DeformSyncNet: Deformation Transfer via Synchronized Shape Deformation Spaces

MINHYUK SUNG*, Adobe Research
ZHENYU JIANG†, The University of Texas at Austin
PANOS ACHLIOPTAS, Stanford University
NILOY J. MITRA, University College London and Adobe Research
LEONIDAS J. GUIBAS, Stanford University

Fig. 1. We propose DeformSyncNet to jointly learn an idealized canonical latent space, encoding all possible deformations from one shape to any other, as well as an individualized linear deformation space, realized in a particular way for each specific shape. These individual deformation spaces are *synchronized* via the connection to the canonical space, resulting in consistent deformations across an entire shape category. Here, a user's manipulations on a shape is projected to a learned plausible shape space (left) while maintaining the shape structure such as part connectivity (red circle) and symmetry (green circle) and also instantly transferred to multiple other shapes from the same category (right), *without* leveraging any correspondence information during network training. Note how the edit only affects the other chairs with arms (blue circle).

Shape deformation is an important component in any geometry processing toolbox. The goal is to enable intuitive deformations of single or multiple shapes or to transfer example deformations to new shapes while preserving the plausibility of the deformed shape(s). Existing approaches assume access to point-level or part-level correspondence or establish them in a preprocessing phase, thus limiting the scope and generality of such approaches. We propose DeformSyncNet, a new approach that allows consistent and synchronized shape deformations without requiring explicit correspondence information. Technically, we achieve this by encoding deformations into a class-specific idealized latent space while decoding them into an individual, model-specific linear deformation action space, operating *directly* in 3D. The underlying encoding and decoding are performed by specialized (jointly trained) neural networks. By design, the inductive bias of our networks results in a deformation space with several desirable properties, such as path invariance across different deformation pathways, which are then also approximately preserved in real space. We qualitatively and quantitatively evaluate our framework against multiple alternative approaches and demonstrate improved performance.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; *Shape analysis*; **Machine learning approaches**.

Additional Key Words and Phrases: Deformation, Deformation Transfer, Shape Editing, Shape Embedding, 3D Deep Learning

ACM Reference Format:
Minhyuk Sung, Zhenyu Jiang, Panos Achlioptas, Niloy J. Mitra, and Leonidas J. Guibas. 2020. DeformSyncNet: Deformation Transfer via Synchronized Shape Deformation Spaces. *ACM Trans. Graph.* 39, 6, Article 262 (December 2020), 16 pages. https://doi.org/10.1145/3414685.3417783

*This work was equally contributed by M.Sung and Z. Jiang as co-first authors.
†This work was done when Z. Jiang attended Tsinghua University.

Authors' addresses: Minhyuk Sung, Adobe Research; Zhenyu Jiang, The University of Texas at Austin; Panos Achlioptas, Stanford University; Niloy J. Mitra, University College London and Adobe Research; Leonidas J. Guibas, Stanford University.

0730-0301/2020/12-ART262 $15.00
https://doi.org/10.1145/3414685.3417783

1 INTRODUCTION

Shape deformation is an essential task for both computer graphics and computer vision. For example, in shape retrieval applications, one can deform a retrieved shape to better meet the user's desiderata. No matter how large the available 3D shape repositories have become [Stratasys; Trimble; TurboSquid], shape deformation still plays a critical role in shape modeling and shape retrieval because there are infinite imaginable shape variations, and creating novel shapes from scratch is still a taxing task. Due to its importance and broad impact, shape deformation has been extensively studied by the geometry processing community in the last decades. Such work can be broadly grouped into two categories. In the first category, one wants to plausibly deform *a single shape based on user input/interactions* [Igarashi et al. 2005; Lipman et al. 2005; Sorkine et al. 2004; Yumer and Kara 2014]. Specifically, one can allow users to edit shapes using a set of *deformation handles*, which are intuitive and procedural parameters either directly coming from the 3D CAD models or generated from parametrizing or approximating each part of a 3D model [Mo et al. 2019b; Xu et al. 2009; Zheng et al. 2011]. Such deformation handles, however, mostly overparametrize the deformation space making it hard to maintain the plausibility or semantic constraints of a shape under arbitrary parameter changes. Consequently, user edits have to be constantly *projected* to the plausible deformation space [Gal et al. 2009; Liu et al. 2017], which is an error-prone process without strict quality guarantees. In the second category, one may want to apply *deformation transfer*, where algorithmic techniques propagate the deformation prescribed on one shape to another shape, or even to a collection of shapes [Baran et al. 2009; Ben-Chen et al. 2009; Chen et al. 2010; Fish et al. 2014; Ovsjanikov et al. 2011; Sumner and Popovic 2004; Zhou et al. 2010]. While deformation transfer alleviates the burden of individually deforming each shape, such algorithms typically expect explicit correspondences between shapes or deformation handles, which is an unrealistic demand in practice.

Nevertheless and leaving for a moment aside the practical considerations, if one assumes the existence of correspondences between deformation handles, some natural ideas emerge. In the projection's case, the correspondences can help us discover and exploit statistical *correlations* among the deformation parameters, which can be further incorporated in, and improve the quality of the projection [Fish et al. 2014]. In the case of deformation-transfer, one can simply transfer the desired changes at the granularity of handles (i.e., from the source's handle(s) to the target's ones) tapping on the correspondence-induced regularization. These are some of the important reasons of why many previous works assume that correspondences are available as groundtruth supervision [Baran et al. 2009; Ben-Chen et al. 2009; Chen et al. 2010; Sumner and Popovic 2004; Yang et al. 2018; Zhou et al. 2010], or estimate correspondences in a preprocessing phase [Yumer and Kara 2014].

In practice, however, computing dense correspondences is expensive or even ill-defined for heterogeneous collection of shapes, e.g., a collection of chairs that includes swivel and four-legged models. Some 3D datasets provide part-level semantic annotations [Mo et al. 2019c; Yi et al. 2016], but these are insufficient to indicate correspondences for *fine-grained* deformations. For all these reasons, in this work, we propose novel neural networks that allow us to do deformation projection and transfer without relying on *any* precomputed notion of shape correspondences (i.e., not even between handles).

Concretely, we introduce DeformSyncNet, a neural architecture that jointly learns (i) an *idealized canonical latent space* of shape encodings, where all possible deformations from any shape to any other are possible, as well as (ii) *individual* linear deformation spaces for each real shape that reflect the particular ways that the shape may (*or may not* be able to) deform in 3D. These shape-specific deformation spaces are synchronized by being connected through the canonical space. We design the canonical latent space as an *affine* space where shape deformations, arising as vectors connecting latent shape encodings, are the main objects of interest — and have a meaning irrespective of the source and target shapes involved. A deformation is transferred by decoding that latent deformation vector in the context of the new source shape and applying the resulting 3D deformation to that shape in real space. The individual deformation spaces, each of which is represented as a *dictionary* of linear deformations, are linked together so as to encourage the network to share the same parameters for the individual linear functions. Additionally, the properties of the affine space naturally structure the latent space to impose cycle consistency between various deformations, without additional loss functions or regularization during the neural network training.

Our approach can benefit from (but does not require) deformation handles. If we are given deformation handles for a particular shape, these can be easily connected to the individual deformation space we learn. Since, both spaces are linear in our setting, the projection from editing via deformation handles to a plausible shape can be simply computed as an orthogonal projection to a linear learned space. During network training, we also enforce that the learned deformation space for each shape becomes a *subspace* of the given deformation handle space so that it follows the given deformation operations, while capturing common constraints across the shapes. In this way, *correlations* among the deformation handles can emerge in the learned space.

In the experiments, we successfully apply our framework for the purposes of *co-editing* of human-made shapes and show qualitatively intuitive results. Quantitatively, we show that our trained networks outperform other modern (neural-net-based) and classical (ICP-based) shape deformation approaches, both in terms of fitting accuracy and quality of deformation transfer.

In summary, our contributions are:

(i) a novel approach for learning synchronized linear deformation spaces for each shape in a category, without explicit correspondences;
(ii) a unified framework enabling both projection of user-edited shapes to plausible counterparts in the shape space as well as transfer of the deformation to other shapes in the same category; and
(iii) a neural network design that is simple yet outperforms existing deep deformation methods in terms of both quality of fitting and deformation transfer.

2 RELATED WORK

2.1 3D Shape Deformation

3D shape deformation has been a long-standing problem in computer graphics. Earlier work introduced methods enabling interactive free-form deformations, while preserving local characteristics of the shape. Some well-known examples are Laplacian editing [Sorkine et al. 2004] and as-rigid-as-possible manipulation [Igarashi et al. 2005] that regularize deformations to maintain local curvature based on mesh Laplacians and local rigidity, respectively. Recent work focuses more on target-driven deformation, deforming and *fitting* a source shape to a target. For the fitting stage, researchers have used iterated closest point (ICP) iterations [Huang et al. 2017; Li et al. 2008] to establish point-wise correspondences and subsequently minimized an energy function based on them. However, ICP is prone to fail when the source shape is significantly different from the target, violating the ICP locality assumption. Recent neural-net-based learning approaches directly predict the deformation offset either in voxel grid [Hanocka et al. 2018; Jack et al. 2018; Kurenkov et al. 2018; Yumer and Mitra 2016] or for point samples on the source shape [Groueix et al. 2019; Mehr et al. 2019; Wang et al. 2019a] resulting in better fitting accuracy.

Both traditional and recent learning-based approaches do not, however, learn the shape variability from the given dataset, and thus cannot verify the semantic *plausibility* of the deformed shapes. Specifically, they cannot examine whether the deformed shape looks like one of the exemplars in the database. Instead, we train a network by performing target-driven deformation, while simultaneously requiring it to learn the shape variation space. Hence, in interactive editing, we can project the user's input through the deformation handles to the learned shape space and preserve semantic plausibility. We also demonstrate in our experiments that our method performs the target-driven deformation comparably or even better than previous methods.

Note that our method also differs from other deep 3D generative models that learn a shape variation space [Achlioptas et al. 2018; Wu et al. 2016] in that it does not generate shapes from a latent representation. Instead, when decoding, it takes an existing shape as input and generates a variant by deforming it directly in 3D. Hence, our method enables *reusing* existing 3D shape data with their associated meta-information, e.g., mesh structure, color, texture, and part hierarchy — which can be carried along in the deformation.

2.2 Deformation Transfer

Deformation transfer has remained an important problem in 3D shape editing after Sumner and Popovic [2004] introduced the concept in their pioneering work. The goal is to automatically propagate the result of shape deformation performed on one shape to others, thus saving users' time and effort. This problem has been comprehensively investigated in many previous works. While Sumner and Popovic [2004] require dense shape correspondences, Zhou *et al.* [2010] and Yang *et al.* [2018] extended their work to only use keypoint correspondences. Ben-Chen *et al.* [2009] and Chen *et al.* [2010] overcame the limitation of requiring single-component manifold meshes, and proposed cage-based methods to deal with any representations of shapes. Baran *et al.* [2009] improved the method to cope with very different shapes for the transfer (e.g., humans to animals) and require only region-level semantic correspondences.

The main remaining limitation is the necessity of shape correspondences as input (in the level of either points, parts, or cages). Recently, Gao *et al.* [2018] proposed a neural-net-based method that enables inter-class deformation transfer without correspondences across the classes. Their network, however, still needs to have *intra-class* correspondences of shapes during the training. In our work, we aim to deal with a diverse collection of man-made 3D shapes, such as ShapeNet models [Chang et al. 2015], where the correspondences between shapes are not identified or even clearly defined. Also, their work, and a recent work of Yin *et al.* [2019], aim to learn transform of shapes across *different* domains, inspired by analogous work in the image space [Isola et al. 2017; Zhu et al. 2017], whereas our goal is to learn deformation transfer for shapes in the *same* domain.

Another notable exception is the StructEdit work [Mo et al. 2019a] that learns transfer of shape differences without any correspondence supervision. Compared with our method, StructEdit transfers *structural or topological* differences of shapes, which cannot be immediately applied to modify a mesh or a point cloud. It also strongly depends on training with shape data annotated with consistent hierarchies [Mo et al. 2019c]. In contrast, our method focuses on *continuous* deformation and direct editing of existing 3D models.

