Representing Shape Collections with Alignment-Aware Linear Models

Romain Loiseau, Tom Monnier, Mathieu Aubry, Loïc Landrieu

Representing Shape Collections With Alignment-Aware Linear Models

Romain Loiseau[1, 2] romain.loiseau@enpc.fr
Tom Monnier[1] tom.monnier@enpc.fr
Mathieu Aubry[1] mathieu.aubry@enpc.fr
Loïc Landrieu[2] loic.landrieu@ign.fr

[1]LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, France
[2]LASTIG, Univ. Gustave Eiffel, ENSG, IGN, F-94160 Saint-Mande, France

Abstract

In this paper, we revisit the classical representation of 3D point clouds as linear shape models. Our key insight is to leverage deep learning to represent a collection of shapes as affine transformations of low-dimensional linear shape models. Each linear model is characterized by a shape prototype, a low-dimensional shape basis and two neural networks. The networks take as input a point cloud and predict the coordinates of a shape in the linear basis and the affine transformation which best approximate the input. Both linear models and neural networks are learned end-to-end using a single reconstruction loss. The main advantage of our approach is that, in contrast to many recent deep approaches which learn feature-based complex shape representations, our model is explicit and every operation occurs in 3D space. As a result, our linear shape models can be easily visualized and annotated, and failure cases can be visually understood. While our main goal is to introduce a compact and interpretable representation of shape collections, we show it leads to state of the art results for few-shot segmentation. Code and data are available at: https://romainloiseau.github.io/deep-linear-shapes

1. Introduction

Picture a company acquiring thousands of 3D scans of technical components; how to leverage, organize, or even simply visualize these 3D models? Deep shape analysis techniques have flourished over the last years [26] but, even when motivated by geometric intuitions, these methods and their results remain hard to interpret and interact with. Moreover, they are often limited by the availability of domain and application-specific annotations. Instead of pushing for even more complex architectures, we operate directly in 3D space and revisit the simple linear shape model with a deep learning perspective. As illustrated in Figure 1, we model a collection of 3D shapes with a set of low-dimensional linear shape models. Each linear model is defined by a prototype 3D point cloud and a set of vector basis that can be interpreted as fields of translation vectors for each point of the prototype. By adding a linear combination of this basis vector to the prototype, one can continuously move in a low-dimensional subspace of the shape space.

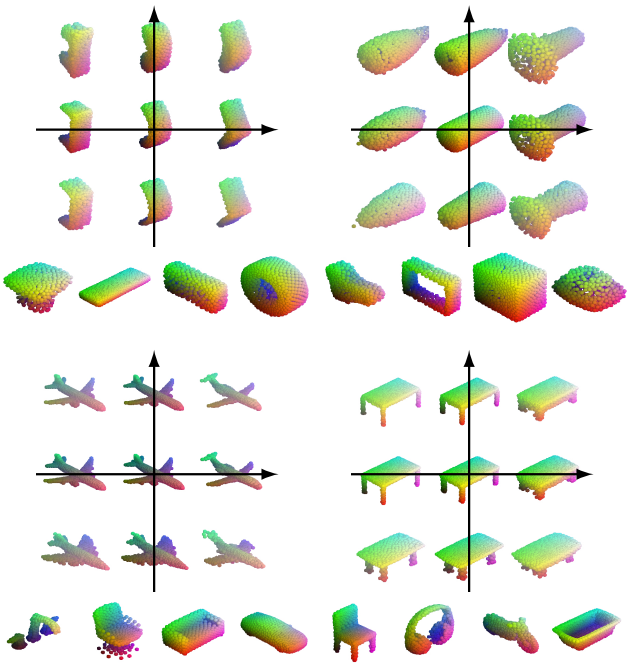


Figure 1: **Discovered linear models.** Our approach discovers without supervision linear shape models from large collections of shapes. We show two examples of two-dimensional families and eight additional prototypes discovered for ABC [34] (top) and ShapeNet [8] (bottom).

We face three key challenges when trying to represent 3D shape collections with such linear models. First, comparing shapes using Chamfer or Earth Mover distances has strong limitations for shape analysis, since they are impacted by simple rigid or affine shape transformations, which cannot be easily represented by linear models. Transformation-invariant distances such as the Gromov-

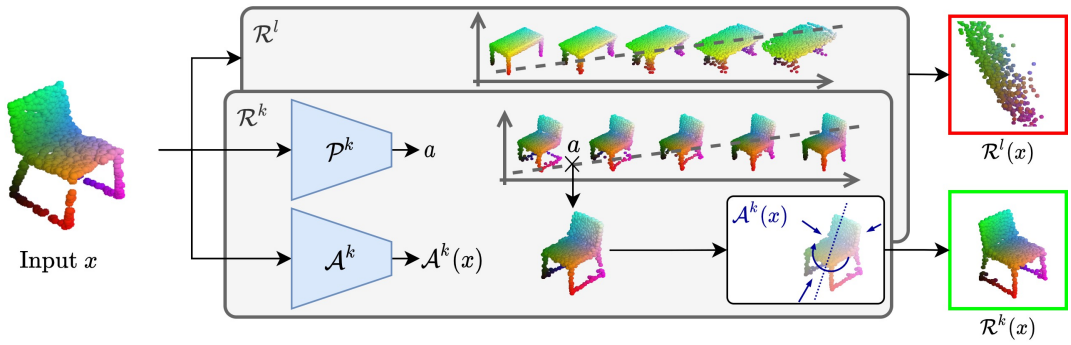


Figure 2: **Method overview.** Given an input point cloud x, we predict for each shape model $\mathcal{R}^k$ the element that best reconstructs the input: the projection network $\mathcal{P}^k$ outputs the coordinates a of a shape in a linear family, and the alignment network $\mathcal{A}^k$ predicts the parameters of an affine transformation $\mathcal{A}^k(x)$ which is applied to the selected shape. The input point cloud is then assigned to the shape model that best reconstructs it, here highlighted in green.

Hausdorff Distance [39] can be defined to overcome this problem, but they are typically very hard to work with. Second, finding the coordinates in the shape basis that best reconstruct a sample according to a given similarity measure is a difficult non-convex problem. Third, operations as simple as averaging are non-trivial for point clouds, and dimensionality reduction techniques such as Principal Component Analysis [59] do not directly apply.

In this work, we present an unsupervised approach that learns small sets of linear shape models to explain large collections of point clouds. We propose to solve this task with a clustering formulation directly in 3D space, where clusters are associated to linear shape families, each modeled as a reference prototype point cloud and a set of basis vectors that can be interpreted as displacement fields. We explore two ways of defining such displacement fields - either using a pointwise parametrization or an implicit one based on parametric differentiable functions of 3D space - and analyze their benefits. In addition, to predict the coordinates of a point cloud in the linear basis and account for shape transformations, we extend the idea from the work of Monnier *et al*. [41] on transformation-invariant image clustering to the setting of 3D shape alignment. By jointly learning linear shape families and parametric functions predicting both shape basis coordinates and alignment parameters, our approach is able to discover rich and meaningful shape models from a collection of point clouds without any supervision.

We believe that our method has strong advantages compared to recent unsupervised 3D shape analysis approaches. First, by manipulating objects directly in 3D space, our results are easy to interpret and visualize. Second, our linear shape models can serve as a mean to explore large collections of raw 3D point clouds. Finally, we show that despite its simplicity, our model yields competitive results for shape clustering and state-of-the-art results for few-shot shape segmentation.

Our contributions can be summarized as follows:

- we present an unsupervised method to represent large point cloud collections with a small set of linear families of shapes;
- we extend the DTI clustering framework to learn linear shape models by introducing projection networks;
- we analyse two different representations for linear shape modeling and show the benefits of representing them with continuous functions of space rather than pointwise displacements;
- we demonstrate qualitative results for visualizing the large unstructured ABC dataset [34] and obtain state-of-the-art few-shot segmentation performances on the standard ShapeNetPart dataset [8].

2. Related work

Point cloud distances and alignment. Classical similarity measures between point clouds include the Chamfer and Earth mover distances. These distances are however not invariant to rigid transformations. The Gromov-Hausdorff distance [39] provides a nice framework to define a transformation-invariance distance, but is difficult to use in practice. Transformation-Invariant distances were also defined for images and used for clustering by Frey and Jojic [18, 20, 21, 19]. The Transformed Component Analysis (TCA) approach [19] would be particularly relevant, although operating with a discrete set of transformations may be too limiting for aligning 3D shapes. Applying them to 3D point clouds would require to align them. This is classicaly done using the the Iterative Closest Point (ICP) algorithm [3]. Instead, we take inspiration from the Deep Transformation-Invariant framework [41] and use neural networks to predict alignment and define similarity.

Linear Shape Modeling. The idea of representing a collection of images using a low-dimensional image basis was first developed for face images [49]. Popularized by the classical eigenfaces model [55], linear models have since been applied to diverse computer vision problems and data. A linear 3D face model was designed in [4] and applied

to new view synthesis. [12] demonstrated applications to medical data. Non-rigid surface-from-motion can also benefit from linear shape basis decomposition to recover 3D shapes [6, 54, 13]. An application of linear modeling to unsupervised 3D keypoint discovery was recently demonstrated in [17]. These linear models are typically learned from a set of examples by principal component analysis, factorization techniques [53], or defined manually [58]. Additionally, some recent works propose to analyse shape collections through implicit representations [32, 64, 14]. In contrast, we propose a learning-based approach to model arbitrary unregistered shapes from large collections of examples, and we use several low-dimensional linear families.

