

StructEdit: Learning Structural Shape Variations

Kaichun Mo^{*1} Paul Guerrero^{*2} Li Yi³ Hao Su⁴ Peter Wonka⁵ Niloy J. Mitra^{2,6} Leonidas Guibas^{1,7}

¹Stanford University ²Adobe Research ³Google Research ⁴UC San Diego
⁵KAUST ⁶University College London ⁷Facebook AI Research

Abstract

Learning to encode differences in the geometry and (topological) structure of the shapes of ordinary objects is key to generating semantically plausible variations of a given shape, transferring edits from one shape to another, and many other applications in 3D content creation. The common approach of encoding shapes as points in a high-dimensional latent feature space suggests treating shape differences as vectors in that space. Instead, we treat shape differences as primary objects in their own right and propose to encode them in their own latent space. In a setting where the shapes themselves are encoded in terms of fine-grained part hierarchies, we demonstrate that a separate encoding of shape deltas or differences provides a principled way to deal with inhomogeneities in the shape space due to different combinatorial part structures, while also allowing for compactness in the representation, as well as edit abstraction and transfer. Our approach is based on a conditional variational autoencoder for encoding and decoding shape deltas, conditioned on a source shape. We demonstrate the effectiveness and robustness of our approach in multiple shape modification and generation tasks, and provide comparison and ablation studies on the PartNet dataset, one of the largest publicly available 3D datasets.

1. Introduction

The shapes of 3D objects exhibit remarkable diversity, both in their compositional structure in terms of parts, as well as in the geometries of the parts themselves. Yet humans are remarkably skilled at imagining meaningful shape variations even from isolated object instances. For example, having seen a new chair, we can easily imagine its *natural* variations with a different height back, a wider seat, with or without armrests, or with a different base. In this paper, we investigate how to learn such shape variations directly from 3D data. Specifically, given a shape collection, we are interested in two sub-problems: first, for any given shape,

*: indicates equal contributions.

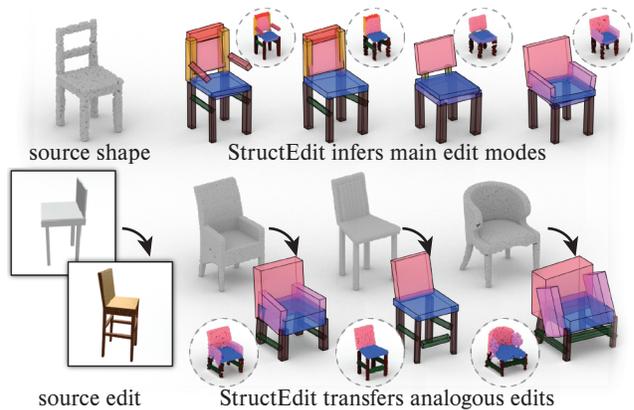


Figure 1. **Edit generation and transfer with StructEdit.** We present StructEdit, a method that learns a distribution of *shape differences* between structured objects that can be used to generate a large variety of edits (first row); and accurately transfer edits between different objects and across different modalities (second row). Edits can be both geometric and topological.

we want to discover the main modes of edits, which can be inferred directly from the shape collection; and second, given an example edit on one shape, we want to transfer the edit to another shape in the collection, as a form of analogy-based edit transfer. This ability is useful in multiple settings, including the design of individual 3D models, the consistent modification of 3D model families, and the fitting of CAD models to noisy and incomplete 3D scans.

There are several challenges in capturing the space of shape variations. First, individual shapes can have different representations as images, point clouds, or surface meshes; second, one needs a unified setting for representing both continuous deformations as well as structural changes (e.g., the addition or removal of parts); third, shape edits are not directly expressed but are only implicitly contained in shape collections; and finally, learning a space of structural variations that is applicable to more than a single shape amounts to learning mappings between different shape edit distributions, since different shapes have different types and numbers of parts (e.g., tables with/without leg bars).

In much of the extant literature on 3D machine learning,

3D shapes are mapped to points in a representation space whose coordinates encode latent features of each shape. In such a representation, shape edits are encoded as vectors in that same space – in other words as differences between points representing shapes. Equivalently, we can think of shapes as “anchored” vectors rooted at the origin, while shape differences are “floating” vectors that can be transported around in the shape space. This type of vector space arithmetic is commonly used [42, 1, 38, 14, 44, 37], for example, in performing analogies, where the vector that is the difference of latent point A from point B is added to point C to produce the analogous point D . The challenge with this view in our setting is that while Euclidean spaces are perfectly homogeneous and vectors can be easily transported and added to points anywhere, shape spaces are far less so. While for continuous variations the vector space model has some plausibility, this is clearly not so for structural variations: the “add arms” vector does not make sense for a point representing a chair that already has arms.

We take a different approach. We consider embedding shape differences or deltas *directly in their own latent space*, separate from the general shape embedding space. Encoding and decoding such shape differences is always done through a variational autoencoder (VAE), in the context of a given source shape, itself encoded through a part hierarchy. This has a number of key advantages: (i) allows compact encodings of shape deltas, since in general we aim to describe local variations; (ii) encourages the network to abstract commonalities in shape variations across the shape space; and (iii) adapts the edits to the provided source shape, suppressing the modes that are semantically implausible.

We have extensively evaluated *StructEdit* on publicly available shape data sets. We introduce a new synthetic dataset with ground truth shape edits to quantitatively evaluate our method and compare against baseline alternatives. We then provide evaluation results on the PartNet dataset [26] and provide ablation studies. Finally, we demonstrate that extensions of our method allow handling of both images and point clouds as shape sources, can predict plausible edit modes from single shape examples, and can also transfer example shape edits on one shape to other shapes in the collection (see Figure 1).

2. Related Work

3D deep generative models have attracted an increasing amount of research efforts recently. Different from 2D images, 3D shapes can be expressed as volumetric representations [42, 15, 47, 9, 17], oct-trees [34, 39], point clouds [11, 1, 22, 36, 48, 31], multi-view depth maps [2], or surface meshes [32, 16, 18]. Beyond low level shape representations, object structure can be modeled along with geometry [27, 38, 35] to focus on the part decomposition of objects or hierarchical structures across object families

during the generation process [43, 23, 25]. Similar to StructureNet [25], we utilize the hierarchical part structure of 3D shapes as defined in PartNet [26]. However, our generative model directly encodes *structural deltas* instead of the shapes which, as we demonstrate, is more suitable for significant shape modifications and edit transfers.