Deformation transfer is closely related to *shape analogies* (originated from image analogies [Hertzmann et al. 2001]), finding a shape x such that $a : b = c : x$ given shapes a, b, and c. Rustamov *et al.* [2013] first introduced a method performing shape analogies based on functional correspondences. More sophisticated recent variations on shape analogies and shape generation based on functional correspondences include [Huang et al. 2019a], [Huang et al. 2019b]. Wu *et al.* [2016] showed how the latent space of 3D GAN can be used for that (similar to word2vec [Mikolov et al. 2013] in machine learning). Compared with these, we neither require correspondences nor decode directly from a latent space without deforming an input shape.

Lastly, we note that similar ideas to deformation transfer have been also studied for other research topics in graphics such as style transfer [Ma et al. 2009; Xu et al. 2010] and motion retargetting [Villegas et al. 2018; Xia et al. 2015], but under the same assumption of that shape correspondences are available.

2.3 Shape Constraint Analysis

While many 3D human-made objects are highly structured, deformation handles accompanied by the 3D models often do not fully acknowledge the underlying structure and thus allow breaking it with arbitrary modifications. As alternatives, researchers have introduced 3D shape analyses extracting the global structure of 3D shapes for editing, such as orthogonality and parallelism of wireframes [Gal et al. 2009] and bounding boxes [Zheng et al. 2011], symmetry [Wang et al. 2011], and articulation of parts [Xu et al. 2009]. Such analyses for *individual* shapes are, however, not able to capture *semantic* constraints or correlations of the handles that can only be observed from families of shapes. Hence, the other line of work extended these ideas to shape *co-analysis* and aimed to discover the semantic relationships among the handles from a

collection of shapes. For example, Kim *et al.* [2013] fit a template part bounding box structure to shapes in order to be able to cluster shapes and find part-level correspondences. Based on such fitted template structures, Ovsjanikov *et al.* [2011] demonstrates ways of exploring the shape space, Zheng *et al.* [2014] investigates co-occurrence of parts, and Fish *et al.* [2014] calculate joint distributions of part box bounding parameters. Yumer *et al.* [2014] also discovers co-constrained abstraction of shapes in the same family. These works, however, analyze shapes based on a *template* structure and thus cannot be scaled to handle 3D models in online repositories that have immense diversity.

3 METHOD

As mentioned in the introduction, key to our approach is an idealized latent space where points represent shapes, and vectors connecting points represent shape differences or deformations. A traditional latent space approach would implement deformation transfer as simply adding the deformation vector to the point representing a new latent source shape (in order to obtain its deformed target version), followed by a decoding of the latter from latent to 3D space (Figure 2). Instead, we propose to decode the deformation vector itself into a deformation *action* which can be applied *directly* in real 3D space to the new shape to be deformed.

In Section 3.1, before we discuss the realization of these operations on real shapes, we introduce an abstract mathematical framework of how things might work when point-wise correspondences across the shapes are known. We then relax these idealized assumptions to generate a specialized set of deformations a real shape can undergo, by developing *shape-specific* deformation action dictionaries. In Section 3.2, we introduce how the learned shape deformation space can be leveraged in a practical shape editing scenario to produce a plausible output shape by *projecting* the user's input to the deformation space. In Section 3.3, we describe how we implement the shared functional form of deformations via neural networks. Finally, in section 3.4, we compare our neural networks with the other recent neural networks learning deformations and discuss the advantages of our approach.

3.1 Abstract Deformation Space

We consider a collection of shapes $\overline{X}$, where each shape $x \in \overline{X}$ is represented as a point cloud with n points (i.e., $\forall x \in \overline{X}, x \in \mathbb{R}^{3n}$). We aim to encode deformations between all ordered pairs in $\overline{X}$ within a low-dimensional continuous canonical latent space. In particular, we create an *affine action space* $\langle X, V, \oplus \rangle$, where $\overline{X} \subseteq X \subseteq \mathbf{R}^{3n}$, V is a k-dimensional vector space, and $\oplus$ is an action of the additive group of V on the set X: $X \times V \rightarrow X$. An affine action space is defined with the following three properties [Gallier 2011; Tarrida 2011]:

(i) (Free action) $x \oplus \vec{0} = x$, for every $x \in X$.
(ii) (Additive action) $(x \oplus u) \oplus v = x \oplus (u + v)$, for every $x \in X$, and every $u, v \in V$.
(iii) (Transitive action) For every $x, y \in X$, there exists a unique $v \in V$ such that $x \oplus v = y$.

Note that the vector space V parametrizes not the shape (point) space but the deformation (vector) space of shape differences. Our goal is to make a vector $v \in V$ be a *anchor-free* representation of deformation, *forgetting* the original shape where the deformation is applied. Thus, the same type of deformation action can be applied to any arbitrary shape $x \in X$ by applying the same deformation parameter $v \in V$ — i.e., the deformation $\overrightarrow{xy} \in V$ such that $x \oplus \overrightarrow{xy} = y$ can be transferred to the other shape $z \in X$ by computing $z \oplus \overrightarrow{xy}$.

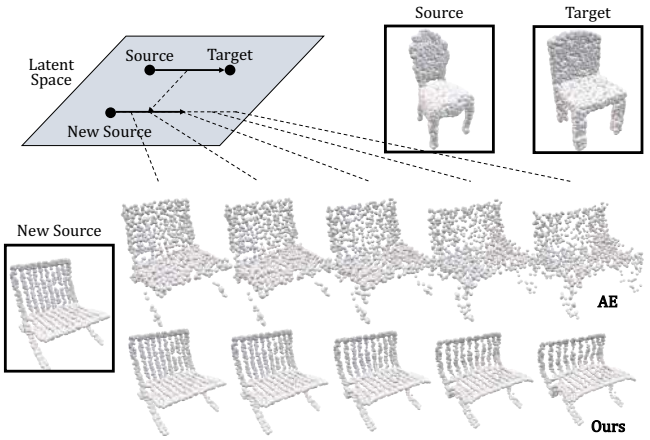


Fig. 2. Comparison of latent vector transfer between an autoencoder (AE) latent space [Achlioptas et al. 2018] and our latent space. The vector from source shape to target is transferred to another source shape, and several points are sampled along the direction of the deformation vector, at scales 0.25, 0.5, 1.0, 1.5, and 2.0. In the autoencoder latent space, the vector does not convey the *difference* between two shapes and, sometimes, even pushes the new shape outside the valid region, making it less plausible. In our latent space, the difference between the source and target shapes is properly transferred, regardless of the location of the new source shape.

We choose an affine space as our latent space of deformations because of its desirable properties, facilitating exploration of the space in downstream applications. From the basic properties above, the following properties can also be derived (Refer to Gallier *et al.* [2011] and Tarrida *et al.* [2011] for proofs. $\overrightarrow{xy} \in V$ denotes a deformation vector from shape x to y, i.e., $x \oplus \overrightarrow{xy} = y$.):

(i) (Identity) $\overrightarrow{xx} = \vec{0}$, for every $x \in X$.
(ii) (Anticommutativity) $\overrightarrow{xy} = -\overrightarrow{yx}$, for every $x, y \in X$.
(iii) (Transitivity) $\overrightarrow{xy} + \overrightarrow{yz} + \overrightarrow{zx} = \vec{0}$, for every $x, y, z \in X$.
(iv) (Parallelogram law) $\overrightarrow{xy} = \overrightarrow{zw} \Leftrightarrow \overrightarrow{xz} = \overrightarrow{yw}$, $\forall\, x, y, z, w \in X$.

The challenge here is how to make the vectors in V encode the *same* type of deformation(s) across all the shapes in X. Consider a shape autoencoder, where $\mathcal{E} : \mathbb{R}^{3n} \rightarrow \mathbb{R}^k$ and $\mathcal{D} : \mathbb{R}^k \rightarrow \mathbb{R}^{3n}$ are encoding and decoding neural networks, respectively. Given these mappings, the simplest way to create an affine space for the shape differences is to take the Euclidean embedding space $\mathbb{R}^k$ as the vector space V of the affine space: $x \oplus v := \mathcal{D}(\mathcal{E}(x) + v)$.

A *vanilla* autoencoder, however, fails to structure the embedding space in a way that a vector indicates the same type of deformation *everywhere* over the space. Figure 2 shows an example of taking a vector from the source to target shape in the embedding space and adding this to a new source shape, with different scales along the vector. This fails to make the new shape adapt the deformation from the source to target, transforming it into implausible shapes.

We aim to design the latent space so that such vector addition can properly transfer the shape difference.

Linear Deformation Representations. Let us first assume that point-wise correspondences are known for all pairs of shapes — to understand a simpler setting. In this case, we can specify a deformation as an *offset vector* for corresponding points. When the points are ordered in a consistent way so that corresponding points across the shapes have the same index in the order, we can consider a *linear* deformation function as an action $\oplus$ of the affine space:

$$x \oplus v \coloneqq \mathbf{A}v + x, \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{3n \times k}$, with k being the dimension of V. The deformation for a $v \in V$ is now explicitly defined as adding per-point offsets ($\mathbf{A}v$) to the source point cloud, and the same v produces the same offsets for the corresponding points across all the shapes. This action on shapes is free ($x \oplus \vec{0} = x$), transitive if k is smaller or equal to $3n$, yet large enough to capture all possible differences of shapes in X, and also is in the additive group of V over X. In the context of an autoencoder, this can be understood as constructing a latent shape space to be decoded to a *linear subspace* $\mathbf{A}$ of the shape point clouds, so that a free vector v over that latent space describes the same point displacements in $3D$ space, regardless of the original shape. Alternatively, one can simply interpret $\mathbf{A}$ as a set of principal axes of the point clouds $x \in \overline{X}$, in the context of PCA.

Now we consider the case when the point-wise correspondences are *unknown*, and possibly even *ill-defined*. [1] Hence, in this case, it is not possible to define a canonical linear deformation function for all shapes with a single matrix $\mathbf{A}$. Instead, we propose to predict a matrix $\mathbf{A}$ for each individual shape $x \in X$ using a neural network $\mathcal{F} : X \to \mathbb{R}^{3n \times k}$. By denoting the output for the input shape x as $\mathbf{A}_x$, the action $\oplus$ is now written as follows:

$$x \oplus v \coloneqq \mathbf{A}_x v + x. \tag{2}$$

The action is still free and transitive (with a large enough $k \leq 3n$) but is *not* in the additive group anymore, since now $\mathbf{A}_x$ is source-dependent; $\mathbf{A}_x$ and $\mathbf{A}_y$ for different shapes x and y may be inconsistent. To impose the property the additive action above, we jointly train a shape encoder $\mathcal{E} : X \to \mathbb{R}^k$ along with the above network $\mathcal{F}$ so that the deformation vector $\overrightarrow{xy}$ from shape x to y is given by $\mathcal{E}(y) - \mathcal{E}(x)$, while the matrix $\mathbf{A}_x$ is predicted as $\mathcal{F}(x)$. The deformation from the point cloud x to y is now computed as follows:

$$d(x \to y) \coloneqq \mathcal{F}(x)\,(\mathcal{E}(y) - \mathcal{E}(x)) + x\,, \tag{3}$$

where the deformation vector is computed in latent space through $\mathcal{E}$, decoded into real space by $\mathbf{A}_x$ as a set of point offsets in an x-specific way, and added to the original point cloud for x.

This utilization of the two networks $\mathcal{E}$ and $\mathcal{F}$ realizes our overall plan: learning (i) a canonical affine deformation space from $\mathcal{E}$; and (ii) an individual linear (affine) deformation space for each shape from $\mathcal{F}$, and connecting these individual shape action spaces by sharing the same source latent space. The individual deformation spaces, defined by the matrices $\mathbf{A}_x$, are thus *synchronized* across the shapes via the connection through the canonical latent deformation space.

Empirically, even without the joint network training, the matrix $\mathbf{A}_x$ can be predicted consistently for *similar* input shapes due to the characteristics of neural networks when learning a *smooth* function — several recent works have proposed unsupervised methods finding shape correspondences based on this network property [Genova et al. 2019; Groueix et al. 2018; Li et al. 2019; Sung et al. 2018; Tulsiani et al. 2017; Zhao et al. 2019]. However, we found that enforcing the property of the additive action regularizes the $\mathbf{A}_x$ matrices to become consistent even when the shapes are dissimilar (see Section 4.1 and Figure 5).

