Deep Manifold Prior

Matheus Gadelha   Rui Wang   Subhransu Maji
{mgadelha, ruiwang, smaji}@cs.umass.edu

University of Massachusetts, Amherst

Abstract. We present a prior for manifold structured data, such as surfaces of 3D shapes, where deep neural networks are adopted to reconstruct a target shape using gradient descent starting from a random initialization. We show that surfaces generated this way are smooth, with limiting behavior characterized by Gaussian processes, and we mathematically derive such properties for fully-connected as well as convolutional networks. We demonstrate our method in a variety of manifold reconstruction applications, such as point cloud denoising and interpolation, achieving considerably better results against competitive baselines while requiring no training data. We also show that when training data is available, our method allows developing alternate parametrizations of surfaces under the framework of AtlasNet [14], leading to a compact network architecture and better reconstruction results on standard image to shape reconstruction benchmarks.

1 Introduction

In recent years a variety of approaches have been proposed to generate manifold data such as surfaces of 3D shapes using deep networks. The goal of this work is to characterize how the choice of the network architecture impacts the properties of the resulting surfaces. We present and analyze a *deep manifold prior*, an approach to represent a manifold as a collection of transformations (atlas) of an Euclidean space parameterized using deep networks (Section 3). We show that random networks induce smooth surfaces whose limiting behavior can be understood in terms of a Gaussian process (GP) [6, 23, 39]. We analyze how the different network architectures affect the distribution of position, normals and curvature of surfaces (Section 4). We also derive the properties of implicit surfaces induced by the level-set of a scalar field $\{f(x) = c\}$ parameterized using a deep network.

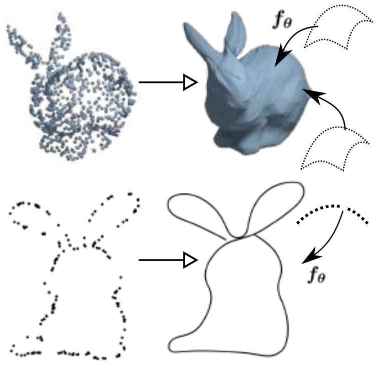

Fig. 1: **Deep manifold prior**. Points interpolated by using deep networks to map points in a 2D grid (top) and 1D grid (bottom) to the target shape (a 3D surface and a 2D curve respectively). The networks are randomly initialized and trained to minimize the Chamfer distance to the target.

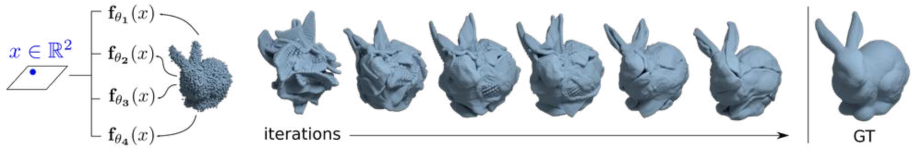
$x \in \mathbb{R}^2$
$\mathbf{f}_{\theta_1}(x)$
$\mathbf{f}_{\theta_2}(x)$
$\mathbf{f}_{\theta_3}(x)$
$\mathbf{f}_{\theta_4}(x)$
iterations
GT

Fig. 2: **Manifold reconstruction pipeline.** Manifold parametrizations are encoded by neural networks ($\mathbf{f}_{\theta_\mathbf{i}}$) and trained to minimize the reconstruction error with respect to the noisy target (left). Prior induced by the neural networks makes the generated surface much closer to the ground-truth (right), without ever seeing any additional training data.

As a concrete application we study the problem of interpolating and denoising point clouds sampled from contours or surfaces of shapes, as seen in Figures 1 and 2. The manifold parametrization allows us to efficiently sample point clouds, which can be combined with a Chamfer metric to measure a reconstruction error with respect to the sampled data. We show that smooth surfaces are obtained when the parameters of the networks are learned to minimize the reconstruction error starting from a *random initialization* (Figure 2). The approach is also effective for the level-set formulation, where the objective is to learn a deep network that correctly classifies points as *inside* or *outside* the surface. However, an advantage of the explicit parametrization is that it does not require the notion of what is inside. In addition we introduce a *regularization* that reduces self-intersections, overlaps, and distortion of the parametrization, which is desirable for applications such as texture mapping (Section 3). Our approach requires *no* prior learning, works across a range of 3D shapes, and outperforms strong baselines for point cloud denoising, such as Screened Poisson Surface Reconstruction (SPSR) and Robust Implicit Moving Least Squares (RIMLS). It is also more lightweight than approaches that operate on volumetric representations of 3D shapes (Section 5).

Our analysis sheds lights on the impressive performance of several recently proposed architectures for 3D surface generation, such as MRTNet [12], AtlasNet [14], FoldingNet [42], and Pixel2Mesh [38], as well as implicit surface approaches [4, 13, 22, 26]. These can be be interpreted as different ways of parameterizing a manifold. In particular, AtlasNet generates a 3D shape as a collection of surfaces, each represented as a transformation of a unit grid using a fully-connected network. However, the generated pieces exhibit significant overlap which results in a poor surface reconstruction and is less desirable for applying materials and textures to the surface (Section 5). The proposed regularization alleviates this problem. Moreover, by replacing the fully-connected networks of AtlasNet with convolutional variants we improve the performance on standard benchmarks for shape generation [7] with networks that have a fraction of the parameters, faster inference time, as well as smaller memory footprint (Section 5).

2 Related Work

Manifold 3D shape generation 3D shape generation is an active area of research with methods that generate 3D shapes as volumetic representations such as occupancy grids [7, 11, 15, 27, 34, 36, 41], signed distance functions [4, 13, 22, 26], mutliview depth and normals [20, 21, 31, 33], or point clouds [1, 9, 10, 12]. Our work is closely related to techniques for generating 3D shapes through a predefined connectivity or parametrization structure over the surface of the shape. Pixel2Mesh [38] utilizes graph convolutional networks to generate meshes that are homeomorphic to a sphere. AtlasNet [14] and FoldingNet [42] learn a parametrization of a surface by adopting deep networks to transform point coordinates in a 2D plane to the shape surface. Specifically, each point is generated as $\left(f_\theta^1(x), f_\theta^2(x), f_\theta^3(x)\right)$ where f_θ^i is a deep network and $x = (x_1, x_2)$ is a point in the unit grid. Alternate approaches [4, 13, 22, 26] represent the surface as the level-set of a scalar field, $f(x) = 0, x \in \mathbb{R}^3$, e.g., of the signed distance function. While these have been applied for shape generation by training on 3D shape datasets, our goal is to analyze the role of these parameterizations as an *implicit prior* for manifold denoising and interpolation tasks.

Deep implicit priors Our work is related to the deep image prior [37] that generates images as a convolutional network transformation of a random signal on a unit grid. By optimizing the randomly initialized network to minimize a reconstruction loss with respect to the noisy target, their approach was shown to yield excellent denoising results. Our approach generalizes this idea to manifold data, which is more appropriate for interpolating and denoising contours and surfaces (see Figure 6 for a comparison). Our work is also related to the recently proposed deep geometric prior [40]. Their approach was used to estimate a surface from point cloud data by partitioning the surface into small overlapping patches and reconstructing the local manifold using a deep network. Consistency in the overlapping regions was enforced by minimizing the Earth Movers distance (EMD). In contrast to their work, we learn a small collection of non-overlapping parametrizations (atlas) by minimizing a regularized term and Chamfer distance, which is much more efficient than EMD. We also consider diverse tasks such as point cloud denoising, interpolation, and shape reconstruction across a category where the atlases needs to be consistent across instances. Finally, we present a theoretical analysis of the local properties of the generated surface by analyzing its limiting behavior as a Gaussian process.

