General Hypernetwork Framework for Creating 3D Point Clouds

Przemyslaw Spurek, Maciej Zieba, Jacek Tabor, and Tomasz Trzcinski, *Senior Member, IEEE*

Abstract—In this work, we propose a novel method for generating 3D point clouds that leverages the properties of hypernetworks. Contrary to the existing methods that learn only the representation of a 3D object, our approach simultaneously finds a representation of the object and its 3D surface. The main idea of our HyperCloud method is to build a hypernetwork that returns weights of a particular neural network (target network) trained to map points from prior distribution into a 3D shape. As a consequence, a particular 3D shape can be generated using point-by-point sampling from the prior distribution and transforming the sampled points with the target network. Since the hypernetwork is based on an auto-encoder architecture trained to reconstruct realistic 3D shapes, the target network weights can be considered to be a parametrization of the surface of a 3D shape, and not a standard representation of point cloud usually returned by competitive approaches. We also show that relying on hypernetworks to build 3D point cloud representations offers an elegant and flexible framework. To that point, we further extend our method by incorporating flow-based models, which results in a novel HyperFlow approach.

Index Terms—Hypernetworks, 3D point cloud processing, generative modeling

1 Introduction

Today many registration devices, such as LIDARs and depth cameras, are able to capture not only RGB channels but also depth estimates. As a result, 3D objects registered by those devices and geometric data structures representing them, called point clouds, become increasingly important in contemporary computer vision applications, including autonomous driving [1] or robotic manipulation [2]. To enable the processing of point clouds, researchers typically transform them into regular 3D voxel grids or collections of images [3], [4]. This, however, increases the memory footprint of object representations and leads to significant information losses. On the other hand, representing 3D objects with the parameters of their surfaces is not trivial due to the complexity of mesh representations and combinatorial irregularities. Last but not least, point clouds can contain a variable number of data points corresponding to one object and registered at various angles, which requires the methods that process them to be permutation and rotation invariant.

One way of addressing the above challenges related to point cloud representations is to subsample the point clouds and enforce permutation invariance within the model architecture, as it was done in DeepSets [5] or PointNet [6], [7]. Although it works perfectly fine when point clouds are given as *an input* of the model, it is not obvious how to apply this approach for variable size *outputs*. A recently introduced family of methods solves this problem by relying on generative models that return probability distribution of the points on the object surface, instead of an exact set of points [8], [9]. The most successful methods that follow this path, such as PointFlow [8] and Conditioned Invertible Flow [9], are based on a flow architecture that allows obtaining a representation of 3D object surfaces.

The main limitation of the existing flow-based models is the fact that their training relies on a conditioning mechanism which, in turn, requires more complex architectures, an increased number of parameters, and a significant amount of structural fine-tuning. Moreover, flow-based methods cannot be trained on probability distributions without compact support. For instance, it is not possible to train a flow-based model on a 3D ball since computing a cost function using log-likelihood returns infinity as a result and can therefore lead to numerical instability of the entire training procedure. Additionally, flow-based models require the dimensionality of input and output data to be identical.

In our previous work [11] we show that one can circumvent the above shortcomings of the flow models by using a HyperCloud model. HyperCloud builds on the approach

- *Przemyslaw Spurek and Jacek Tabor are with Jagiellonian University, 31-007 Kraków, Poland. E-mail: przemyslaw.spurek@gmail.com, jacek.tabor@uj.edu.pl.*
- *Maciej Zieba is with the Wroclaw University of Science and Technology, 50-370 Wroclaw, Poland, and also with the Tooploox, 53-601 Wroclaw, Poland. E-mail: maciej.zieba@pwr.edu.pl.*
- *Tomasz Trzcinski is with the Warsaw University of Technology, 00-661 Warsaw, Poland, and with the Jagiellonian University, 31-007 Kraków, Poland, and also with the Tooploox, 53-601 Wroclaw, Poland. E-mail: t.trzcinski@ii.pw.edu.pl.*

Manuscript received 6 October 2020; revised 2 November 2021; accepted 20 November 2021. Date of publication 26 November 2021; date of current version 3 November 2022.
The work of Przemyslaw Spurek was supported by the National Centre of Science (Poland) under Grant 2019/33/B/ST6/00894. The work of Jacek Tabor was supported by the National Centre of Science (Poland) under Grant 2017/25/B/ST6/01271. The work of Tomasz Trzcinski was supported in part by the National Centre of Science (Poland) under Grant 2020/39/B/ST6/01511 and in part by the Foundation for Polish Science under Grant POIR.04.04.00-00-14DE/18-00 co-financed by the European Union under the European Regional Development Fund. The work of Maciej Zieba was supported by the National Centre of Science (Poland) under Grant 2020/37/B/ST6/03463. The authors have applied a CC BY license to any Author Accepted Manuscript (AAM) version arising from this submission, in accordance with the grants' open access conditions.
(Corresponding author: Przemyslaw Spurek.)
Recommended for acceptance by E. Kalogerakis.
Digital Object Identifier no. 10.1109/TPAMI.2021.3131131

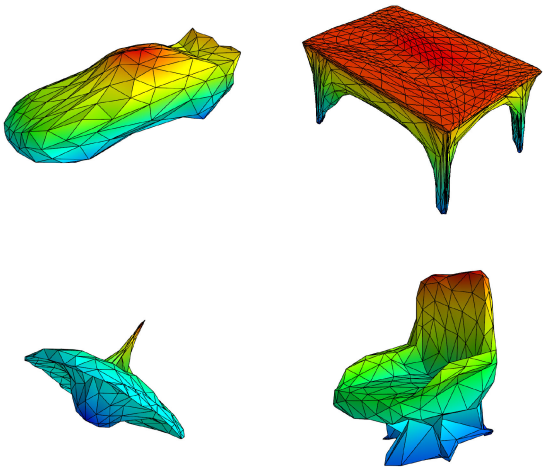

Fig. 1. Mesh representations generated by our HyperCloud method. Contrary to the existing methods that return point cloud representations sparsely distributed in 3D space, our approach allows creating a continuous 3D object representation in the form of meshes.

of [10] and combines it with a hypernetwork [12], [13] that outputs weights of a generative model, the so-called *target network*. The target network can then be used to create an arbitrary number of points (depending on its architecture returned by a hypernetwork), instead of fixed-size sets. The proposed model is a hypernetwork, whereas previous works encoded the prior distribution transformation as a latent vector. Our HyperCloud method is much easier to train than the competing algorithms, as it requires a smaller number of hyperparameters and does not put any constraints on the input probability distribution and its Jacobian. Methods that use log-likelihood as a cost function cannot be trained on probability distributions with compact support. Finally, as presented in Fig. 3, the HyperCloud method returns a continuous mesh representation of 3D objects at virtually no cost in the quality of reconstructions (see Fig. 1).

In this paper, we postulate that using hypernetworks to build powerful 3D point representations offers an elegant and flexible framework and, to that end, we introduce a more general method for creating such representations that also encompasses the existing flow models. To satisfy the requirements of the flow models, we need to introduce the prior distribution of our target network that offers non-compact support. On the other hand, representing objects by modeling their surfaces inspires us to use probability distributions that allow a straightforward transformation of a 3D point cloud into a mesh, as it is done HyperCloud via the so-called *triangulation trick*, as shown in Fig 3.

Thus, we consider a point cloud as a sample from a distribution on object surfaces with additive noise introduced by a registration device, such as LIDAR. To model this distribution, we propose a new Spherical Log-Normal function, which mimics the topology of 3D objects and provides non-compact support.

This, in turn, enables effective utilization of a flow-based model as a part of a hypernetwork, instead of a fully-connected neural network as we do in HyperCloud. The resulting general framework which we call HyperFlow produces state-of-the-art generative results both for point clouds and mesh representations, while reducing the training time and corresponding memory footprint of the model by over an order of magnitude with respect to the competing flow-based methods.

To summarize, we have extended our previous work in the following way:

- We have introduced a new HyperFlow generative framework that encompasses previous hypernetwork-based models while allowing incorporation of powerful flow-based architectures.
- To achieve that, we have proposed a new Spherical Log-Normal distribution which models a point cloud density with non-compact support and, hence, can be effectively used by a flow-based model.
- The resulting method offers a significant reduction in training time and memory footprint with respect to the more complex flow-based models while preserving state-of-the-art generative capabilities.

The remainder of this paper is structured as follows: Section 2 discusses related works. In Section 3, we present our HyperCloud approach. In Section 4 we introduce Spherical Log-Normal probability distribution that enables the generalization of HyperCloud framework in order to encompass flow-based models and in Section 5 we introduce the generalized HyperFlow method for building 3D point cloud representations. Finally, Section 6 presents the results of evaluations and we conclude this work in Section. 7.

2 Related Work

Introducing deep learning in the context of 3D point cloud representations allowed improving performance in various discriminative tasks, including classification [5], [6], [7], [14] and segmentation [6]. Despite those successes, generating 3D point clouds with deep learning models remains a challenging task.

Point Clouds Reprehension of 3D Objects. Due to the irregular format of point cloud representation, most researchers transform such data to regular 3D voxel grids or collections of images. In [4], the authors propose the voxelized representation of an input point cloud. Other approaches use multi-view 2D images [3] or occupancy grid calculation [15], [16]. Modeling volumetric objects in a general-adversarial manner is also considered in [17] for the 3D-GAN model.

Another approach to generative models for point cloud converts a point distribution to a $N \times 3$ matrix by sampling a pre-defined number of N points from the distribution so that existing generative models are applicable. Such a solution can be applied in the VAE framework [18] as well as adversarial auto-encoders (AAEs) [10]. In the above methods, auto-encoders and GANs are trained with loss functions that directly optimize the distance between two point sets, e.g., using Chamfer distance (CD) or earth mover's distance (EMD). In [19], the authors apply auto-regressive models [20] with a discrete point distribution to generate one point at a time, also using a fixed number of points per shape.