Relation to Deep Functional Dictionaries. The matrix $\mathbf{A}_x$ can be interpreted as a *dictionary* of deformations (point-wise displacements), and in this context, we call the network $\mathcal{F}$ a *deformation dictionary predictor* in the rest of the paper. The idea of learning dictionaries of functions on shapes is analogous to the Deep Functional Dictionaries work by Sung *et al.* [2018], but there are two main differences. First, our dictionaries are learned from pairwise deformations, not from the input functions over the shapes (thus, out work is purely self-supervised). Second, we synchronize our dictionaries by explicitly learning a mapping of the dictionaries to a canonical space, instead of solely relying on the smoothness of the learned functions.

3.2 Deformation Projection

Many 3D models in online repositories are either described with geometric primitives or spline surfaces or are decomposed into smaller parts that can be easily approximated with bounding primitives [Mo et al. 2019c; Sung et al. 2017; Yi et al. 2017]. This information provides the user with intuitive deformation handles, although mostly they *overparametrize* the deformation space, meaning that arbitrary changes of the parameters do not always give a valid shape. When assuming that the deformation function of the given handles is *linear* — we observed that most deformation parameters coming from the human-made 3D models are translation and scaling of parts, which are linear deformations — in our framework, we can easily project the user input on the given handles to the learned deformation space using simple computation. Additionally, we can guide our networks during training to learn the deformations that are present in the given parameter space.

Projecting Shape Editing to Deformation Space. Let the input linear deformation function (defined by the deformation handles) denoted as $\mathbf{B}_x\,(z + z_0)$, where $\mathbf{B}_x \in \mathbb{R}^{3n \times m_x}$, m_x is the number of deformation handles, and $z_0 \in \mathbb{R}^{m_x}$ is the default parameters. We also have the learned linear deformation function for the shape: $\mathbf{A}_x v + x$ (Equation 2), where $x = \mathbf{B}_x z_0 \in \mathbb{R}^{3n}$ is the initial shape and $\mathbf{A}_x \in \mathbb{R}^{3n \times k}$ gives a *valid* variation space of x. The goal of the projection is this: given the user input via the deformation handles $z \in \mathbb{R}^{m_x}$, we find $v \in \mathbb{R}^k$ (a valid variation) that minimizes the difference of point-wise offsets from the edited shape to the valid shape: $\|\mathbf{B}_x z - \mathbf{A}_x v\|_2^2$. In particular, we consider a shape editing scenario where the user edits the shape through *some* of the deformation handles and prescribes values for them at each time, and then the system automatically

[1] For a heterogeneous collection of shapes, particularly for human-made objects, the correspondences may not be clearly defined for some pairs, either geometrically or semantically.

projects the edit to the valid space. Thus, given the *equality* constraints to the deformation handles $\mathbf{C}z = z_{\text{in}}$, where $\mathbf{C} \in \mathbb{R}^{m_x \times m_x}$ is a matrix indicating the selected handles and $z_{\text{in}} \in \mathbb{R}^{m_x}$ is the sparse user inputs, we find $\widehat{z} \in \mathbb{R}^{m_x}$ defined as follows:

$$\begin{aligned} \widehat{z} := \underset{z}{\operatorname{argmin}} \min_{v} \|\mathbf{B}_x z - \mathbf{A}_x v\|_2^2 \\ \text{s.t.} \quad \mathbf{C}z = z_{\text{in}} \,. \end{aligned} \tag{4}$$

When $\mathbf{B}'_x$ denotes the matrix $\mathbf{B}_x$ except for the *columns* of the selected handles, Equation 4 can be written in a following form:

$$\widehat{z'} := \underset{z'}{\operatorname{argmin}} \min_{v} \left\| (\mathbf{B}'_x z' + c) - \mathbf{A}_x v \right\|_2^2 , \tag{5}$$

where $\widehat{z'}$ is the best values for the rest of the parameters, and $c = \mathbf{B}\mathbf{C}z_{\text{in}}$. For any point cloud $y \in \mathbb{R}^{3n}$, the projection distance to the linear space of $\mathbf{A}_x$ is computed as:

$$\min_{v} \|\mathbf{A}_x v - y\|_2^2 = \left\| \left(I - \mathbf{A}_x \mathbf{A}_x^{\dagger}\right) y \right\|_2^2 ,$$

where $\mathbf{A}_x^{\dagger}$ is a pseudoinverse of $\mathbf{A}_x$ (note that $\mathbf{A}_x \mathbf{A}_x^{\dagger} \neq I$ since $\mathbf{A}_x$ is a thin matrix, i.e., $3n \gg m_x$). Hence, Equation 5, again, can be rewritten as follows:

$$\widehat{z'} := \underset{z'}{\operatorname{argmin}} \left\| \left(I - \mathbf{A}_x \mathbf{A}_x^{\dagger}\right) (\mathbf{B}'_x z' + c) \right\|_2^2 , \tag{6}$$

The solution of this linear regression $\widehat{z'}$ is defined as:

$$\widehat{z'} = \mathbf{P}_x^{\dagger} \mathbf{Q}_x c, \tag{7}$$

where $\mathbf{P}_x = \left(I - \mathbf{A}_x \mathbf{A}_x^{\dagger}\right) \mathbf{B}'_x$, $\mathbf{Q}_x = -\left(I - \mathbf{A}_x \mathbf{A}_x^{\dagger}\right)$, and $\mathbf{P}_x^{\dagger}$ is pseudoinverse of $\mathbf{P}_x$. Note that, when the user edits through one deformation at each time, $\mathbf{P}_x^{\dagger}$ and $\mathbf{Q}_x$ can be precomputed for each handle.

Some deformation handles may have inequality constraints; e.g., scale parameters must be positive numbers. The least squares problem in Equation 6 can also be quickly solved with inequality constraints using standard techniques such as interior-point methods or active-set methods.

Projecting Network Output to Deformation Handle Space. At training time, we can also project the output deformed shape of the networks to the given deformation handle space to make the deformation *describable* with the given handles (the effect of this option is investigated in Section 4.5):

$$d_{\text{proj}}(x \to y) := \mathbf{B}_x \mathbf{B}_x^{\dagger} \left(\mathcal{F}(x) \left(\mathcal{E}(y) - \mathcal{E}(x)\right) + x\right) . \tag{8}$$

We remark that our framework can easily exploit any type of linear deformation handles, even when they are inconsistent and their numbers are different across the shapes.

3.3 Neural Network Design and Losses

For the networks $\mathcal{E}$ and $\mathcal{F}$ in Section 3.1, we can use any neural network architecture that processes point clouds and produces a global shape feature and per-point features, respectively. Note that every *three* rows of $\mathbf{A}_x \in \mathbb{R}^{3n \times k}$ determine the offset of each point *independently* and thus can be predicted as a per-point feature. In our experiments (Section 4), we use PointNet [Qi et al. 2017]; its classification architecture for the encoder $\mathcal{E}$ and the segmentation architecture for the deformation dictionary predictor $\mathcal{F}$. Figure 3 shows the entire pipeline. A Siamese structure of the encoder $\mathcal{E}$ takes a pair of source $x \in \overline{X}$ and target $y \in \overline{X}$ shapes as input and predicts the deformation vector $(\mathcal{E}(y) - \mathcal{E}(x)) \in \mathbb{R}^k$. The dictionary predictor $\mathcal{F}$ takes *only* the source shape as input and predicts the linear deformation dictionary $\mathcal{F}(x) \in \mathbb{R}^{3n \times k}$. For faster convergence in training, we normalize each column in the dictionary $\mathcal{F}(x)$ to have unit norm. The deformation of the source shape fitting to the target is predicted by computing point offsets $(\mathcal{F}(x)\,(\mathcal{E}(y) - \mathcal{E}(x))) \in \mathbb{R}^{3n}$ and adding it to the source shape as described in Equation 3. Additionally, if deformation handles are provided for each shape, the deformed source shape can also be projected to their space as described in Equation 8. The fitting error from the output deformed source shape $d(x \to y)$ to the target shape y is measured using Chamfer Distance (cf., [Achlioptas et al. 2018; Fan et al. 2017; Groueix et al. 2019; Wang et al. 2019a; Yifan et al. 2020]):

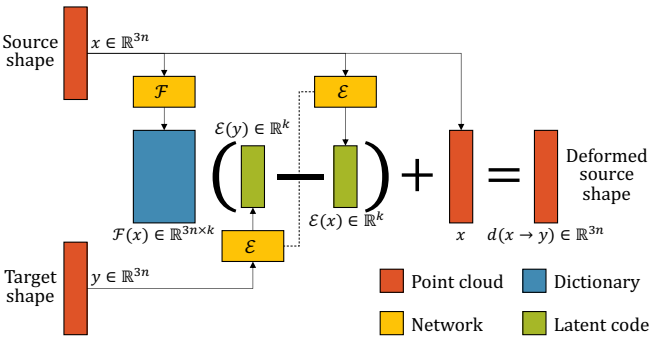


Fig. 3. Neural network pipeline. The encoder $\mathcal{E}$ takes both the source and target point clouds, and computes the latent deformation vector as $(\mathcal{E}(y) - \mathcal{E}(x))$. The deformation dictionary predictor $\mathcal{F}$ takes only the source shape and predicts the dictionary $\mathcal{F}(x)$. The outputs of these two modules are multiplied and added to the source point cloud to produce the deformed shape as $x \to x + \mathcal{F}(x)(\mathcal{E}(y) - \mathcal{E}(x))$.

$$\mathcal{L}_F := \text{Ch}\left(d(x \to y), y\right) . \tag{9}$$

We also follow the idea of Wang *et al.* [2019a] and Yifan *et al.* [Yifan et al. 2020] to preserve *symmetries* of man-made objects in the deformation. When a global reflection symmetry axis is given for the source shape, we flip the output deformed shape along the axis and minimize Chamfer distance to this:

$$\mathcal{L}_R := \text{Ch}\left(d(x \to y), \mathbf{R}\left(d(x \to y)\right)\right) , \tag{10}$$

where $\mathbf{R}$ is the mirroring operation along the given reflection axis.

Additionally, we also support sparsity regularization losses to enforce structure in the output dictionaries. For example, in shape editing scenarios, the user may want to identify the essential correlations among the given deformation handles. To discover them, we can apply $l1$-loss to the output dictionary matrices *after* projecting them to the given deformation handle space (as in Equation 8):

$$\mathcal{L}_{S1} := \frac{1}{k} \|\mathbf{B}_x^{\dagger} \mathcal{F}(x)\|_1 . \tag{11}$$

This loss can encourage the columns in the projected dictionary matrices to be sparse, so that they can capture strongly correlated deformation handles. Also, while we preset the number of columns

in the dictionary k during training (*i.e.*, the dimension of the latent space), we can find the minimal set of the columns necessary to handle all possible deformations by imposing a column-wise $l2,1$-loss to the projected dictionaries [Ding et al. 2006; Nie et al. 2010]:

$$\mathcal{L}_{S2,1} := \frac{1}{k}\|\mathbf{B}_x^{\dagger}\mathcal{F}(x)\|_{2,1}, \tag{12}$$

which makes the norm of each column to be close to zero [2]. Empirically, we found that this loss also plays the role of $l1$-loss, sparsifying each column, even while making the training more stable. Hence, we only employ this $l2,1$-loss in our final training loss, which is defined as follows:

$$\mathcal{L} = \mathcal{L}_F + \mathcal{L}_R + w_{S2,1}\mathcal{L}_{S2,1}, \tag{13}$$

where $w_{S2,1}$ is a weight for the sparsity loss. In Section 4.1, we analyze the effect of the sparsity loss.

Please note that we do not use any *cycle-consistency* loss since the consistency of the dictionaries $\mathcal{F}(x)$ across the shapes automatically emerges during network training, as discussed in Section 3.1. In Section 4.3, we empirically evaluate cycle-consistency with a dataset where, as ground-truth, the point correspondences are known across the shapes. The experiment demonstrates that our method, without any loss function for consistency, performs even better results compared with the network of Groueix *et al.* [2019], which explicitly optimizes using cycle-consistency losses.

3.4 Comparison with Other Neural Deformations

Recent neural networks learning target-driven deformations, such as 3DN [Wang et al. 2019a], Cycle Consistency [Groueix et al. 2019], and Neural Cages [Yifan et al. 2020], have an architecture analogous with ours in that they take a source-target pair of shapes as input, compute a latent vector for the shape difference, and apply it to the source shape. We discuss the main differences between these methods and ours, and describe how the differences affect performance and usage in downstream applications.