Structure-aware 3D shape editing is a long-standing research topic in shape analysis and manipulation. Early works [20, 46] analyzed individual input shapes for structural constraints by leveraging local but adaptive deformation to adjust shape edits according to its content. Global constraints were subsequently used in the form of parallelism, orthogonality [13], or high-level symmetry and topological variations [41, 6]. However, analyzing shapes in isolation can lead to spuriously detected structural constraints and cannot easily be generalized to handle large number of shapes. Hence, followup works [28, 12, 50] analyze a family of 3D shapes to decipher the shared underlying geometric principles. Recently, utilizing deep neural networks, Yumer and Mitra [51] learn how to generate deformation flows guided by some high-level intentions through 3D volumetric convolutions, while free-form deformation is learned [21, 19] to capture how shapes tend to deform within a category, or predict 3D mesh deformation [40] conditioned on an input target image, with high-level deformation priors encoded through networks. However, by preserving input shape topology, these works greatly limit the possible edit space. Instead, we develop a deep neural network to capture the common structural variations within shape collections, and enable versatile edits with both geometric and topological changes.

Shape deformation transfer aims at transferring deformation imposed to a source shape onto a target shape. This requires to address how to represent shape edits, and how to connect the source and target pairs so that the edit are transferable. Early works used deformation flow or piecewise affine transformation to represent shape edits with explicit correspondence information [33], or via functional spaces to represent shape differences [10, 30]. Correspondence is established either pointwise [33, 49, 52, 24], or shapes are associated using higher-level abstractions like cages [5, 8], patches [3], or parts [45]. Recent efforts adapt neural representations for shapes via latent vector spaces, and then generate codes for shape edits by directly taking differences between latent shape representations. They either represent the source and target shapes in the same latent space and directly transfer the edit code [42, 1, 38], or learn to transform the edit code from source shape domain to target shape domain [14]. Shape edit transfer is also related to motion re-targeting [44, 37] where the shape deformations are usually restricted to topology-preserving changes. In contrast, we directly encode shape deltas, leading to more consistent edit transfers, even with significant topological changes.

3. Method

Shape differences, or *deltas* ΔS_{ij} , are defined as a description of the transformation of a source shape S_i into a target shape S_j . Our goal is to learn a generative model of the conditional distribution $p(\Delta S_{ij}|S_i)$ that *accurately* captures all ΔS_{ij} in the dataset, and has a high degree of *consistency* between the conditional distributions of different source shapes (see Figure 5).

3.1. Representing Shapes

We represent a shape as a hierarchical assembly of parts that captures both the geometry and the structure of a shape. The part assembly is modeled as an n -ary tree $S := (\mathbf{P}, \mathbf{H})$, consisting of a set of *parts* $\mathbf{P} := (P_1, P_2, \dots)$ that describe the geometry of the shape, and a set of edges \mathbf{H} that describes the *structure* of the shape. See Figure 2 (left), for an example. Each part is represented by an oriented bounding box $P := (c, q, r, \tau)$, where c is the center, q a quaternion defining its orientation, r the extent of the box along each axis, and τ is the semantics of the part, chosen from a pre-defined set of categories. These parts form a hierarchy that is modeled by the edges \mathbf{H} of an n -ary tree. Starting at the root node that consists of a bounding box for the entire shape, parts are recursively divided into their constituent parts, with edges connecting parents to child parts. A chair, for example, is divided into a backrest, a seat, and a base, which are then, in turn, divided into their constituent parts until reaching the smallest parts at the leaves.

3.2. Representing Shape Deltas

Given a source shape S_i and a target shape S_j , we first find corresponding parts in both shapes, based on parameters, semantics, and hierarchy. We find the part-level transformation assignment $\mathbf{M} \subset \mathbf{P} \times \mathbf{P}'$ among the parts $\mathbf{P} \in S_i$ and $\mathbf{P}' \in S_j$ starting at the children of the root parts, and then recursively match children of the matched parts, until reaching the leaves. We only match pairs with the same semantics, using a linear assignment based on the distance between the part bounding boxes. As a measure of similarity between two parts, we cannot directly use the distance between their box parameters, as multiple parameter sets describe the same bounding box. Instead, we measure the distance between point clouds sampled on the bounding boxes:

$$\delta_{\text{box}}(P_k, P_l) = \delta_{\text{ch}}(\mathbf{X}(P_k), \mathbf{X}(P_l)), \quad (1)$$

where \mathbf{X} is a function that samples a bounding box with a set of points. We use the Chamfer distance [4, 11] to measure the distance between two point clouds.

Given the assignment \mathbf{M} , the shape delta $\Delta S := (\Delta \mathbf{P}, \mathbf{P}^-, \mathbf{P}^+, \mathbf{H}^+)$ consists of three sets of *components*: a set of *part deltas* $\Delta \mathbf{P} = \{\Delta P_1, \Delta P_2, \dots\}$ that model geometric transformations from source parts to corresponding

target parts, a set of *deleted parts* \mathbf{P}^- , and a set of *added parts* \mathbf{P}^+ . Additional edges \mathbf{H}^+ describe edges between added parts and their parents. Note that the parents can also be other added parts. Each part in the source shape can either be associated with a part delta or be deleted (see Figure 2).

A part delta $\Delta P = (\Delta c, \Delta q, \Delta r)$ defines the parameter differences between a source part and the corresponding target part. Deleted parts \mathbf{P}^- are the source parts that are not assigned to any target part. Added parts \mathbf{P}^+ along with their structure \mathbf{H}^+ form zero, one, or multiple sub-trees that extend the n -ary tree of the source shape. Note that edges \mathbf{H} that are not adjacent to an added part are not stored in the shape delta but inferred from the source shape.

Applying a shape delta to a source shapes, an operation we denote as $S_i + \Delta S_{ij}$, gives us the target shape S_j :

$$S_j := S_i + \Delta S_{ij} = \{P + \Delta P \mid P \in \mathbf{P} \setminus \mathbf{P}^-\} \cup \mathbf{P}^+, (\mathbf{H} \setminus \mathbf{H}^-) \cup \mathbf{H}^+, \quad (2)$$

where $P + \Delta P = (c + \Delta c, \Delta q * q, r + \Delta r, \tau)$ is a modified part, and \mathbf{H}^- is the set of edges that is adjacent to any removed part. Note that our shape delta representation encodes both structural and geometric differences between the two shapes involved and represents a *minimal* program for transforming the source shape to the target shape.