An important class of generative models of 3D point clouds are Energy-Based Models (EBMs). Energy-based generative ConvNets [21] approximate the explicit probability distribution of data in the form of an energy function parametrized by a convolutional neural network. In such a case, new point clouds can be generated using Monte Carlo Markov

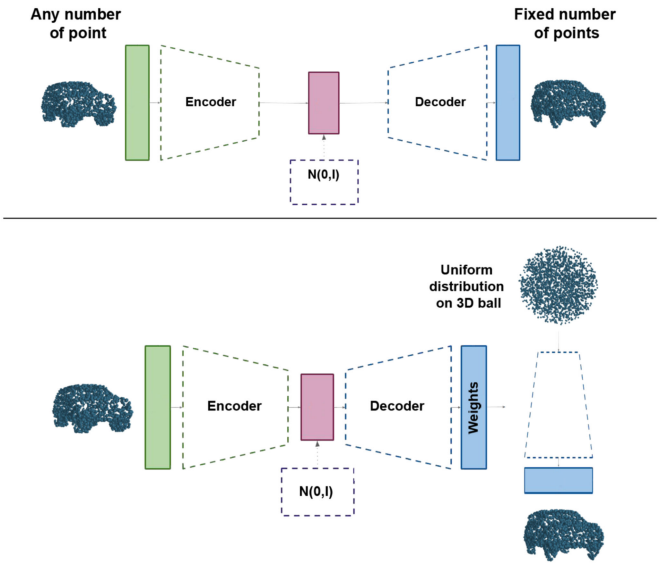

Fig. 2. *Top:* The baseline approach for generating 3D point clouds returns a fixed number of points [10]. *Bottom:* Our HyperCloud method leverages a hypernetwork architecture that takes a 3D point cloud as an input while returning the parameters of the *target network*. Since the parameters of the target network are generated by a hypernetwork, the output dataset can be variable in size. We can sample any number of points from the uniform distribution on a 3D ball and transfer them to surface of an object. As a result, we obtain a continuous parametrization of the surface of the object and a more robust representation of its mesh.

Chain (MCMC) sampling. Such an architecture was used to generate images [21], videos [22], [23], [24], and 3D voxels [25], [26]. One of the most recent models, Generative PointNet (GPN), applies this approach to 3D point clouds [27].

All the above methods learn to produce a fixed number of points for each shape, but they do not parametrize the surface of the shapes. Treating a point cloud as a fixed-dimensional matrix has several drawbacks. First, the model is restricted to generating a fixed number of points. Getting more points for a particular shape requires separate up-sampling models such as [28], [29].

In [8], the authors propose a principled probabilistic framework to generate 3D point clouds by modeling them as a distribution of distributions. PointFlow uses two-level of distributions where the first level is the distribution of shapes, and the second level is the distribution of points given a shape. PointFlow uses continuous normalizing flow [30], [31] for both of these tasks.

Instead of directly parametrizing the distribution of points in a shape, PointFlow models this distribution as an invertible parameterized transformation of 3D points from a prior distribution (e.g., a 3D Gaussian an). Intuitively, under this model, generating points for a given shape involves sampling points from a generic Gaussian prior distribution and then moving them according to this parameterized transformation to their new location in the target shape. Such a solution has many advantages over the classical approaches, which only produce a cloud of points, nevertheless it is limited in multiple ways. The most important limitation is the fact that they use log-likelihood as a cost function and, in consequence, cannot be trained on probability distributions with compact support. This significantly reduces the utility of flow-based models as, for instance, using a 3D ball distribution as a prior one returns infinite

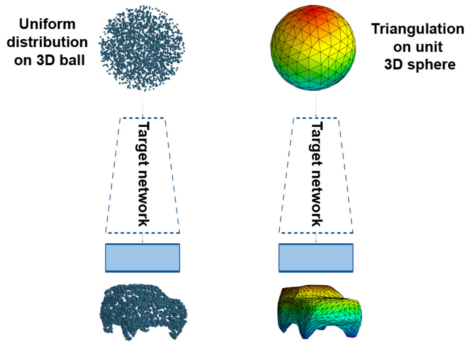

Fig. 3. Scheme of producing mesh representations with HyperCloud. When using 3D *ball* distribution, our method can generate 3D point clouds filled with data points, while when given 3D *sphere* distribution, it transforms samples from the sphere to surfaces of 3D objects - a feature highly desirable in the context of 3D mesh rendering.

values and therefore leads to numerical instability of the training. In this work, we show that once this constraint is dropped thanks to using a fully connected neural network, we can directly model 3D point cloud surfaces and hence create their continuous mesh representations.

Mesh Representation of 3D Objects. One of the most challenging tasks in 3D point cloud generation is producing mesh representations using only raw 3D point cloud in training. Mesh is a set of vertices joined together with edges that enable a piece-wise planar approximation of a surface.

A mesh of an object can be obtained with a transformation of a mesh on a unit sphere [11], [32]. However, such methods are limited, and they reconstruct objects that are topologically the same as spheres.

Patch-based approaches [33], [34], [35] are much more flexible and enable modeling virtually any surface, including those with a non-disk topology. This is achieved using parametric mappings to transform 2D patches into a set of 3D shapes. The first deep neural network which models 2D manifold into 3D space was FoldingNet [14]. FoldingNet uses a single patch to model the surface of an object. In AtlasNet [36], the authors introduced a method that used several patches to model a mesh. Such elements in the atlas are trained independently. Subsequently, these patches are not stitched together, causing discontinuities appearing as holes or intersections patches. Therefore in the case of AtlasNet, the authors also use a sphere to construct a mesh.

3 HyperCloud: Hypernetwork for Generating 3D Point Clouds

In this section, we present our HyperCloud model for generating 3D point clouds. HyperCloud encompasses previously introduced approaches: the auto-encoder based generative model proposed in [10] and the hypernetwork proposed in [12]. Before we present our solution, we will briefly describe these two approaches.

3.1 Adversarial Auto-Encoders for 3D Point Clouds

Let us start with the auto-encoder architecture for 3D point clouds. Let $\mathcal{X} = \{X_i\}_{i=1,\dots,n} = \{(x_i, y_i, z_i)\}_{i=1,\dots,n}$ be a given dataset containing point clouds. The basic goal of an auto-encoder is to transport the data through a typically, but not necessarily, lower dimensional latent space $\mathcal{Z} \subseteq \mathbb{R}^D$ while

minimizing the reconstruction error. Thus, we search for an encoder $\mathcal{E}: \mathcal{X} \rightarrow \mathcal{Z}$ and decoder $\mathcal{D}: \mathcal{Z} \rightarrow \mathcal{X}$ functions, which minimizes the reconstruction error between X_i and its reconstructions $\mathcal{D}(\mathcal{E}(X_i))$.

For point cloud representation, the crucial step is to define proper reconstruction loss that can be used in the autoencoding framework. In the literature, two common distance measures are successively applied for reconstruction purposes: Earth Mover's (Wasserstein) Distance [37], and Chamfer pseudo-distance [38].

Earth Mover's Distance (EMD) is a metric between two distributions based on the minimal cost that must be paid to transform one distribution into the other. For two equally sized subsets $X_1 \subset \mathbb{R}^3$ and $X_2 \subset \mathbb{R}^3$ their EMD is defined as

$$EMD(X_1, X_2) = \min_{\phi: X_1 \rightarrow X_2} \sum_{x \in X_1} c(x, \phi(x)),$$

where ϕ is a bijection and $c(x, \phi(x))$ is the cost function and can be defined as

$$c(x, \phi(x)) = \frac{1}{2} \|x - \phi(x)\|_2^2.$$

The second metric, Chamfer pseudo-distance (CD), measures the squared distance between each point in one set to its nearest neighbor in the other set

$$CD(X_1, X_2) = \sum_{x \in X_1} \min_{y \in X_2} \|x - y\|_2^2 + \sum_{x \in X_2} \min_{y \in X_1} \|x - y\|_2^2.$$

An auto-encoder based generative model is a classical auto-encoder model with a modified cost function, which forces the model to be generative, i.e., ensures that the data transported to the latent space comes from the prior distribution (typically Gaussian one) [39], [40], [41]. Thus, to construct a generative auto-encoder model, we add a measure of the distance of a given sample from the prior distribution to its cost function.

Variational Auto-encoders (VAE) are generative models that are capable of learning an approximated data distribution by applying variational inference [39]. To ensure that the data transported to latent space $\mathcal{Z}$ are distributed according to standard normal density, we add the distance from standard multivariate normal density

$$\text{cost}(X; \mathcal{E}, \mathcal{D}) = Err(X; \mathcal{D}(\mathcal{E}(X))) + \lambda D_{KL}(\mathcal{E}(X), N(0, I)),$$

where D_{KL} is the Kullback–Leibler divergence [42].

The main limitation of VAE models is that the regularization term requires a particular prior distribution to make the KL divergence tractable. In order to deal with that limitation, the authors of [43] introduce an Adversarial Auto-encoder (AAE) that utilizes adversarial training to force a particular distribution on $\mathcal{Z}$ space. The model assumes an additional neural network - the discriminator, which is responsible for distinguishing between fake and true samples, where the true samples are sampled from an assumed prior distribution and fake samples are generated via an encoding network.

In [10], the authors propose an approach to Adversarial Auto-encoders dedicated to the 3D point clouds. Because the input of the model is a set of points, they use an $\mathcal{E}$ PointNet model [6] that is invariant to permutations as an encoder. They receive the same distribution for all possible orderings of points from X. Since the discriminator is not permutation invariant mapping D (as it is a simple MLP model), the authors utilize an additional function that provides one-to-one mapping for the points stored in X.