Learning Shape-Dependent Variation Space. While both our network and the others are trained to learn deformation from one shape to the other, we not only learn how to fit the input to the target but also discover the plausible variation space of the input shape, which is represented as a linear space with the deformation dictionary $\mathbf{A}_x$. Hence, in shape editing, when the deformation handles are given as linear deformation functions, the user's input can be easily projected back to the learned variation space as described in Section 3.2 and also demonstrated in Section 4.1. This is an additional capability of our method compared to the other methods. Moreover, our deformation dictionary $\mathbf{A}_x$ captures strong correlations among the given deformation handles with additional regularization loss (see Equation 12), and thus provides more intuitive interpretation of the learned deformation space, as discussed in Section 4.1.

Factorizing Deformation Representation. We also found that the factorization of shape offset representation into a source-dependent dictionary $\mathcal{F}(x)$ and a latent vector for a pair of shapes $(\mathcal{E}(y) - \mathcal{E}(x))$ gives a better performance in the fitting. See Section 4.2 for quantitative evaluations. While Neural Cages [Yifan et al. 2020] have a similar factorization to ours, the others (3DN [Wang et al. 2019a] and Cycle Consistency [Groueix et al. 2019]) immediately combine the information of the pair to the source shape in the network without separately manipulating the source shape beforehand.

[2]Note that we normalize the dictionary columns *before* projection, and thus the norm of dictionary columns *after* the projection can still be minimized (but cannot be zero).

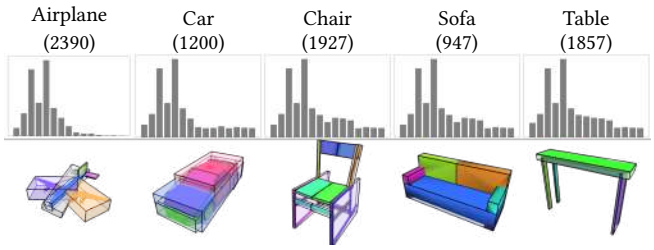


Fig. 4. ShapeNet part bounding box dataset. Each column shows the number of shapes, a histogram of the number of components, and a sample object with part bounding boxes for each category. We preprocess raw CAD models as described in Section 4.1 and collect models with number of components is in the range of [2, 16].

Enforcing Affine Space Properties. To attain the affine space properties in the latent space, in our framework, the deformation from one shape to the other is represented as the *subtraction* of two latent codes $(\mathcal{E}(y) - \mathcal{E}(x))$, as described in Section 3.1. In all the other methods, however, the deformation is learned by first *concatenating* two latent codes and processing it in the next layers. In Section 4.2, we conduct an ablation study, changing the computation of the latent deformation vector in our framework, similarly with the other methods. The results demonstrate that our architecture enforcing the affine space properties produces more plausible shapes in the deformation transfer compared with the other methods and the architecture in the ablation study.

4 RESULTS

We utilize our framework in the context of shape *co-editing* application. Also, we quantitatively evaluate our method and compare against state-of-the-art shape deformation techniques.

4.1 Qualitative Evaluations

Dataset and Network Training. We experiment with five categories from the ShapeNet repository [Chang et al. 2015], namely Airplane, Car, Chair, Sofa, and Table. Most of the 3D models in ShapeNet are composed of smaller parts that have simpler geometry. Thus, we compute oriented bounding boxes for parts using the Trimesh library [Dawson-Haggerty et al.] and take *translation* and anisotropic *scales* along each local coordinate as our deformation handles. Figure 4 shows the number of shapes in each category (in parentheses), the distribution of the number of components in each shape (first row), and a sample object with part bounding boxes (second row). Before computing the bounding boxes, we run a preprocessing to merge small components to their closest neighbors and also combine overlapping components, as illustrated in the work of Sung *et al.* [2017]. However, we do not group symmetric parts in order to have more freedom in editing each part. (Symmetries will

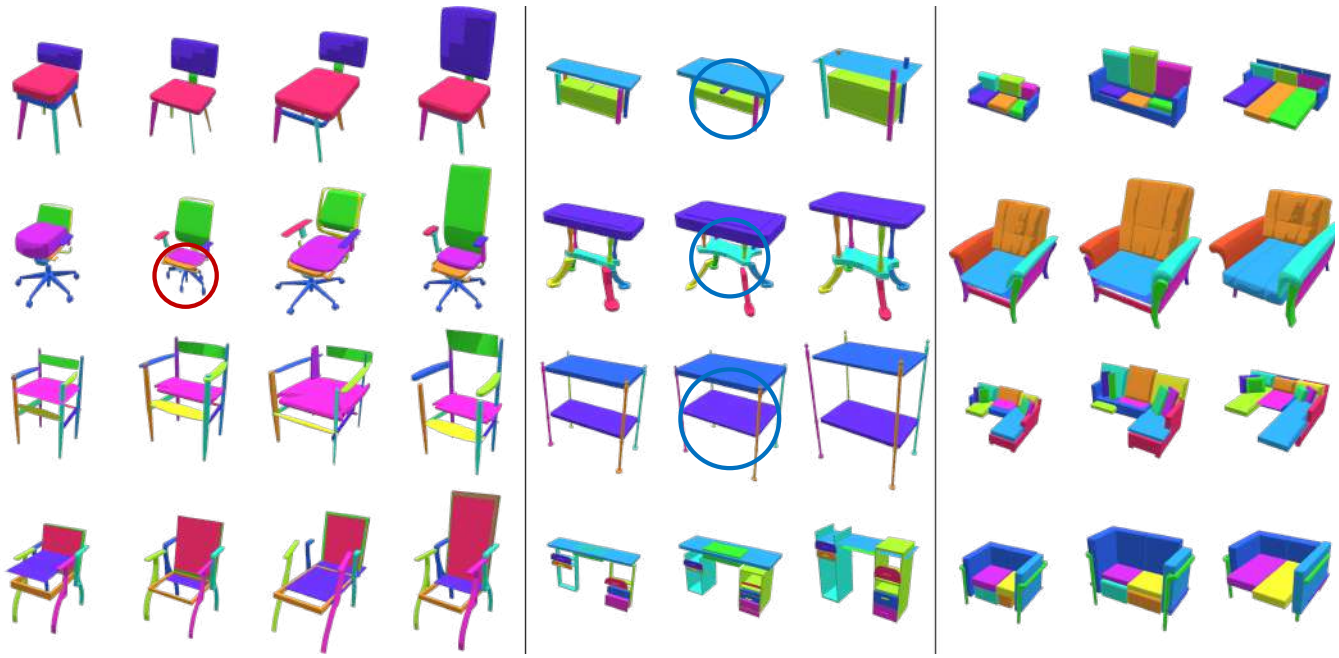


Fig. 5. Elements in the learned deformation dictionary. The columns are deformations along some elements in the dictionary (columns of $\mathbf{A}_x$). Each element shows a natural *mode* of shape variations changing local parts, and the variations of each element are also *consistent* across the shapes despite the difference in styles of the shapes. The second column of chairs shows scaling of swivel leg (red circle), and the element does not affect the shapes not including the swivel leg. Also, the second column of tables lifts up the shelf in the middle (blue circle) and makes no change if the part does not exist. Refer to the supplemental video for animated visualizations.

be *learned* by our networks, as illustrated in Section 4.1 and 4.4.) After preprocessing, we take models with a number of component parts in the range of $[2, 16]$. We do not normalize the scale of each shape individually to see natural shape variations (e.g., making a chair to a bench without decreasing the height), but all shapes are in the range of $[-0.5, 0.5]$ in each axis.

We train the networks for each category. We sample $2K$ points randomly over each shape and feed the networks with random pairs of the source and target shapes. Training, validation, and test sets are randomly split with the 85-5-10 ratio. We set the dimension of latent space (i.e., the number of dictionary elements) k to 64 and the weight of sparsity regularization loss $w_{S2,1}$ to 10.

Since we aim to deal with *uncurated* datasets where no manual annotations are present, we do *not* use any part labels in ShapeNet (unlike those used by Mo *et al.* [2019a]) or any correspondence supervision in network training. Also, such part labels are often insufficient to infer correspondences across the deformation handles.

Deformation Dictionaries. We illustrate the learned deformation dictionary $\mathbf{A}_x$ in Figure 5. Each column visualizes shapes deformed along some elements in the dictionaries. Thanks to the sparsity regularization loss (Equation 12), the elements show local and natural deformation *modes* and correlations among the given deformation handles. For example, the first and third columns of chairs translate and scale the seat along up and front directions, respectively, while preserving connectivities with the other parts. The last column also elongates the back part upward. Similar local deformations are also observed in sofas. Interestingly, the second column scales only swivel leg (red circle), and it does not affect the shape with a different type of leg. Similarly, in the second column of tables, the element translates the shelf in the middle along the up direction and does not make any changes if the shape does not include the shelf. The supplementary video shows how the shapes vary along the dictionary elements in animations.

Shape Co-Editing. We also describe how our framework can be employed for interactive shape co-editing and demonstrate qualitative results. We consider a scenario that the user modifies a source shape through one of the given deformations handles at each step. Given the user input, the system (i) first automatically *snaps* the edited shape to the plausible shape space while fixing the modified parameter as described in Section 3.2. When solving the least square in Equation 6, we use a constraint that all scale parameters should be greater than zero. Then, (ii) the projected deformation is transferred to the desired new source shape in the same category.

Figure 1 and 6 show some examples of the deformation projection and transfer. In Figure 6, given a source shape (first column), we randomly select one of the deformation handles and perturb the parameter (second column). If the selected handle is a translation along one of the local coordinates of a box (blue arrow in the figure), we vary the parameter in the range of $[-0.5 + t, 0.5 + t]$, where t is the given number. If the select handle is a scaling (red arrow in the figure), we pick a random number in the range of $[0.5s, 1.5s]$, where s is the default scale. Then, the modified shape is projected to

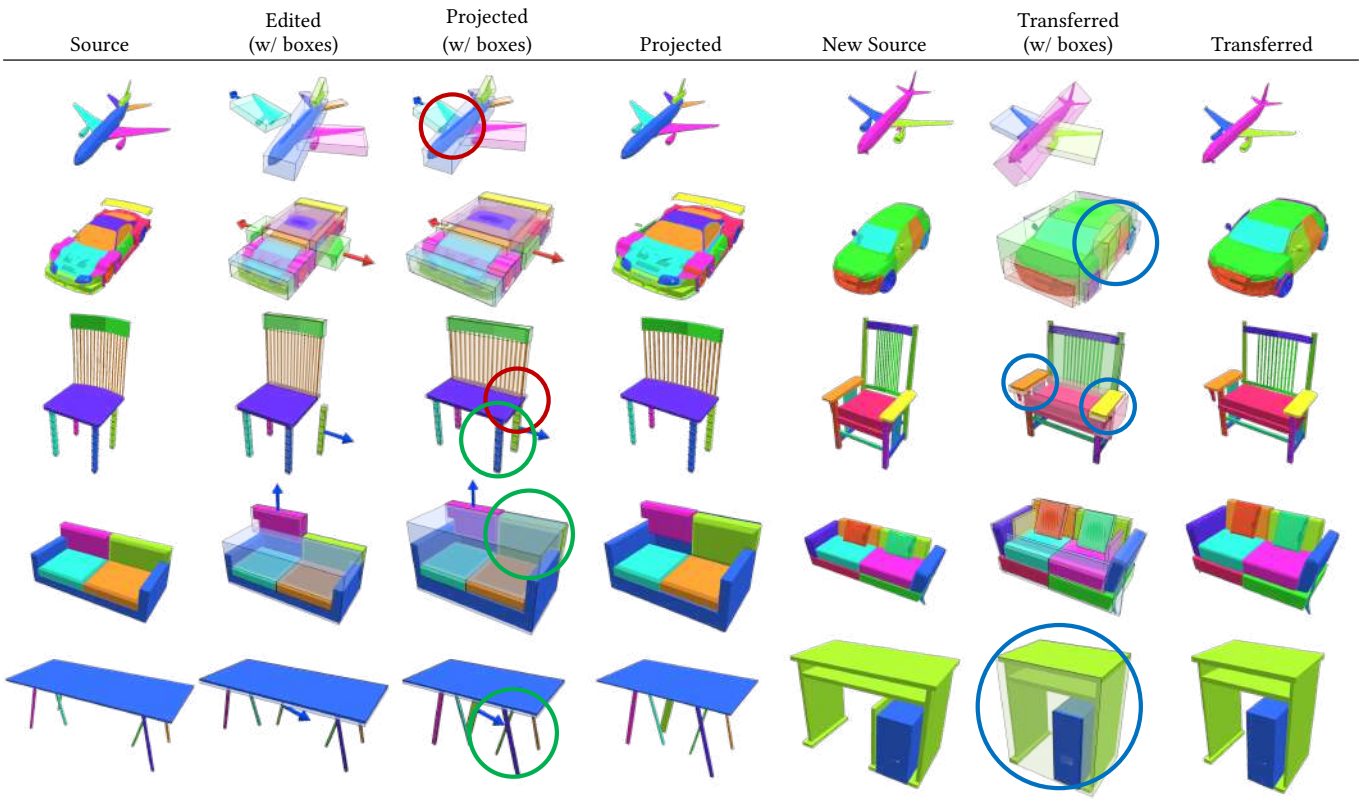


Fig. 6. Shape editing projection and transfer results. The first column is the source shape, the second column is the user edit (blue and red arrows mean translation and scaling along the direction, respectively), the third and fourth columns are the results of projection (with and without the part bounding boxes), the fifth column is the new source shape, and the sixth and seventh columns are the results of transferring the projected deformation to the new source shape (with and without the part bounding boxes). The projection adjusts the rest of the deformation handles while preserving part connectivity (red circles) and symmetry (green circles). Also, the projected deformations are naturally transferred to the new shape despite the different part structures (blue circles). See the supplemental video for more examples.