3.3. Conditional Shape Delta VAE

We train a conditional variational autoencoder (cVAE) consisting of an encoder e that encodes a shape delta into a latent code $z := e(S_i, \Delta S_{ij})$, and a decoder d , that decodes a latent code back into a shape delta $\Delta S_{ij} := d(S_i, z)$. Both the encoder and decoder are conditioned on S_i . Providing access to the source shape encourages the cVAE to learn a distribution of deltas conditional on the source shape. Both the encoder e and decoder d make use of a shape encoder $z_s := f(S_i)$ to generate z_s and intermediate features of source shape S_i .

The encoders and decoders are specialized networks that operate on trees of parts or shape delta components, and are applied recursively on the respective trees. We set the dimensionality of the latent code z and all intermediate feature vectors computed by the encoders and decoders to 256.

Source Shape Encoder. The encoder computes two feature vectors for each part in a source shape, respectively encoding information about the part and its subtree.

The box features v_k^{box} are computed for the geometry of each part of the source shape using the encoder f_{box} :

$$v_k^{\text{box}} := f_{\text{box}}([c_k, q_k, r_k]), \quad (3)$$

where $[c_k, q_k, r_k]$ denotes the concatenation of the box parameters of Part P_k . The subtree features v_k^{tree} at each part are computed with a recursive encoder. For the leaf parts,

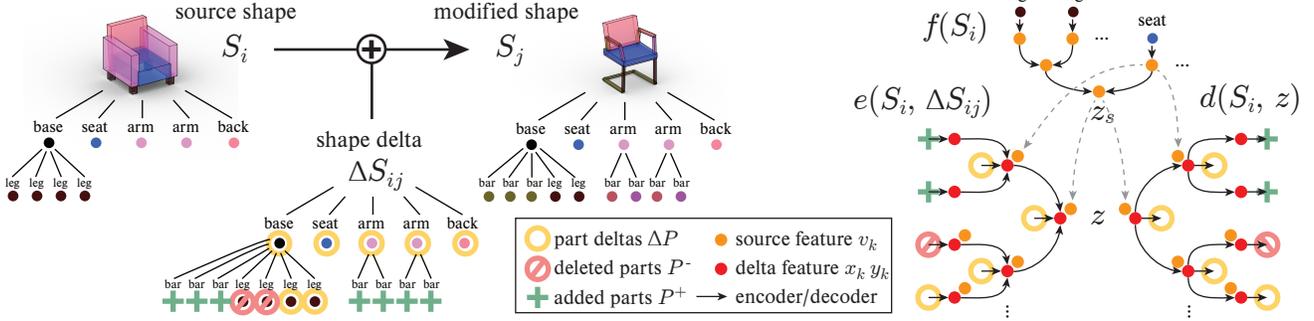


Figure 2. **Shape deltas and the conditional shape delta VAE.** On the left, we show a source shape and a modified shape. Both are represented as hierarchical assemblies of individual parts, where each part is defined by a bounding box and a semantic category (see colors). The shape delta describes the transformation of the source into the modified shape with three types of components: part deltas, deleted parts, and added parts. We learn a distribution of these deltas, conditioned on the source shape, using the conditional VAE illustrated on the right side. Shape deltas are encoded and decoded recursively, following their tree structure, yielding one feature per subtree (red circles). Conditioning on the source shape is implemented by recursively encoding the source shape, and feeding the resulting subtree features (orange circles) to the corresponding parts of the encoder and decoder.

we define $v_k^{\text{tree}} = v_k^{\text{box}}$. For non-leaf parts, we recursively pool their child features with encoder f_{tree} :

$$v_k^{\text{tree}} := f_{\text{tree}}(\{[v_l^{\text{tree}}, \tau_l]\}_{P_l \in \mathbf{C}_k}), \quad (4)$$

where \mathbf{C}_k is the set of child parts, τ is the semantic label of a part, and the square brackets denote concatenation. The encoder f_{tree} uses a small PointNet [29] with max-pooling as symmetric operation. The PointNet architecture can encode an arbitrary number of children and ensures that f_{tree} is invariant to the ordering of the child parts. See the Supplementary for architecture details.

Shape Delta Encoder. The shape delta encoder computes a feature y_i for each component in sets $\Delta\mathbf{P}$, \mathbf{P}^- , \mathbf{P}^+ of the shape delta ΔS_{ij} . Each feature describes the component and its sub-tree. The feature of the root component is used as latent vector z for the shape delta. We encode a shape delta recursively, following the tree structure of the source shape extended by the added parts. Components in the set of part additions \mathbf{P}^+ and their edges \mathbf{H}^+ are encoded analogous to the source shape encoder:

$$y_k := f_{\text{tree}}(\{[y_l, \tau_l]\}_{P_l \in \mathbf{C}_k^+}), \quad (5)$$

where \mathbf{C}_k^+ are child parts of P_k that include newly added parts, and $y_k = f_{\text{box}}(P_k)$ for added parts that are leaves.

Part deltas $\Delta\mathbf{P}$ and deletions \mathbf{P}^- modify existing parts of the source shape. For components in both of these sets, we encode information about the part delta and the corresponding source part side-by-side, using the encoder c_{part} :

$$y_k := c_{\text{part}}([y_k^{\Delta\text{box}}, y_k^{\text{tree}}, \rho_k, v_k^{\text{box}}, \tau_k]), \quad (6)$$

where ρ is a one-hot encoding of the shape delta component type (delta or deletion), and $y^{\Delta\text{box}} = c_{\Delta\text{box}}(\Delta P)$ is a feature describing the part delta. For deleted parts, we

set $y^{\Delta\text{box}}$ to zero. v_k^{box} is the feature vector describing the box geometry of the source part, defined in Eq. 3. Finally, features of the child components are pooled by the encoder c_{tree} :

$$v_k^{\text{tree}} = c_{\text{tree}}(\{y_l\}_{P_l \in \mathbf{C}_k^+}). \quad (7)$$

The encoder c_{tree} is implemented as a small PointNet and returns zeros for leaf nodes that do not have children.

Shape Delta Decoder. The shape delta decoder reconstructs a part delta ΔP or a deletion P^- for each part of the source shape, and recovers any added nodes P^+ and their edges H^+ . The shape delta is decoded recursively, starting at the root part of the source shape. We use two decoders to compute the feature vectors; one decoder for part deltas and deleted parts, and one for added parts.

