LIU L., XIONG Y.: Photo-inspired Model-driven 3D Object Modeling. *ACM Trans. Graph.* **30**, 4 (jul 2011), 80:1–80:10. [2](#)
- [ZCOM13] ZHENG Y., COHEN-OR D., MITRA N. J.: Smart variations: Functional substructures for part compatibility. *Computer Graphics Forum* **32**, 2 (2013), 195–204. [2](#)