The probability distribution assumed for the latent space can be more complex than $N(0, I)$ and not given in an explicit form. Some autoencoders try to learn some more sophisticated distributions directly from the data. Such solutions may utilize techniques like VampPrior [44] or incorporate continuous [8] or discrete [45] normalizing flows.

Due to large techniques of enforcing probability distribution on the latent space, the cost function of the model can be formulated in the more general form

$$\text{cost}(X; \mathcal{E}, \mathcal{D}) = Err(X; \mathcal{D}(\mathcal{E}(X))) + Reg(\mathcal{E}(X), P), \tag{1}$$

where Err is Earth Mover's (Wasserstein) Distance or Chamfer pseudo-distance and Reg is a function that forces latent space to be from some known or trainable distribution P. For known distributions like the Gaussian one, Kullback–Leibler divergence or adversarial training can be used for regularization.

In our work, we propose to enrich the presented regularized autoencoder by replacing the decoder with the hypernetwork. The goal of the hypernetwork is to transform the latent representation of the point cloud to the weights of the so-called target network. The goal of the target network is to transform the samples from the assumed prior to the points that represent 3D shapes without assuming the arbitrary fixed number of points. Roughly speaking, in our case, hypernetwork produces a parametrization of the respective generative model.

3.2 Hyper-Network

Hyper-networks, introduced in [12] are defined as neural models that generate weights for a separate target network solving a specific task. The authors aim to reduce the number of trainable parameters by designing a hyper-network with a smaller number of parameters than the target network. By making an analogy between hyper-networks and generative models, the authors of [46], use this mechanism to generate a diverse set of target networks approximating the same function.

Hyper-networks can also be used for functional representations of images [13]. In such a concept by means of a functional (or deep) representation of an image, the authors introduce a function (neural network) $I: \mathbb{R}^2 \rightarrow \mathbb{R}^3$ which given a point (with arbitrary coordinates) (x, y) in the plane returns the point in $[0, 1]^3$ representing the RGB values in a continuous domain of the color of the image at a location (x, y). In such a framework, images are represented by functions (neural networks) that transfer pixel locations $\mathbb{N} \times \mathbb{N}$ into color space. More precisely, the neural network generates an RGB color for each pixel position. In this case, the entire image is created by processing each pixel location to obtain the corresponding color values.

3.3 HyperCloud

Inspired by the above methods, we propose our HyperCloud model that uses a hyper-network to output weights

of a generative network to create 3D point clouds, instead of generating them directly with the decoder, as done in [10]. More specifically, we present a parameterization of the surface of 3D objects as a function $S : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, which given a point from the prior distribution (x, y, z) returns the point on the surface of the objects. Roughly speaking, instead of producing a 3D point cloud, we would like to produce many neural networks (a different neural network for each object) that model the surfaces of objects.

In practice, we have one neural network architecture that uses different weights for each 3D object. More precisely, we model function $T_\theta : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ (neural network with weights θ), which takes an element from the prior distribution $\mathcal{P}$ and transfers it onto an element on the surface of the object. In our work, we use the transformation between a uniform distribution on the 3D ball and the object. This choice of a distribution allows one to create a continuous mesh representation. The key idea behind this is that the distribution does not have compact support. Roughly speaking, the Gaussian distribution does not have a smooth border.

In consequence, we can produce as many points as we need (we can sample an arbitrary number of points from the uniform distribution of the unit ball and transfer them by the target network). Thanks to the target network, the model can be trained on point clouds containing a different number of points.

Furthermore, we can produce a continuous mesh representation of the object. All elements from the ball are transformed into a 3D object. In consequence, the unit sphere is transformed into the surface of the object. Now we can produce meshes without a secondary mesh rendering procedure. It is obtained by simply feeding our neural network by the vertices of a sphere mesh, see Fig 3. As a result, we obtain meshes of 3D objects. The sharpness of the borders is a direct consequence of compact support probability distribution of the input prior. Since flow-based models cannot handle this family of priors and require infinite support distributions, the representations generated with those models are of lower quality.

The target network is not trained directly. We use a hyper-network $H_\phi : \mathbb{R}^3 \supset X \rightarrow \theta$, which for a point-cloud $X \subset \mathbb{R}^3$ returns weights θ to the corresponding target network T_θ. Thus, a point cloud X is represented by a function

$$T((x, y, z); \theta) = T((x, y, z); H_\phi(X)).$$

To use the above model, we need to train the weights ϕ of the hypernetwork. For this purpose, we minimize the distance between point clouds like Chamfer distance (CD) or earth mover's distance (EMD) over the training set of points clouds. More precisely, we take an input point cloud $X \subset \mathbb{R}^3$ and pass it to H_ϕ. The hypernetwork returns weights θ to target network T_θ. Next, the input point cloud X is compared with the output from the target network T_θ (we sample the correct number of points from the prior distribution and transfer them by the target network). As a hypernetwork, we use a permutation invariant encoder that is based on PointNet architecture [6] and a modified decoder to produce weights instead of row points. The architecture of T_θ consists of: an encoder ($\mathcal{E}$) which is a PointNet-like network that transports the data to lower-dimensional latent space $\mathcal{Z} \in \mathbb{R}^D$ and a decoder ($\mathcal{D}$) (fully connected network), which transfers the latent space to the vector of weights for the target network. In our framework, hypernetwork $T_\theta(X)$ represents our autoencoder structure $\mathcal{D}(\mathcal{E}(X))$. Assuming $T_\theta(X) = \mathcal{D}(\mathcal{E}(X))$, we train our model by minimizing the cost function given by Equation (1).

3.4 Limitations

Using the hypernetwork as part of our HyperCloud method, we obtain a simple yet effective approach for modeling 3D point clouds. Instead of relying on more complex flow-based modules, such as the Continuous Normalization Flow (CNF) [30] in PointFlow [8], we use a hypernetwork to return in HyperClouda fully connected *target network* that maps a uniform distribution on a 3D ball to a 3D point cloud. Relying on a hypernetwork instead of conditioning on a CNF module with the autoencoder latent space [8] reduces the number of CNF function parameters in our model. As a consequence, we reduce the training time and corresponding memory footprint of the model by over an order of magnitude with respect to the competing PointFlow. However, although much more efficient, our resulting HyperCloud model does not offer the flexibility that is required to fully reconstruct complex 3D shapes, and hence may result in inferior performance compared to the competing flow-based models. The most straightforward way to address this limitation is to substitute a conventional fully connected target network with a CNF module. Yet this substitution is not possible without modifying the underlying probability distribution, since flow-based models cannot be trained on compact support priors. In the next sections, we show how to mitigate this limitation by introducing a novel probability distribution function that can be later integrated into a general framework for building 3D point cloud representations.

4 Spherical Log-Normal distribution and the triangulation trick

In this section, we introduce Spherical Log-Normal distribution that offers non-compact support required by flow-based modules by modeling the density of point clouds around the surfaces of 3D objects. We then show how it can be also used in the context of generating 3D meshes via the so-called *triangulation trick*.

Since our approach relies on flow-based models, a density distribution has to fulfill several conditions to be used in practice. First of all, flow-based methods cannot be trained on probability distributions with compact support. For instance, it is not possible to train a flow-based model on a 3D ball, as proposed in HyperCloud [11], since computing the log-likelihood cost function used in flows would return infinity for this distribution. As a result, the model does not converge due to numerical instability. Second, we would like to model the probability distribution of the surface (mesh representation), which is two-dimensional (the border of a 3D object). Therefore, a Gaussian distribution in $\mathbb{R}^3$ is not a good choice since it models only elements in 3D. Finally, the density distribution should be topologically coherent with the density of the modeled object. More precisely, because of the way registration devices sample space around object surfaces, point clouds are populated with the