For part deltas $\Delta\mathbf{P}$ and deleted parts \mathbf{P}^- , the decoder d_{tree} computes x_k from parent feature and source part:

$$x_k := d_{\text{tree}}([z, x_p, v_k^{\text{box}}, v_k^{\text{tree}}, \tau_k]), \quad (8)$$

where x_p is the feature vector of the parent part, v_k^{box} and v_k^{tree} are the features describing the source part and source part subtree defined in Equations 3 and 4. We include the latent code z of the shape delta in each decoder to provide a more direct information pathway. We then classify the feature x_k into one of two types (delta or deletion), using the classifier $\rho'_k = d_{\text{type}}(x_k)$. For part deltas, we reconstruct the box difference parameters $\Delta P'_k = (\Delta c'_k, \Delta q'_k, \Delta r'_k) = d_{\Delta\text{box}}(x_k)$ with the decoder $d_{\Delta\text{box}}$. For deleted parts, no further parameters need to be reconstructed.

Feature vectors for added parts \mathbf{P}^+ are computed by the decoder d_{add} that takes as input the parent feature and outputs a list of child features. This list has a fixed length n , but we also predict an *existence probability* p_k for each feature in the list. Features with $p_k < 0.5$ are discarded. In our

experiments we decode 10 features and probabilities per parent. The decoder d_{add} is defined as

$$(x_{k_1}, \dots, x_{k_n}, p_{k_1}, \dots, p_{k_n}) := d_{\text{add}}(x_p), \quad (9)$$

where x_p is the parent feature, x_{k_i} are the child features with corresponding existence probabilities p_{k_i} . We realize the decoder as a single layer perceptron (SLP) that outputs n concatenated features, followed by another SLP that is applied to each of the n with shared parameters to obtain the n features and probabilities. Once the feature x_k for an added part is computed, we reconstruct the added part $P'_k = (c'_k, q'_k, r'_k, \tau') = d_{\text{box}}(x_k)$ with the decoder d_{box} . We stop the recursion when the existence probability for all child parts falls below 0.5. To improve robustness, we additionally classify a part as leaf or non-leaf part, and do not apply d_{add} to the leaf nodes. We use two instances of this decoder that do not share parameters, one for the added parts that are children of part deltas, and one for added parts that are children of other added parts.

3.4. Loss and Training

We train our cVAE with a dataset of $(S_i, \Delta S_{ij})$ pairs. Each shape delta ΔS_{ij} is an edit of the source shape S_i that yields a target shape $S_j = S_i + \Delta S_{ij}$. Both source shapes and target shapes are part of the same shape dataset. Shape deltas represent *local* edits, so we can create shape deltas by computing the difference between pairs of shapes in local neighborhoods: $\{S_j - S_i \mid S_j \in \mathcal{N}_i\}$, where \mathcal{N}_i denotes the local neighborhood of shape S_i (see Section 4).

We train the cVAE to minimize the reconstruction loss between the input shape delta ΔS_{ij} and the reconstructed shape delta $\Delta S'_{ij} = d(S_i, e(S_i, \Delta S_{ij}))$:

$$\begin{aligned} \mathcal{L}_{\text{total}} &:= \mathbb{E}_{(S_i, \Delta S_{ij}) \sim p(S_i, \Delta S_{ij})} [\mathcal{L}_{\Delta S}(\Delta S_{ij}, \Delta S'_{ij})] \\ \mathcal{L}_{\Delta S}(\Delta S_{ij}, \Delta S'_{ij}) &= \lambda \mathcal{L}_{\Delta \mathbf{P}} + \mathcal{L}_{\mathbf{P}^+} + \mathcal{L}_{\rho} + \beta \mathcal{L}_v. \end{aligned} \quad (10)$$

The reconstruction loss consists of four main terms, corresponding to the component types $(\Delta \mathbf{P}, \mathbf{P}^+)$, a classification loss \mathcal{L}_{ρ} for the predicted components into one of the component types, and the variational regularization \mathcal{L}_v . Since we do not decode parameters for deleted parts \mathbf{P}^- , there is no loss term for these components beyond the classification loss. Empirically, we set $(\lambda, \beta) := (10, 0.05)$.

The *part delta loss* $\mathcal{L}_{\Delta \mathbf{P}}$ measures the reconstruction error of the bounding box delta:

$$\mathcal{L}_{\Delta \mathbf{P}}(\Delta S_{ij}, \Delta S'_{ij}) := \sum_{\Delta P'_k \in \Delta \mathbf{P}'} \delta_{\text{box}}(P_k + \Delta P_k, P_k + \Delta P'_k),$$

where δ_{box} is the bounding box distance defined in Eq. 1. Correspondences between the input part deltas ΔP_k and reconstructed part deltas $\Delta P'_k$ are known, since each part delta corresponds to exactly one part of the source shape.

The *classification loss* \mathcal{L}_{ρ} is defined as the cross entropy H between component type ρ and reconstructed type ρ' :

$$\mathcal{L}_{\rho}(\rho_k, \rho'_k) := \sum_{\Delta \mathbf{P}' \cup \mathbf{P}'^-} H(\rho_k, \rho'_k). \quad (11)$$

The *added part loss* $\mathcal{L}_{\mathbf{P}^+}$ measures the reconstruction error for the added parts. Unlike part deltas and deleted parts, added parts do not correspond to any part in the source shape. Using the inferred assignment $\mathbf{M} \subset \mathbf{P}^+ \times \mathbf{P}'^+$ (see Section 3.2) – matched parts share indices, and \mathbf{P}'^+_m denotes the set of added parts in the reconstructed shape delta that have a match – the loss $\mathcal{L}_{\mathbf{P}^+}$ is defined as:

$$\mathcal{L}_{\mathbf{P}^+}(\Delta S_{ij}, \Delta S'_{ij}) := \sum_{P'_k \in \mathbf{P}'^+_m} \mathcal{L}_m + \sum_{P'_k \in \mathbf{P}'^+} H(\mathbb{1}_{P'_k \in \mathbf{P}'^+_m}, p_k), \quad (12)$$

the first term defines the reconstruction loss for all matched parts, while the second term defines the loss for the existence probabilities p_k of both matched and unmatched parts (see Eq. 9). The indicator function $\mathbb{1}$ returns 1 for matched parts and 0 for unmatched parts. The loss for matched parts \mathcal{L}_m measures box reconstruction error, the part semantics, and the leaf/non-leaf classification of a part:

$$\begin{aligned} \mathcal{L}_m(\Delta S_k, \Delta S'_k) &:= \mu \delta_{\text{box}}(P_k, P'_k) + \\ &H(\tau_k, \tau'_k) + \gamma H(\mathbb{1}_{P_k \in \mathbf{P}_{\text{leaf}}}, l'_k), \end{aligned} \quad (13)$$

where τ_k is the semantic label of part P_k , l_k is the predicted probability for part P_k to be a leaf part, and \mathbf{P}_{leaf} is the set of leaf parts. We set (μ, γ) to $(20, 0.1)$.