Embedding a manifold Our work is related to techniques for embedding manifolds into a low-dimensional Euclidean space (*e.g.*, IsoMap [35] or LLE [28]). Our approach parameterizes the inverse mapping from the Euclidean space to the data manifold using a deep network. Interestingly, invertability can be guaranteed by using networks with easy to compute inverses (*e.g.*, NICE [8] or GLOW [19]). In computer graphics, a number of techniques have been developed for shape surface denoising and reconstruction. Screened Poisson Reconstruction [17] constructs an implicit surface on a 3D volumetric grid based on oriented point samples by solving the Poisson equation. Approaches based on Moving Least Squares [2, 25, 29] reconstruct a surface by estimating an approximation of each local patch, similar to the deep geometric prior [40] approach. Our approach outperforms these baselines by a significant margin (Table 1).

Deep networks and Gaussian processes A Gaussian process (GP) is commonly viewed as a prior over functions. Let T be an index set (*e.g.*., $T \in \mathbb{R}^d$), let $\mu(t)$ be a real-valued mean function and $K(t, t')$ be a non-negative definite kernel or covariance function on T. If $f \sim GP(\mu, K)$, then, for any finite number of indices $t_1, ..., t_n \in T$, the vector $(f(t_i))_{i=1}^n$ is Gaussian distributed with mean vector $(\mu(t_i))_{i=1}^n$ and covariance matrix $(K(t_i, t_j))_{i,j=1}^n$. Neal [23] showed that a two-layer network with infinite number of hidden units approaches a GP. The mean and covariance of commonly used non-linearities have been derived in several subsequent works [6, 39]. We use this machinery to analyze the limiting GP of deep manifold priors.

3 Method

Background Our focus is to define priors over *manifolds*. We first introduce some basic notation. A *n-manifold* is a topological space $\mathcal{M}$ for which every point in $\mathcal{M}$ has a neighborhood homeomorphic to the Euclidean space $\mathbb{R}^n$. Let $\mathcal{U} \subset \mathcal{M}$ and $\mathcal{V} \subset \mathbb{R}^n$ be open sets. A homeomorphism $\phi : \mathcal{U} \rightarrow \mathcal{V}, \phi(u) = (x_1(u), x_2(u), ..., x_n(u))$ is a *coordinate system* on $\mathcal{U}$ and $x_1, x_2, ..., x_n$ are *coordinate functions*. The pair $\langle \mathcal{U}, \phi \rangle$ is a *chart*, whereas $\zeta = \phi^{-1}$ is a *parameterization* of $\mathcal{U}$. An *atlas* on $\mathcal{M}$ is a collection of charts $\{\mathcal{U}_\alpha, \phi_\alpha\}$ whose union covers $\mathcal{M}$. Intuitively, surfaces are 2-manifolds where as contours are 1-manifolds. Thus the dimensionality of the input of the parameterization or the output of the chart corresponds to the order n of the manifold. Atlases can be used to represent manifolds that cannot be decomposed using a single parametrization (*e.g.*, the surface of a sphere can be diffeomorphically mapped to two planes but not one.)

General framework In our work we will replace the search over $\mathcal{U}$ by a search over the parameters θ of the DNN f_θ that encodes the parameterization $f_\theta = \zeta = \phi^{-1}$. More specifically, given a set of points $P \in \mathcal{M}$, we aim to recover the manifold $\mathcal{M}$ by computing the following:

$$\theta^* = \arg\min_\theta \mathcal{L}_C(f_{\theta, x \sim \mathbb{R}^n}(x), P). \tag{1}$$

The approximated manifold can then be reconstructed in the domain on which it is embedded f_{θ^*}. In practice, we restrict x to the unit hypercube $[0, 1]^n$. Here $\mathcal{L}$ is a loss function that computes a discrepancy between sets. Thus, reconstructing a manifold represented by an atlas of k charts is done by computing the following:

$$\theta_1^*, \theta_2^*, ...\theta_k^* = \arg\min_{\theta_1, \theta_2, ...\theta_k} \mathcal{L}_C(\bigcup_{i=1}^k f_{\theta_i}(x), P) \tag{2}$$

Parameterization We explore two choices of parameterizations of the coordinate function $f_\theta(x)$ as a deep neural network. The first uses a multi-layer perception (MLP) to represent the parameterization explicitly: the network receives as an input a value $x \in \mathbb{R}^n$ and outputs the coordinates of point in the manifold. We use ReLU non-linearities throughout the network, except for the last layer where we use tanh. This representation is analogous to the ones used in [14, 42]. The second choice is to encode $\mathcal{M}$ directly through a convolutional network $g(z)$, where z is a stationary signal (Gaussian noise). We use 2D convolutional layers followed by ReLU activations and bilinear upsampling, except for the last layer where we use tanh. The convolutional parametrization induces a stationary prior (see Supplementary for details), and we observe the resulting architectures are more memory-efficient and compact than the first choice.

Loss function A key part of our method is computing a distance between two sets of points P_1 and P_2. Such distance metric needs to be differentiable and reasonably efficient to compute, since the cardinality of the sets might be large. Thus, similarly to previous work [12, 14, 38, 42], we employ the Chamfer distance $\mathcal{L}_C$ defined as follows:

$$\mathcal{L}_C(P_1, P_2) = \sum_{p_1 \in P_1} \min_{p_2 \in P_2} \|p_1 - p_2\|_2^2 + \sum_{p_2 \in P_2} \min_{p_1 \in P_1} \|p_1 - p_2\|_2^2 \, .$$

Stretch regularization Representing the manifold as a set of multiple parameterizations output by DNNs has some drawbacks. First, there is no guarantee that the charts are invertible, which means that a surface generated by f_θ might contain self-intersections. Second, multiple charts might be representing the same region of the manifold. In theory this is not a problem as long as overlapping regions are consistent. However, in practice this consistency is hard to achieve when point clouds are sparse and noisy. We propose to alleviate those issues by penalizing the stretch of the computed parameterization. Let $\mathcal{N}(w)$ be the neighborhood of w in $\mathbb{R}^n$, the *stretch regularization* $\mathcal{L}_S$ can be defined as follows:

$$\mathcal{L}_S(\theta) = \mathbb{E}_{x \sim [0,1]^n} \left[\sum_{x' \in \mathcal{N}(x)} \|f_\theta(x) - f_\theta(x')\|_2^2 \right] . \tag{3}$$

Notice that we can compute the neighbors of x ahead of time which makes the computation significantly cheaper. In practice, we sample x from a set of predefined regularly spaced values in $[0, 1]$ – a regular grid in the 2D case. Now we can define our full loss function as follows.

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_C(\boldsymbol{f}_{\boldsymbol{\theta}, x \sim \mathbb{R}^n}(x), P) + \lambda \mathcal{L}_S(\boldsymbol{\theta}), \tag{4}$$

where $\boldsymbol{\theta} = \theta_1, \theta_2, ...\theta_k$ and $\boldsymbol{f_\theta}(x) = \bigcup_{i=1}^{k} f_{\theta_i}(x)$.