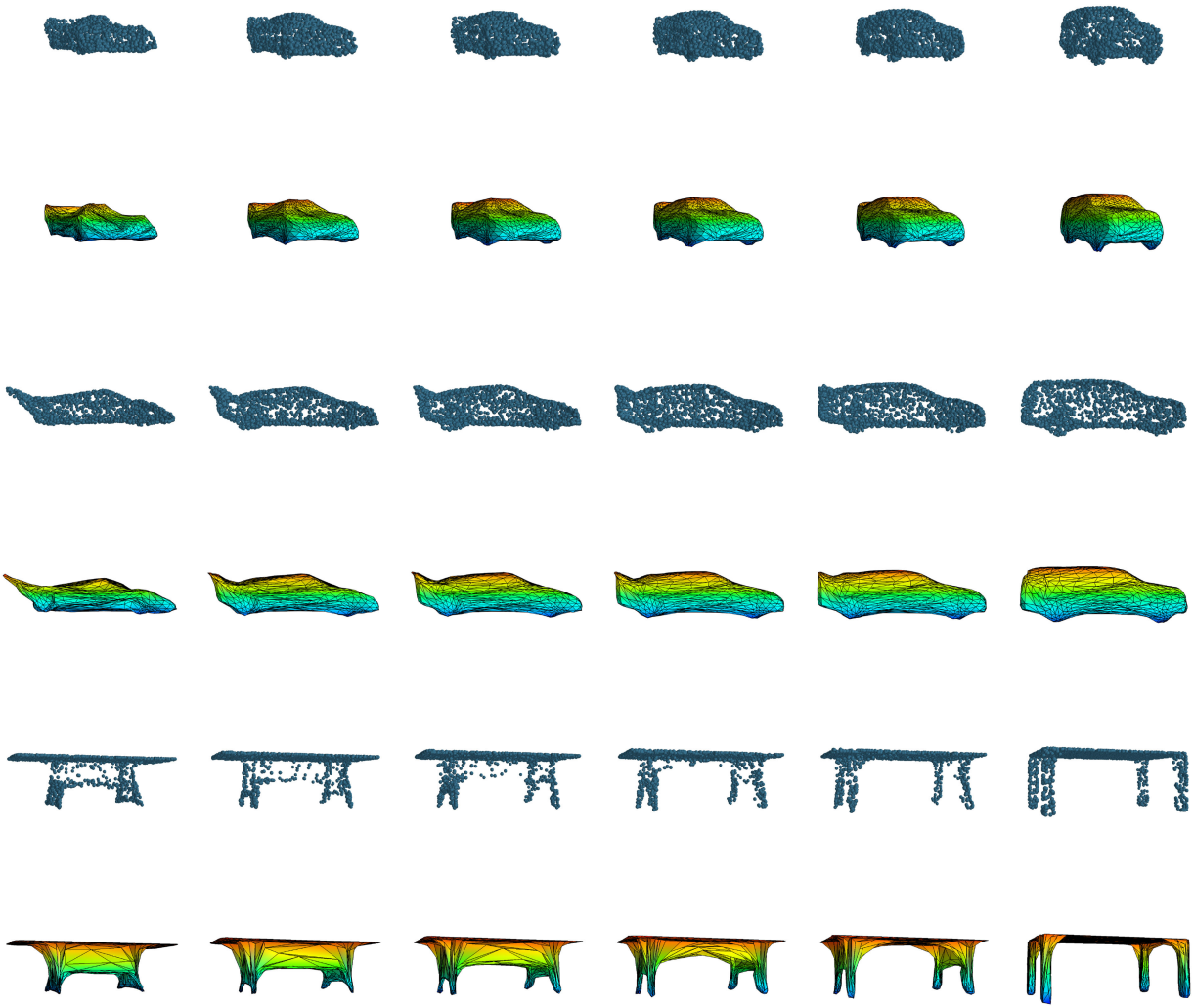

Fig. 4. Interpolations between two 3D point clouds and its mesh representations.

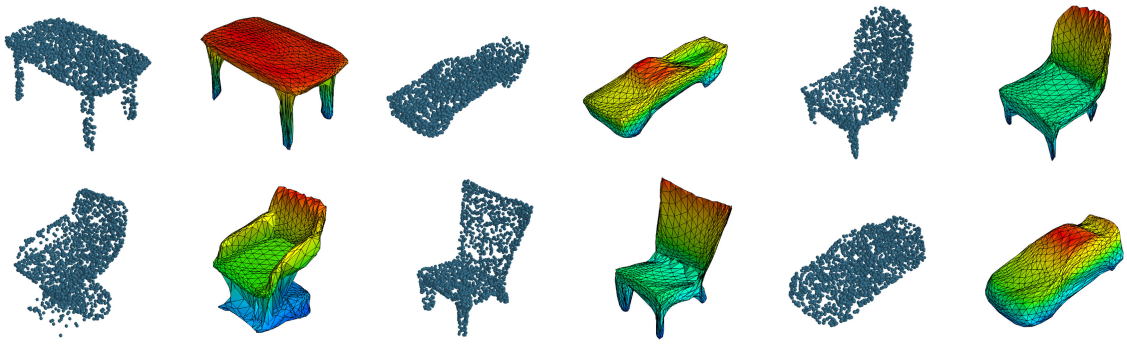

Fig. 5. 3D point clouds and their mesh representations produced by HyperCloud.

highest density around object edges and are missing points within the object structure. Modeling this density with a distribution that does not allow discontinuities is infeasible as per Theorem 1 [47].

Theorem 1. *There is no continuous invertible map between the 3-D ball and the 2-D sphere that respects the boundary.*

For modeling the surface of an object with a continuous, invertible map, one shall consider the topology of the object [31], [48], [49]. To learn a transformation that is continuous, invertible, and provides the results close to object boundary, one has to choose a prior that is topologically similar to the expected point cloud, *i.e.* has the same number of discontinuities.[1] Therefore, we construct a probability distribution on a sphere without compact support.

1. Continuous normalizing flows (FFJORD [31]) are able to approximate discontinuous density functions. This, however, remains insufficient to model high-quality 3D point clouds while generating continuous parametrization of object surfaces. Consequently, in our approach, we propose a density distribution without compact support and with a single discontinuity, which corresponds to the topology of 3D objects represented with point clouds.

4.1 Spherical Log-Normal Distribution on $\mathbb{R}^n$.

A probability distribution on a sphere in $\mathbb{R}^n$ can be constructed by using one-dimensional density distribution, which takes only positive real values $f : \mathbb{R}_+ \to \mathbb{R}_+$. In such a case, we can define spherical density distribution as

$$f_n : \mathbb{R}^n \ni x \to \frac{1}{\mathrm{vol}(S_{n-1})\|x\|^{n-1}} f(\|x\|), \tag{2}$$

where $\mathrm{vol}(S_{n-1})$ is a surface area of a n-dimensional unitary sphere and f is a one-dimensional density, which takes only positive real values. We use one-dimensional density distribution $f : \mathbb{R}_+ \to \mathbb{R}_+$ along the radius of unit sphere in all directions. In our model, we use a Log-normal distribution $f(r) = \frac{1}{r} \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(\log r - \mu)^2}{2\sigma^2})$ that is a continuous probability distribution of a random variable, whose logarithm is normally distributed and, hence, provides non-compact support.

4.2 Spherical Log-Normal Distribution in $\mathbb{R}^3$.

To develop an intuition behind the proposed distribution, we start with a simple visualization in $\mathbb{R}^2$. Fig. 8 shows the level sets and sample from Spherical Log-Normal distribution with different parameters σ. The spherical Log-Normal distribution does not have compact support and can therefore be used in a flow-based architecture. Furthermore, we can force the distribution to concentrate as close as possible to 2D sphere boundaries.

In $\mathbb{R}^3$, our Spherical Log-Normal distribution is defined as

$$f_3(x) = \frac{1}{2(2\pi)^{3/2}\sigma\|x\|^3} \exp\left(-\frac{(\log\|x\| - \mu)^2}{2\sigma^2}\right). \tag{3}$$

In order to use our distribution in a flow-based model, we need to compute its log-likelihood function

$$\log f_3(x) = -\log\left(2(2\pi)^{\frac{3}{2}}\right) - \log\sigma - 3\log\|x\| - \frac{(\log\|x\| - \mu)^2}{2\sigma^2}. \tag{4}$$

Finally, sampling elements from our Spherical Log-Normal distribution can be done by following a simple procedure. First sample r from one-dimensional Gaussian $N(0, 1)$ and then sample x from n-dimensional Gaussian $N(0, I)$. A sample form Spherical Log-Normal can be obtained by the following equation: $\exp(\mu + \sigma \cdot r) \cdot \frac{x}{\|x\|}$.

We avoid numerical instabilities of training by applying a straightforward strategy to find the correct values of σ parameter: we start with an arbitrarily large value of σ and reduce it linearly during the training.

4.3 Triangulation Trick

To model 3D object surfaces as meshes using the HyperCloud generative model, we need to investigate the relationship between point clouds and object surfaces. In principle, a point cloud representing a 3D object can be considered a set of samples located on the surface of the object with additive noise introduced by a registration device. We use our Spherical Log-Normal function to model this distribution with peak density around the object surfaces (in 2D, around circle edges, in 3D close to the surface of the sphere) and limited by the radius of the distribution. Once we obtain a parametrized distribution of a point cloud that models the object surface together with the registration noise, we can produce a mesh with a simple operation which we call the *triangulation trick*.

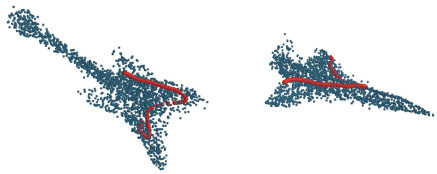

Fig. 6. Example of interpolation of the 3D object representation space in the target network. Our hypernetwork architecture allows us to work with a single object, represented as a distribution of points on a single 3D point cloud, hence we can browse the space of potential 3D objects by interpolating representation space in the target network, instead of doing so in the latent auto-encoder space, as typically done.

The triangulation trick involves transferring vertices of a sphere mesh through a target network in the same way as 3D points, as shown in Fig 3. Since the target network transforms a sample from a Spherical Log-Normal distribution into a 3D point cloud, when we feed it with a sphere triangulation, it outputs a mesh. In fact, when we substitute samples from the Spherical Log-Normal distribution with sphere vertices, we effectively assume minimal registration noise. Processing vertices by the target network pre-trained on point clouds allows us to directly generate denoised mesh representation of object surfaces and obtain a high-quality 3D object rendering. The generative character of our HyperCloud model enables the construction of the entire mesh by processing only vertices with a target network, without the need for information about the connections between them, as is done in traditional rendering methods.

Fig. 7 presents reconstructions obtained using the Gaussian and the Spherical Log-Normal distributions. We look at the cross-sections of the reconstructions to observe the main differences in how the input distribution is transformed into a final model by the target network. For the Gaussian distribution, its tails are transformed into object details, such as wingtips and airplane rear aileron. Therefore, we cannot claim that the peak density models the surface of the object, while its tails model the registration noise. For the Spherical Log-Normal, its distribution tails are spread along object surfaces, modeling the registration noise. This allows us to produce the final mesh through the triangulation trick, effectively denoising 3D mesh-based object representation and yielding high-quality results, as shown in Fig. 9.

5 HyperFlow: Hypernetwork and Continuous Normalizing Flows for generating 3D point clouds

In this section, we present our general framework for creating 3D point clouds, together with their mesh-based representations, dubbed HyperFlow. We show how HyperFlow leverages the Spherical Log-Normal to encompass both flow-based and conventional neural networks. Now we are ready to introduce the HyperFlow model that leverages a hypernetwork framework to train a Continuous Normalizing Flow [31] target network and generate 3D point clouds together with its mesh-based representation.