4. Experiments

We evaluate our main claims with three types of experiments. To show that encoding shape deltas more *accurately* captures the distribution of deltas $p(\Delta S_{ij} | S_i)$ compared to encoding shapes, we perform *reconstruction* and *generation* of modified shapes using our method, and compare to a state-of-the-art method for directly encoding shapes. To show that encoding shape deltas gives us a distribution that is more *consistent* between different source shapes, we perform *edit transfer* and measure the consistency of the transferred edits. Additionally, we show several applications. Ablation studies are provided in the supplementary.

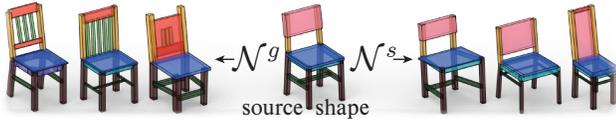
Shape Distance Measures. We use two distance measures between two shapes. The *geometric distance* d_{geo} between two shapes is defined as the Chamfer distance δ_{ch} between two point clouds of size 2048 sampled randomly on the bounding boxes of each shape. The *structural distance* d_{st} is defined by first finding a matching \mathbf{M} between the parts of two shapes (Section 3.2), and then counting the total number of unmatched parts in both shapes, normalized by the number of parts in the first shape.

Datasets. We train and evaluate on datasets that contain pairs of source shapes and shape deltas ($S_i, \Delta S_{ij}$). To create these datasets, we start with a dataset of structured shapes that we use as source shapes, and then take the difference to neighboring shapes to create the deltas.

The first dataset we use for training and evaluation is the PartNet dataset [26] generated from a subset of ShapeNet [7] with annotated hierarchical decomposition of each object into labelled parts (see Section 3.2). We train separately on the categories chair, table, and furniture. There are 4871 chairs, 5099 tables, and 862 cabinets in total. We use the official training and test splits as used in [25].

We define neighborhoods \mathcal{N} as k -nearest neighbors, according to two different metrics:

- *Geometric neighborhoods* \mathcal{N}^g based on the geometric distance d_{geo} contain edits with structural variations.
- *Structural neighborhoods* \mathcal{N}^s based on a structural distance d_{st} contain edits with geometric variations.



The inset shows both neighborhoods with size $k = 3$ for a source shape. We set $k = 100$ in our training sets and choose $k_{\text{test}} = 20$ for our test sets to obtain approximately the same neighborhood radius.

To evaluate the *consistency* of our conditional shape delta distributions between different source shapes, we need ground truth correspondence between edits of different source shapes. In absence of any suitable benchmark, we introduce a new synthetic dataset where source shapes and edits are created procedurally, giving us the correspondence between edits by construction. The synthetic dataset consists of three shape groups: stools, sofas, and chairs. These groups are closed with respect to the edits. Inside each group, all edits have correspondences. Between groups, only some edits have correspondences. For example, stools have no correspondence for edits that modify the backrest, but do have correspondences for edits that modify the legs. For details on the procedural generation, please see the supplementary.

Baselines. We compare our method to StructureNet [25], a method that learns a latent space of shapes with the same hierarchical structure as ours*, but does not encode edits explicitly. We represent edits as difference vectors in latent space that we can add to different source shapes (see supp. material for details). As upper bound for our error metrics, we additionally compare to a baseline that only models identity edits and always returns the source shape. A key advantage of StructEdit over these baselines or any

*For fairness, we do not use the relationship edges of StructureNet here. The supplementary material shows that results with edges are similar.

		\mathcal{N}^g				\mathcal{N}^s			
		chair	table	furn.	avg.	chair	table	furn.	avg.
E_r^{geo}	ID	1.000	1.000	1.001	1.000	1.000	1.000	0.999	1.000
	SN	0.886	0.972	0.875	0.911	0.656	0.492	0.509	0.553
	SE	0.755	0.805	0.798	0.786	0.531	0.414	0.434	0.459
E_r^{st}	ID	0.946	0.940	0.951	0.945	1.107	1.341	1.124	1.191
	SN	0.264	0.370	0.388	0.340	0.734	1.469	0.915	1.039
	SE	0.082	0.151	0.139	0.124	0.136	0.246	0.183	0.188

Figure 3. **Edit Reconstruction.** We compare geometric and structural reconstruction errors for the identity baseline (ID), StructureNet (SN), and StructEdit (SE) on neighborhoods \mathcal{N}^g and \mathcal{N}^s . Visual examples are shown above. Reconstructing deltas rather than absolute shapes leads to more accurate reconstructions.

hand-crafted edit transfer method is enabling semantically-valid translations of structural or geometric edits between different source shapes.

4.1. Edit Reconstruction

To measure reconstruction performance, we train our architecture without the variational regularization on the PartNet dataset, and the evaluation is based on geometric distances d_{geo} and structural distances d_{st} :

$$E_r^* = \frac{1}{r_{\mathcal{N}}} d_*(S_i + \Delta S_{ij}, S_i + \Delta S'_{ij}), \quad (14)$$

where ΔS_{ij} is the input delta, and $\Delta S'_{ij}$ the reconstructed delta. We normalize the distances by the average distance $r_{\mathcal{N}}$ of a neighbor from the source shape in the given dataset.

Results for both metrics and both neighborhoods are given in Figure 3. Geometric neighborhoods \mathcal{N}_g have large structural variations, but low geometric variations. For geometric distances, the source shape is therefore already a good approximation of the neighborhood, and the identity baseline in the table performs close to the other methods. In contrast, with structural distances we see a much larger spread. For structural neighborhoods \mathcal{N}_s , most of the neighbors share a similar structure. Here, StructureNet’s reconstruction errors of the source shape become apparent, showing a structural error E_r^{st} comparable to the identity baseline. StructEdit, on the other hand, only needs to encode local shape deltas. We benefit from the large degree of consistency between the deltas of different source shapes, allowing us to encode local neighborhoods more accurately. This effects of this benefit can also be confirmed visually in the upper part of Figure 3.