Deep Learning for 3D Analysis. Neural networks successfully tackled numerous challenges in 3D shape analysis. The main approaches can be broadly classified depending on the representation of 3D data they leverage, voxels [11], point clouds [44], graphs [35], surfaces [7, 24], structured models [37, 23], or more recently implicit volumetric models [43, 10, 40]. They have been successfully applied to tasks as diverse as classification [38], segmentation [52], shape generation [24, 51], matching [42], denoising [29] and compression [30]. In this paper, we focus on 3D point clouds but our approach is general and could be extended to other representations.

The key idea to use Multi-Layer Perceptron (MLP) on point clouds was initially proposed for shape classification and segmentation by Qi *et al*. [44] with an architecture called PointNet, and for 3D point cloud generation in [16]. To answer the difficulty of annotating 3D data, new approaches are able to perform self-supervised and unsupervised feature learning [62, 46, 28] and low-shot segmentation [25, 57, 22]. Especially related to ours is the recent 3D capsule approach [63] that explicitly tries to design a shape representation invariant to 3D transformations and can be applied to many tasks. However, the latent representations learned with 3D capsules and the associated generation process are difficult to interpret. Also related to ours is the approach of Deprelle *et al*. [15] which proposes a 3D shape reconstruction model obtained by combining and transforming learned elementary structures. This method shares similarities with ours as it allows us to learn prototypes of parts of shapes. However, it focuses on reconstruction accuracy, uses a single prototype per part and mainly follows the black-box AtlasNet [24] deformation framework.

3. Modeling Shape Collections

Our goal is to explain a collection of N point clouds $x_1, \dots, x_N$ with a small set of K shape models. For simplicity, we assume that all point clouds have the same number M of points. We propose to solve this task with a clustering formulation described in Sec. 3.1. We then describe how we model alignment (Sec. 3.2) and linear shape families (Sec. 3.3) resulting in our final modeling. Finally, we present how we parametrize our linear shape models and give some training details (Sec. 3.4).

3.1. Method overview

We build a set of K shape models $\mathcal{R} = \{\mathcal{R}^1, \dots, \mathcal{R}^K\}$. Each $\mathcal{R}^k$ maps a sample point cloud x to a reconstructed point cloud $\mathcal{R}^k(x)$ which can be interpreted as the approximation of x by the corresponding model. We denote by d a distance between point clouds which measures the quality of a reconstruction. We use the Chamfer distance in all our experiments. We learn the shape models $\mathcal{R}$ by minimizing the loss

$$\mathcal{L}(\mathcal{R}) = \sum_{x \in x_1, \dots, x_N} \min_{k=1}^{K} d\left(x, \mathcal{R}^k(x)\right) \ , \tag{1}$$

which can be interpreted as a clustering objective defined as the sum of the reconstruction errors with optimal cluster assignment.

Prototype model. The simplest form of $\mathcal{R}^k$ is a constant function: $\mathcal{R}^k_{\text{proto}}(x) = c^k \in \mathbb{R}^{M \times 3}$ where each c^k can be seen as a prototype point cloud. Such prototype point clouds can be learned by minimizing $\mathcal{L}$ with batch Stochastic Gradient Descent (SGD). This amounts to performing stochastic K-means [5] for 3D point clouds. Note that this is a weak reconstruction model, however, the goal of this paper is not to learn the most faithful reconstruction, but rather to summarize the collection.

3.2. Alignment-Aware Model

A clear limitation of the prototype model is that it does not take into account simple geometric transformations of the point clouds, such as rigid transformations. For example, point clouds can be close to a model's prototype c^k according to the distance d, while a rotated or translated version of the same point cloud is far away. We would like both point clouds to be associated with the same shape model. To address this issue, we incorporate in each model $\mathcal{R}^k$ an affine alignment component. In practice, we use neural networks $\mathcal{A}^k$ - which we refer to as *alignment networks* - whose goal is to predict an affine transformation $\mathcal{A}^k(x)$ aligning the prototype c^k with a target point cloud x. This results in an alignment-aware model $\mathcal{R}^k_{\text{align}}$ defined by:

$$\mathcal{R}^k_{\text{align}}(x) = \mathcal{A}^k(x)\left[c^k\right] \ , \tag{2}$$

where the affine transformation $\mathcal{A}^k(x)$ is applied to each point of the prototype point cloud c^k. The alignment networks $\mathcal{A}^1, \dots, \mathcal{A}^K$ can be trained alongside the prototypes $c^1, \dots, c^K$ by minimizing Equation 1. This model can be seen as an extension of the recent Deep Transformation-Invariant (DTI) clustering framework [41] developed for

images to point clouds. Indeed, our alignment models can be understood as defining an approximation of an affine-invariant version of the distance d according to which the clustering is performed. In this paper, we rather view these networks as an integral part of the shape models.

Note that different transformation models could be considered. In our experimental analysis, we study variations of the model using weaker transformations, such as rigid transformations or scaling, and show the benefits of the affine model. On the contrary, one could consider complex deformations parametrized by deep networks, such as the ones used in FoldingNet [62] or AtlasNet [24], which would surely lead to higher accuracy reconstructions. However, such transformations completely change the geometry of a point cloud and are hard to interpret.

3.3. Linear Shape Modeling

Our goal in this section is to model changes in objects more subtle than those that can be modeled by affine transformations, such as the angle of the wings of an airplane, while maintaining the model intepretability. We propose to associate a linear shape family to each prototype point cloud.

Linear shape families. For each model k, we define a linear shape family as a pair formed by (i) a prototype point cloud c^k in $\mathbb{R}^{M\times 3}$ and (ii) a set v^k of D basis vectors $v^k = \{v_1^k, \dots, v_D^k\}$, where each $v_i^k \in \mathbb{R}^{M\times 3}$ associates to each point of the prototype a 3D vector and can be interpreted as displacement fields. Each (c^k, v^k) defines a continuous collection of shapes covered by translating the points of c^k along the directions defined by v^k. Each element u of the linear family (c^k, v^k) is characterized by a vector a in $\mathbb{R}^D$ defining its coordinates in the linear shape family:

$$u = c^k + \sum_{i=1}^{D} a_i \, v_i^k \; . \tag{3}$$

The vector a can be interpreted as the set of amplitudes to apply to the displacement fields $\{v_1^k, \dots, v_D^k\}$. Note this formally describes an affine space but we follow the convention of previous works and refer to it as linear.[1] Also note that we do not explicitly enforce linear independence between basis vectors, but their high dimensionality ($M\times 3$) leads to such independence in practice.

Projection networks. If we had access to ordered point clouds, *i.e.* lists of M points in $\mathbb{R}^3$ where the i-th points are in correspondence, we would be able to use the L_2 distance to measure point clouds similarity. In this case, computing the coordinates of the element of the linear family closest to a target point cloud would simply amount to performing Euclidean projection. This is however not the case for unordered point clouds, for which the notion of distance is more complicated. For common point cloud similarity measures such as the Chamfer distance, finding the closest point cloud in a linear family is a difficult non-convex optimization problem. This task is made even harder by the fact that we use our alignment networks to transform the elements of the family before comparing them with the input cloud.

Therefore, we propose to leverage deep learning to estimate which element of a linear family is the closest to a target point cloud after alignment. More specifically, we associate to each linear family (c^k, v^k) a neural network $\mathcal{P}^k$ which aims at associating to a given input sample the coordinates of the element in the linear family minimizing the distance d. The output of the network $\mathcal{P}^k(x) \in \mathbb{R}^D$ is interpreted as the coordinates a of the point cloud defined in Equation 3. By analogy with the L_2 distance case, we refer to these networks as *projection networks*.

Full model. We define our final shape model $\mathcal{R}$ as a collection of models $\mathcal{R}^k_{\text{full}}$ each composed of a linear family (c^k, v^k), an alignment network $\mathcal{A}^k$ and a projection network $\mathcal{P}^k$. Given a target point cloud x, our model reconstructs it by (i) selecting an element of the linear family (c^k, v^k) through the projection network $\mathcal{P}^k$, and (ii) aligning it with the target using the transformation predicted by the alignment network $\mathcal{A}^k$. More formally, we write each shape model as:

$$\mathcal{R}^k_{\text{full}}(x) = \mathcal{A}^k(x) \left[c^k + \sum_{i=1}^{D} \left[\mathcal{P}^k(x)\right]_i \, v_i^k \right], \tag{4}$$

where $\left[\mathcal{P}^k(x)\right]_i$ refers to the i-th component of $\mathcal{P}^k(x)$ and the affine transformation $\mathcal{A}^k(x)$ is applied to each point of the point cloud independently. Again, we optimize jointly the $c^k, v^k, \mathcal{A}^k$ and $\mathcal{P}^k$ to minimize the reconstruction loss defined in Equation (1).

3.4. Parameterization and training details

We first describe how we parametrize the linear families, then provide implementation details such as networks architecture and our curriculum learning strategy.