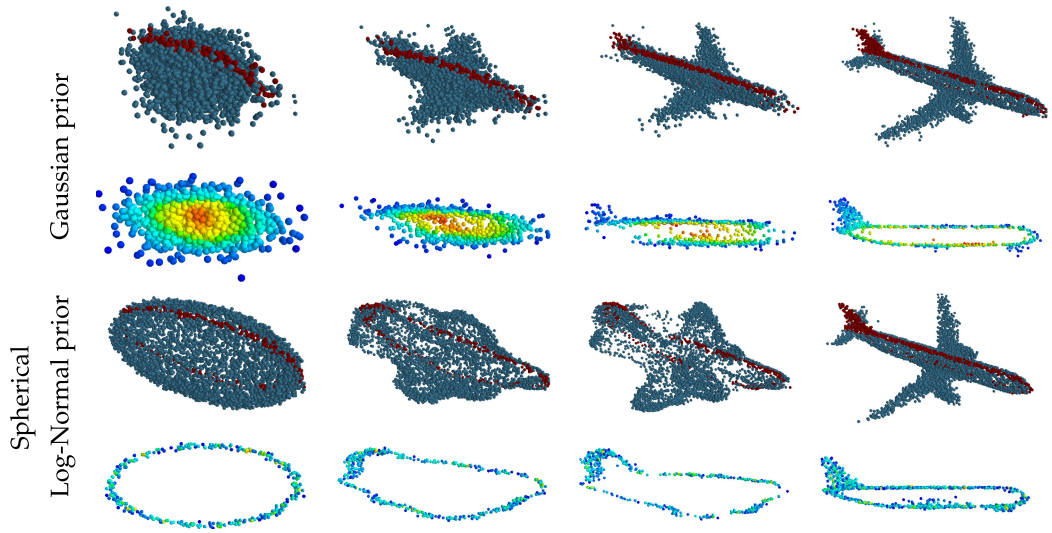

Fig. 7. We compare how the prior density is modified for the model with a Gaussian prior **(the upper two rows)** and Spherical Log-Normal **(the bottom two rows)**. In the first and the third row we show how the flow model transforms the original density into the target dataset. The second and the fourth row show the cross-sections along the plane depicted by red points. For the Gaussian distribution, target space points are not distributed evenly across the object (a central part of Gaussian distribution is transformed into the bottom of the plane, while its tails are used to model wingtips). For the Spherical Log-Normal, target space points are distributed evenly across the object, showcasing that our approach truly models the distribution of the points along object surfaces.

5.1 Continuous Normalizing Flow

Generative models are one of the fastest-growing areas of deep learning. Variational Autoencoders (VAE) [39] and Generative Adversarial Networks (GAN) [50] are the most popular approaches. However, yet another model has gained popularity – namely the Normalizing Flow (NF) [48]. A flow-based generative model is constructed by a sequence of invertible transformations. Unlike the other two methods mentioned previously, the model explicitly learns the data distribution, and therefore the loss function is simply the negative log-likelihood.

The Normalizing Flow (NF) [48] is able to model complex probability distributions. The normalizing flow transforms a simple prior distribution (usually the Gaussian one) $\mathcal{P}(Y)$ into a complex one (represented by data distribution X) by applying a sequence of invertible transformation functions: $f_1, \ldots, f_n : Y \rightarrow X$. By flowing through a chain of transformations $x = F(y) = f_n \circ f_{n-1} \circ \ldots \circ f_1(y)$, we obtain a probability distribution of the final target variable. Consequently, the probability density of the output variable is given by the change of variables formula

$$\log P(x) = \log P(y) - \sum_{k=1}^{n} \log \left| \det \frac{\partial f_k}{\partial y_{k-1}} \right|, \tag{5}$$

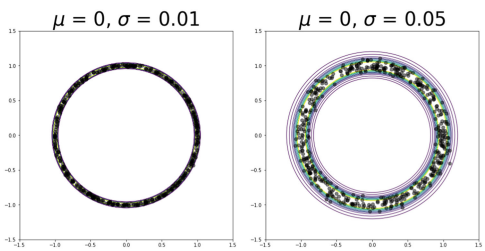

Fig. 8. Level sets and samples from the Spherical Log-Normal distribution with different parameters σ and $\mu = 0$. Since the Spherical Log-Normal distribution does not have compact support, it can be used in flow-based architectures.

where y can be computed from x using the inverse flow: $y = f_1^{-1} \circ f_2^{-1} \circ \ldots \circ f_n^{-1}(x)$. In such a framework, both the inverse map and the determinant of the Jacobian should be computable.

The continuous normalizing flow [30] is a modification of the above approach, where instead of a discrete sequence of iterations, we allow the transformation to be defined by a solution to a differential equation $\frac{\partial y(t)}{\partial t} = f(y(t), t)$, where f is a neural network that has an unrestricted architecture. The Continuous Normalizing Flow (CNF) $F_\theta : Y \ni y \rightarrow x \in X$ is a solution of differential equations with the initial value problem $y(t_0) = x$, $\frac{\partial y(t)}{\partial t} = f_\theta(y(t), t)$. In such a case we have

$$\begin{aligned} F(y) = F_\theta(y(t_0)) &= y(t_0) + \int_{t_0}^{t_1} f_\theta(y(t), t) dt, \\ F_\theta^{-1}(x) &= x + \int_{t_1}^{t_0} f_\theta(y(t), t) dt, \end{aligned} \tag{6}$$

where f_θ defines the continuous-time dynamics of the flow F_θ and $y(t_1) = x$.

The log probability cost function with prior distribution and density g can be computed by

$$\mathcal{C}_F(X; g, \theta) = \sum_{x \in X} \log g(F_\theta^{-1}(x)) - \int_{t_0}^{t_1} \operatorname{Tr}\left(\frac{\partial f_\theta}{\partial y(t)} \right) dt. \tag{7}$$

In PointFlow [8] the authors show that the CNF can be used for modeling 3D objects. Instead of directly parametrizing the distribution of points in a shape (fixed size 3D point cloud), PointFlow models this distribution as an invertible parameterized transformation of 3D points from a prior distribution (e.g., a 3D Gaussian). Intuitively, under this model, generating points for a given shape involves sampling points from a generic Gaussian prior and then moving them according to this parameterized transformation to their new location in the target shape.

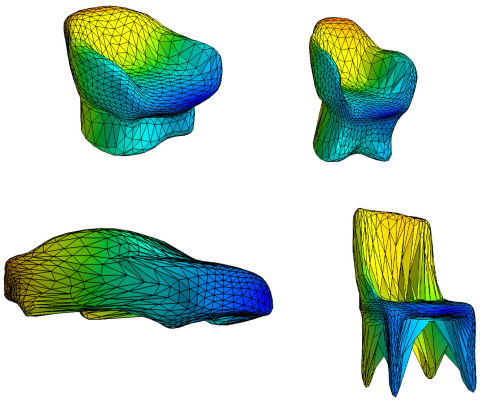

Fig. 9. Mesh representations generated by our HyperFlow method. Contrary to the existing methods that return point cloud representations sparsely distributed in 3D space, our approach allows creating a continuous 3D object representation in the form of meshes.

5.2 HyperFlow

In this section, we present details of our novel model dubbed HyperFlow, which encompasses and extends prior works by training continuous normalizing flow modules to model 3D point cloud distributions with a hypernetwork framework. Our model is an extension of HyperCloud, which uses flow architecture as a target network.

We adapt the log-likelihood cost function to a hypernetwork framework. We therefore introduce our HyperFlow model that consists of two main parts. The first one is a hypernetwork that outputs weights of another neural network. The second one is a target network that models the distribution of elements on the surface of a 3D object. Using autoencoder terminology, we define three elements: an encoder, a decoder, and a prior distribution. The encoder $\mathcal{E}_\phi : \mathcal{X} \rightarrow \mathcal{Z}$ can reduce data dimensionality by mapping it to a lower-dimensional latent space $\mathcal{Z} \subseteq \mathbb{R}^D$. We follow [51] and use a simple permutation-invariant encoder to predict $\mathcal{E}_\phi$.

We use $\mathcal{P}_\mathcal{Z}$ over shape representations proposed by PointFlow [8]. The assumed probability distribution on the latent pace can be more complex than the commonly used $N(0, I)$ and not given in an explicit form. In such a framework, we use an additional continuous normalizing flow G_ψ, which transfers the latent space into a Gaussian prior. Finally, we propose to use a decoder that returns weights of the target network $\mathcal{D}_\theta : \mathcal{Z} \ni z \rightarrow \Theta$, instead of 3D points, as done in [8], [9]. The resulting hypernetwork contains an encoder $\mathcal{E}_\phi$, a decoder $\mathcal{D}_\theta$ and a flow G_ψ.

The hypernetwork takes as an input a point cloud $X \subset \mathbb{R}^3$ and returns weights Θ to f_Θ that define the continuous-time dynamics of the flow F_Θ. The CNF takes an element from the prior distribution $\mathcal{P}$ and transfers it to an element on the surface of the object. In our work, we use a Free-form Jacobian of Reversible Dynamics (FFJORD) [31] and transformation between the Spherical Log-Normal distribution and the 3D object. As presented in Section 4, this choice of distribution function allows one to create a continuous mesh representation with the triangulation trick.

HyperFlow is trained by optimizing the following objective function

$$l_{HF}(X; \mathcal{E}, \mathcal{D}) = \mathcal{C}_F(X; f_3, \mathcal{D}(\mathcal{E}(X))) + Reg(\mathcal{E}(X, P). \quad (8)$$

$\mathcal{C}_F$ is CNF log probability cost function given by eq. (6) with the Spherical Log-Normal density f_3 as a prior. Instead of direct pasteurization θ, we use a hyper model $\mathcal{D}$ that predicts parameters of the target function inside CNF. Reg is a regularization term responsible for forcing latent representation $\mathcal{E}(X)$ to an assumed prior P. The regularization can be performed via KL divergence, adversarial training, or by incorporating an additional CNF as in the PointFlow approach. We utilize the PointFlow method for the experimental part to keep the methodological consistency between key reference approaches.