	\mathcal{N}^g				\mathcal{N}^s				
	chair	table	furn.	avg.	chair	table	furn.	avg.	
E_{qc}^{geo}	ID	1.822	1.763	1.684	1.756	1.629	1.479	1.446	1.518
	SN _{0.2}	1.760	2.076	1.626	1.821	1.308	1.208	1.243	1.253
	SN _{0.5}	1.722	2.068	1.558	1.783	1.241	1.103	1.135	1.160
	SN _{1.0}	1.768	2.189	1.554	1.837	1.232	1.057	1.017	1.102
	SE	1.593	1.655	1.561	1.603	1.218	1.000	1.015	1.078
E_{qc}^{st}	ID	1.281	1.215	1.288	1.261	1.437	1.303	1.442	1.394
	SN _{0.2}	1.081	0.878	1.015	0.991	1.466	3.484	1.414	2.121
	SN _{0.5}	0.871	0.729	0.873	0.824	1.373	3.300	1.204	1.959
	SN _{1.0}	0.751	0.667	0.726	0.715	1.763	3.622	1.167	2.184
	SE	0.559	0.524	0.741	0.608	0.609	0.451	0.676	0.579

Figure 4. **Edit Generation.** The visual results show generated deltas applied to source shapes and the table compares the learned distribution of deltas to the ground truth distribution. An aggregated quality and coverage error is shown, using both geometric and structural distances. Our method benefits from the consistency of delta distributions, giving an improved performance.

4.2. Edit Generation.

Next, we compare our learned distribution $p(\Delta S'_{ij}|S_i)$ to the ground truth distribution $p(\Delta S_{ij}|S_i)$ using two measures. The *coverage error* E_c measures the average distance from a ground truth sample to the closest generated sample, while the *quality error* E_q measures the average distance from a generated sample to the closest ground truth sample.

$$E_c^* := \frac{1}{r_{\mathcal{N}}|\mathbf{S}|k_{\text{test}}} \sum_{S_i \in \mathbf{S}} \sum_{S_j \in \mathcal{N}_i} \min_{\Delta S'_{ij}} d_*(S_i + \Delta S'_{ij}, S_j)$$

$$E_q^* := \frac{1}{r_{\mathcal{N}}|\mathbf{S}|N_{\Delta}} \sum_{S_i \in \mathbf{S}} \sum_{\Delta S'_{ij}} \min_{S_j \in \mathcal{N}_i} d_*(S_i + \Delta S'_{ij}, S_j),$$

where the generated shape delta is sampled according to the learned distribution $\Delta S'_{ij} \sim p(\Delta S'_{ij}|S_i)$. We use $N_{\Delta} = 100$ samples per source shape, and average over all source shapes in the test set \mathbf{S} . k_{test} is the neighbor count of each neighborhood \mathcal{N}_i in the test set. We evaluate the quality and coverage errors with both geometric distances d_{geo} and structural distances d_{st} . The coverage and quality metrics can be combined by adding the two metrics for each source shape, giving the Chamfer distance between the generated samples and the ground truth samples of each source shape, denoted as E_{qc}^* , where $*$ can be geo or st.

Figure 4 compares the results of our method to the baselines. In the table, the identity baseline has low quality error, but extremely high coverage error, since it approxi-

mates the distribution of deltas for each source shape with a single sample near the mode of the distribution. StructureNet and StructEdit approximate the distribution more accurately. Since in StructureNet, we cannot learn neighborhoods explicitly, we sample from Gaussians in latent space that are centered at the source shape, with sigmas 0.2, 0.5, and 1.0. Larger sigmas improve coverage at the cost of quality. StructEdit encodes deltas explicitly, allowing us to learn different types of neighborhoods and to make use of the similarity between the delta distributions at different source shapes. *This is reflected in a significantly lower error in nearly all cases.* The supplementary provides separate quality and coverage for each entry.

Figure 4, top, shows examples of multiple edits generated for several source shapes. Learning an accurate distribution of shape deltas allows us to expose a wide range of edits each source shape. Our method can learn different types of neighborhoods, corresponding different types of edits. We can see that properties of these neighborhoods are preserved in our learned distribution: geometric neighborhoods \mathcal{N}^g preserve the overall proportions of the source shape and have a large variety of structures; while the reverse is true for structural neighborhoods \mathcal{N}^s . We show interpolations between edits in the supplementary.

4.3. Edit Transfer

Edit transfer maps a shape delta ΔS_{ij} from a source shape S_i to a different source shape S_k . First, we encode the shape delta conditioned on the first source shape, and decode it conditioned on the other source shape: $\Delta S'_{kl} = d(S_k, e(S_i, \Delta S_{ij}))$. Since the two source shapes generally have a different geometry and structure, the edit needs to be adapted to the new source shape by the decoder. The two edits should perform an analogous operation on both shapes. Our synthetic dataset provides a ground truth for analogous shape deltas. Shapes in this dataset are divided into groups of 96 shapes, and the shape delta ΔS_{ij} between any pair of shapes (S_i, S_j) in a group has a known analogy in all of the other groups. When transferring an edit, we measure the geometric distance d_{geo} and structural distance d_{st} of the modified shape from the ground truth analogy.

$$E_t^* = \frac{1}{r_{\mathcal{N}}} d_*(S_k + \Delta S_{kl}, S_k + \Delta S'_{kl}), \quad (15)$$

where ΔS_{kl} is the ground truth analogy and $\Delta S'_{kl}$ the predicted analogy. In case an edit is not transferable, like adding an armrest to a shape that already has an armrest, we define an identity edit that leaves the source shape unchanged.