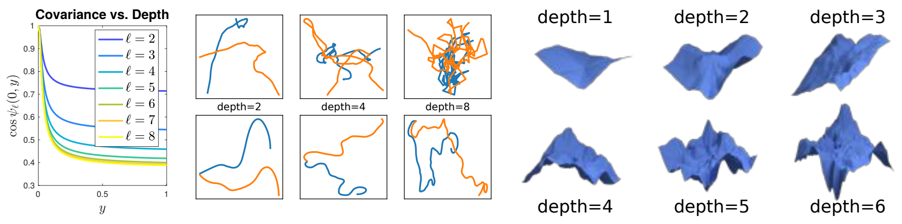

Fig. 3: **Characterizing the deep manifold prior. (left)** a plot demonstrating the relationship between the network depth and the covariance function for the limiting GP. **(middle)** Random curves generated by the coordinate (top rows) and arc-length (bottom rows) parametrizations using deep networks with varying depths. **(right)** Random surfaces generated by deep networks of varying depths.

Manifolds as deep level-sets An alternative approach is to represent d-manifold as the level-set of a scalar function over $d+1$ dimensions. For example, a surface can be represented as the level set, $f(x) = 0$, where $x \in \mathbb{R}^3$. Prior work [4, 13, 22, 26] has explored this approach to generate a 3D surface by approximating its signed distance function. Level-set formulation can naturally handle shapes with different topologies, but require the knowledge of what is inside the surface, which can be challenging to estimate for imperfect point-cloud data. In this work, we also characterize and experiment with the manifold prior induced by the level-set of a deep network $f_\theta(x) = 0$ initialized randomly.

4 Limiting GP for the Deep Manifold Prior

Consider the case when the manifold coordinates are parametrized using a deep network $f_\theta(x)$. We show that random networks, *e.g.*, whose parameters are drawn i.i.d. from a Gaussian distribution, produces smooth manifolds. This is done by analyzing the limiting behavior of the function as a Gaussian process. In practice this is a good approximation to networks that are relatively shallow and have hundreds of hidden units in each layer.

Concretely, the mean $\mathbb{E}_\theta[f_\theta(x)]$ and covariance $\mathbb{E}_\theta[f_\theta(x)f_\theta(y)^T]$ of the parameterization characterize the structure of the generated manifold. For example, the covariance function of a smooth manifold decays slowly as a function of distance in the input space compared to a rough one. Following prior work [6, 23, 39], we first derive the mean and covariance for a two layer network with a scalar output. We then generalize the analysis to vector outputs and multi-layer networks.

Consider a two-layer fully-connected network on an input $x \in \mathbb{R}^n$. Let H be the number of units in the hidden layer represented using parameters $U = (u_1, u_2, \ldots u_H)$ where $u_j \in \mathbb{R}^n$ and the second layer has one output parameterized by weights $v \in \mathbb{R}^H$. Denote the non-linearity applied to each unit as the scalar function $h(\cdot)$. The output of the network is: $f(x) = \sum_{k=1}^{H} v_k h(u_k^T x)$. When the parameters U and v are drawn from a Gaussian distributions $N(0, \sigma_u^2 \mathbb{I})$ and

$N(0, \sigma_v^2 \mathbb{I})$ respectively, we have:

$$\mathbb{E}_{U,v}[f(x)] = \mathbb{E}_{U,v}\left[\sum_{k=1}^{H} v_k h\left(u_k^T x\right)\right] = 0,$$

since U and v are independent and zero mean. Similarly, the covariance function $K(x,y)$ can be shown to be:

$$K(x,y) = \mathbb{E}_{U,v}[f(x)f(y)] = H\sigma_v^2 \mathbb{E}_U\left[h\left(u_k^T x\right)h\left(u_k^T y\right)\right].$$

This follows since each u_k is drawn i.i.d, each v_k is independent and drawn identically from a Gaussian distribution with zero mean. The quantity $V(x,y) = E_u\left[h(u^T x)h(u^T y)\right]$ can be computed analytically for various transfer functions. Williams [39] showed that when $h(t) = \text{erf}(t) = 2/\sqrt{\pi}\int_0^t e^{-t^2} dt$, then

$$V_{\text{erf}}(x,y) = \frac{2}{\pi}\sin^{-1}\frac{x^T \Sigma y}{\sqrt{(x^T \Sigma x)(y^T \Sigma y)}}. \tag{5}$$

Here $\Sigma = \sigma^2 \mathbb{I}$ is the covariance of u. For the ReLU non-linearity $h(t) = \max(0,t)$, Cho and Saul [6] derived the expectation as:

$$V_{\text{relu}}(x,y) = \frac{1}{\pi}\|x\|\|y\|\left(\sin\psi + (\pi - \psi)\cos\psi\right), \tag{6}$$

where $\psi = \cos^{-1}\left(\frac{x^T y}{\|x\|\|y\|}\right)$. We refer the reader to [6,39] for kernels corresponding of other transfer functions.

An application of the Central Limit Theorem shows that by letting σ_v^2 scale as $1/H$ and $H \to \infty$, the output of a two layer convolutional network converges to a Gaussian distribution with zero mean and covariance

$$K_1(x,y) = E_{U,v}\left[f(x)f(y)\right] = V(x,y). \tag{7}$$

Hence the limiting behavior of the DNN can be approximated as a Gaussian process with a zero mean and covariance function $K(x,y) = V(x,y)$.

Extending to multiple outputs The above analysis can be extended to the case when the function $f(x)$ is vector valued. For example a 2-manifold in 3D can be represented as $f(x) = (f^1(x), f^2(x), f^3(x))$, with $x \in \mathbb{R}^2$. In our case, the functions share a common backbone and each $f^i(x)$ is constructed from the outputs of the last hidden layer parameterized with weights v^i, *i.e.*, $f^i(x) = \sum_{k=1}^{H} v_k^i h(u_k^T x)$. From the earlier analysis we have that each $f^i(x)$ has zero mean in expectation. And the covariance between dimension i and j of f is:

$$K_1^{i,j}(x,y) = \mathbb{E}_{U,v_i,v_j}\left[f^i(x)f^j(y)\right] = V(x,y)\,\mathbf{1}[i=j].$$

This follows from the fact that each v_k^i is independent and drawn from a zero mean distribution. Thus, the covariance is a diagonal matrix with entries $V(x,y)$ in its the diagonal.

Extending to multiple layers The analysis can be extended to multiple layers by recursively applying the formula for the two-layer network. Denote $K_\ell(x,y)$ as the covariance function of a scalar valued fully-connected network with $\ell+1$ layers and $J(\theta) = \sin\theta + (\pi - \theta)\cos\theta$. Following [6] for the ReLU non-linearity we have the following recursion:

$$K_{\ell+1}(x,y) = \frac{1}{\pi}\left(K_\ell(x,x)K_\ell(y,y)\right)^{1/2} J\left(\psi_\ell\right).$$

Where $\psi_\ell(x,y) = \cos^{-1}\frac{K_\ell(x,y)}{\sqrt{K_\ell(x,x)K_\ell(y,y)}}$ and $K_0(x,y) = x^T y$. Note that if in each layer we add a bias term sampled from a $N(0,\sigma_b^2)$ the covariance changes to $K_\ell(x,y) + \sigma_b^2$ and the mean remains unchanged at zero.