5.3 Relation to the Existing Models

Compared to previous models like PointFlow, we propose to use hyper-networks instead of embedding-based conditioning. Thanks to that approach, the target model responsible for generating a shape does not share parameters across all possible shapes, but each of the shapes receives a dedicated target model. As a consequence, the number of parameters of the target model is significantly lower, which is especially important for continuous flows, where a large number of forward passes is executed by an ODE solver. We receive comparable reconstruction results to PointFlow (in terms of EMD) with a significant reduction of parameters from over $0.5M$ parameters (PointFlow uses $512 - 512 - 512$ configuration of the target flow) to over 2500 parameters of the target flow (we use $64 - 16 - 64$ configuration).

We also think that the selection of base distribution for our model is a valuable contribution. Compared to PointFlow, we preserve the probabilistic features of the model and, thanks to proper base distribution selection and the triangular trick, we are able to generate meshes, while PointFlow fails to do that (see Table 3). On the other hand, AtlasNet receives better reconstruction results, but it is not trained in a probabilistic framework, it has no generative capabilities, and it is impossible to evaluate the importance of each of the points on the surface (no likelihood measure).

6 Experiments

This section describes the experimental results of the proposed generative models in various tasks, including 3D points cloud and mesh generation, learning representations, and interpolation.

6.1 Metrics

Following the methodology for evaluating generative fidelity and diversification among the samples provided in [51] and [8], we have applied the following criteria for evaluation: Jensen-Shannon Divergence, Coverage, Minimum Matching Distance 1-nearest Neighbor Accuracy.

Jensen-Shannon Divergence (JSD): a measure of the distance between two empirical distributions P and Q, defined as

$$JSD(P\|Q) = \frac{KL(P\|M) + KL(Q\|M)}{2}, \text{ where } M = \frac{P+Q}{2}.$$

Coverage (COV): a measure of generative capabilities in terms of richness of generated samples from the model. For two point cloud sets $X_1, X_2 \subset \mathbb{R}$ the coverage is defined as a

TABLE 1
Generation Results

Category	Methods	JSD	MMD		COV		1-NNA	
			CD	EMD	CD	EMD	CD	EMD
Airplane	r-GAN	7.44	0.261	5.47	42.72	18.02	93.58	99.51
	l-GAN (CD)	4.62	0.239	4.27	43.21	21.23	86.30	97.28
	l-GAN (EMD)	**3.61**	0.269	3.29	**47.90**	50.62	87.65	85.68
	PC-GAN	4.63	0.287	3.57	36.46	40.94	94.35	92.32
	PointFlow	4.92	**0.217**	3.24	46.91	48.40	**75.68**	**75.06**
	HyperCloud (ours)	4.84	0.266	3.28	39.75	43.70	93.80	88.95
	HyperFlow (ours)	5.39	0.226	**3.16**	46.66	**51.60**	80.12	76.42
	Training set	6.61	0.226	3.08	42.72	49.14	70.62	67.53
Chair	r-GAN	11.5	2.57	12.8	33.99	9.97	71.75	99.47
	l-GAN (CD)	4.59	2.46	8.91	41.39	25.68	64.43	85.27
	l-GAN (EMD)	2.27	2.61	7.85	40.79	41.69	64.73	65.56
	PC-GAN	3.90	2.75	8.20	36.50	38.98	76.03	78.37
	PointFlow	1.74	2.42	7.87	**46.83**	**46.98**	**60.88**	**59.89**
	HyperCloud (ours)	2.73	2.56	**7.84**	41.54	46.67	68.20	68.80
	HyperFlow (ours)	**1.50**	**2.30**	8.01	44.71	46.37	63.36	70.24
	Training set	1.50	1.92	7.38	57.25	55.44	59.67	58.46
Car	r-GAN	12.8	1.27	8.74	15.06	9.38	97.87	99.86
	l-GAN (CD)	4.43	1.55	6.25	38.64	18.47	63.07	88.07
	l-GAN (EMD)	2.21	1.48	5.43	39.20	39.77	69.74	68.32
	PC-GAN	5.85	1.12	5.83	23.56	30.29	92.19	90.87
	PointFlow	**0.87**	**0.91**	**5.22**	44.03	46.59	**60.65**	**62.36**
	HyperCloud (ours)	3.09	1.07	5.38	40.05	40.05	84.65	77.27
	HyperFlow (ours)	1.07	1.14	5.30	**45.74**	**47.44**	64.63	62.78
	Train set	0.86	1.03	5.33	48.30	51.42	57.39	53.27

MMD-CD scores are multiplied by 10^3; MMD-EMD scores and JSDs are multiplied by 10^2.

fraction of points in X_2 that are the nearest neighbor to some points in X_1 in the given metric.

Minimum Matching Distance (MMD): Since COV only takes the closest point clouds into account and does not depend on the distance between the matchings , an additional metric has been introduced. For point, cloud sets X_1, X_2, MMD is a measure of similarity between point clouds in X_1 to those in X_2.

1-Nearest Neighbor Accuracy (1-NNA) is a testing procedure characteristic for evaluating GANs. We have considered two sets: set S_g composed of generated point clouds and test (reference) point clouds, S_r. We have picked some generated point cloud X from S_g and have found the corresponding nearest neighbor in $S_{-X} = S_r \bigcup S_g - \{X\}$, the set that aggregates both training and sampled shapes excluding the considered point cloud X. The 1-NNA is the leave-one-out accuracy of the 1-NN classifier

$$1 - NNA = \frac{\sum_{X \in S_g}[N_X \in S_g] + \sum_{Y \in S_r}[N_Y \in S_r]}{|S_g| + |S_r|}.$$

For each sample, the 1-NN classifier classifies it as coming from S_r or S_g according to the label of its nearest sample. The perfect situation occurs when the classifier is unable to distinguish between real and generated point clouds, which means that the value of the criterion is close to 50%.

6.2 Generation on 3D Point Clouds

We examine the generative capabilities of the provided HyperCloud and HyperFlow in comparison to the existing reference approaches. In this experiment, we follow the methodology provided in [8]. For HyperCloud, we have utilized the hypernetwork architecture trained with EMD reconstruction loss together with the continuous flow on latent representation instead of simple KLD regularization. We have compared the results with the existing solutions: raw-GAN [51], latent-GAN [51], PC-GAN [52] and PointFlow [8]. We have trained each model using point clouds from one of the three categories in the ShapeNet dataset: *airplane*, *chair*, and *car*. We have followed the exact evaluation pipeline provided in [8].

The results are presented in Table 1. HyperCloud obtains comparable results to the other models that utilize EMD reconstruction loss with the advantage of sampling an arbitrary number of points. The model was outperformed by PointFlow and HyperFlow that do not utilize EMD as reconstruction loss and use a more complex continuous flow for a point-level generation. However, HyperFlow is capable of generating meshes via the triangulation trick and is more effective in terms of training time and memory consumption. Fig. 10 displays a comparison between our HyperFlow method and the competing PointFlow. We have evaluated the architectures used in the previous sections that have obtained the best quantitative results for a fair comparison. The models have been trained on the *car* dataset. Our HyperFlow approach has led to a significant reduction in both training time and memory footprint due to a more compact flow architecture enabled by a hypernetwork framework.

6.3 Generation of 3D Meshes

The main advantage of our method compared to reference solutions is the ability to generate both 3D point clouds and meshes without any post-processing stage. In Fig. 5, we present a point cloud as well as a mesh representation generated by the same model. Thanks to using a uniform distribution on the 3D ball, we can easily construct a mesh. All elements from the ball are transformed into a 3D object. In consequence, the unit sphere is transformed into the surface of the object. As it was mentioned, we can produce meshes without a secondary meshing procedure. It is obtained by

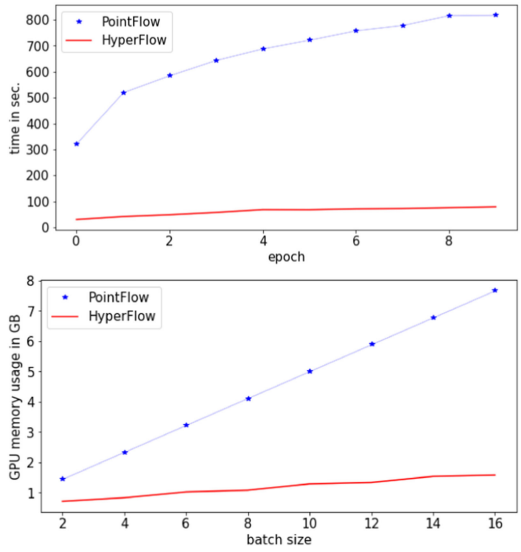

Fig. 10. Comparison of training times and GPU memory used by PointFlow and HyperFlow. Our HyperFlow method offers over an order of magnitude decrease in both training time and memory.

propagating the triangulation of the 3D sphere through the target network, see Fig 3.

In the case of a Gaussian prior, we can use a similar procedure, but it is nontrivial to select the optimal sphere radius, which will be used by the generation of a mesh (contrary to our hyper models, in PointFlow there is no default for radius R). If the chosen radius is too small, the constructed mesh lies *inside* the point cloud, and consequently, we lose small outlying elements of the object, e.g., chair legs. On the other hand, if the chosen sphere radius is large, some small elements of the 3D object will be merged, e.g., four legs of a chair will merge into one.

For the evaluation of the quality of mesh grid representation, we propose the following experiment. Instead of sampling the points from the assumed prior distribution, we sample them from a given surface (sphere with the assumed radius). Next, we calculate the standard quality measures of generated point clouds considered in the previous experiment. Since all models except PointFlow listed in Table 1 work only on a fixed number of points, we compare our results only with PointFlow.