the learned plausible shape space (third and fourth column). Given another source shape (fifth column), the projected deformation is transferred (sixth and seventh columns) by taking the latent deformation vector and applying it to the new source shape with the learned shape-dependent action. The results show that the projection adjusts the rest of the parameters in a way to maintain the shape structure such as symmetry and part connectivity. For instance, right-wing of the airplane (first row) and a leg of the chair (third row) and the table (last row) are properly attached to the other parts after the projection while preserving symmetry. Also, the deformation on the source shape is naturally transferred to the new source shape even when the geometry or the deformation handles are different. For example, the tables in the last row have different part structure, but the translation of legs is naturally adopted to the new shape as decreasing the width. Refer to the supplemental video for more examples.

Effect of Sparsity Regularization Losses. We analyze the effect of sparsity regularization loss in Equation 12 by varying its weight. Figure 7 (a) shows the mean fitting error (Chamfer distance) of $1k$ random chair pairs when varying the weight from 0.1 to 10. While the fitting error increases with a larger weight, the errors with large weights are smaller than the ones of baseline deformation methods (see Section 4.2). In Figure 7 (b) and (c), we show how many non-zero elements and columns we obtain in the projected dictionaries $\mathbf{B}_x^{\dagger}\mathcal{F}(x)$ when increasing the weight and also varying the threshold of zero. The numbers of non-zero elements and columns dramatically decrease, indicating that the network discovers more strongly correlated deformation handles, while still learning all possible deformations.

4.2 Quantitative Evaluations Using ShapeNet

Next, we quantitatively evaluate our method using ShapeNet [Chang et al. 2015] dataset and compare with the other baseline methods.

Dataset and Network Training. For the quantitative evaluations, we use the same preprocessed dataset of ShapeNet with Groueix *et al.* [2019]. This dataset contains five categories: Airplane, Car, Chair, Lamp, and Table. The difference of this dataset with the one used in Section 4.1 is that it normalizes each point cloud to be fit in a uniform cube, which size is 2 for each axis. All networks including ours and baselines are trained as described in Section 4.1, by taking $2k$ random

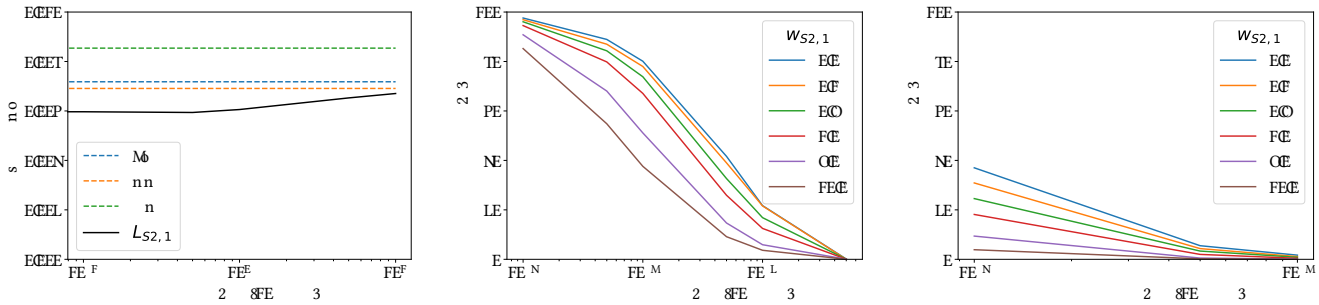


Fig. 7. Effect of Sparsity Regularization Losses. (a) Fitting error comparison when varying the weight of sparsity regularization loss in Equation 12. The x-axis is the weight (in log-10 scale), and the y-axis is the fitting error measured as Chamfer distance. The dot lines indicate the fitting errors of baseline methods: 3DN [Wang et al. 2019a], Cycle Consistency (CC) [Groueix et al. 2019] and Neural Cages (NC) [Yifan et al. 2020]. (b) The percentage of non-zero *elements* with varying weights. The x-axis is the zero threshold (in log-10 scale). (c) The number of non-zero *columns* with varying weights.

sample points in each shape and feeding random source-target shape pairs in each category. To maximize the fitting capability, in this experiment, we set the dimension of the latent space in our framework, k, to 512 and disable sparsity regularization loss ($w_{S2,1} = 0$). Also, in our framework, we do not use the projection to the deformation handle space described in Equation 8 during the network training, and the effect of the projection is analyzed in Section 4.5.

Baselines. We compare our method with non-rigid ICP by Huang *et al.* [2017] and three recent neural-network-based deformation methods mentioned in Section 3.4: 3DN [Wang et al. 2019a], Cycle Consistency (CC) [Groueix et al. 2019], and Neural Cages (NC) [Yifan et al. 2020]. We take point clouds as input to all networks and do not use the differentiable mesh sampling operator introduced in 3DN [Wang et al. 2019a], which can be easily plugged into any of the networks. All the baseline neural-net-based methods have a part in their architecture to compute deformation (*i.e.* shape difference) from the given source and target shapes and apply it to the source shape. Thus, we implement the deformation transfer of the other methods as applying the given source-target pair information to the other source shape, similarly with our method.

Ablation Study. We test the impact of enforcing affine space properties by modifying the part of computing latent deformation vector in our networks. As mentioned in Section 3.4, instead of taking the *difference* of two latent codes ($\mathcal{E}(y) - \mathcal{E}(x)$), we concatenate latent codes of source and target shapes and process it to produce the deformation vector [3].

Target-Driven Deformation and Transfer. We compare the methods in two aspects: fitting accuracy of deformations and plausibility of deformation transfer results.

For the fitting accuracy, we perform target-driven deformation, deforming a *source* shape to fit it to a *target* shape, with random 1,000 pairs of source and target test shapes in each category. Then, we measure the average Chamfer distance between the target and the deformed source shape (Fitting CD). Also, inspired by Groueix *et*

[3]Within the PointNet [Qi et al. 2017] classification architecture, we concatenate global features of the shapes produced from the max-pooling layer and then pass it through the same next MLP layers. The final output replaces ($\mathcal{E}(y) - \mathcal{E}(x)$) in our original architecture.

Table 1. Comparisons of fitting accuracy of deformation and plausibility of deformation transfer results. We compare our DeformSyncNet (DSN) with Non-rigid ICP (NR-ICP) [Huang et al. 2017], 3DN [Wang et al. 2019a], Cycle Consistency (CC) [Groueix et al. 2019] and Neural Cages (NC) [Yifan et al. 2020]. Concat* is an ablation study computing the latent deformation vector by concatenating two latent codes of shapes instead of taking the difference. As evaluation metrics for the plausibility and the fitting accuracy of deformation transfer results, we use mean Intersection over Union (mIoU) and Fitting Chamfer Distance (CD) of semantic parts for the former, and Minimum Matching Distance (MMD) and Coverage (Cov) for the latter. MMD and Cov are measured based on Chamfer distance. Bold is the best result, and underscore is the second-best result. See Section 4.2 for details.

Category		Airplane	Car	Chair	Lamp	Table
mIoU (%) ↑ is better	NR-ICP	66.7	61.2	77.5	65.5	66.0
	3DN	58.4	48.3	58.1	45.9	46.6
	CC	67.8	61.1	77.6	**65.6**	65.4
	NC	67.5	61.2	**78.2**	64.7	**66.9**
	DSN	**68.0**	**61.8**	77.7	64.8	66.2
	Concat*	58.6	52.9	59.1	51.7	46.6
Fitting CD ($\times 10^{-3}$) ↓ is better	NR-ICP	8.99	8.35	26.86	61.58	44.51
	3DN	2.54	5.04	6.32	14.15	8.51
	CC	3.26	4.18	9.81	30.65	14.61
	NC	6.73	7.49	18.82	41.40	25.80
	DSN	1.95	4.21	5.90	13.28	8.05
	Concat*	**1.65**	**3.98**	**4.91**	**9.87**	**6.48**
MMD-CD ($\times 10^{-3}$) ↓ is better	3DN	13.31	8.15	56.18	116.95	107.14
	CC	7.44	6.45	26.83	56.89	53.39
	NC	**6.10**	**4.96**	21.93	52.07	33.16
	DSN	6.40	5.54	**21.43**	**39.36**	**27.67**
	Concat*	17.85	8.84	52.45	150.40	129.40
Cov-CD (%) ↑ is better	3DN	16.0	6.9	6.6	8.1	10.0
	CC	30.3	14.1	30.9	27.6	25.7
	NC	30.6	**37.5**	39.5	32.3	35.9
	DSN	**31.0**	33.1	**41.3**	**34.1**	**38.9**
	Concat*	3.1	2.4	3.1	3.0	2.6

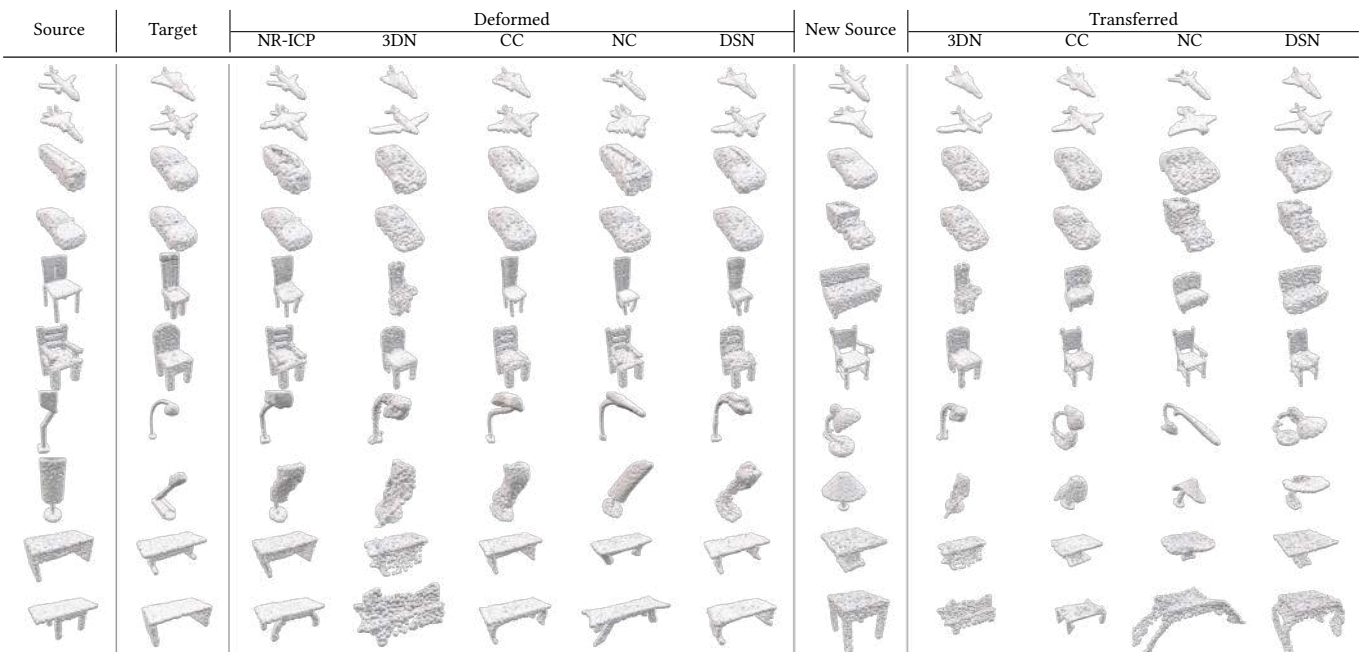


Fig. 8. Examples of qualitative evaluation results. The first and second columns are the source and target shapes, the next five columns are deformed source shapes fitted to the target, resulted by different methods, the next eighth column is the new source shape, and the next four columns are the results of deformation transfer from the source-target pair to the new source. The baseline methods compared with our DeformSyncNet (DSN) are Non-rigid ICP (NR-ICP) [Huang et al. 2017], 3DN [Wang et al. 2019a], Cycle Consistency (CC) [Groueix et al. 2019] and Neural Cages (NC) [Yifan et al. 2020].