4.1 Discussion and Analysis

The above analysis shows that random networks induce certain priors over the coordinates of the manifold. The effect of increasing the depth of the network can be seen by visualizing how the covariance $\cos\psi_\ell(x,y)$ varies as a function of depth. Figure 3 plots $\cos\psi_\ell(x,y)$ at $x = 0$ for a curve as a function of the depth of the network for $\sigma_b = 0.01$. The covariance decays faster with depth, indicating that the deeper networks produce manifolds with higher spatial frequencies (or curvatures). This can also be seen in Figure 3 which shows random curves (middle) from a surfaces (right) for networks with varying depths.

One potential drawback of fully-connected network parameterization is that the generated manifold does not have a stationary (translationally invariant) covariance function. A covariance function $K(x,y)$ is stationary if it can be written as $K(x,y) = k(x-y)$. On the other hand, a convolutional network that produces coordinates through a series of convolutional layers operating on a random noise has a stationary covariance [5]. This is identical to the approach for generating natural images in the deep image prior [37] and we explore this alternative in Section 5.2.

Normals and curvature While we have shown that the outputs $f(x)$ induced by random networks is a GP in the limit, what can be said about intrinsic properties such as normals and curvature? Consider the curve $\gamma(t) = (x(t), y(t))$. Since derivatives are linear operators, it follows that distribution of derivatives, $\dot{x}$ and $\dot{y}$, are also Gaussian [30]. The curvature is given by $\kappa = (\ddot{x}\dot{y} - \ddot{y}\dot{x})/(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}$. Unfortunately, since each of the derivatives converge to a zero mean Gaussian distribution, the limiting distribution of the curvature κ does not exist. The pathology arises because the parameterization has a speed ambiguity, *i.e.*, replacing t with any monotonic function of t results in the same curve. To avoid this one can directly parametrize the derivatives as $\dot{x} = \cos(f(t))$ and $\dot{y} = \sin(f(t))$ where f is a deep network. This is an arc-length (unit speed) parametrization since $\dot{x}^2 + \dot{y}^2 = 1$. Once the derivatives are generated, the curve can be reconstructed by integration, *i.e.*, $x = \int_0^t \cos(f(t))dt$. In this case the limiting distribution of the coordinates, normal, and curvature all exist and are also

	Surface	Contour	Implicit	RIMLS [25]	SPSR [17]
bunny	**2.71E-04**	6.64E-04	5.52E-04	1.43E-03	3.96E-04
dragon	**4.18E-04**	6.12E-04	1.20E-03	1.65E-03	1.46E-02
car	**2.73E-04**	4.57E-04	6.83E-02	1.50E-03	2.10E-03
cup	**2.59E-04**	5.80E-04	2.64E-02	1.74E-03	1.00E-02
mobius	**3.51E-04**	4.95E-04	3.26E-03	1.96E-03	1.89E-02
chair	**3.95E-04**	4.22E-04	7.32E-03	2.09E-03	2.58E-02
spiral	1.05E-03	**7.31E-04**	1.64E-02	2.98E-03	7.90E-02
ring	5.69E-04	**5.54E-04**	4.81E-02	2.46E-03	3.76E-02
avg.	**4.48E-04**	5.65E-04	2.13E-02	1.98E-03	2.36E-02

Table 1: **Quantitative results for point cloud denoising.** *Surface*, *Contour* and *Implicit* represent different *deep manifold priors* based on a 2-manifold, 1-manifold and level-set paramertization.

GPs. We derive the mean and covariance function in the Supplementary material. Figure 3-middle shows draws from the GP with direct (top) and arc-length (bottom) parametrizations. One can see that arc-length parametrizations lead to more length-uniform curves.

Unlike curves, it is much more challenging to design arc-length parametrizations of surfaces. The difficulty arises due to the fact the gradients need to satisfy additional constraints for the surface to be integrable [32]. Hence, we directly parameterized the coordinate function and proposed the stretch regularization to minimize distortion. Alternatives ways of parameterizing the surface to satisfy properties such as conformality [24] is left for future work.

Deep level-set prior Finally, the GP analysis applies in a straightforward manner to the level-set formation $f_\theta(x) = 0$ where f_θ is a ReLU network mapping the 3D position $x \in \mathbb{R}^3$ to a scalar. The induced distribution over the scalar field is a GP for random networks. Since for a differentiable function f with non-zero gradient, the gradient is orthogonal to the level set, one can characterize the surface by analyzing the gradient field ∇f. The limiting distribution over the gradient field is also a GP and one can estimate the mean and convariance functions by a similar analysis (see Supplementary material for details). However, the training objective of the level-set prior is different from the explicit parameterization as the network must classify points as inside or outside the surface. This supervision can be challenging to obtain from noisy data, especially for thin structures. We provide a comparison with this approach in Section 5.

5 Experiments

In this section we will present quantitative and qualitative results for applying the manifold prior to multiple manifold reconstruction tasks. All the experiments in this paper were implemented using Python 3.6 and PyTorch. Computation was performed on TitanX GPUs.

	S1R	S8R	S1	S8	-	-
avg.	4.48E-03	**4.48E-04**	2.75E-03	1.35E-03	-	-

	C1R	C8R	C1	C8	RIMLS [25]	SPSR [17]
avg.	1.08E-03	5.77E-04	1.00E-03	5.82E-04	1.98E-03	2.36E-02

Table 2: **Ablation studies.** Comparison between different variations of our approach. Naming follows the following convention: S corresponds to a 2-manifold parameterization (surface), whereas C corresponds to a 1-manifold (contour). The following number (1 or 8) corresponds to the number of parameterizations. A R letter is added if stretch regularization was used ($\lambda = 1.0$).

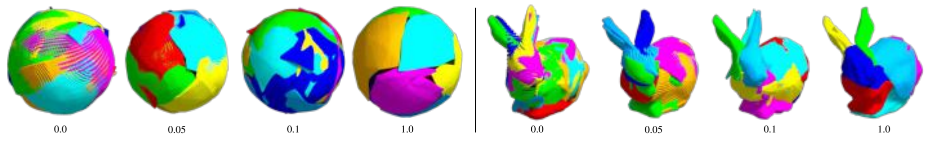

Fig. 4: **Effect of the regularization weight on the reconstructed manifold.** For this experiment, we use our method to reconstruct a sphere using an atlas with 8 charts and render each one with a different color. Without any regularization, there is a significant amount of deformation applied to each surface (hence the space between the points) and a considerable amount of overlap between different parts. As the regularization weight increases, those aspects are noticeably reduced.

5.1 Denoising and Interpolation

Benchmark Our benchmark consists of 8 different 3D shapes with diverse characteristics. The shapes are normalized to fit a unit cube and 16K points are sampled on their surfaces. The point positions are perturbed by a Gaussian noise with standard deviation 2×10^{-3} and zero mean. Figure 7 shows the ground-truth shapes as well as their noisy counterpart. Since the level-set representation and the baseline methods (RIMLS [25], SPSR [17]) require normal information, we estimate the normal for every point by using the local frame defined by its nearest neighbors. We experimented multiple numbers of neighbors for both baselines and used the value that led to the best results: 20 neighbors for SPSR and the level-set representation, 30 neighbors for RIMLS. The network used in the level-set representation follows the same architecture and training protocol as the one used for the explicit parametrizations (described in the next paragraph). However, it is trained to predict every point as outside (+1) or inside the surface (-1). Points with positive values are generated by translating every point in the point cloud along the normal direction for a distance $\epsilon = 2 \times 10^{-3}$. Points with negative values are generated in the same way, but applying a displacement to the opposite direction. For RIMLS, we used a relative spatial filter size of 10, 15 projection iterations and a volumetric grid with 200^3 resolution. For SPSR, we used an octree with depth 7 and 8 iterations.