As mentioned above, we can use the PointFlow model to produce mesh representation similarly by feeding the target network by triangulation on a sphere. In our experiment, consistently with the standard used for hypothesis testing, we have used 95%, 98%, and 99% confidence spheres for 3D Gaussian distribution, see Table 3. As we can see, the default Gaussian prior is not suitable for producing a continuous representation of the boundary. As can be seen in Table 3, PointFlow that uses a Gaussian distribution as a prior provides results inferior to HyperCloud and HyperFlow, while the HyperFlow method offers the best performance, thanks to using Spherical Log-Normal as a prior instead of a compact support distribution function as in HyperCloud.

TABLE 3
The Values of Quality Measures of 3D Representations Obtained by Sampling From a Sphere with a Given Radius R for *Airplane*, *Chair* and *Car* Shapes

Sphere R	JSD	MMD		COV	
		CD	EMD	CD	EMD
Airplane					
PointFlow					
R=2.795	22.26	0.49	6.65	44.69	20.74
R=3.136	26.46	0.60	6.89	39.50	19.01
R=3.368	29.65	0.68	6.84	40.49	16.79
HyperCloud (ours)					
R=1	9.51	0.45	5.29	30.60	28.88
HyperFlow (ours)					
R=1	**6.55**	**0.38**	**3.65**	**40.49**	**48.64**
Chair					
PointFlow					
R=2.795	19.28	4.28	13.38	36.85	20.84
R=3.136	22.52	4.89	14.47	32.47	17.22
R=3.368	24.68	5.36	14.97	31.41	17.06
HyperCloud (ours)					
R=1	4.32	**2.81**	9.32	40.33	40.63
HyperFlow (ours)					
R=1	**4.26**	3.33	**8.27**	**41.99**	**45.32**
Car					
PointFlow					
R=2.795	16.59	1.6	8.00	20.17	17.04
R=3.136	20.21	1.75	7.80	21.59	17.32
R=3.368	24.10	1.96	8.35	18.75	17.04
HyperCloud (ours)					
R=1	**5.20**	**1.11**	6.54	**37.21**	28.40
HyperFlow (ours)					
R=1	5.77	1.39	**5.91**	28.40	**37.21**

It can be seen that HyperCloud and HyperFlow preserve the good quality of sampled point clouds, while PointFlow has difficulties in obtaining good quality representations from the sphere.

6.4 Unsupervised Representation Learning

In this experiment, we have evaluated the quality of latent space representation of our models. We have followed the experimental settings from previous works [8], [51] and have trained our model using the full ShapeNet dataset. Next, we have evaluated the quality of latent representation by training a linear SVM classifier on top of it using ModelNet10 and ModelNet40 datasets. In this experiment we have also considered 3DGAN [17], AtlasNet [36], FoldingNet [14], Generative PointNet (GPN) [27] and Generative VoxelNet (GVoxelNet) [26] as reference methods. We have provided the results of empirical evaluation of our model in Table 2. Our models have achieved an accuracy that is comparable to the results achieved by the original version of l-GAN, but they have been was worse than the results achieved by PointFlow and l-GAN trained with a new setting. However, in our experiments, we have not used

TABLE 2
Unsupervised Feature Learning

	3DGAN	FoldingNet	l-GAN (EMD)	l-GAN (CD)	GPN	GVoxelNet	AtlasNet	l-GAN-2 (EMD)	l-GAN-2 (CD)	PointFlow	HyperFlow	HyperCloud
MN10 (%)	91.0	94.4	**95.4**	**95.4**	93.7	92.4	91.9	92.8	92.2	**93.7**	91.0	90.5
MN40 (%)	83.3	**88.4**	84.0	84.5	-	-	86.6	87.0	86.7	86.8	84.7	84.9

Models are first trained on ShapeNet to learn shape representations, which are then evaluated on ModelNet10 (MN10) and ModelNet40 (MN40) by comparing the accuracy of off-the-shelf SVMs trained using the learned representations. l-GAN-2 was trained and evaluated using PointFlow experimental settings.

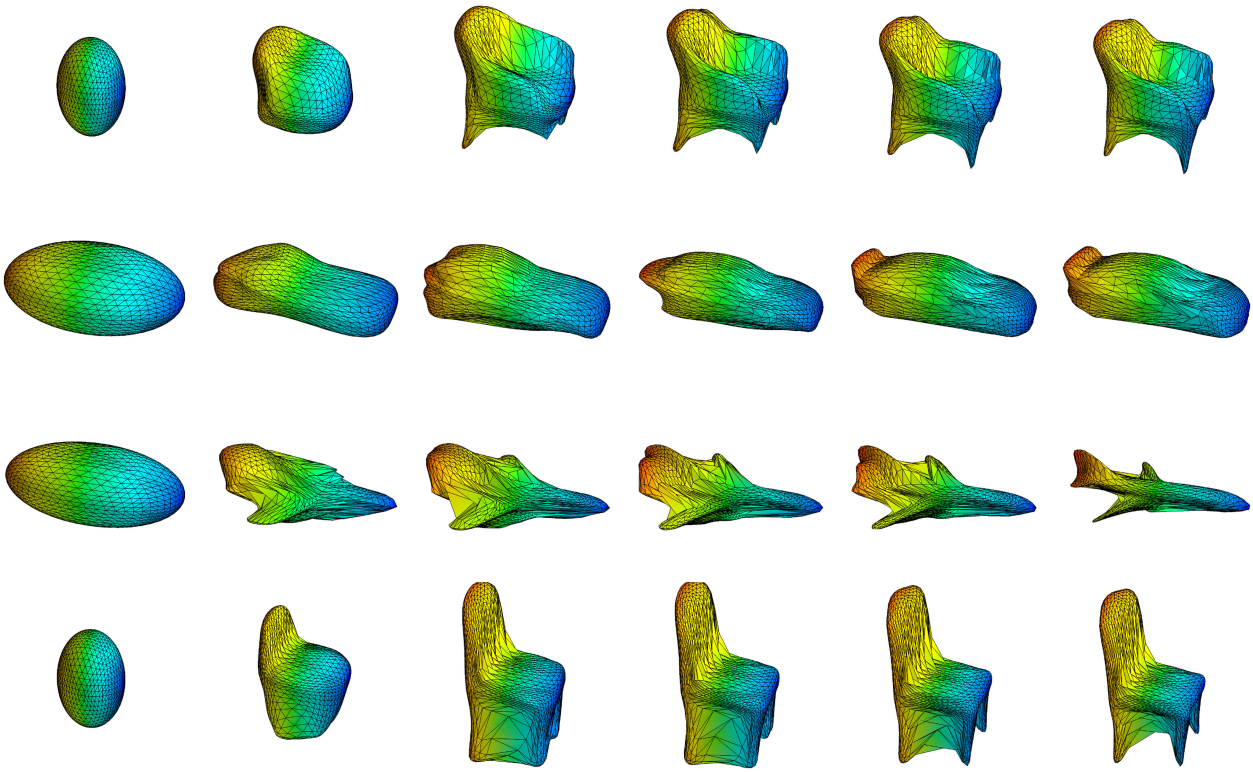

Fig. 11. We show how the triangulation on the sphere is transformed into a mesh of the object. Thanks to the so-called triangulation trick, we obtain object meshes. Since we use a CNF as a target network, we can visualize a continuous transformation between a uniform sphere and the surfaces of objects.

preprocessed ModelNet datasets in the same pipeline as in PointFlow, but in the way recommended in [51].

6.5 Interpolation

In our hyper model, we can construct two types of interpolation. Since we have two different prior distributions: the Gaussian one in the hypernetwork architecture (latent of auto-encoder) and the uniform distribution on the unit sphere in the target network, see Fig. 2. First of all, we can take two 3D objects and obtain a smooth transition between them, see Fig. 4. For each point cloud, we can generate mesh representation. Therefore we can also produce interpolation between the meshes.

Our hypernetwork architecture allows us to work with a single object, represented as a distribution of points on a single 3D point cloud. One interesting consequence of this feature is that we can browse the space of potential 3D objects by interpolating representation space in the target network instead of doing so in the latent auto-encoder space, as is typically done. Fig. 6 shows an example of such interpolation.

6.6 Flow Transformations of Meshes

In this part of the experiment, we show the idea of direct mesh generation via the triangulation trick. We show how the locations of points are shifted during the integration process, while preserving the connections between points.

In our HyperFlow model, generating points for a given shape involves sampling points from a Spherical Log-Normal prior and then moving them according to this parameterized transformation to their new location in the target shape. In Fig. 11 we present such a transformation. Since our model allows producing meshes, we show how a mesh from a uniform sphere is transformed into a mesh on the object.

6.7 Reconstruction Results

In this subsection, we evaluate how well our model can learn the underlying distribution of points. We present reconstruction results for ShapeNet (*airplane*, *car*, *chair*). In this experiment, we compare HyperFlow with the current state-of-the-art AtlasNet [36] where the prior shape is either a sphere or a set of patches. Furthermore, we also make a comparison with l-GAN [53] and PointFlow [8]. We follow the experiment set-up in PointFlow and report the performance in both CD and EMD in Table 4. Since these two metrics depend on the scale of point clouds, we also report the upper bound in the "oracle" column. The upper bound is produced by computing the error between two different point clouds with the same number of points sampled from the same ground truth meshes.

We observe that HyperFlow achieves comparable results (in terms of EMD) to PointFlow that was also trained via

TABLE 4
Reconstruction Results

Dataset	Metric	l-GAN		AtlasNet		PointFlow	HyperFlow	Oracle
		CD	EMD	Sphere	Patches			
Airplane	CD	1.020	1.196	1.002	**0.969**	1.208	1.603	0.837
	EMD	4.089	**2.577**	2.672	2.612	2.757	2.763	2.062
Chair	CD	9.279	11.21	**6.564**	6.693	10.120	13.761	3.201
	EMD	8.235	6.053	5.790	**5.509**	6.738	6.724	3.297
Car	CD	5.802	6.486	**5.392**	5.441	6.531	7.353	3.904
	EMD	5.790	4.780	4.587	**4.570**	5.126	4.837	3.251

CD is multiplied by 10^4 and EMD is multiplied by 10^2.