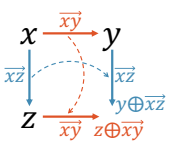
al. [2019], we evaluate the fittings by finding the closest points from the deformed source shape to the target and measuring mean Intersection over Union (mIoU) of semantic parts provided by Yi *et al.* [2016]. This metric demonstrates how beneficial each method at unsupervised co-segmentation or few-shot segmentation.

For the plausibility of deformation transfer results, the quantitative evaluation is challenging since it is not precisely determined without involving human perception. In our evaluation, we follow the ideas of Achlioptas *et al.* [2018] evaluating generative models. Achlioptas *et al.* introduce two *data-driven* evaluation metrics: Minimal Matching Distance (MMD-CD) and Coverage (Cov-CD). Minimal Matching Distance is the average of the Chamfer distance from each generated shape to its the closest shape in the reference dataset, indicating how likely the output shape looks like a real shape. Coverage is the proportion of the shapes in the reference dataset that are closest from each generated shape, showing how many variations are covered by the generated shapes. For these two, we randomly choose 200 pairs of source and new source shapes, and for each of the 10 target shapes, we transfer the source-to-target deformation to the new source shape. Then, we measure the metrics by taking the training set as the reference dataset; the averages for all target shapes are reported.

We report all results in Table 1. Bold is the best result, and underscore is the second-best result. Note that our DeformSyncNet is the only method that shows outstanding performance both in the fitting accuracy and in the plausibility of deformation transfer results. For the Fitting CD and mIoU, DeformSyncNet gives better accuracy in most of the categories compared with the other methods. The network in the ablation study, concatenating two latent codes, is the only one that shows the better fitting accuracy than DeformSyncNet in terms of Fitting CD, but its performances in other metrics are poor, particularly in MMD-CD and Cov-CD. For the MMD-CD and Cov-CD, our DeformSyncNet also outperforms the other methods. The only competitor is Neural Cages [Yifan et al. 2020], but it gives inferior accuracy of the fitting, as shown in Fitting CD.

Figure 8 illustrates some results of quantitative evaluation. Our DeformSyncNet shows the best fitting results in most of the cases and also captures well the difference between the source and target shapes in the transfer. For instance, the source and target chairs in the fifth row have differences in the width and height of the back part, and these differences are correctly transferred to the new shape, which has a different global structure. Also, in the sixth row, ours combines chair arms of the source shape to seat in the deformation, and this change is properly transferred to the new shape. The other methods tend not to transfer the *difference* but to do target-oriented deformation, making the new source shape close to the target shape.

Parallelogram Consistency Test. One of the beneficial properties of affine space is the *path invariance*, producing the same deformation regardless of the pathway from the starting point to the destination. This property is desirable in the downstream applications since it allows the user to explore the latent shape variation space freely without specifying the order of steps. We test this property by taking a random



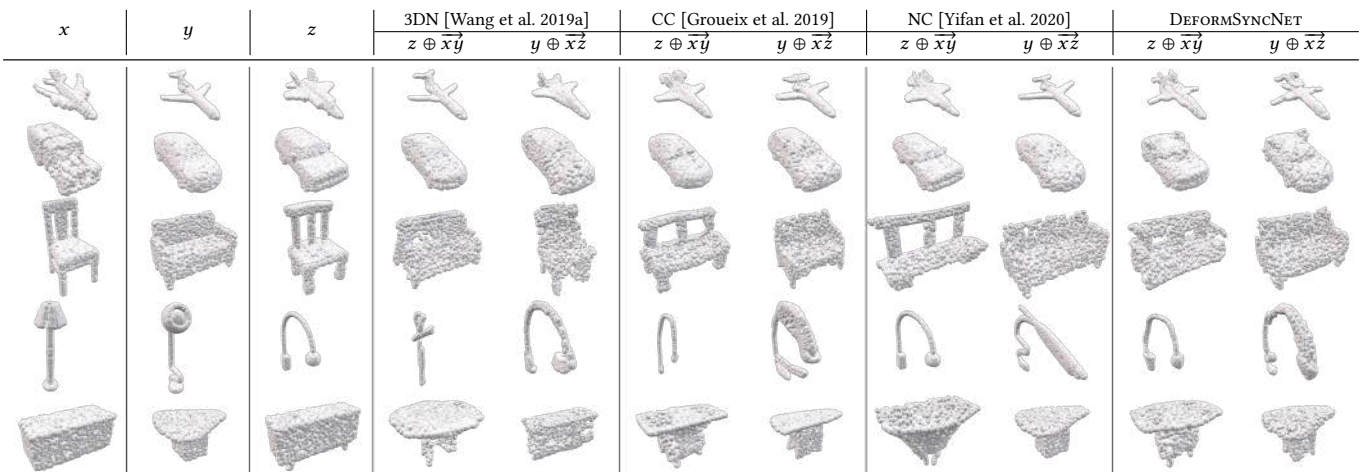


Fig. 9. Qualitative results of parallelogram consistency test. Our DeformSyncNet provides the most consistent result of deformation given two different pathways: $z \oplus \overrightarrow{xy}$ and $y \oplus \overrightarrow{xz}$. See Section 4.2 for the details.

parallelogram in the latent space. A triplet of shapes x, y, and z are given, and deformations are transferred in two ways: $\overrightarrow{xy}$ to z and $\overrightarrow{xz}$ to y (see inset). Then, we verify whether these two results are the same by measuring Chamfer distance.

The quantitative results comparing with other methods are reported in Table 2. Our DeformSyncNet gives much smaller differences between two pathways of deformations compared with the other methods. Cycle Consistency [Groueix et al. 2019] using regularization losses for the consistency provides the second-best results in all categories, but its fitting distance is much larger than ours. Figure 9 illustrates some of the qualitative results.

Table 2. Quantitative results of parallelogram consistency test. We measure Chamfer distance between two deformation results (going to the same destination but with different pathways): $z \oplus \overrightarrow{xy}$ and $y \oplus \overrightarrow{xz}$. Our DeformSyncNet (DSN) gives the minimal difference between the two results compared with the other methods.

Category		Airplane	Car	Chair	Lamp	Table
CD ($\times 10^{-3}$) ↓ is better	3DN	13.30	7.73	77.58	194.44	133.77
	CC	6.98	5.54	26.23	46.20	57.77
	NC	8.56	7.21	31.48	138.73	89.58
	DSN	**2.28**	**4.28**	**8.07**	**22.69**	**14.33**
	Concat*	16.70	7.98	69.58	103.80	122.50

4.3 Quantitative Evaluations and User Study Using Parametric 3D Models

For further quantitative analysis and user study, we experiment with parametric 3D models provided by Schulz *et al.* [2017] and compare our method with the other methods.

Dataset and Network Training. The dataset of Schulz *et al.* [2017] contains 74 parametric models, each of which has its own deformation parametrization. For each model, we uniformly sample $2k$ points and generate $1k$ variants with random parameters; 128 shapes out of them are used as the test set. The main difference of this dataset with ShapeNet in Section 4.2 is that point correspondences are *known* across the shapes since they are generated from the same parametric models. The point correspondences are *not used* as supervision in any experiment, but will be used as ground-truth in evaluations. During training, we do *not* project the deformed shape to the given deformation parameter space (see Equation 8).

Two-Way Consistency Evaluation. As discussed in Section 3.3, we empirically verify the effect of our network design enforcing consistency in the deformation, compared with Cycle Consistency in Groueix *et al.* [2019] leveraging dedicated loss functions for the consistency and also the other baseline methods. We train the networks for the shapes of *each* parametric model and measure the two-way consistency error for a point cloud pair (x, y) as follows:

$$\frac{1}{n}\sum_i \| (d_i(x \to y) - x_i) + (d_i(y \to x) - y_i) \|^2,$$

where $x_i \in \mathbb{R}^3$ and $(d_i(x \to y) - x) \in \mathbb{R}^3$ are i-th point of the point cloud x and its new position after deformation toward y, respectively, n is the number of points, and here we assume that all point clouds are ordered consistently based on the given point correspondences. We computed the mean of this two-way consistency error for 50 pairs, which are the first 50 of the largest Chamfer distances among $1k$ randomly generated pairs. Figure 10 (a) shows a comparison between the results of our method and the baseline methods. Each dot indicates the consistency error for a single parametric model, and the x-axis and y-axis are for our DeformSyncNet and the other methods, respectively. Our DeformSyncNet gives smaller two-way consistency errors than the other methods including Cycle Consistency in most cases: 70, 58, and 71 cases out of the 74 models compared with 3DN, Cycle Consistency, and Neural Cages, respectively.

Quantitative Evaluation of Deformation Transfer. Using the given deformation parametrization, we also quantitatively evaluate the performance of deformation transfer. If we choose all of the source,

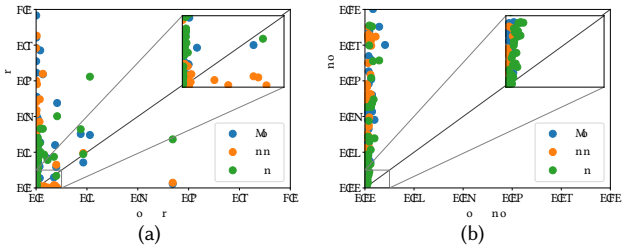


Fig. 10. Two-way consistency and deformation transfer results using parametric models from Schulz *et al.* [2017]. (a) Comparison of two-way cycle consistency errors between DeformSyncNet and the other methods. Each dot is a result of one of 74 parametric models, and x- and y-axes are DeformSyncNet and the other methods, respectively. Note that in this visualization, above the diagonal line indicates that the corresponding method performs worse than ours. Compared with 3DN, Cycle Consistency (CC), and Neural Cages (NC), ours gives smaller errors in 70, 58, and 71 cases, respectively, out of total 74 cases. (b) Comparison of deformation transfer errors (Chamfer distance between the prediction and the ground truth). Again, ours gives smaller errors in 74, 69, and 73 cases compared with 3DN, Cycle Consistency (CC), and Neural Cages (NC), respectively.

target, and destination (the new shapes to transfer deformation) shapes from the same parametric model, the ground truth of the deformation transfer can be computed by transferring the difference of parameters from source to target to the destination shape. Given the 50 test pairs per model above, we randomly pick another shape as the destination and compute Chamfer distance between the predicted and the ground truth shape. Figure 10 (b) illustrates the mean Chamfer distance; x-axis is DeformSyncNet, and y-axis is the other methods. In most cases, DeformSyncNet gives a magnitude smaller distances compared with the other methods. The numbers of cases out of the 74 models when DeformSyncNet outperforms 3DN, Cycle Consistency, and Neural Cages are 74, 69, and 73, respectively.