Linear family parametrization. While the prototypical point cloud c^k is modeled directly using learnable parameters in $\mathbb{R}^{M\times 3}$, the basis vectors v_i^k can be parametrized in two different ways:

- *Pointwise parametrization:* for each model k, we represent v^k as vectors of learnable parameters of size $D \times (M \times 3)$ that can directly be interpreted as D pointwise displacement vectors of the prototype c^k.
- *Implicit parametrization:* we use implicit parametric functions of the 3D space modeled as neural networks to

[1] An analogy can be made with the face reconstruction model EigenFace [56]: c is equivalent to the *mean face*, and v to the eigenfaces.

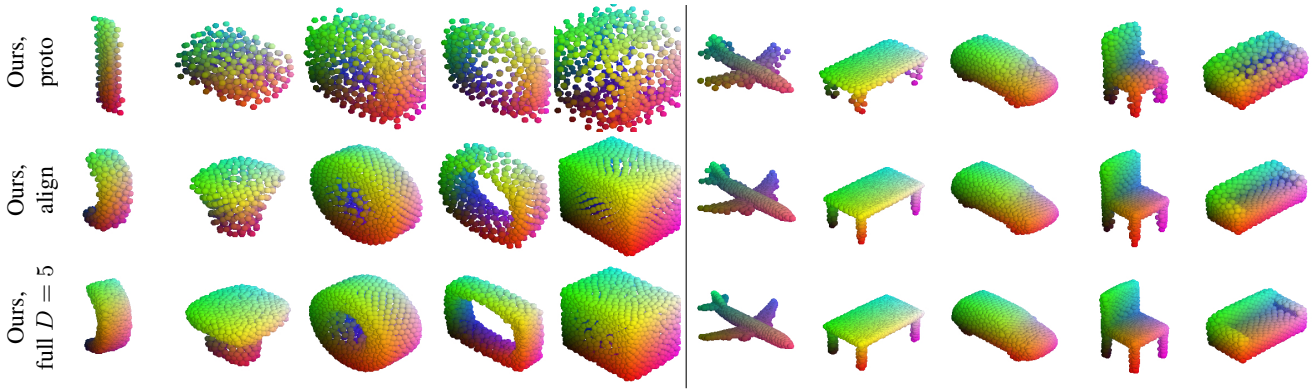


Figure 3: **Learned prototypes and comparisons.** We compare the prototypes from our different shape modeling discovered in ABC [34] (left, 5 shape models out of 10) and ShapeNetCore [8] (right, 5 shape models out of 55). Note how sharp the prototypes become when the shape modeling complexity increases, respectively with alignment-awareness and 5-dimensional linear families.

define the displacement fields. More precisely, for each model k and basis dimension i, we learn a parametric function $\mathcal{V}_i^k : \mathbb{R}^3 \mapsto \mathbb{R}^3$ mapping any point in the 3D space to a displacement direction. Writting $[c^k]_p$ the 3D coordinates of the p-th point of prototype c^k, the 3D coordinates $[v_i^k]_p$ of the i-th basis vector associated to the point p are $[v_i^k]_p = \mathcal{V}_i^k([c^k]_p)$.

Intuitively, the pointwise parametrization seems better suited for modeling complex and discontinuous transformations within a shape family such as the appearance/disappearance of object parts. On the contrary, the transformations learned with implicit parametrizations are derived from continuous functions of the 3D space and can be expected to be more regular.

We compare both settings in Section 4.2, and show that pointwise parametrizations provide better shape reconstructions, but that implicit parametrization yields more interpretable transformations preserving semantic correspondences. Thus, unless specified otherwise, we use the implicit parametrization of the basis in the rest of the paper.

Architecture. For each model k, the alignment network $\mathcal{A}^k$ takes as input a point cloud and outputs a vector in $\mathbb{R}^{12}$ corresponding to a linear 3D operator and a translation vector applied to each point of the model. The projection network $\mathcal{P}^k$ also takes a point cloud as input and outputs a vector in $\mathbb{R}^D$ that is interpreted as coordinates in the linear family (c^k, v^k). These networks share a common PointNet [44] backbone encoder which acts as a global feature extractor. This shared encoder starts with a sequence of three linear layers with batch normalization [31] and ReLU activation acting on points independently and sequentially generating representations of size 64, 128 and 1024, and ends with a max-pooling over all points. This encoder is then followed by $2 \times K$ Multi-Layer Perceptrons (MLPs) corresponding to each prediction task (alignment or projection) and each shape model. Each MLP has one hidden layer of size 128. The implicit parametrizations $\mathcal{V}_i^k : \mathbb{R}^3 \mapsto \mathbb{R}^3$ are MLPs with 2 hidden layers of size 128.

Curriculum learning. Inspired by the curriculum learning of [41], we propose to learn our models by gradually increasing the models complexity. We first learn raw prototype models ($\mathcal{R}^k_{\text{proto}}$), an optimization which corresponds to performing a gradient-based K-means algorithm in the 3D space. Second, we augment each model with alignment awareness ($\mathcal{R}^k_{\text{align}}$). Finally, we gradually increase the linear families dimension up to the desired one, resulting in our final shape model ($\mathcal{R}^k_{\text{full}}$).

Implementation details. Our implementation - which will be released upon publication - uses PyTorch, Torch-Points3D [9], and an efficient CUDA implementation of the Chamfer distance which significantly speeds up training. With $K = 10$ prototypes and $D = 5$, our model has $4.6M$ parameters. For comparison, the reconstruction models proposed by Wang *et al.* [57] and Groueix *et al.* [25] have respectively $2.6M$ and $10.0M$ parameters. See our supplementary material for additional details.

4. Experiments

In this section, we analyze the benefits of our method to represent shape collections, first qualitatively (Section 4.1) then quantitatively (Section 4.2). Finally, we demonstrate that it leads to results on par with state of the art for few-shot and low shot shape segmentation (Section 4.3).

4.1. Qualitative results

We demonstrate the potential of our method for exploring large shape collections.

Datasets. The ShapeNet dataset [8] is a large collection of over $50K$ 3D models organized along 55 common object categories such as chairs, airplanes, or cars. The ABC dataset [34] is a very large collection of Computer-Aided

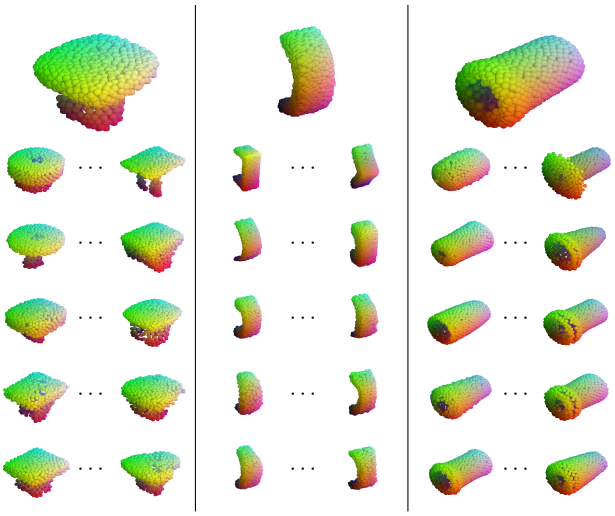


Figure 4: **Basis vectors.** Examples of linear shape models obtained after training on ABC with $D = 5$. The prototype is represented at the top and each row corresponds to one of the dimension of the linear families. The models' basis vectors correspond to complex morphological changes.

Design (CAD) models of diverse mechanical object parts, such as screws or pipes. We used the first six chunks from this dataset and considered the connected components of each mesh as separate objects ($\approx 70K$ shapes). We apply our approach using 55 shape models for ShapeNet and 10 for ABC. For both datasets, we uniformly sample points on the objects' surface to obtain point clouds.

Prototypes. We present in Figure 3 examples of prototypes learned when successively adding different components of our method. The first line, denoted "Ours, proto", represents the linear families' prototypes learned during the first stage of our training ($\mathcal{R}_{\text{proto}}$). The second line, denoted "Ours, align", displays the learned prototypes after the second stage of our training ($\mathcal{R}_{\text{align}}$), during which affine alignment networks are learned jointly with their model's prototype. Finally, the third line denoted "Ours, full $D = 5$" illustrates the prototypes learned at the last stage of training ($\mathcal{R}_{\text{full}}$) alongside linear shape families of dimension 5 and their associated projection networks. We show the center of each linear shape model, defined by taking the median amplitude a_i in each dimension i when considering all point clouds associated with the model, *i.e.* point clouds for which this model outputs the best reconstruction.

The prototypes learned with the Chamfer distance (first line) appear noisy, hinting that they are not well aligned with the shapes they try to approximate. When adding alignment networks, we obtain the prototypes of the second line, which are much cleaner, outlining the interest of using a transformation-invariant model, as well as the fact that our approach can effectively learn such a model. Finally, the prototypes obtained with our full method are even sharper and smoother, indicating that linear shape families can better model the associated point clouds.

Our results on the ABC dataset outline the capacity of our full model to differentiate between different types of shapes, as prototypes correspond to different object types. By looking at the prototypes, one can grasp at a glance the diversity of shapes contained in this large-scale dataset.

Linear shape models. In Figure 4, we illustrate some of the linear shape models learned on ABC (more results for both datasets are in the supplementary material). The top row shows the center of the linear shape models, and the subsequent lines illustrate the five basis vectors. For each model and each basis vector, we represent two shapes whose amplitudes for the considered dimensions are set to the 5-th and 95-th percentile values of all point clouds associated to the model, while the other amplitudes remain at the median value. Again, we can see how the different dimensions give insights on the diversity of shapes within the dataset.