Experimental setup Our method performs denoising by minimizing Equation 4. In this framework, P is the noisy point cloud we are trying to reconstruct and $\boldsymbol{f_\theta}$ is a neural network. In all experiments we use a neural network with 3 fully connected layers, where the layers have 256, 128 and 64 hidden units, respectively. The output of the networks is a point in $\mathbb{R}^3$. The input can be either a point in $\mathbb{R}$ (1-manifold) or $\mathbb{R}^2$ (2-manifold). We use *ReLU* activations followed by batch normalization at each layer, except for the last, where we use a *tanh* non-linearity. We vary the architecture of $\boldsymbol{f_\theta}$ with respect to the number of parameterizations (1 or 8) and dimensionality (1 or 2). Additionally, we try each one of these architectural variations with $\lambda = 0$ and $\lambda = 1.0$. When using 8 parametrizations, 4096 points are sampled per parametrization. When using just one parametrization, 16K points are sampled. We optimize our objective through gradient descent using the Adam optimizer with learning rate 10^{-3}. For evaluation, we uniformly sampled 16K points in the computed manifold (represented as a triangular mesh) and compute the Chamfer distance with respect to the ground-truth.

Results and discussion. Our methods significantly outperform the baselines for most of the shapes. Quantitative results can be seen in Table 1 and the qualitative results are shown in Figure 7. The numbers are computed using 8 parametrizations (for surfaces and curves) and $\lambda = 1.0$. A comparison between different variations of our approach is displayed in Table 2. RIMLS, SPSR and level-set representations (*Implicit* in Table 1) have trouble reconstructing point clouds with a significant amount of noise. This is due to the fact that those methods rely on accurate surface normal estimates to infer inside/outside regions of the shape. Besides, RIMLS and methods based on implicit functions (SPSR and level-set representations) work better when dealing with closed surfaces. Shapes that are better approximated by contours (ring, spiral, chair's legs) are particularly challenging for those approaches. On the other hand, the networks parametrizing explicit functions (*Surface* and *Contour* in Table 1) are able to adapt to different structures and present a fair performance across a diverse set of shapes.

The results in Table 2 suggest that using multiple parametrizations gives a better approximation than just using a single one. This happens because complex shapes are easier to represent by multiple parametrizations. For example, while using a single 2-manifold parametrization, the ring tends to be approximated by a disk, which significantly increases the reconstruction error when the points are uniformly sampled over the final mesh. This behavior is illustrated in Figure 5. Our ablation studies also indicate that using stretch regularization helps parametrizations of both surfaces and contours. Figure 4 shows the effect of stretch regularization for two different shapes. As the regularization weight increases, the overlap between different parameterizations becomes smaller. When overlaps exist, the manifold representation is suboptimal – the same regions are being generated multiple times.

Interpolation We also explored using the manifold prior for point cloud interpolation. This experiment follows the same experimental setup as denoising. However, instead of perturbing the points with Gaussian noise, we randomly select 1K points out of 16K. Interpolation is performed by minimizing Equa-

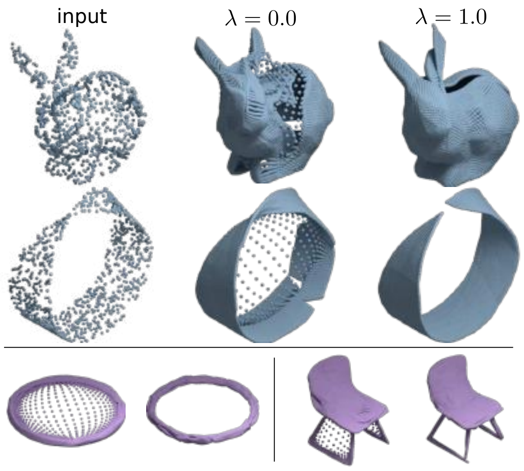

Fig. 5: *Interpolation results on the top.* Stretch regularization ($\lambda = 1.0$) helps generate smoother surfaces. *On the bottom, denoising using one vs. multiple parametrizations.* Shapes on the left were reconstructed using a single parameterization, whereas shapes on the right used 8 parameterizations. Using multiple parameterizations helps reconstruct complex shapes.

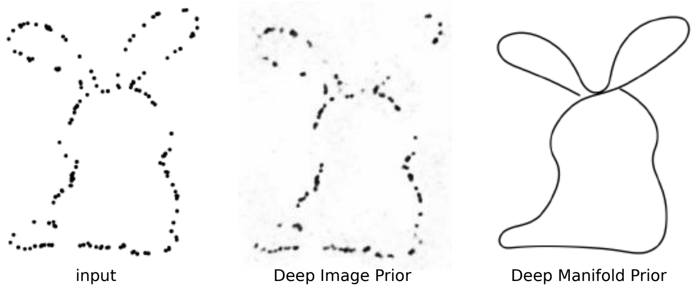

Fig. 6: **Comparison to the deep image prior [37].** Image-based prior (middle) is not able to connect the dots in the input image (left). On the other hand, the manifold prior is able to reasonably interpolate the dotted drawing.

tion 4. Results can be seen in Figure 5. For these experiments we use a single parameterization and include stretch regularization, without which the surface has holes and significant folds. Our method is able to reconstruct reasonable surfaces from a small set of points.

Comparison with the deep image prior We also compare our approach to the deep image prior [37] for interpolating points in 2D images. Results are presented in Figure 6. We use the same architecture from [37] while minimizing the mean squared error with respect to the image pixels. For the manifold prior, we use a single 1-manifold parameterization following the architecture described before, differing only in the dimensionality of the output: points this this case are in

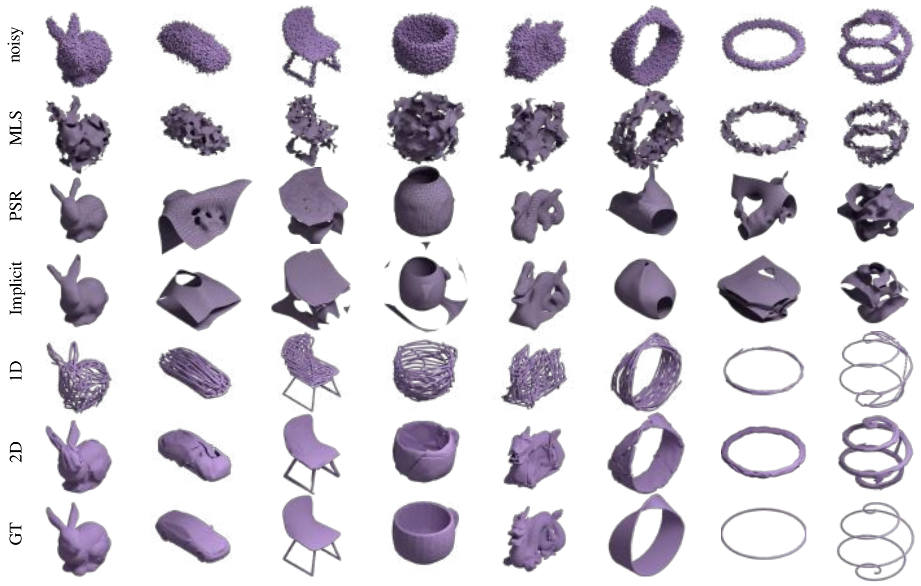

Fig. 7: **Qualitative comparison between different denoising methods.** Rows display different methods, whereas columns display different shapes. Baseline methods do not perform as well as the deep manifold prior, even for closed surfaces like the bunny (first column) and the dragon (fifth column). As we can see, 2-manifold parameterizations are better for reconstructing surfaces, whereas 1-manifold counterparts reconstruct the curves (last two columns) more acurattely.

