Statistical Edge Detection And UDF Learning For Shape Representation

Virgile Foy, Fabrice Gamboa, Reda Chhaibi

Statistical Edge Detection And UDF Learning For Shape Representation

Virgile Foy *[1], Fabrice Gamboa †[1], and Reda Chhaibi ‡[1]

[1]Institut de Mathématiques de Toulouse, Université Paul Sabatier, 118 Route de Narbonne, Toulouse, 31400, France

May 4, 2024

Abstract

In the field of computer vision, the numerical encoding of 3D surfaces is crucial. It is classical to represent surfaces with their Signed Distance Functions (SDFs) or Unsigned Distance Functions (UDFs) [AL20, CZ19, GFZ+20, MON+19]. For tasks like representation learning, surface classification, or surface reconstruction, this function can be learned by a neural network, called Neural Distance Function [CPM+20]. This network, and in particular its weights, may serve as a parametric and implicit representation for the surface. The network must represent the surface as accurately as possible. In this paper, we propose a method for learning UDFs that improves the fidelity of the obtained Neural UDF to the original 3D surface. The key idea of our method is to concentrate the learning effort of the Neural UDF on surface edges. More precisely, we show that sampling more training points around surface edges allows better local accuracy of the trained Neural UDF, and thus improves the global expressiveness of the Neural UDF in terms of Hausdorff distance.

To detect surface edges, we propose a new statistical method based on the calculation of a p-value at each point on the surface. Our method is shown to detect surface edges more accurately than a commonly used local geometric descriptor [PGK02].

Keywords: Computer Vision, Edge Detection, Goodness-Of-Fit Tests, Neural Networks, Representation Learning

*virgile.foy@math.univ-toulouse.fr

†https://www.math.univ-toulouse.fr/ gamboa/

‡https://www.math.univ-toulouse.fr/ rchhaibi/

Contents

1 Introduction

1.1 Industrial Context

Many problems in science and engineering require solving complex boundary value problems. The geometric shape of one part of a system may have a significant influence on the system's performance. For example, the geometric shape of the turbine blades of an aircraft engine significantly influence the engine's performance [Roy96]. In those cases, the geometry of the parts must be determined very precisely. The simulation codes used to design the parts can be very costly in terms of time and power [CSR24]. The ultimate goal of our work is to obtain real-time estimates of the outputs of these codes. This would significantly reduce the design time and allow better exploration of different geometries for potentially higher performance. Such approach has also motivated the seminal works [RLR+20] and [PFS+19] that have inspired our research.

In order to build such a model, we isolate the part from the industrial system and consider it as a shape in $\mathbb{R}^3$. We have access to a set of these shapes and we attempt to construct a parametric and implicit representation of these shapes. Parametric in the sense that each shape from the dataset is mapped to a combination of parameters called latent representation. Implicit means that the shapes are numerically encoded by their distance functions, from which they are the zero-level set (see Section 1.2). Practically, each shape is represented by the weights of a neural network called Neural Unsigned Distance Function (Neural UDF, see Sections 2.2 and 3.4). Each Neural UDF represents each shape because it has been trained to reproduce the shape's true UDF. This latent representation can be used as the input data of a regression model to predict numerical simulations outputs.

In this paper, we only focus on the problem of learning an UDF by a neural network. In particular, we propose to train the Neural UDF on points sampled in a smart way. Our objective is that the Neural UDF represents each shape more accurately.

1.2 Encoding 3D Shapes With Implicit Field

Some active research fields like computer vision, robotics, and numerical simulations require the ability to manipulate three-dimensional data. The 3D scenes to be processed consist of sets of 3D objects. Here we focus on the envelopes of these objects, specifically 3D watertight surfaces (see Definition 1.1).

Definition 1.1 (Watertight surface)
*Let $\mathcal{V}$ be a subset of $\mathbb{R}^3$ and $\mathcal{B}$ its boundary. $\mathcal{B}$ is a **watertight surface** i.i.f. the three following conditions are fulfilled:*

- *$\mathcal{V}$ is bounded*
- *$\mathcal{V}$ is a connected set*
- *$\mathbb{R}^D \setminus \mathcal{V}$ is a connected set*

Given a task to be performed on a set of surfaces (representation learning, regression, classification,...), a crucial preliminary question is the digital format used to encode the surfaces. This format may be imposed by the data source or chosen to suit the learning approach.

The most common representation is meshing. Meshes are particularly used in CFD or design. It is also well-suited for training graph networks [SGT+09, LCBF21, LFBC22]. In this work, we only used surface meshes in order to sample points from the surfaces and compute the surface UDF (see Section 3.3).

Point cloud is also an important type of 3D geometric data structure. This type of data is becoming increasingly available as acquisition democratizes with stereo cameras and LiDAR. However, extracting geometric features from a point cloud is challenging because the number of sampled points and the order in which they are recorded can vary within a given set of surfaces. Many Geometric Deep Learning approaches are based on PointNet model [QSMG17]. This model allows to learn shape representation for regression and classification tasks. In our work, we use point cloud representation for edge detection, surface reconstruction and visualization.