Reconstructions. In Figure 5, we show examples of reconstructed shapes from ShapeNet (airplanes, cars, and chairs) for four different linear shape families. As expected, the model is able to reconstruct objects precisely while remaining visually interpretable. Again, more examples can be seen in supplementary material.

4.2. Quantitative Analysis for Clustering and Reconstruction

The qualitative results described in the previous section outline the potential of our approach for visualizing and analyzing large, unstructured, and diverse shape collections. We now provide a more quantitative analysis of these results on the standard ModelNet10 dataset [61].

Data and evaluation. ModelNet10 contains 3991 train and 909 test aligned 3D point clouds obtained from CAD models of 10 different classes. We use this dataset both in its original aligned version and also with added random rotations around the z-axis to evaluate the capacity of our method to represent unaligned data. Unless specified otherwise, the results are given for the original dataset. We trained the different variants of our method with 10 reconstruction models on train and test shapes of ModelNet10. We evaluate in Table 1 the clustering accuracy and reconstruction error measured by the Chamfer Distance. To measure the quality of the resulting clustering, we assign to each model the majority label of its associated point clouds from the train set. The accuracy of the classification is then defined by assigning to test shapes the label of the model giving the best reconstruction.

Alignment. We compute the performance of our models only defined by prototypes ("Ours proto"), and then train models with alignments of different complexities ("Ours,

Table 1: **Results on ModelNet10.** We present results with 10 linear shapes models, first for different restrictions of the alignment networks, then for different basis vector configurations. The steps in the curriculum training of our model are in **bold**. We report clustering accuracy in % ('Accuracy') and the Chamfer distance multiplied by 10^3 ('CD'), Results are averaged over five runs.

			Accuracy	CD
		Ours, proto	$\mathbf{63.9 \pm 1.5}$	$\mathbf{20.0 \pm 0.4}$
		... with supervision	79.0 ± 0.2	23.5 ± 0.0
Ours, align		Rigid transformation (6D)	64.6 ± 5.2	16.2 ± 0.1
		Trans. + Iso. Scaling (4D)	71.5 ± 4.1	15.0 ± 0.1
		Trans. + Aniso. Scaling (6D)	74.1 ± 3.0	10.4 ± 0.1
		Linear (9D)	71.85 ± 4.7	11.1 ± 0.1
		Affine (12D)	$\mathbf{75.9 \pm 3.0}$	$\mathbf{9.7 \pm 0.0}$
		... with supervision	88.9 ± 0.5	11.2 ± 0.0
Ours, full	$D=1$	Pointwise parametrization	74.3 ± 1.7	7.9 ± 0.0
		Implicit parametrization	$\mathbf{77.5 \pm 2.8}$	$\mathbf{8.1 \pm 0.0}$
		... with supervision	89.7 ± 0.6	9.5 ± 0.0
	$D=5$	Pointwise parametrization	75.1 ± 1.7	5.7 ± 0.0
		Implicit parametrization	$\mathbf{77.0 \pm 3.4}$	$\mathbf{5.9 \pm 0.0}$
		... with supervision	90.4 ± 1.0	7.8 ± 0.0
FoldingNet [62]			76.3 ± 7.5	$\mathbf{3.5 \pm 0.0}$

align"). We first evaluate a model whose alignment networks are restricted to a rigid transformation ("Rigid transformation (6D)"), with rotations parametrized with quaternions. We also evaluate models with a scaling and a translation ("Trans. + Iso. Scaling (4D)"), axis-aligned scalings and a translation ("Trans + Aniso. Scaling (6D)"), a linear transformation ("Linear (9D)"), and finally an affine transformation ("Affine (12D)"). We observe that using alignment networks allows significant clustering improvement in terms of accuracy and reconstruction quality. Moreover, restricting the output of the alignment networks leads to a lower performance: even for centered and rotation-aligned data such as ModelNet, allowing complex alignments benefits both clustering and reconstruction.

Linear families. We then evaluate models with affine alignment but different linear basis ("ours, full"). We compare the results between one-dimensional ($D = 1$) and five-dimensional ($D = 5$) linear families as well as between basis vectors learned in the pointwise and implicit parametrization (see Section 3.3). Increasing the dimension of the shape families improves the reconstruction error but slightly decreases the clustering accuracy with the implicit parametrization. This can be explained by the models becoming too expressive, resulting in point clouds from different classes being associated with the same model.

Table 2: **Non-aligned data.** Clustering Accuracy ('Accuracy', in %) and reconstruction error ('CD', Chamfer distance multiplied by 10^3) obtained with 10 linear shapes models on the rotated version of ModelNet10. Δ_{CD} is the difference of reconstruction error when training the same model on the aligned or unaligned datasets.

	Accuracy	Δ_{Accuracy}	CD	Δ_{CD}
Ours, proto	41.2 ± 3.4	-22.7	30.1 ± 0.1	-10.1
Ours, align	61.8 ± 3.3	-14.1	11.0 ± 0.1	-1.3
Ours, full $D = 1$	65.2 ± 6.7	-12.3	9.3 ± 0.0	-1.2
Ours, full $D = 5$	$\mathbf{68.8 \pm 7.9}$	$\mathbf{-8.2}$	$\mathbf{6.7 \pm 0.0}$	$\mathbf{-0.8}$

Baseline and supervised upper bound. As a baseline, we performed k-means clustering in feature space using the implementation of FoldingNet [62] proposed by [50]. The resulting accuracy is comparable to that of our best models'. However, FoldingNet relies on learning black-box deep deformations of a planar patch, and the resulting shape family and generation process are thus harder to interpret than ours.

We also trained our model in a supervized manner by associating a class to each model, and only training each model on point clouds from their class ("with supervision" lines, in light gray). As expected, this "oracle" setting performs better in terms of clustering accuracy, but with lower reconstruction quality. This can be explained by the presence of classes with high variability such as *chairs* which require several families to fully cover, and similar classes such as *desks* and *tables* which can be well reconstructed by a single family.

Non-aligned data. In Table 2, we report our approach's performance when trained on ModelNet10 with random rotations. We observe that adding alignment networks to the model results in significantly better metrics compared to simple prototypes. Our full models with alignment are able to reach reconstruction qualities almost comparable to the equivalent models trained on aligned shapes. Similarly, the drop in clustering performance is reduced when adding the alignment networks and linear shape families. This outlines the capacity of our models to handle raw unaligned data. We present in the appendix illustrations of the prototypes learned in this setting.

4.3. Application to few/low-shot Segmentation

Our linear shape models can perform semantic segmentation by transferring point labels from the model's prototype to the reconstructed point cloud. More precisely, given an input point cloud x, we identify the model k with the lowest reconstruction error. We then compute $\tilde{x} = \mathcal{R}^k_{\text{full}}(x)$, the point cloud reconstructed by this model. We transfer the point annotation from the prototype c^k to $\tilde{x}$. Finally, each point of x is assigned the label of the closest point of $\tilde{x}$. This strategy is especially meaningful in a few-shot setting, since only the prototypes need to be annotated.

Table 3: **10-shot segmentation.** We report pointwise IoU for 9 classes and the average IoU over all 16 classes of ShapeNet-Part. See text for details.

		airplane	bag	cap	car	chair	lamp	laptop	mug	table	avg
Shared encoder	Gadelha *et al.* 2020 [22]	—	—	—	—	—	—	—	—	—	74.1
	Ours, full $D = 5$ (random)	71.7	70.6	**84.0**	62.1	78.8	68.7	93.1	87.5	70.6	72.5
	Ours, full $D = 5$ (prototype)	**79.4**	**73.0**	81.8	**72.1**	**83.6**	**76.1**	**94.7**	**89.8**	**76.2**	**77.4**
One encoder per class	Wang *et al.* 2020 [57]	67.3	74.4	**86.3**	—	83.4	68.7	93.8	90.9	74.2	—
	Groueix *et al.* 2019 [25]	67.1	—	—	61.4	78.9	65.8	—	—	66.1	—
	Ours, full $D = 5$ (random)	72.2	66.0	75.5	63.0	79.1	68.9	93.1	84.2	69.4	—
	Ours, full $D = 5$ (prototype)	**80.0**	**79.7**	76.1	**72.0**	**83.6**	**77.1**	**94.9**	**91.1**	**75.9**	—

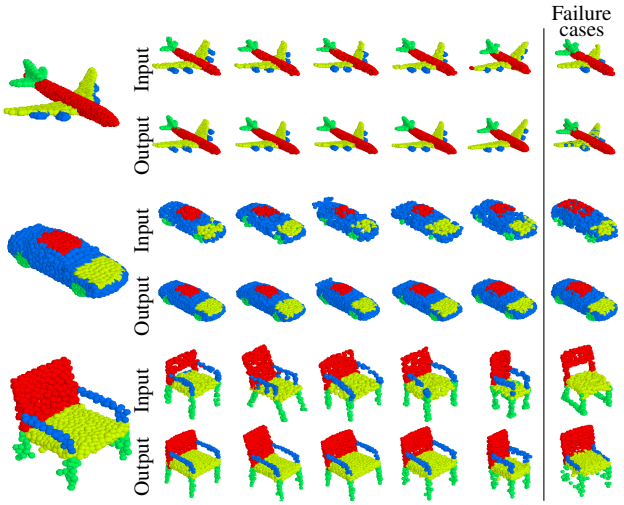


Figure 5: **Reconstruction results.** Examples of samples annotated pointwise with our semantic-segmentation method ($D = 5$). We visually selected failure cases where semantic regions were wrongly predicted. Prototypes are represented on the left column, the "input" lines display input samples with ground truth annotations and the "output" lines our reconstruction with pixel labels propagated from the prototype.