$\mathbb{R}^2$ instead of $\mathbb{R}^3$. Coordinates of the black pixels in the input image are used to form a point cloud and the manifold is computed by minimizing Chamfer distance with respect to it.

5.2 Learning from data

Finally, we show how the insights presented in the earlier sections, in particular convolutional parameterization and stretch regularization, can also improve generative models of 3D shapes when trained on a large collection of shapes.

To measure the effect of the stretch regularization in a learning-based scenario, we train a model using the same architecture as AtlasNet [14] on a subset of $50,000$ shapes across 13 categories of the ShapeNet dataset [3]. Adding stretch regularization did not significantly impact the Chamfer metric – error of 1.46×10^{-3} and 1.47×10^{-3} with and without regularization. However, the results are qualitatively better. As seen in Figure 8 the regularization reduces the stretch and overlap of the generated surfaces, and eliminates artifacts where holes are incorrectly filled.

We also train a convolutional decoder with stretch regularization on the single-view reconstruction benchmark [7]. Our approach called ConvAtlas is compared against AtlasNet and MRTNet [12] in Table 4. For a fair comparison, we

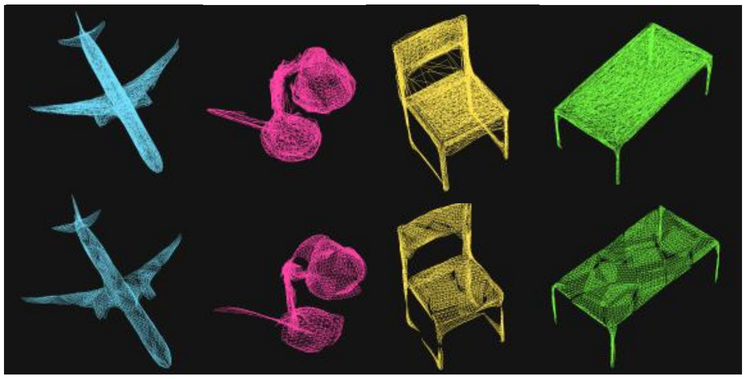

Fig. 8: **Autoencoder results**. Results on using AtlasNet [14] trained w/o (top) and w/ (bottom) stretch regularization. The latter results in meshes with reduced deformation and overlap, and removes artifacts where the chair's back is incorrectly filled.

Architecture	mean/cat.	mean/inst.	#params.
MRTNet	4.80	4.26	81.6M
AtlasNet	4.74	4.38	42.6M
ConvAtlas	**4.53**	**4.00**	**14.5M**

Table 3: **Quantitative results for single-view image-to-shape reconstruction.** The table reports the mean Chamfer distance metric (scaled by 10^3) computed per category and per instance.

use 4K points for evaluation across all methods. ConvAtlas outperforms both approaches in terms of per-category and per-instance error, and also leads to more compact models. Per-category results and experimental details are in the Supplementary material.

6 Conclusion

We presented a manifold prior induced by deep neural networks. Our experiments show that the prior can be effectively used for a variety of manifold reconstruction tasks: denoising, interpolation and single-view reconstruction. Besides, we analyzed the influence of the architecture in the characteristics of the prior by posing the models as GP. In conjunction to the prior induced by deep networks, we showed that using a stretch regularization procedure enables better manifold approximation and improves the quality of the generated meshes, reducing large deformations and overlaps between different parameterizations.

Acknowledgements. The authors would like to thank Daniel Sheldon for helpful discussions related to Gaussian processes. This work is supported in part by NSF grants #1908669 and #1749833. Our experiments were performed in the UMass GPU cluster obtained under the Collaborative Fund managed by the Massachusetts Technology Collaborative.

A Convolutional Parametrizations

In the main paper, we experimented with fully connected architectures for representing manifold parametrizations. However, parametrizations represented by convolutional architectures also induce a prior useful for manifold reconstruction tasks. In this section, we show experiments with denoising and single-view reconstruction. We start by defining a `ConvBlock`, which consists of a bilinear upsampling layer followed by a 2D-conv, batch normalization [16] and Leaky ReLU activation (slope=0.2). Every convolutional layer uses filter size 3×3, stride 1 and the number of filters is exactly half the number of its input channels. In other words, at every `ConvBlock`, the output tensor spatially doubles the size of its input tensor, but only has half the number of channels. This pattern follows throughout the whole network, except for the last layer, where the output layer always have 3 channels, representing the (x, y, z) point coordinates.

A.1 Denoising

The denoising experiments follow the same procedure described in the main paper, except for the network architecture. Instead of using a fully connected model, we employ a network with 3 `ConvBlock`s, starting from an input tensor with shape $4 \times 4 \times 512$ whose values are drawn from a standard gaussian distribution. The output of each parametrization is a tensor with shape $32 \times 32 \times 3$, which we can treat as a point cloud with 1024 and use Chamfer distance in the same way as described in Section 5. We also use the position of the points in the output tensor to define the local neighborhood utilized in the stretch regularization. Results are presented in Figure 9. As we can see, convolutional parametrizations also induce a useful prior for manifold reconstructions and, similarly to the other parametrizations, it is significantly better than the baselines. Quantitatively, using convolutional parametrizations in the denoising yields slightly worse results than using fully connected networks – in terms of Chamfer distance, 4.58×10^{-4} vs. 4.48×10^{-4}.

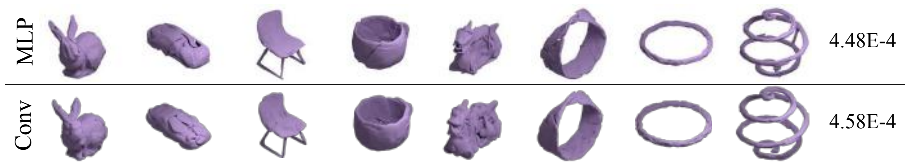

Fig. 9: Comparison of Conv and MLP networks for denoising. Average error across shapes to the right. Both models use 8 parametrizations and stretch regularization. Zoom for details.