Figure 5 compares the transfer error of our method to baseline. In absence of ground truth edit correspondence for PartNet we qualitatively compare edit transfers on PartNet in the top rows. Our transferred edits better mirror the given edit, both in the properties of the source shape that it

\mathcal{N}		chair	sofa	stool	c. \rightarrow s.	c. \rightarrow st.	avg.
E_t^{geo}	Identity	1.002	0.938	0.892	0.892	0.938	0.932
	StructureNet	0.868	0.764	0.721	0.888	1.307	0.910
	StructEdit	0.586	0.566	0.599	0.572	0.698	0.604
E_t^{st}	Identity	0.941	1.328	0.333	0.333	1.328	0.853
	StructureNet	0.208	0.161	0.025	0.671	0.871	0.387
	StructEdit	0.005	0.001	0.003	0.002	0.123	0.027

Figure 5. **Edit Transfer.** We transfer the edit of the source shape in the top row to analogous edits in the remaining rows, using StructureNet (SN) and StructEdit (SE) on PartNet. The table quantifies the transfer error between shapes of the same group (columns 1 – 3), or between different groups (columns 4 and 5) of the synthetic dataset. Our explicit encoding of edits improves transfer quality.

modifies, and in the properties that it preserves. The table provides quantitative evaluation on the synthetic dataset. We show both transfers within the same group, and transfers between different groups, where some edits are not applicable and need to be mapped to identity. Our method benefits from consistency between deltas and achieves lower error.

4.4. Raw Point Clouds and Images.

Our latent space of shape deltas can be used for several interesting applications, such as the edit transfers we showed earlier. Here we demonstrate two additional applications.

First, we explore variations for raw, unlabelled point clouds. We can transform can transform the point cloud into our shape representation using an existing method [26], generate and apply edits, and then apply the changes back to the point cloud. For details please see the supplementary. Results on several raw point clouds sampled from ShapeNet [7] meshes are shown in Figure 6.

Second, we create cross-modal analogies, between images and point clouds. The images can be converted to our shape representation using StructureNet [25]. This allows us to define an edit from a pair of images, and to transfer this edit to a point cloud, using the same approach as described previously. Details are given in the supplementary. Results for several point clouds and image pairs are shown in Figure 7 on data from the ShapeNet dataset.

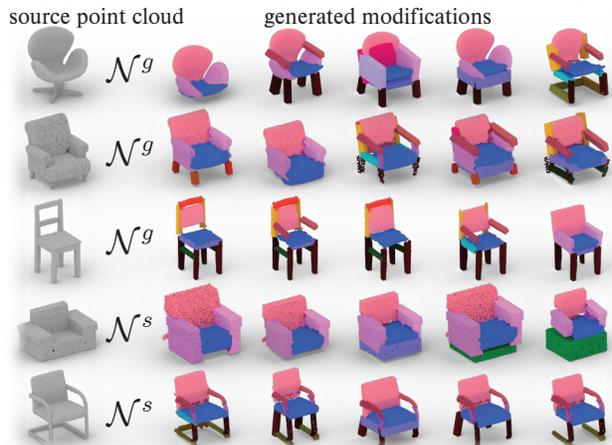


Figure 6. **Exploring Point Cloud Variations.** Edits for point clouds can be created by transforming a shape into a point cloud, applying the edit, and passing the changes back to the point cloud.

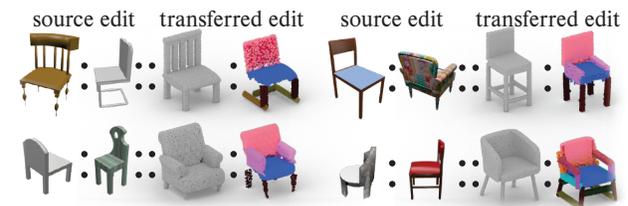


Figure 7. **Cross-Modal Analogies.** We can transfer edits between modalities by transforming different shape modalities to our structured shape representation.

5. Conclusion

We presented a method to encode shape edits, represented as shape deltas, using a specialized cVAE architecture. We have shown that encoding shape deltas instead of absolute shapes has several benefits, like more accurate edit generation and edit transfer, which has important applications in 3D content creation. In the future, we would like to explore additional neighborhood types, add sparsity constraints to modify only a sparse set of parts, and encode chains of edits. While we demonstrated consistency of shape deltas in their latent space, our method remains restricted to class-specific transfers. It would be interesting to try to collectively train across different by closely-related shape classes.

Acknowledgements

This research was supported by NSF grant CHS-1528025, a Vannevar Bush Faculty Fellowship, KAUST Award No. OSR-CRG2017-3426, an ERC Starting Grant (SmartGeometry StG-2013-335373), ERC PoC Grant (SemanticCity), Google Faculty Awards, Google PhD Fellowships, Royal Society Advanced Newton Fellowship, and gifts from Adobe, Autodesk, Google, and the Dassault Foundation.

References

- [1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017. [2](#)
- [2] Amir Arsalan Soltani, Haibin Huang, Jiajun Wu, Tejas D Kulkarni, and Joshua B Tenenbaum. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1511–1519, 2017. [2](#)
- [3] Ilya Baran, Daniel Vlasic, Eitan Grinspun, and Jovan Popović. Semantic deformation transfer. In *ACM Transactions on Graphics (TOG)*, volume 28, page 36. ACM, 2009. [2](#)
- [4] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'77*, pages 659–663, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc. [3](#)
- [5] Mirela Ben-Chen, Ofir Weber, and Craig Gotsman. Spatial deformation transfer. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 67–74. ACM, 2009. [2](#)
- [6] Martin Bokeloh, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun. An algebraic model for parameterized shape editing. *ACM Transactions on Graphics (TOG)*, 31(4):78, 2012. [2](#)
- [7] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. [6](#), [8](#)
- [8] Lu Chen, Jin Huang, Hanqiu Sun, and Hujun Bao. Cage-based deformation transfer. *Computers & Graphics*, 34(2):107–118, 2010. [2](#)
- [9] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016. [2](#)
- [10] Etienne Corman, Justin Solomon, Mirela Ben-Chen, Leonidas Guibas, and Maks Ovsjanikov. Functional characterization of intrinsic and extrinsic geometry. *ACM Transactions on Graphics (TOG)*, 36(4):59a, 2017. [2](#)
- [11] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017. [2](#), [3](#)
- [12] Noa Fish, Melinos Averkiou, Oliver Van Kaick, Olga Sorkine-Hornung, Daniel Cohen-Or, and Niloy J Mitra. Meta-representation of shape families. *ACM Transactions on Graphics (TOG)*, 33(4):34, 2014. [2](#)
- [13] Ran Gal, Olga Sorkine, Niloy J Mitra, and Daniel Cohen-Or. iwires: an analyze-and-edit approach to shape manipulation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 33. ACM, 2009. [2](#)
- [14] Lin Gao, Jie Yang, Yi-Ling Qiao, Yu-Kun Lai, Paul L Rosin, Weiwei Xu, and Shihong Xia. Automatic unpaired shape deformation transfer. In *SIGGRAPH Asia 2018 Technical Papers*, page 237. ACM, 2018. [2](#)
- [15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. [2](#)
- [16] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018. [2](#)
- [17] J Gwak, Christopher B Choy, Animesh Garg, Manmohan Chandraker, and Silvio Savarese. Weakly supervised generative adversarial networks for 3d reconstruction. *arXiv preprint arXiv:1705.10904*, 2017. [2](#)
- [18] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):90, 2019. [2](#)
- [19] Dominic Jack, Jhony K Pontes, Sridha Sridharan, Clinton Fookes, Sareh Shirazi, Frederic Maire, and Anders Eriksson. Learning free-form deformations for 3d object reconstruction. In *Asian Conference on Computer Vision*, pages 317–333. Springer, 2018. [2](#)
- [20] Vladislav Kraevoy, Alla Sheffer, Ariel Shamir, and Daniel Cohen-Or. Non-homogeneous resizing of complex models. In *ACM Transactions on Graphics (TOG)*, volume 27, page 111. ACM, 2008. [2](#)
- [21] Andrey Kurenkov, Jingwei Ji, Animesh Garg, Viraj Mehta, JunYoung Gwak, Christopher Choy, and Silvio Savarese. Deformnet: Free-form deformation network for 3d shape reconstruction from a single image. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 858–866. IEEE, 2018. [2](#)
- [22] Chun-Liang Li, Manzil Zaheer, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov. Point cloud gan. *arXiv preprint arXiv:1810.05795*, 2018. [2](#)
- [23] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. 36(4), 2017. [2](#)
- [24] Chongyang Ma, Haibin Huang, Alla Sheffer, Evangelos Kalogerakis, and Rui Wang. Analogy-driven 3d style transfer. In *Computer Graphics Forum*, volume 33, pages 175–184. Wiley Online Library, 2014. [2](#)
- [25] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *ACM Transactions on Graphics (TOG)*, *Siggraph Asia 2019*, 38(6):Article 242, 2019. [2](#), [6](#), [8](#)
- [26] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019. [2](#), [6](#), [8](#)