Few-shot Segmentation. In this setting, where we only use a few annotations for each class and train our model with only the reconstruction loss as described earlier, we consider two methods to annotate the prototypes:

- *Random*. We randomly pick one sample from the train set for each model and propagate its labels to their nearest points of the aligned prototype.
- *Prototype*. We align all samples from the train set for each model's prototype and label each point with majority voting. This second setting is meant to emulate the *manual* annotation of the 10 prototypes. While this is not directly comparable to other approaches, it outlines the crucial advantage given by our approach, which identifies a small set of prototype shapes that can be annotated instead of using random samples. Some prototypes annotated in this manner can be seen in Figure 5.

We use the densely annotated ShapeNetPart [47] to evaluate the segmentation performance of our few-shot segmentation scheme. We report in Table 3 the performance of our 10-shot segmentation scheme for nine classes of ShapeNet-Core, and the average performance over all 16 classes. As mentioned in Section 3.4, all the alignment and projection networks share a common PointNet [44] encoder which acts as a global feature extractor. To compare with previous works that use either a shared model or a different model per class, we present results using either a single encoder for all classes or one encoder per class. Using only 10 samples from the dataset to annotate our prototypes, we observe that the annotation from random samples performs on par or better than state-of-the-art approaches. Annotating prototypes (using all training samples) significantly outperforms all methods. This shows that our approach can be used to precisely and densely annotate large shape datasets with minimal human intervention. We also observe some failure cases shown in the last column of Figure 5: since our model can only move points and not add or subtract them, shapes with optional parts, such as the arms of chairs, may be mislabeled.

Low-shot Segmentation. Our model can also be trained in a low-shot setting, yielding a slight improvement of +1 and +2 mIoU compared to 3D capsules [63] when trained with only 1% or 5% of annotated shapes. More details on these results are provided in the supplementary material.

5. Conclusion

We presented a new take on linear shape models with deep learning, representing large un-annotated collections of 3D shapes. Our alignment-aware model produces concise, expressive and interpretable overviews of unaligned point clouds collections. We show that our method leads to state-of-the-art results for few-shot segmentation.

Acknowledgements This work was supported in part by ANR project READY3D ANR-19-CE23-0007 and HPC resources from GENCI-IDRIS (Grant 2020-AD011012096). We thank François Darmon, Damien Robert, Vivien Sainte Fare Garnot and Yang Xiao for inspiring discussions and valuable feedback.

Supplementary material

6. Implementation details

Our implementation uses PyTorch, Torch-Points3D [9], and an efficient CUDA implementation of the Chamfer distance which significantly speeds up training. Code and data are available at: https://romainloiseau.github.io/deep-linear-shapes

Training strategy. We use the Adam optimizer [33] with a learning rate of 0.001, a batch size of 64, and neither weight decay nor data augmentation. Our model takes point clouds in $\mathbb{R}^{1024\times3}$ as input for all experiments, except for the few-shot segmentation task that takes point clouds in $\mathbb{R}^{2048\times3}$ as input.

Curriculum learning. Inspired by the curriculum learning strategy of [41], we propose to learn our models by gradually increasing the models complexity. We first learn raw prototype models ($\mathcal{R}^k_{\text{proto}}$), an optimization which corresponds to performing a gradient-based K-means algorithm in the 3D space. Second, we augment each model with alignment awareness ($\mathcal{R}^k_{\text{align}}$). Finally, we gradually increase the linear families dimension up to the desired one, resulting in our final shape model ($\mathcal{R}^k_{\text{full}}$). Curriculum learning allows the model to choose the number of displacement fields D according to the complexity of the studied dataset. Early stopping occurs when the benefit of adding a new degree of liberty (*i.e.* increasing D by one) does not meet a criterion on the loss or on a validation task, see Figure 7.

Alignment networks and basis vectors are initially set to identity and zero, respectively. When unfreezing a new module (alignment or a dimension of projection), the learning rate for the new weights is initially set to a tenth of the learning rate applied for the rest of the network, and gradually increased over 50 epochs to the global learning rate. This "warm-up" heuristic helps the network learn more smoothly from one step of the curriculum to the next.

Initialization strategy. As it is the case for many clustering algorithms, initialization can be critical. In our case, we initialize the prototype point clouds with samples of the training set chosen according to a k-means++ strategy [2] with respect to the Chamfer distance.

Cluster reassignment. To prevent empty clusters, we reassign at the end of each epoch any cluster that was selected fewer times than 20% of the expected size of clusters (N/K) in the evenly distributed cluster assignment of Equation 1. Clusters are reassigned by selecting and duplicating another cluster. The duplicated cluster is chosen with a probability proportional to the mean of its reconstruction error over the last epoch. To break the symmetry, we add Gaussian noise with variance 10^{-4} to both its prototype and vector basis. The alignment and projection networks are copied without adding noise. We decrease the reassignment threshold tenfold after each curriculum step in order to preserve less populated but expressive clusters.

Table 4: Low-shot supervised segmentation results on ShapeNetPart. We report the IoU averaged over all classes.

Training data	SONet [36]	3D-PointCapsNet [63]	Ours full $D = 5$
1%	64	67	68
5%	69	70	72

Memory and Speed. With $K = 10$ prototypes and $D = 5$, our model has $4.6M$ parameters. For comparison, the reconstruction models proposed by Wang *et al.* [57] and Groueix *et al.* [25] have respectively $2.6M$ and $10.0M$ parameters. Our model can be trained on a single NVIDIA GeForce RTX 2080Ti within a few hours on the $3\,991$ samples of ModelNet10, and in less than a day on ShapeNetCore. Inference on all samples from ShapeNetCore ($\approx 50k$ shapes) takes less than 4 minutes.

Choice of K. The number of models can be automatically selected through usual model selection heuristics such as the Bayesian Information Criterion (BIC), as we show in Figure 8. Being entirely unsupervized, there is no restriction on how linear families relate to classes: complex classes can be represented by several models, and similar classes by a single family. However, as demonstrated in our clustering experiments, when the number of classes and models are the same, linear families and classes tend to be assigned on a one-to-one basis

7. Low-shot setting

Low-Shot Semantic Segmentation. Our models can learn to perform semantic segmentation from a small number of annotated examples. We first initialize a set number of prototypes per class with random examples from the training set. This allows us to associate each prototype's point with a *part* semantic label. We then perform our standard training scheme, but with an altered Chamfer distance, which can only match points with the same part label from the true and reconstructed point clouds. At inference time, we can associate a part label to each point of the input shape by taking the points' closest neighbor in their reconstructed shape. This setting is supervised in the sense that we use the point labels explicitly during training. As presented in Table 4, our model trained on only 1 and 5% of the annotated

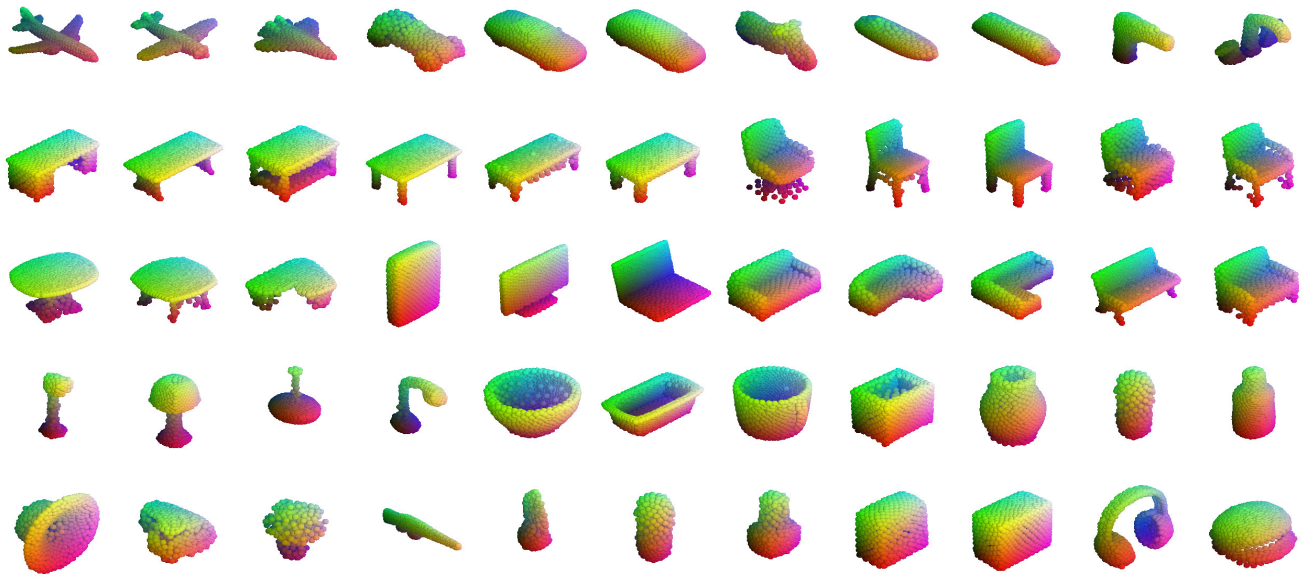


Figure 6: **Modeling ShapeNet.** Prototypes from all 55 linear shape models learned on ShapeNet [8], with our final 5-dimensional model "Ours, full $D = 5$". In this figure, the prototypes have been manually rearranged with respect to their semantics. Note that some diverse classes such as tables or chairs are modeled by several models.