NLL optimization. The remaining approaches were trained by optimizing reconstruction measures, therefore, they perform better in terms of the considered criteria.

7 Conclusions

In this work, we present a novel approach to representing point clouds of 3D objects with parameters of target networks trained by a hypernetwork. More specifically, we are able to build variable size representations of point clouds not only when they are input into the model but also when they are returned as an output. Our proposed method not only gives high-quality 3D object representations, but also allows for the creation of realistic 3D meshes. Finally, the framework presented in this work encompasses many existing approaches, including flow-based models, and hence it can be used in a multitude of real-life applications, including LIDAR data reconstruction and autonomous driving, and it can also open new research areas related to generative models.

References

[1] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D object detection from point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7652–7660.

[2] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Trans. Automat. Sci. Eng.*, vol. 12, no. 2, pp. 398–409, Apr. 2015.

[3] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 945–953.

[4] Z. Wu *et al.*, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1912–1920.

[5] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3391–3401.

[6] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 652–660.

[7] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5099–5108.

[8] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan, "Pointflow: 3D point cloud generation with continuous normalizing flows," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 4541–4550.

[9] M. Stypułkowski, M. Zamorski, M. Zieba, and J. Chorowski, "Conditional invertible flow for point cloud generation," 2019, *arXiv:1910.07344*.

[10] M. Zamorski *et al.*, "Adversarial autoencoders for compact representations of 3D point clouds," *Comput. Vis. Image Understanding*, Elsevier, vol. 193, 2020, Art. no. 102921.

[11] P. Spurek, S. Winczowski, J. Tabor, M. Zamorski, M. Zieba, and T. Trzciński, "Hypernetwork approach to generating point clouds," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 9099–9108.

[12] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *Proc. 5th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=rkpACe1lx

[13] S. Klocek, L. Maziarka, M. Wołczyk, J. Tabor, J. Nowak, and M. Śmieja, "Hypernetwork functional image representation," in *Proc. Int. Conf. Artif. Neural Netw.*, 2019, pp. 496–510.

[14] Y. Yang, C. Feng, Y. Shen, and D. Tian, "Foldingnet: Point cloud auto-encoder via deep grid deformation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 206–215.

[15] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2012.

[16] D. Maturana and S. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 922–928.

[17] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 82–90.

[18] M. Gadelha, R. Wang, and S. Maji, "Multiresolution tree networks for 3D point cloud processing," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 103–118.

[19] Y. Sun, Y. Wang, Z. Liu, J. E. Siegel, and S. E. Sarma, "PointGrow: Autoregressively learned point cloud generation with self-attention," early access, May 14, 2020, doi: 10.1109/WACV45572.2020.9093430.

[20] A. Van den Oord *et al.*, "Conditional image generation with pixelCNN decoders," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4790–4798.

[21] J. Xie, Y. Lu, S.-C. Zhu, and Y. Wu, "A theory of generative ConvNet," in *Proc. Int. Conf. Mach. Learn.* , 2016, pp. 2635–2644.

[22] J. Xie, S.-C. Zhu, and Y. Nian Wu, "Synthesizing dynamic patterns by spatial-temporal generative ConvNet," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7093–7101.

[23] J. Xie, S.-C. Zhu, and Y. N. Wu, "Learning energy-based spatial-temporal generative ConvNets for dynamic patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 2, pp. 516–531, Feb. 2021.

[24] T. Han, E. Nijkamp, X. Fang, M. Hill, S.-C. Zhu, and Y. N. Wu, "Divergence triangle for joint training of generator model, energy-based model, and inferential model," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 8670–8679.

[25] J. Xie, Z. Zheng, R. Gao, W. Wang, S.-C. Zhu, and Y. N. Wu, "Learning descriptor networks for 3D shape synthesis and analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8629–8638.

[26] J. Xie, Z. Zheng, R. Gao, W. Wang, S. -C. Zhu, and Y. N. Wu, "Generative VoxelNet: Learning energy-based models for 3D shape synthesis and analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Dec. 15, 2020, doi: 10.1109/TPAMI.2020.3045010.

[27] J. Xie, Y. Xu, Z. Zheng, S.-C. Zhu, and Y. N. Wu, "Generative PointNet: Deep energy-based learning on unordered point sets for 3D generation, reconstruction and classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14976–14985.

[28] W. Yifan, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung, "Patch-based progressive 3D point set upsampling," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5958–5967.

[29] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-Net: Point cloud upsampling network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2790–2799.

[30] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 6571–6583.

[31] W. Grathwohl, R. T. Chen, J. Betterncourt, I. Sutskever, and D. Duvenaud, "FFJORD: Free-form continuous dynamics for scalable reversible generative models," in *Proc. Int. Conf. Learn. Representations*, 2018.

[32] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2Mesh: Generating 3D mesh models from single RGB images," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 52–67.

[33] J. Bednarik, S. Parashar, E. Gundogdu, M. Salzmann, and P. Fua, "Shape reconstruction by learning differentiable surface representations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 4716–4725.

[34] Z. Deng, J. Bednařík, M. Salzmann, and P. Fua, "Better patch stitching for parametric surface reconstruction," in *Proc. Int. Conf. 3D Vis.*, 2020, pp. 593–602.

[35] P. Spurek, S. Winczowski, M. Zieba, T. Trzciński, and K. Kania, "Modeling 3D surface manifolds with a locally conditioned atlas," 2021, *arXiv:2102.05984*.

[36] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "A papier-mâché approach to learning 3D surface generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 216–224.

[37] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *Int. J. Comput. Vis.*, vol. 40, no. 2, pp. 99–121, 2000.

[38] M.-P. Tran, "3D contour closing: A local operator based on chamfer distance transformation," 2013. [Online]. Available: https://hal.archives-ouvertes.fr/file/index/docid/803416/filename/cclose_tran.pdf

[39] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. Int. Conf. Learn. Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6114

[40] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, "Wasserstein auto-encoders," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

[41] S. Knop, P. Spurek, J. Tabor, I. Podolak, M. Mazur, and S. Jastrzebski, "Cramer-wold auto-encoder," *J. Mach. Learn. Research*, vol. 21, no. 164, pp. 1–28, 2020. [Online]. Available: http://jmlr.org/papers/v21/19-560.html
[42] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, 1951.
[43] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," 2015, *arXiv:1511.05644*.
[44] J. Tomczak and M. Welling, "VAE with a VampPrior," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2018, pp. 1214–1223.
[45] R. v. d. Berg, L. Hasenclever, J. M. Tomczak, and M. Welling, "Sylvester normalizing flows for variational inference," in *Proc. 34th Conf. Uncertainty Artif. Intell.*, 2018, pp. 393–402.
[46] A.-S. Sheikh, K. Rasul, A. Merentitis, and U. Bergmann, "Stochastic maximum likelihood optimization via hypernetworks," *CoRR*, vol. abs/1712.01141, 2017. [Online]. Available: http://arxiv.org/abs/1712.01141
[47] A. Dill, C.-L. Li, S. Ge, and E. Kang, "Getting topology and point cloud generation to mesh," *CoRR*, abs/1912.03787, 2019.
[48] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1530–1538.
[49] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen, "Invertible residual networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 573–582.
[50] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
[51] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3D point clouds," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 40–49.
[52] C.-L. Li, M. Zaheer, Y. Zhang, B. Poczos, and R. Salakhutdinov, "Point cloud GAN," 2018, *arXiv:1810.05795*.
[53] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3D point clouds," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 40–49.

Przemysław Spurek received the master's degree in mathematics and the PhD degree in computer science from the Jagiellonian University, Krakow, Poland, in 2009 and 2014, respectively. He is currently an assistant professor with the Institute of Computer Science, Jagiellonian University.

Maciej Zieba received the master's degree in computer science from the Blekinge Institute of Technology, Sweden, and the master's degree in economics and the PhD degree in computer science from the Wroclaw University of Science and Technology. He is currently an AI researcher with Tooploox and an associate professor with the Wroclaw University of Science and Technology. He was the co-author of a variable number of research papers published in significant journals and presented on the top ML conferences including NeurIPS, ICLR, and ICML. His research interests include deep learning, especially generative models, and representation learning.

Jacek Tabor received the master's and PhD degrees in mathematics from the Jagiellonian University, Krakow, Poland, in 1997 and 2000, respectively. From 1997–1998 he was on Fulbright Scholarship with the SUNY at Buffalo. He is currently a professor with the Institute of Computer Science, Jagiellonian University.

Tomasz Trzcinski (Senior Member, IEEE) received the MSc degree in research on information and communication technologies from Universitat Politècnica de Catalunya, the MSc degree in electronics engineering from Politecnico di Torino in 2010, the PhD dgree in computer vision from École Polytechnique Fédérale de Lausanne in 2014, and the DSc degree (habilitation) from the Warsaw University of Technology in 2020. Since 2015, he has been an associate professor with the Warsaw University of Technology, where he leads a Computer Vision Lab, and an assistant professor with the Jagiellonian University of Cracow. In 2017, he was a visiting scholar with Stanford University and in 2019, with Nanyang Technological University. Previously, he was with Google in 2013, Qualcomm in 2012, and Telefónica in 2010. He is currently an associate editor for *IEEE Access* and *MDPI Electronics*. He is an expert of National Science Centre and Foundation for polish science. He is also a chief scientist with Tooploox and a co-founder of Comixify, a technology startup focused on using machine learning algorithms for video editing.