User Study for Deformation Transfer. We further assess the quality of deformation transfer results by taking the destination shape from a *different* parametric model (but in the same category) with the source and target shapes. Since we cannot compute the ground truth in this case, we conducted a user-study on Amazon Mechanical Turk. Among the 74 parametric models, we grouped 8 Airplane, Chair, and Table models per category and trained all the networks for each group, while still taking the source and target shape from the same parametric model. But in the test time, the destination shape is randomly selected from the other model in the same group — note that we take the source and target shapes from the same parametric model so that the participants can clearly identify the difference. From the same 50 test pairs per model above, in total 1,200 (50×8 models $\times 3$ groups), we randomly select 10% (120 pairs), assign the random destination shape, and create user study questions as shown in Figure 11. For each question, we ask human subjects (Turkers) to select *at least one* among four shown objects: the outputs of 3DN, Cycle Consistency, Neural Cages, and our DeformSyncNet. The associations between the methods and the objects are hidden to the Turkers, and the order of the objects is randomized. The Turkers are also encouraged to choose *multiple* objects if they think more than one options are equally good. In total, we collected 1,200

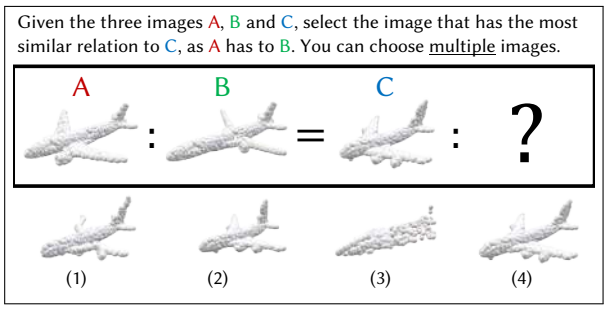


Fig. 11. User study example. The source (A) and target (B) shapes in the first row are sampled from the same parametric model, while the destination (C) shape is sampled from the other parametric model in the same category. The results of 3DN, Cycle Consistency, Neural Cages, and our DeformSyncNet are shown in the second row with a randomly permuted order. Multiple choices were allowed.

responses from a pool of 100 different participants (each question was answered by 10 distinct Turkers).

In the results, our DeformSyncNet was selected in 54.7% of all 1,200 responses, whereas 3DN, Cycle Consistency, Neural Cages were selected in 30.9%, 4.0%, and 46.6%, respectively. (Note that we allow *multiple* choices, and thus the sum of percentages is greater than 100.) The performance gap between DeformSyncNet and Neural Cages (the second best) is statistically significant as the McNemar's contingency test [McNemar 1947] between the two empirical (binomial) distributions has a p-value of $5e-4$ for the null hypothesis of that the two distributions have an equal marginal. We also remark that for each of the 10 responses collected for each question, DeformSyncNet was the most preferred one among all methods in 59 out of 120 questions, and also it tied up with the other most preferred one(s) in 17 questions.

4.4 Shape Structure Discovery

We examine the capability of our method of discovering shape structure, such as symmetry and part connectivity. To simplify the test, we employ a procedural model of tables from Tian *et al.* [Tian et al. 2019], which has a rectangular top and four legs at the corner. Once we train our networks with 5000 random samples of tables, in test time, we randomly perturb the default shape in 2000 times and project them back to the learned shape space as described in Section 4.1. Figure 12 shows examples of the perturbation and projections. When measuring the difference of symmetric scales, x, y, and z scales of legs, and the gap between the top and legs, the average ratios compared to the scale of the default shape along each axis are 9.60e-3, 6.94e-3, 9.60e-3, and 3.17e-2, respectively.

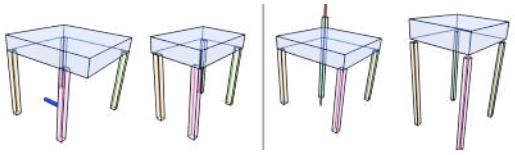


Fig. 12. Examples of perturbed (left) and projected (right) synthetic tables.

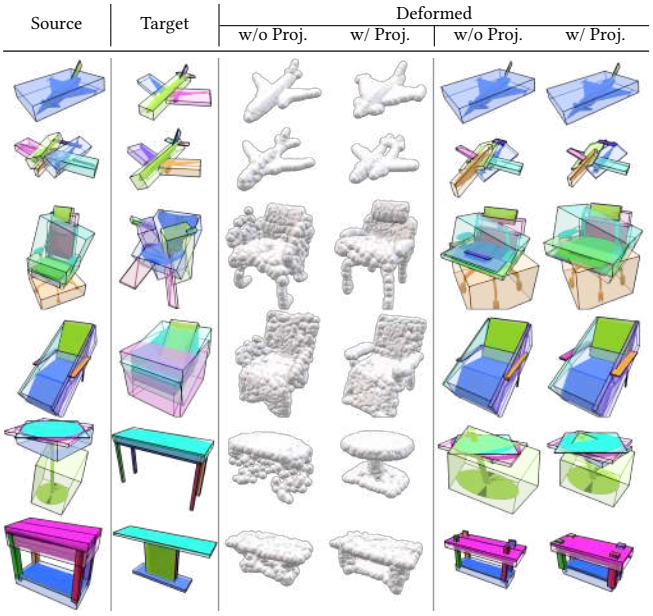


Fig. 13. Qualitative analysis of the effect of deformation projection to the deformation handle space during the network training. The third and fourth columns demonstrate that the training with the projection gives more plausible results in deformation. However, if the results are projected in test time, the difference becomes negligible.

4.5 Effect of Projection in Training

When the deformation handles are given for the shapes in the training dataset, we can try to project the deformed source shape to the given deformation handle space during the network training, as explained in Section 3.2. As well as the sparsity loss regularizing the dictionaries (in Equation 12), we can also project the deformed shape to the given space (Equation 8) and measure the fitting loss with it (changing Equation 9 to $\mathrm{Ch}(d_{\mathrm{proj}}(x \to y), y)$). Interestingly, a qualitative analysis demonstrates that the projection during the training guides the network to learn more plausible variations close to the input shape. Figure 13 shows some results when training our network with and without the projection during the training using the ShapeNet dataset and part bounding boxes in Section 4.1. In test time, the network model trained with the projection provides more reasonable variations of the source shapes (third and fourth columns), while this visual improvement does not make a significant change in the fitting distance to the target (Table 3). If we perform the same projection in *test* time (fifth and sixth columns) as well, the difference between two cases, training with and without the projection, becomes negligible despite the discrepancy in the results before the project.

4.6 Extension to Non-Linear Deformations

While our framework is designed to decode the latent space to a *linear subspace* of point cloud offsets, we also demonstrate that our framework can be extended to decode each axis of the latent space to a *non-linear* trajectory — but without guaranteeing the affine properties. The matrix multiplication in Equation 3 can be rewritten

Table 3. Quantitative result of training DeformSyncNet with and without the projection to the deformation handle space. The projection during the training does not make notable changes in the quantities of all the deformation transfer evaluation metrics. See Section 4.1 for the details of the evaluation metrics.

Category		Airplane	Car	Chair	Sofa	Table
Fitting CD ($\times 10^{-3}$)↓	w/o Proj.	3.53	**2.67**	6.82	4.75	7.91
	w/ Proj.	**3.27**	2.82	**5.93**	**3.76**	**7.55**
MMD-CD ($\times 10^{-3}$)↓	w/o Proj.	**1.12**	**1.97**	**7.04**	3.30	**6.43**
	w/ Proj.	1.13	2.11	7.25	**3.17**	6.50
Cov-CD (%)↑	w/o Proj.	**40.8**	36.5	**39.1**	38.8	40.3
	w/ Proj.	39.1	**37.5**	38.1	**41.2**	**40.9**

Table 4. Results of learning rotational motions in Shape2Motion dataset [Wang et al. 2019b] and fitting error comparison. DSN (L) and DSN (C) indicate DeformSyncNet learning linear and circular trajectories, respectively. The case learning circular trajectories performs better when the dimension of the latent space k is set to the number of rotating parts (DoF, the first two rows). The case learning linear trajectories also performs well if the dimension of latent space becomes high, e.g., $k = 64$ (the last two rows). Neural Cages (NC) fails to learn the rotational motions.

Model		Carton	Eyeglasses	Swing
CD ($\times 10^{-3}$) ↓ is better	DSN (L,k =DoF)	0.14	0.33	0.41
	DSN (C,k =DoF)	**0.12**	**0.20**	**0.32**
	3DN	0.04	0.08	**0.06**
	CC	0.09	0.26	0.78
	NC	1.94	0.71	7.04
	DSN (L,$k = 64$)	**0.03**	**0.03**	0.09
	DSN (C,$k = 64$)	**0.03**	0.04	0.08

as the following *per-point* function:

$$d_i(x \to y) := \sum_j \{\mathcal{F}_{ij}(x) \left(\mathcal{E}_j(y) - \mathcal{E}_j(x)\right)\} + x_i, \tag{14}$$

where $\mathcal{F}_{ij}(x) \in \mathbb{R}^3$ is the j-th offset at the i-th point, $\mathcal{E}_j(x) \in \mathbb{R}$ is j-th element of $\mathcal{E}(x)$, and x_i is the position of i-th point of x. We generalize this formulation by redefining $\mathcal{F}_{ij}(x)$ as a *function* describing an *arc-length* trajectory with the parameter $\left(\mathcal{E}_j(y) - \mathcal{E}_j(x)\right)$:

$$d_i(x \to y) := \sum_j \mathcal{F}_{ij}\left(x, \mathcal{E}_j(y) - \mathcal{E}_j(x)\right) + x_i. \tag{15}$$

For example, a *uniform circular* trajectory with a parameter t is formulated as follows:

$$\mathcal{F}_{ij}(x, t) := \left(\exp\left([t\mathcal{R}_{ij}(x)]_\times\right) - I_{3\times 3}\right)\left(x_i - C_{ij}(x)\right), \tag{16}$$

where $\mathcal{R}_{ij}(x) \in \mathbb{R}^3$ is the rotation vector describing the axis and angle, and $C_{ij}(x) \in \mathbb{R}^3$ is the rotation center; $[\cdot]_\times$ is the cross product matrix of the input vector. This formulation means that each element in the latent vector $\left(\mathcal{E}_j(y) - \mathcal{E}_j(x)\right)$ shared across all the points indicates a scale of the rotation angle or just a rotation angle if $\mathcal{R}_{ij}(x)$ is normalized to a unit vector.

We test this extension using the Shape2Motion dataset [Wang et al. 2019b] where the 3D models are annotated with movable parts and their motion parameters, e.g., parameters of rotation and/or

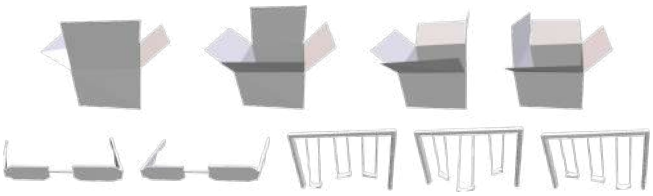


Fig. 14. The learned dictionaries of *rotational* motions using Shape2Motion dataset [Wang et al. 2019b]. Each element in the dictionaries describes the motion of each *independent* folding part, such as the flaps of the carton. See supplementary video for animation.

translation. We picked three models in different categories, carton, eyeglasses, and swing, which include rotations and generated $1k$ shape variations (128 out of them is the test set) by uniformly sampling rotation angles. Table 4 shows the comparison of fitting errors between the cases of learning linear and circular trajectories (the first two rows). The dimension of the latent space k is set to the degree of freedom of each model. The case learning circular trajectories gives a smaller fitting error. If we set a large number for the dimension of the latent space k (the last two rows), however, the network learning linear trajectories can also provide a very small fitting error by encoding all the rotational motions in the high-dimensional latent space. Note that Neural Cages [Yifan et al. 2020] which also only learns linear deformation offsets fails in this dataset (the fifth row).

We also observe that our network can discover independent rotation motions in the learned dictionaries, particularly when the rotation vector $\mathcal{R}_{ij}(x)$ is normalized to a unit vector; this normalization regularizes each rotation element to represent a *same-angle* rotation. Figure 14 visualizes the learned rotation dictionaries where each folding part is identified at each element. An animation can also be seen in the supplementary video.

Note that, among the affine properties in Section 3.1, the transitivity property is not guaranteed when learning the non-linear deformations; i.e., the latent vector may not be uniquely determined given a specific deformation. However, the property can be practically enforced when leveraging regularizations making each element to be unique.