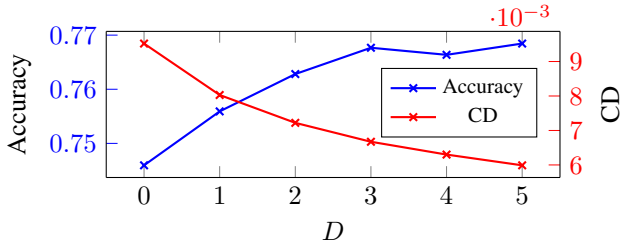


Figure 7: **Influence of D on ModelNet.** The reconstruction error (CD) decreases with added degrees of freedom. In contrast, the clustering Accuracy stops increasing when $D >= 3$, hinting that we have reached a sufficient level of complexity.

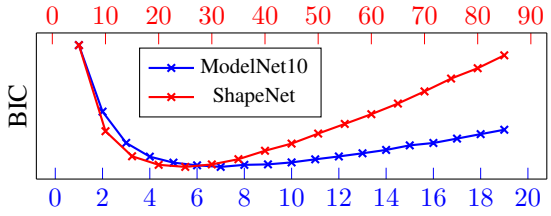


Figure 8: **Model Selection.** We can select the number of clusters K using the BIC. We obtain 7 clusters for ModelNet and 30 for ShapeNet, which is consistent with the shapes' diversity.

shapes yields an improvement of $+1$ and $+2$ average IoU points respectively, compared to 3D-Capsule [63]. In contrast to this more complex model, our linear shape models remain viewable and interpretable.

8. Self-supervised classification

Self-supervised classification. To assess the capacity of our approach to extract relevant information from 3D models, we evaluate it in a standard self-supervised feature learning setup. We train our method on ShapeNetCore with 55 5-dimensional families, extract features from our results, and train and evaluate a linear SVM on the ModelNet10/40 train-test split following standard practices. We define three features that can be extracted from our model. A first type of features is defined as the soft-minimum of the distance between an input point and the reconstruction predicted by each linear shape model, with a temperature taken here as 100 ("Distances"). These first features are complemented by concatenating the coordinates predicted by the projection networks ("Distances and coordinates"). Finally, we directly use the features from our point cloud encoder ("Embedding").

We present the results obtained with these different features in Table 5, and compare to approaches specifically designed for self-supervised classification. While our method's performance is below most of these dedicated approaches, our results are still promising. Interestingly, we

can see that adding shape coordinates to the distances significantly boosts the results, and even outperforms the latent embedding learned by the encoder, which outlines that our learned shape spaces are informative and meaningful.

Table 5: Results of the self-supervised classification task. We report the accuracy of a linear SVM trained on the training set of ModelNet using as input feature the reconstruction error to 55 linear shape families of dimension 5 augmented or not by the predicted coordinates, or the latent vector outputted by the point cloud encoder. In parenthesis, we report the name of the backbone network used (PointNet [44], PointNet++ [45], or VGG19 [48]).

	$N_{\text{feat.}}$	MN40	MN10
Ours, full $D = 5$ (PointNet)			
Distances	55	70.5	86.2
Distances and coordinates	330	86.8	90.9
Embedding	512	86.2	89.6
3D-GAN [60]	7168	83.3	91.0
VIP-GAN [27] (VGG19)	512	90.2	92.2
FoldingNet [62]	512	88.4	94.4
Latent-GAN [1] (PointNet)	512	84.5	95.4
Rec-Space [46] (PointNet)	512	87.3	91.6
Multi-Task [28]	512	89.1	—
Label-Efficient [22] (PointNet++)	512	89.8	—

9. Learned Linear Shape Models

In this section, we show qualitative examples of learned linear shape models. In Figure 6, we represent all 55 models learned on ShapeNet [8], with our final 5-dimensional model "Ours, full $D = 5$". This illustrates how our model can be used to represent concisely a diverse and complex dataset such as ShapeNet without any supervision. In Figure 9, we show the 10 models learned on our subset of ABC [34] for the three steps of our curriculum strategy, illustrating the benefit of both alignment networks and linear families to learn such a diverse shape dataset. We also display the vector basis for all 5 learned dimensions, representing the richness of each linear shape family.

Lastly, we represent in Figure 10 the models learned on ModelNet with random rotations. We observe that when alignment networks are used, the obtained prototypes are similar to the ones obtained on the aligned version of the dataset. This shows that our approach can be used successfully on raw, un-aligned datasets.

10. Reconstruction results

We show some reconstruction results in Figure 11 and Figure 12 for ABC and ShapeNet respectively. For each model, we represent some sample shapes for which the model provides the reconstruction with the lowest error. Viewing our approach in terms of clustering, this amounts to showing elements from the clusters associated with each model. Note that in Figure 12, our linear models are associated with rich subsets of shapes which remain mostly semantically homogeneous.

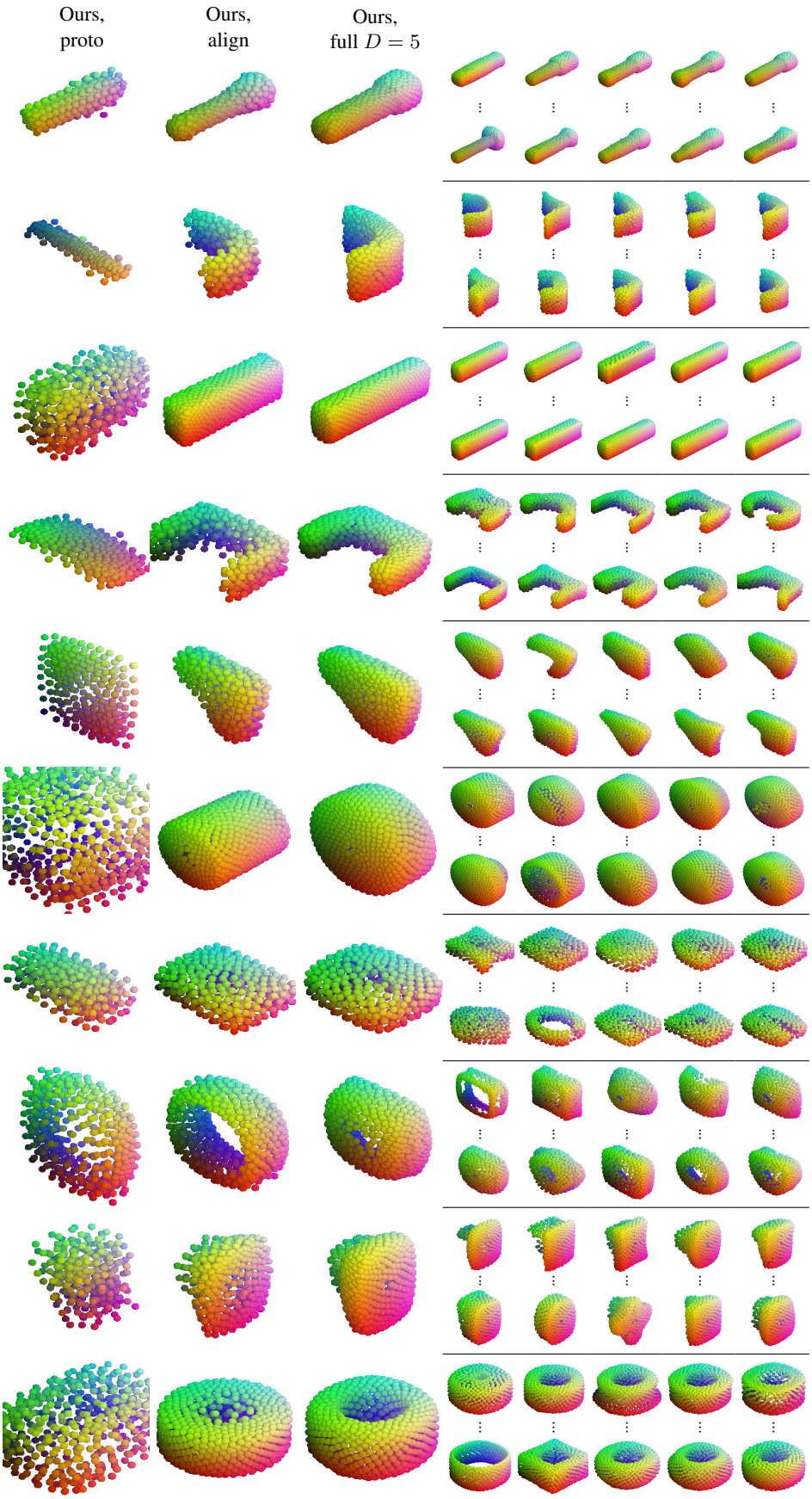
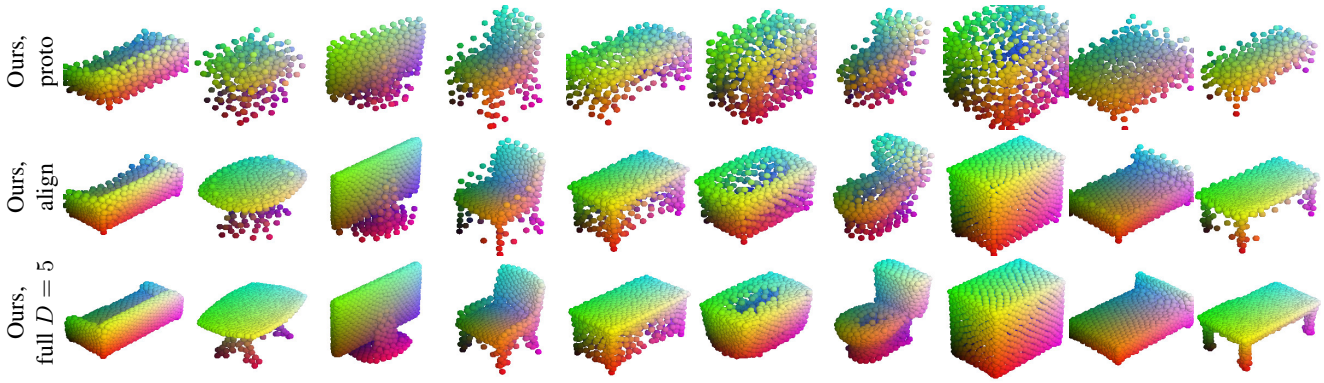
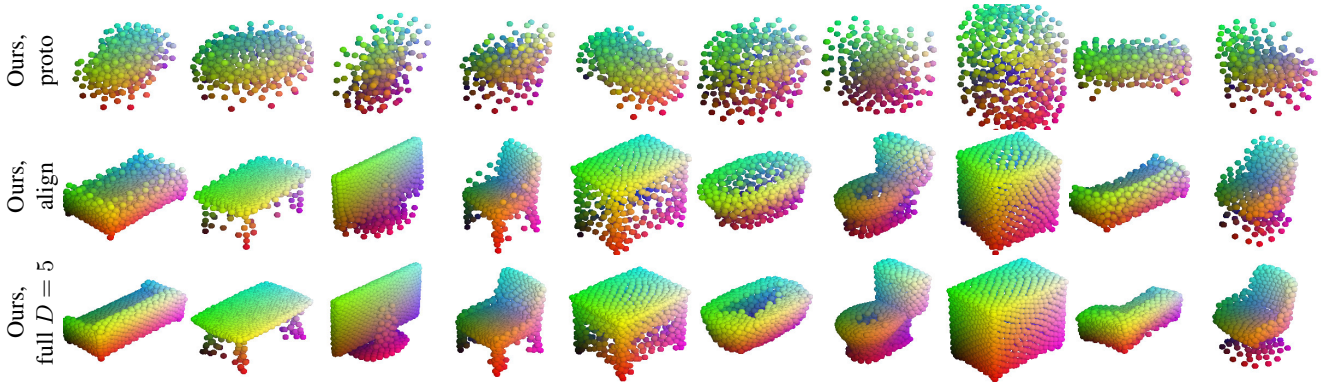