- [27] Charlie Nash and Christopher KI Williams. The shape variational autoencoder: A deep generative model of part-segmented 3d objects. In *Computer Graphics Forum*, volume 36, pages 1–12. Wiley Online Library, 2017. 2
- [28] Maks Ovsjanikov, Wilmot Li, Leonidas Guibas, and Niloy J Mitra. Exploration of continuous variability in collections of 3d shapes. In *ACM Transactions on Graphics (TOG)*, volume 30, page 33. ACM, 2011. 2
- [29] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 4
- [30] Raif M Rustamov, Maks Ovsjanikov, Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Leonidas Guibas. Map-based exploration of intrinsic shape differences and variability. *ACM Transactions on Graphics (TOG)*, 32(4):72, 2013. 2
- [31] Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions. *arXiv preprint arXiv:1905.06292*, 2019. 2
- [32] Ayan Sinha, Asim Umesh, Qixing Huang, and Karthik Ramani. Surfnet: Generating 3d shape surfaces using deep residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6040–6049, 2017. 2
- [33] Robert W Sumner and Jovan Popović. Deformation transfer for triangle meshes. *ACM Transactions on graphics (TOG)*, 23(3):399–405, 2004. 2
- [34] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2088–2096, 2017. 2
- [35] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations*, 2019. 2
- [36] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3d point clouds via graph convolution. 2018. 2
- [37] Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. Neural kinematic networks for unsupervised motion retargetting. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2
- [38] Hao Wang, Nadav Schor, Ruizhen Hu, Haibin Huang, Daniel Cohen-Or, and Hui Huang. Global-to-local generative model for 3d shapes. In *SIGGRAPH Asia 2018 Technical Papers*, page 214. ACM, 2018. 2
- [39] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive o-cnn: a patch-based deep representation of 3d shapes. In *SIGGRAPH Asia 2018 Technical Papers*, page 217. ACM, 2018. 2
- [40] Weiye Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 3dn: 3d deformation network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1038–1046, 2019. 2
- [41] Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiqian Cheng, and Yueshan Xiong. Symmetry hierarchy of man-made objects. In *Computer graphics forum*, volume 30, pages 287–296. Wiley Online Library, 2011. 2
- [42] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems*, pages 82–90, 2016. 2
- [43] Zhijie Wu, Xiang Wang, Di Lin, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Sagnet: Structure-aware generative network for 3d-shape modeling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2019)*, 38(4):91:1–91:14, 2019. 2
- [44] Shihong Xia, Congyi Wang, Jinxiang Chai, and Jessica Hodgins. Realtime style transfer for unlabeled heterogeneous human motion. *ACM Transactions on Graphics (TOG)*, 34(4):119, 2015. 2
- [45] Kai Xu, Honghua Li, Hao Zhang, Daniel Cohen-Or, Yueshan Xiong, and Zhi-Quan Cheng. Style-content separation by anisotropic part scales. In *ACM Transactions on Graphics (TOG)*, volume 29, page 184. ACM, 2010. 2
- [46] Weiwei Xu, Jun Wang, KangKang Yin, Kun Zhou, Michiel Van De Panne, Falai Chen, and Baining Guo. Joint-aware manipulation of deformable models. In *ACM Transactions on Graphics (TOG)*, volume 28, page 35. ACM, 2009. 2
- [47] Xinchun Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in Neural Information Processing Systems*, pages 1696–1704, 2016. 2
- [48] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4541–4550, 2019. 2
- [49] Jie Yang, Lin Gao, Yu-Kun Lai, Paul L Rosin, and Shihong Xia. Biharmonic deformation transfer with automatic key point selection. *Graphical Models*, 98:1–13, 2018. 2
- [50] Mehmet Ersin Yumer, Siddhartha Chaudhuri, Jessica K Hodgins, and Levent Burak Kara. Semantic shape editing using deformation handles. *ACM Transactions on Graphics (TOG)*, 34(4):86, 2015. 2
- [51] M Ersin Yumer and Niloy J Mitra. Learning semantic deformation flows with 3d convolutional networks. In *European Conference on Computer Vision*, pages 294–311. Springer, 2016. 2
- [52] Kun Zhou, Weiwei Xu, Yiyang Tong, and Mathieu Desbrun. Deformation transfer to multi-component objects. In *Computer Graphics Forum*, volume 29, pages 319–325. Wiley Online Library, 2010. 2