A.2 Single-view Reconstruction

In this subsection we present quantitative and qualitative results for single-view image-to-shape using convolutional paramterizations. We also train a con-

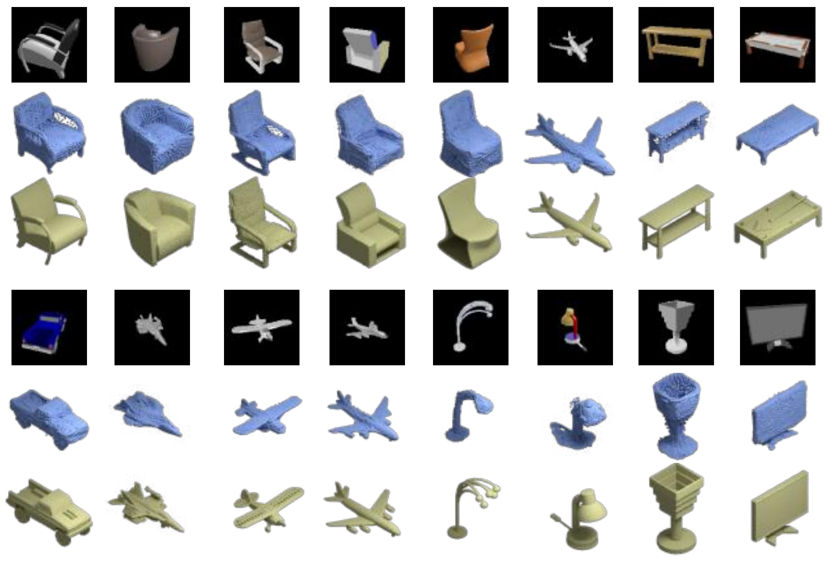

Fig. 10: **Image-to-shape reconstruction results from the test set.** The images shown are the input (black background), our results (32K points, rendered blue), and ground truth (rendered in light green). Qualitatively, our method is able to generate high-resolution point clouds faithfully capturing fine geometric details such as the chair legs, arms, airplane engines, monitor stands etc.

volutional decoder with stretch regularization on the single-view reconstruction benchmark [7]. This follows the same experimental setup as previous papers [7, 9, 12, 14]. However, unlike AtlasNet [14], our network is trained in one stage, without the need to train the decoder in an auto-encoder setting before fine-tuning it with an image CNN in a second step. We used Adam optimizer [18] with learning rate of 10^{-3}. The model is trained for 40 epochs and the learning rate is divided by 2 every 5 epochs. We use ResNet-18 as image encoder and 32 convolutional parameterizations. Even though we use more parameterizations than AtlasNet, the total number of parameters is smaller (see Table 6. The evaluation results per category are presented in Table 4. Compared to MRTNet, our model performs better in 12 out of 13 categories. Compared to AtlasNet, our method is better or ties (the firearm category) in 7 out of 13 categories. Overall our approach outperforms AtlasNet in per-category mean by 0.21, a relative improvement of 4.4%. Also note that our model outperforms AtlasNet mainly in categories with a large number of examples (tables, cars, airplanes, chairs). As a result, if average over instances, our method has a per-instance mean of 4.0, vs. 4.38 by AtlasNet – a relatively improvement of 8.7%.

Ablation studies. Table 5 shows a quantitative comparison between a few architectural variations. We start by analyzing a variation of our network that generates the same number of points (using a single decoder) as MRTNet (4K

	pla.	ben.	cab.	car	cha.	mon.	lam.	spe.	fir.	cou.	tab.	cel.	wat.	**mean**
AtlasNet [14]	2.17	**3.39**	**3.93**	3.40	4.56	**5.05**	12.24	8.79	**2.15**	**4.58**	4.15	**3.25**	3.93	4.74
MRTNet [12]	2.25	3.68	4.73	**2.55**	4.06	6.07	11.15	8.84	2.25	4.98	4.45	3.72	3.64	4.80
Ours (32 dec.)	**2.06**	3.40	4.46	2.60	**3.76**	5.94	**10.66**	**8.38**	**2.15**	4.64	**3.96**	3.45	**3.40**	**4.53**

Table 4: **Quantitative results for single-view image-to-shape reconstruction.** The table reports Chamfer distance metric (scaled by 10^3) computed per category, and the mean of all categories. For each method 4K points were used to compute the distance.

points) and the same image encoder (vgg-16). The performance of this variation is 0.05 worse than MRTNet, but it has an order of magnitude less parameters than MRTNet. Another variation is to still use a single decoder but generate a higher-resolution point cloud (16K points). This variation results in improved Chamfer distance, by 0.1, than the first variation, indicating that the increased resolution does improve reconstruction accuracy. Again, even when the number of generated points is higher than 4K, our evaluation is done by randomly selecting 4K points, for fair comparison. The last row in the table is our default setting (32 decoders outputting a total of 32K points). The number of network parameters are reported in Table 6. Even though the number of points our network generates is 8 times that of MRTNet, its size is only about 1/6 of MRTNet, since our network does not need to represent multiple resolutions at each layer. Compared to AtlasNet, our network is about 1/3 of its size, due to the efficiency of using a fully convolutional architecture. Despite using a much smaller number of parameters, our network outperforms MRTNet (in terms of Chamfer distance metric) by 0.27, and AtlasNet by 0.21.

Architecture	mean/cat.	mean/inst.
MRTNet	4.80	4.26
1 dec./vgg16/4k	4.85	4.30
1 dec./res18/16k	4.75	4.22
32 dec./res18/32k	**4.53**	**4.00**

Table 5: **Architecture variations and evaluation results.** The table reports per-category mean and per-instance mean for MRTNet, and three variations of our methods: single decoder with 4K output points, 16K output points, and 32 decoders with 32 output points. For all cases, the Chamfer distance is calculated using 4K sample points, and results are scaled by 10^3.

Qualitative Results. Figure 10 shows image-to-shape reconstruction results for images from the test dataset. Overall our method is able to accurately capture fine geometric details such as the chair legs, arms, airplane engines, monitor stands etc. The number of points (32K) is considerably higher than previous work (e.g. 1K by [9] and 4K by [12]). Some specific shapes, such as lamps and

Method	#parameters
AtlasNet	42.6M
MRTNet	81.6M
Ours (1 dec.)	2.49M
Ours (1 residual dec.)	5.79M
Ours (32 dec.)	14.5M

Table 6: **Comparing the # of network parameters.**

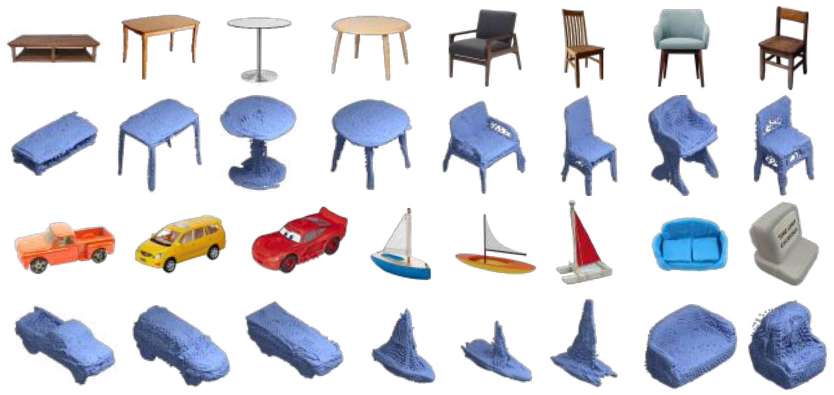

Fig. 11: **Image-to-shape reconstruction results on Internet photos.** We test our method on real photos downloaded from the Internet and the results are rendered in blue. The test images here are considerably different from the training set. Our method achieves reasonable results with accurate geometric details. The last image (computer) represents a category that has not been seen during training.

jet fighters, present significant challenges for the network as the input images do not contain all the visual details. Nonetheless our method is able to produce a reasonable approximation.

Test on real images. The test set images are synthetically rendered and as such they look similar to the training images. To evaluate our method on real images we use photos downloaded from the Internet, as shown in [12]. They are processed by removing the background so only the foreground object remains. Figure 11 shows the results. The top row in the figure shows furniture objects, which demonstrate that even though the network is trained using synthetic images rendered with artificial lighting and materials, the model is able to generalize well to real shading, lighting, and materials. The second row shows additional objects where the shading is considerably different from training images. In particular, the last image (desktop computer) is in a category that the training has never seen. Nonetheless the reconstructed shape is reasonable.