Figure 9: **Modeling ABC.** 10 prototype from linear shape models learned on the ABC dataset [34]. Note that the prototypes are smoother and sharper when using alignment networks and 5-dimensional linear families. On the right-most columns, we illustrate the 5 dimensions of the linear family of each shape model. Each linear family spans a rich subset of the space of shapes.



(a) 10 prototypes of the linear shape models learned from the **aligned ModelNet10** dataset.



(b) 10 prototypes of the linear shape models learned from the **rotated ModelNet10** dataset.

Figure 10: **Modeling ModelNet10.** Prototype learned on ModelNet's [61] aligned version (a) and with random z-axis rotations (b). In this figure, the models are manually rearranged to be in correspondence across the two experiments. Note how our model without alignment networks ("Ours, proto") is unable to learn meaningful prototypes on un-aligned data. In contrast, our models with alignment networks learn sharp and informative prototypes despite the rotations. This shows that alignment networks allow our model to handle a raw, un-aligned dataset to produce a compact overview of its shape diversity.

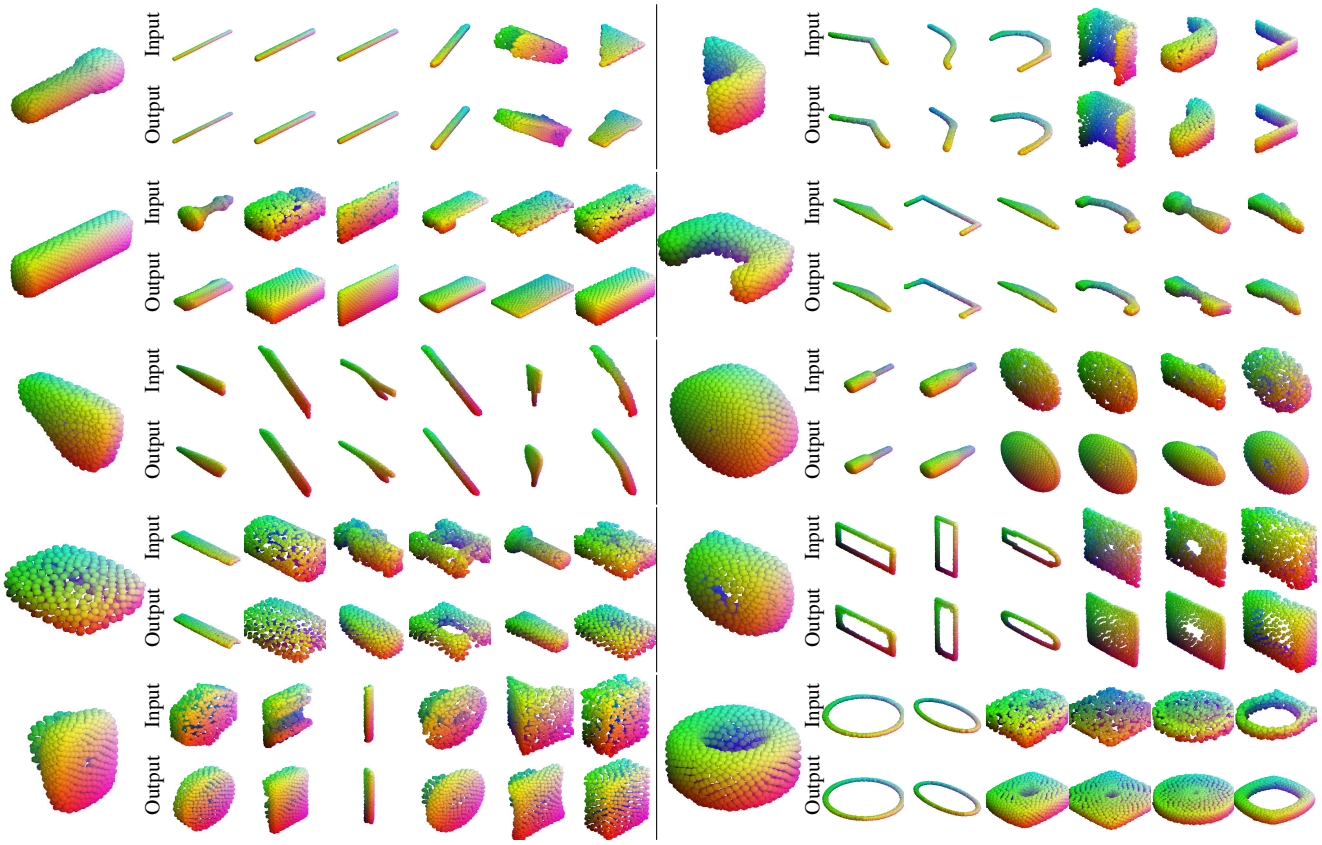


Figure 11: **Visualizing Reconstructions on ABC.** The left-most columns represent prototypes from all 10 linear models learned on the ABC dataset. For each prototype, we select 6 samples for which this model gives the best reconstruction ("Input", top line). We then represent the associated reconstruction provided by the model ("Output", top line). Each family represents a wide variety of morphologically homogeneous shapes: round rings, square rings, bent archs, cylinders, etc... Looking at the prototypes gives us a concise overview of the shape diversity.