5 CONCLUSION AND FUTURE WORK

We have proposed DeformSyncNet, a neural-network-based framework learning a synchronized linear deformation space for each shape. The synchronization is achieved without supervised correspondences but by connecting each shape-specific deformation space with an idealized canonical latent space, where all possible deformations are encoded. From this latent space, an encoded deformation is realized directly on each shape through per-point offsets via a shape-specific action decoding. As applications, our framework demonstrates (i) deformation projection, snapping an edited shape by the user to plausible shape space, and (ii) deformation transfer, adopting the modification performed on one shape to other shapes in the same category.

Our framework has several limitations. While we leverage the deformation handle information during the network training via projection, it is only exploited in the loss function but not fed as input to the network. Since most of the deformation handles are associated with not the whole but a part of the shape, part-level features related to them can provide additional information for the deformation space of the shape. Also, we take point clouds as input and use Chamfer distance as the only supervision. A more advanced backbone architecture and regularization losses for handling meshes can help learn more plausible deformations [Gao et al. 2018; Wang et al. 2019a]. We also introduced the extension of our framework to non-linear deformation in Section 4.6, but enforcing the affine properties remains to be explored.

Furthermore, the variations of *articulated* shapes may include *hierarchical* structure, such that the variation of a smaller part is factorized from the variation of a larger part. Such a structure might be better understood by finding elements in the deformation dictionary not in parallel, but sequentially. Finally, analogies, i.e., a transfer of the difference between two shapes to a third, can be extended to cross-domain cases, such as 3D shapes from/to images [Mo et al. 2019b] or natural language [Achlioptas et al. 2019].

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their comments and suggestions. N. J. Mitra acknowledges the support of ERC PoC Grant, Google Faculty Award, Royal Society Advanced Newton Fellowship, and gifts from Adobe. L. J. Guibas acknowledges the support of a Vannevar Bush Faculty Fellowship, a Samsung GRO grant, a Google Daydream Research Award, and gifts from the Adobe, Autodesk, and Snap corporations.

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2018. Learning Representations and Generative Models for 3D Point Clouds. In *ICML*.
Panos Achlioptas, Judy Fan, X.D. Robert Hawkins, D. Noah Goodman, and J. Leonidas Guibas. 2019. ShapeGlot: Learning Language for Shape Differentiation. In *ICCV*.
Ilya Baran, Daniel Vlasic, Eitan Grinspun, and Jovan Popovic. 2009. Semantic Deformation Transfer. In *ACM SIGGRAPH*.
Mirela Ben-Chen, Ofir Weber, and Craig Gotsman. 2009. Spatial Deformation Transfer. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.
Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. ShapeNet: An Information-Rich 3D Model Repository. arXiv:1512.03012
Lu Chen, Jin Huang, Hanqiu Sun, and Hujun Bao. 2010. Cage-based deformation transfer. *Computer & Graphics* (2010).
Dawson-Haggerty et al. [n.d.]. *trimesh*. https://trimsh.org/
Chris Ding, Ding Zhou, Xiaofeng He, and Hongyuan Zha. 2006. R1-PCA: Rotational Invariant L1-Norm Principal Component Analysis for Robust Subspace Factorization. In *ICML*.
Haoqiang Fan, Hao Su, and Leonidas Guibas. 2017. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *CVPR*.
Noa Fish, Melinos Averkiou, Oliver van Kaick, Olga Sorkine-Hornung, Daniel Cohen-Or, and Niloy J. Mitra. 2014. Meta-representation of Shape Families. In *ACM SIGGRAPH*.
Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. 2009. iWIRES: an analyze-and-edit approach to shape manipulation. In *ACM SIGGRAPH*.
Jean Gallier. 2011. *Geometric Methods and Applications*. Springer.
Lin Gao, Jie Yang, Yi-Ling Qiao, Yu-Kun Lai, Paul L. Rosin, Weiwei Xu, and Shihong Xia. 2018. Automatic Unpaired Shape Deformation Transfer. In *ACM SIGGRAPH Asia*.
Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman, and Thomas Funkhouser. 2019. Learning Shape Templates with Structured Implicit Functions. In *ICCV*.
Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. 2018. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *CVPR*.

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. 2019. Deep Self-Supervised Cycle-Consistent Deformation for Few-Shot Shape Segmentation. In *Eurographics Symposium on Geometry Processing*.

Rana Hanocka, Noa Fish, Zhenhua Wang, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2018. ALIGNet: Partial-Shape Agnostic Alignment via Unsupervised Learning. *ACM Transactions on Graphics* (2018).

Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. 2001. Image Analogies. In *ACM SIGGRAPH Asia*.

Haibin Huang, Evangelos Kalogerakis, Siddhartha Chaudhuri, Duygu Ceylan, Vladimir G. Kim, and Ersin Yumer. 2017. Learning Local Shape Descriptors from Part Correspondences with Multiview Convolutional Networks. *ACM Transactions on Graphics* (2017).

Ruqi Huang, Panos Achlioptas, Leonidas Guibas, and Maks Ovsjanikov. 2019a. Limit Shapes–A Tool for Understanding Shape Differences and Variability in 3D Model Collections. In *Eurographics Symposium on Geometry Processing*.

Ruqi Huang, Marie-Julie Rakotosaona, Panos Achlioptas, Leonidas J. Guibas, and Maks Ovsjanikov. 2019b. OperatorNet: Recovering 3D Shapes From Difference Operators. In *ICCV*.

Takeo Igarashi, Tomer Moscovich, and John F. Hughes. 2005. As-Rigid-as-Possible Shape Manipulation. In *ACM SIGGRAPH*.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. In *CVPR*.

Dominic Jack, Jhony K. Pontes, Sridha Sridharan, Clinton Fookes, Sareh Shirazi, Frederic Maire, and Anders Eriksson. 2018. Learning Free-Form Deformations for 3D Object Reconstruction. In *ICCV*.

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

Andrey Kurenkov, Jingwei Ji, Animesh Garg, Viraj Mehta, JunYoung Gwak, Christopher Bongsoo Choy, and Silvio Savarese. 2018. DeformNet: Free-Form Deformation Network for 3D Shape Reconstruction from a Single Image. In *WACV*.

Hao Li, Robert W. Sumner, and Mark Pauly. 2008. Global Correspondence Optimization for Non-Rigid Registration of Depth Scans. In *Eurographics Symposium on Geometry Processing*.

Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas Guibas. 2019. Supervised Fitting of Geometric Primitives to 3D Point Clouds. In *CVPR*.

Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, and David Levin. 2005. Linear Rotation-Invariant Coordinates for Meshes. In *ACM SIGGRAPH*.

Jerry Liu, Fisher Yu, and Thomas Funkhouser. 2017. Interactive 3D Modeling with a Generative Adversarial Network. arXiv:1706.05170

Chongyang Ma, Haibin Huang, Alla Sheffer, Evangelos Kalogerakis, and Rui Wang. 2009. Analogy-driven 3D style transfer. In *Eurographics*.

Q. McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* (1947).

Eloi Mehr, Ariane Jourdan, Nicolas Thome, Matthieu Cord, and Vincent Guitteny. 2019. DiscoNet: Shapes Learning on Disconnected Manifolds for 3D Editing. In *ICCV*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *NeurIPS*.

Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas Guibas. 2019a. StructEdit: Learning Structural Shape Variations. arXiv:1911.11098

Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy J. Mitra, and Leonidas Guibas. 2019b. StructureNet: Hierarchical Graph Networks for 3D Shape Generation. In *ACM SIGGRAPH Asia*.

Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. 2019c. PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding. In *CVPR*.

Feiping Nie, Heng Huang, Xiao Cai, and Chris Ding. 2010. Efficient and Robust Feature Selection via Joint $l2$, 1-Norms Minimization. In *NeurIPS*.

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

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*.

Raif M. Rustamov, Maks Ovsjanikov, Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Leonidas Guibas. 2013. Map-Based Exploration of Intrinsic Shape Differences and Variability. In *ACM SIGGRAPH*.

Adriana Schulz, Ariel Shamir, Ilya Baran, David I.W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. 2017. Retrieval on Parametric Shape Collections. In *ACM SIGGRAPH*.

Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. 2004. Laplacian Surface Editing. In *Eurographics Symposium on Geometry Processing*.

Stratasys. [n.d.]. GrabCAD Community. https://grabcad.com/library

Robert W. Sumner and Jovan Popovic. 2004. Deformation Transfer for Triangle Meshes. In *ACM SIGGRAPH*.

Minhyuk Sung, Hao Su, Vladimir G. Kim, Siddhartha Chaudhuri, and Leonidas Guibas. 2017. ComplementMe: Weakly-supervised Component Suggestions for 3D Modeling. In *ACM SIGGRAPH Asia*.

Minhyuk Sung, Hao Su, Ronald Yu, and Leonidas Guibas. 2018. Deep Functional Dictionaries: Learning Consistent Semantic Structures on 3D Models from Functions. In *NeurIPS*.

A.R. Tarrida. 2011. *Affine Maps, Euclidean Motions and Quadrics*. Springer.

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. 2019. Learning to Infer and Execute 3D Shape Programs. In *ICLR*.

Trimble. [n.d.]. 3D Warehouse. https://3dwarehouse.sketchup.com/

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

TurboSquid. [n.d.]. TurboSquid. https://www.turbosquid.com/

Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. 2018. Neural Kinematic Networks for Unsupervised Motion Retargetting. In *CVPR*.

Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 2019a. 3DN: 3D Deformation Network. In *CVPR*.

Xiaogang Wang, Bin Zhou, Yahao Shi, Xiaowu Chen, Qinping Zhao, and Kai Xu. 2019b. Shape2Motion: Joint Analysis of Motion Parts and Attributes from 3D Shapes. In *CVPR*.

Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhi-Quan Cheng, and Y. Xiong. 2011. Symmetry Hierarchy of Man-Made Objects. In *Eurographics*.

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

Shihong Xia, Congyi Wang, Jinxiang Chai, and Jessica Hodgins. 2015. Realtime style transfer for unlabeled heterogeneous human motion. *ACM Transactions on Graphics* (2015).

Kai Xu, Honghua Li, Hao Zhang, Daniel Cohen-Or, Yueshan Xiong, and Zhi-Quan Cheng. 2010. Style-Content Separation by Anisotropic Part Scales. In *ACM SIGGRAPH Asia*.

Weiwei Xu, Jun Wang, KangKang Yin, Kun Zhou, Michiel van de Panne, Falai Chen, and Baining Guo. 2009. Joint-aware Manipulation of Deformable Models. In *ACM SIGGRAPH*.

Jie Yang, Lin Gao, Yu-Kun Lai, Paul L. Rosin, and Shihong Xia. 2018. Biharmonic deformation transfer with automatic key point selection. *Graphical Models* (2018).

Li Yi, Leonidas Guibas, Aaron Hertzmann, Vladimir G. Kim, Hao Su, and Ersin Yumer. 2017. Learning Hierarchical Shape Segmentation and Labeling from Online Repositories. In *ACM SIGGRAPH*.

Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. 2016. A Scalable Active Framework for Region Annotation in 3D Shape Collections. In *ACM SIGGRAPH Asia*.

Wang Yifan, Noam Aigerman, Vladimir Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. 2020. Neural Cages for Detail-Preserving 3D Deformations. arXiv:1912.06395

Kangxue Yin, Zhiqin Chen, Hui Huang, Daniel Cohen-Or, and Hao Zhang. 2019. LOGAN: Unpaired Shape Transform in Latent Overcomplete Space. In *ACM SIGGRAPH Asia*.

Ersin Yumer and Levent Burak Kara. 2014. Co-Constrained Handles for Deformation in Shape Collections. In *ACM SIGGRAPH Asia*.

Ersin Yumer and Niloy J. Mitra. 2016. Learning Semantic Deformation Flows with 3D Convolutional Networks. In *ECCV*.

Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 2019. 3D Point Capsule Networks. In *CVPR*.

Youyi Zheng, Daniel Cohen-Or, Melinos Averkiou, and Niloy J. Mitra. 2014. Recurring Part Arrangements in Shape Collections. In *Eurographics*.

Youyi Zheng, Hongbo Fu, Daniel Cohen-Or, Oscar Kin-Chung Au, and Chiew-Lan Tai. 2011. Component-wise Controllers for Structure-Preserving Shape Manipulation. In *Eurographics*.

Kun Zhou, Weiwei Xu, Yiying Tong, and Mathieu Desbrun. 2010. Deformation Transfer to Multi-Component Objects. In *Eurographics*.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *ICCV*.