Shape correspondence. Once trained, our network learns to generate shapes with corresponding structures. We demonstrate this with the following experiment.

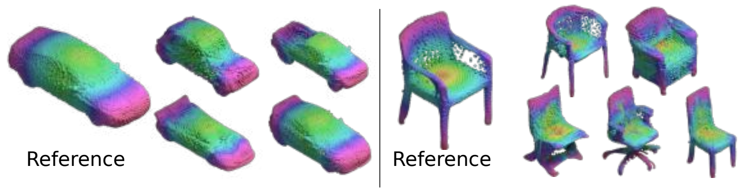

Fig. 12: **Visualizing Shape Correspondences.** Our network learns approximate shape correspondences even though the training is not supervised with such information. The shapes shown here are generated by 32 decoders.

First, we randomly select a point cloud generated by our network and call it a reference shape. Then, we assign every point in the reference shape a color, where the hue is computed based on the point's distance to the center of gravity of the object. Then this color assignment is propagated to the other point clouds, such that a point at index (i, j) in the output tensor is assigned the same color as the point on the reference shape at the same index. The resulting colorized point clouds are shown in Figure 12. Similar color indicates similar index range in the output tensor. Note that even though the network is not trained explicitly with point correspondences as supervisory signal, it learns to generate corresponding parts in the same regions of the output tensor, as can be seen around the tips of the chairs' arms, legs and seats.

B Limiting distribution for the curvature

We start by parameterizing the derivative of a space curve $\dot{x} = \cos(f(t))$ and $\dot{y} = \sin(f(t))$ where f is a neural network. From the standard analysis we know that $f(t)$ converges to a Gaussian with mean μ and kernel $k(\cdot, \cdot)$. Without loss of generality we can assume that the mean μ is such that $\cos(\mu) \neq 0$ and $\sin(\mu) \neq 0$. This can be achieved by adding a fixed bias term μ to the output of the last layer. To compute the limiting distribution of $\ddot{x}$ and $\dot{y}$ we apply the first order delta method to obtain:

$$\dot{x} \to \mathcal{N}(\cos(\mu), \sigma^2 \sin^2(\mu)), \tag{8}$$

$$\dot{y} \to \mathcal{N}(\sin(\mu), \sigma^2 \cos^2(\mu)). \tag{9}$$

Note we can only apply the first order delta method when the derivatives are not zero. Hence we assumed that μ is set to be a quantity which has this property. Otherwise we need the second-order delta method and the resulting distribution would be χ^2 for one of the derivatives.

Since the derivative is a linear operator it follows that $\ddot{x}$ and $\ddot{y}$ are also GPs. The curvature formula for a arc-length parameterized space curve is $\kappa^2 = \ddot{x}^2 + \ddot{y}^2$. From this it follows that κ^2 is a χ^2 random variable.

Graph parameterization. We also analyze the case where the curve is the graph of a one-dimensional function, i.e., $x = x$,$y = f(x)$. In this case the curvature can be written as $\kappa = \ddot{f}/((1+\dot{f}^2)^{\frac{3}{2}})$. Once again all the derivatives $\dot{f}$ and $\ddot{f}$ are Gaussian random variables. Assume that $(\dot{f}, \ddot{f})$ are distributed according to $N(0, \Sigma)$. Here $\Sigma = [\sigma_{\dot{f},\dot{f}}, \sigma_{\dot{f}\ddot{f}}; \sigma_{\dot{f}\ddot{f}}, \sigma_{\ddot{f}\ddot{f}}]$ denoting the joint covariance distribution. Applying the delta method with $g(a, b) = b/(1 + a^2)^{3/2}$, we get that k is distributed as a Gaussian random variable $N(0, \nabla g^T \Sigma \nabla g)$. Since $\nabla g(a, b)|_{0,0} = [0, 1]$, we have $k \rightarrow N(0, \sigma_{\ddot{f}\ddot{f}})$.

References

1. Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J Guibas. Learning Representations and Generative Models For 3D Point Clouds. In *International Conference on Machine Learning*, 2018.
2. Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics*, 9(1):3–15, 2003.
3. Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*, 2015.
4. Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
5. Zezhou Cheng, Matheus Gadelha, Subhransu Maji, and Daniel Sheldon. A Bayesian Perspective on the Deep Image Prior. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
6. Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.
7. Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *European Conference on Computer Vision*, 2016.
8. Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.
9. Haoqiang Fan, Hao Su, and Leonidas Guibas. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
10. Matheus Gadelha, Subhransu Maji, and Rui Wang. 3d shape generation using spatially ordered point clouds. In *British Machine Vision Conference (BMVC)*, 2017.
11. Matheus Gadelha, Subhransu Maji, and Rui Wang. Unsupervised 3D Shape Induction from 2D Views of Multiple Objects. In *International Conference on 3D Vision (3DV)*, 2017.
12. Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution Tree Networks for 3D Point Cloud Processing. In *ECCV*, 2018.
13. Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *International Conference on Computer Vision*, 2019.

14. Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.
15. Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3d object reconstruction. In *International Conference on 3D Vision (3DV)*, 2017.
16. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, 2015.
17. Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):29, 2013.
18. Diederik P Kingma and Jimmy Ba. ADAM: a method for stochastic optimization. In *International Conference on Learning Representations*, 2014.
19. Durk P Kingma and Prafulla Dhariwal. GLOW: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018.
20. Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning Efficient Point Cloud Generation for Dense 3D Object Reconstruction. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.
21. Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 3d shape reconstruction from sketches via multi-view convolutional networks. In *International Conference on 3D Vision (3DV)*, 2017.
22. Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
23. Radford M Neal. *Bayesian learning for neural networks.* PhD thesis, University of Toronto, 1995.
24. Zeev Nehari. *Conformal mapping.* Courier Corporation, 2012.
25. A Cengiz Öztireli, Gael Guennebaud, and Markus Gross. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer Graphics Forum*, volume 28, pages 493–501. Wiley Online Library, 2009.
26. Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
27. Stephan R. Richter and Stefan Roth. Matryoshka Networks: Predicting 3D Geometry via Nested Shape Layers. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.
28. Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.
29. Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM, 1968.
30. Ercan Solak, Roderick Murray-Smith, William E Leithead, Douglas J Leith, and Carl E Rasmussen. Derivative observations in gaussian process models of dynamic systems. In *Advances in neural information processing systems*, 2003.
31. Amir Arsalan Soltani, Haibin Huang, Jiajun Wu, Tejas Kulkarni, and Joshua Tenenbaum. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *CVPR*, 2017.

32. Héctor J Sussmann. Orbits of families of vector fields and integrability of distributions. *Transactions of the American Mathematical Society*, 180:171–188, 1973.
33. Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3D models from single images with a convolutional network. In *European Conference on Computer Vision (ECCV)*, 2016.
34. Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
35. Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.
36. Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Computer Vision and Pattern Regognition (CVPR)*, 2017.
37. Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
38. Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*, 2018.
39. Christopher KI Williams. Computing with infinite networks. In *Advances in neural information processing systems*, pages 295–301, 1997.
40. Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. Deep geometric prior for surface reconstruction. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
41. Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NIPS*, 2016.
42. Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. FoldingNet: Point Cloud Auto-Encoder via Deep Grid Deformation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.