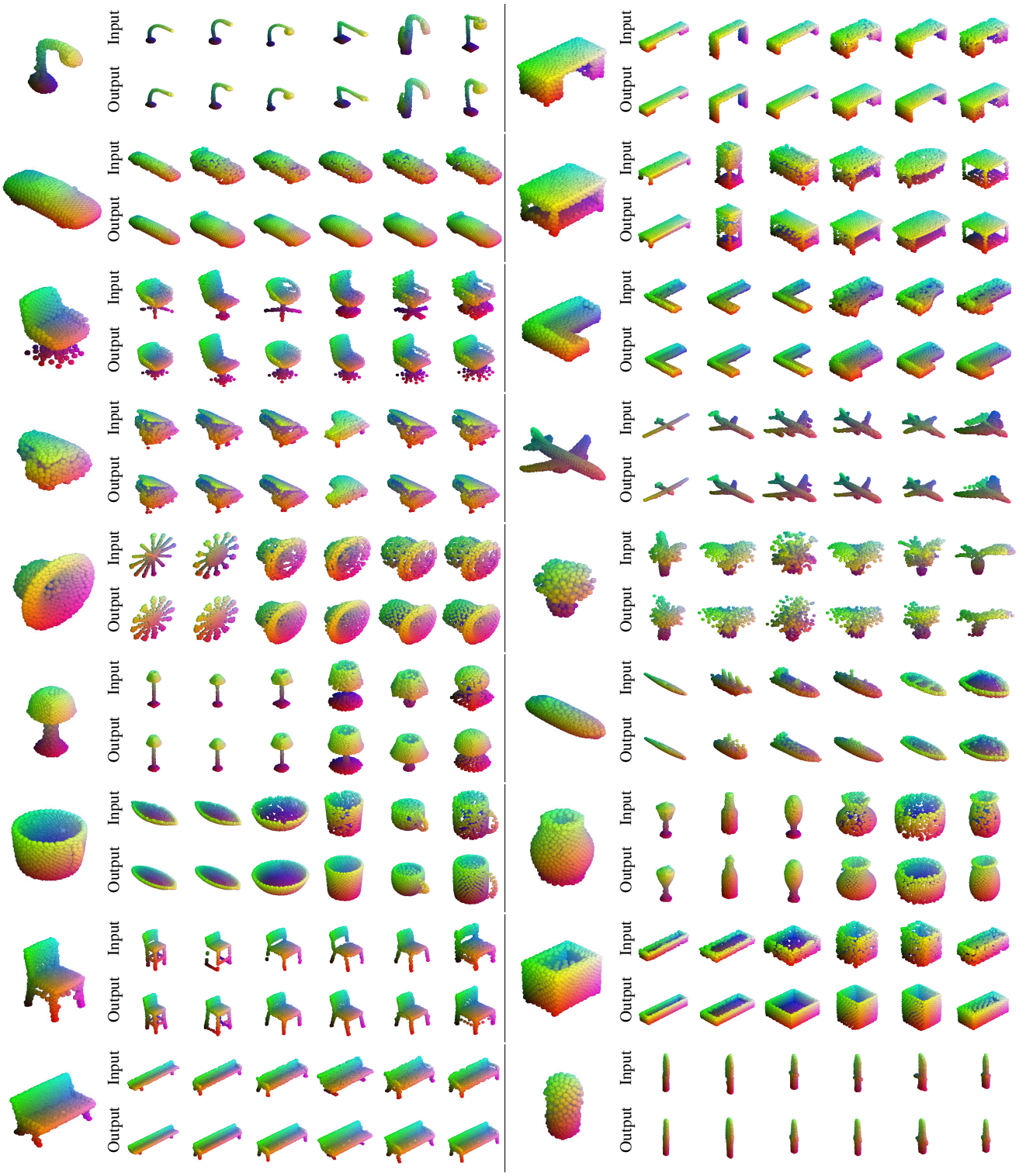


Figure 12: **Visualizing Reconstructions on ShapeNet.** The left-most columns represent prototypes from some of the 55 linear models learned on the ShapeNet dataset. For each prototype, we select 6 samples for which this model gives the best reconstruction ("Input", top line). We then represent the associated reconstruction provided by the model ("Output", top line). We observe that the samples associated with a given model are for the most part semantically homogeneous, and well represented by their prototype.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3D point clouds. In *ICCV*, 2018.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[3] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*. International Society for Optics and Photonics, 1992.

[4] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3D faces. In *SIGGRAPH*, 1999.

[5] Leon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In *NeurIPS*, 1995.

[6] Christoph Bregler, Aaron Hertzmann, and Henning Biermann. Recovering non-rigid 3D shape from image streams. In *CVPR*, 2000.

[7] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *Signal Processing Magazine*, 2017.

[8] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012, Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[9] Thomas Chaton, Chaulet Nicolas, Sofiane Horache, and Loic Landrieu. Torch-points3D: A modular multi-task frameworkfor reproducible deep learning on 3D point clouds. In *3DV*, 2020.

[10] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019.

[11] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019.

[12] Tim F Cootes and Christopher J Taylor. Statistical models of appearance for medical image analysis and computer vision. In *Medical Imaging : Image Processing*. International Society for Optics and Photonics, 2001.

[13] Yuchao Dai, Hongdong Li, and Mingyi He. A simple prior-free method for non-rigid structure-from-motion factorization. *International Journal of Computer Vision*, 2014.

[14] Yu Deng, Jiaolong Yang, and Xin Tong. Deformed implicit field: Modeling 3d shapes with learned dense correspondence. In *CVPR*, pages 10286–10296, 2021.

[15] Theo Deprelle, Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Learning elementary structures for 3D shape generation and matching. In *NeurIPS*, 2019.

[16] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3D object reconstruction from a single image. In *CVPR*, 2017.

[17] Clara Fernandez-Labrador, Ajad Chhatkuli, Danda Pani Paudel, Jose J Guerrero, Cédric Demonceaux, and Luc Van Gool. Unsupervised learning of category-specific symmetric 3d keypoints from point sets. *ECCV*, 2020.

[18] Brendan J Frey and Nebojsa Jojic. Estimating Mixture Models of Images and Inferring Spatial Transformations Using the EM Algorithm. In *CVPR*, 1999.

[19] Brendan J Frey and Nebojsa Jojic. Transformed component analysis: Joint estimation of spatial transformations and image components. In *ICCV*, 1999.

[20] Brendan J Frey and Nebojsa Jojic. Fast, large-scale transformation-invariant clustering. In *NeurIPS*, 2002.

[21] Brendan J Frey and Nebojsa Jojic. Transformation-Invariant Clustering Using the EM Algorithm. *Transactions on Pattern Analysis and Machine Intelligence*, 2003.

[22] Matheus Gadelha, Aruni RoyChowdhury, Gopal Sharma, Evangelos Kalogerakis, Liangliang Cao, Erik Learned-Miller, Rui Wang, and Subhransu Maji. Label-efficient learning on point clouds using approximate convex decompositions. In *ECCV*, 2020.

[23] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *ICCV*, 2019.

[24] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3D surface generation. In *CVPR*, 2018.

[25] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Unsupervised cycle-consistent deformation for shape matching. In *Computer Graphics Forum*, 2019.

[26] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3D point clouds: A survey. *Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[27] Zhizhong Han, Mingyang Shang, Yu-Shen Liu, and Matthias Zwicker. View inter-prediction gan: Unsupervised representation learning for 3D shapes by learning global shape memories to support local view predictions. In *AAAI*, 2019.

[28] Kaveh Hassani and Mike Haley. Unsupervised multi-task feature learning on point clouds. In *CVPR*, 2020.

[29] Pedro Hermosilla, Tobias Ritschel, and Timo Ropinski. Total denoising: Unsupervised learning of 3D point cloud cleaning. In *CVPR*, 2019.

[30] Lila Huang, Shenlong Wang, Kelvin Wong, Jerry Liu, and Raquel Urtasun. Octsqueeze: Octree-structured entropy model for lidar compression. In *CVPR*, 2020.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[32] Chiyu Jiang, Jingwei Huang, Andrea Tagliasacchi, and Leonidas Guibas. Shapeflow: Learnable deformations among 3d shapes. In *NeurIPS*, 2020.

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[34] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *CVPR*, 2019.

[35] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, 2018.

[36] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *CVPR*, 2018.

[37] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *Transactions on Graphics*, 2017.

[38] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 2020.

[39] Facundo Mémoli and Guillermo Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Foundations of Computational Mathematics*, 2005.

[40] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019.

[41] Tom Monnier, Thibault Groueix, and Mathieu Aubry. Deep Transformation-Invariant Clustering. In *NeurIPS*, 2020.

[42] G Dias Pais, Srikumar Ramalingam, Venu Madhav Govindu, Jacinto C Nascimento, Rama Chellappa, and Pedro Miraldo. 3DRegNet: A deep neural network for 3D point registration. In *CVPR*, 2020.

[43] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019.

[44] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*, 2017.

[45] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017.

[46] Jonathan Sauder and Bjarne Sievers. Self-supervised deep learning on point clouds by reconstructing space. In *NeurIPS*, 2019.

[47] Manolis Savva, Angel X Chang, and Pat Hanrahan. Semantically-enriched 3D models for common-sense knowledge. In *CVPR Workshops*, 2015.

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[49] Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America*, 1987.

[50] An Tao. Unsupervised point cloud reconstruction for classific feature learning. https://github.com/AnTao97/UnsupervisedPointCloudReconstruction, 2020.

[51] Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3D reconstruction networks learn? In *CVPR*, 2019.

[52] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*, 2019.

[53] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 1992.

[54] Lorenzo Torresani, Aaron Hertzmann, and Chris Bregler. Nonrigid structure-from-motion: Estimating shape and motion with hierarchical priors. *Transactions on Pattern Analysis and Machine Intelligence*, 2008.

[55] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 1991.

[56] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *CVPR*, 1991.

[57] Lingjing Wang, Xiang Li, and Yi Fang. Few-shot learning of part-specific probability space for 3D shape segmentation. In *CVPR*, 2020.

[58] Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. Linear subspace design for real-time shape deformation. *Transactions on Graphics*, 2015.

[59] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 1987.

[60] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NeurIPS*, 2016.

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

[62] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, 2018.

[63] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3D point capsule networks. In *CVPR*, 2019.

[64] Zerong Zheng, Tao Yu, Qionghai Dai, and Yebin Liu. Deep implicit templates for 3d shape representation. In *CVPR*, 2021.