Other approaches are based on voxel grids [GYLB+19]. This format is well-suited for 3D convolutional networks but is very expensive in terms of memory. We did not use this approach here.

It is also possible to encode a 3D surface in the form of its distance function [AL20, CZ19, GFZ+20, MON+19]. More precisely, this function associates each point in the ambient space with its distance to the closest object in the surface. This function can be signed (Signed Distance Function, SDF) or unsigned (Unsigned Distance Function, UDF). These functions are formally defined in 1.2 and 1.3.

Definition 1.2 (Signed Distance Function)
*Let us consider a watertight surface $\mathcal{B} \subset \mathbb{R}^3$. As it is watertight, one can define $\mathcal{I}_\mathcal{B}$ (resp. $\mathcal{O}_\mathcal{B}$) the inner (resp. outer) part of $\mathcal{B}$. The **Signed Distance Function (SDF)** of $\mathcal{B}$ is the function* $\mathrm{SDF}_\mathcal{B} : \mathbb{R}^3 \to \mathbb{R}$ *such that:*

$$\forall \boldsymbol{x} \in \mathbb{R}^3,\ \mathrm{SDF}_\mathcal{B}(\boldsymbol{x}) := \begin{cases} d(x, \mathcal{B}) \text{ if } \boldsymbol{x} \in \mathcal{O}_\mathcal{B} \\ -d(x, \mathcal{B}) \text{ if } \boldsymbol{x} \in \mathcal{I}_\mathcal{B} \end{cases} \tag{1}$$

Definition 1.3 (Unsigned Distance Function)
*Let $\mathcal{B} \subset \mathbb{R}^3$ be a watertight surface. The **Unsigned Distance Function (UDF)** of $\mathcal{B}$ is the function* $\mathrm{UDF}_\mathcal{B} : \mathbb{R}^3 \to \mathbb{R}$ *such that:*

$$\forall \boldsymbol{x} \in \mathbb{R}^3,\ \mathrm{UDF}_\mathcal{B}(\boldsymbol{x}) := d(\boldsymbol{x}, \mathcal{B}). \tag{2}$$

The surface thus corresponds to the zero-level set of its distance function. We use this format because it gives a continuous representation and can be easily learned using a surrogate model (for example a neural network). In Section 1.3, we give a pratical description of a representation learning model based on SDFs. However, the contributions of this paper relate to methods based on UDFs.

1.3 *DeepSDF*: Autodecoder For Representation Learning

In [PFS+19], Park et. al. propose a model called *DeepSDF* for learning representation of 3D watertight surfaces based on implicit distance function and auto-decoder architecture. This model combines the following two ideas:

1. interpretation of a surface as the zero-level set of its SDF. Train a neural network to predict the values of this function on a dataset of points in $\mathbb{R}^3$.
2. learn a representation $\boldsymbol{z}$ of a surface (encoded by a mesh) in a low-dimensional space, as inspired by autoencoder models [Bal87a].

This Section gives a practical description of the methodology proposed in [PFS+19]. We make this description rough enough so that the main ideas can be easily understood. For more details, we refer to [PFS+19].

In Paragraph 1.3.1, we formulate the problem tackled by *DeepSDF* model. In Paragraph 1.3.2, we describe the dataset used to train the network. In Paragraph 1.3.3, we explain how the netowrk is trained. In Paragraph 1.3.4, we give the procedure used to map any new shape to a point in the latent space. In Paragraph 1.3.5, we explain how one can use DeepSDF model for shape generation.

1.3.1 Problem Statement

Consider a set of watertight surfaces $\mathcal{B}_1, ..., \mathcal{B}_N$ whose Signed Distance Functions $\text{SDF}_1, ..., \text{SDF}_N$ can be calculated at any point of $\mathbb{R}^3$. The SDF of a surface is formally defined in 1.2.

DeepSDF model consists of an auto-decoder network [ZLLM19]. This single network is able to

- estimate the SDF of each of the shapes $\mathcal{B}_1, ..., \mathcal{B}_N$ at any point in $\mathbb{R}^3$.
- maps the shapes $\mathcal{B}_1, ..., \mathcal{B}_N$ to latent codes $\boldsymbol{z_1}, ..., \boldsymbol{z_N} \in \mathbb{R}^L$. Latent dimension L is a hyperparameter of the model that needs to be fixed in advance.

1.3.2 Data preparation

The training dataset is composed of inputs and target ouputs. The inputs are the concatenation of:

- the spatial coordinates of a point in $\mathbb{R}^3$
- a latent code in $\mathbb{R}^L$.

For any point $\boldsymbol{x} \in \mathbb{R}^3$ and any latent code $\boldsymbol{z} \in \mathbb{R}^L$ (refering to a shape $\mathcal{B}$), the associated target output in the dataset is the value of the SDF of $\mathcal{B}$ at the point $\boldsymbol{x}$.

The training dataset $\boldsymbol{X}$ is built as the union of N sub-datasets - one for each shape:

$$\boldsymbol{X} = \bigcup_{j=1}^{N} \boldsymbol{X_j}. \tag{3}$$

Given a shape index $j \in [\![1, N[\![$, let us describe how to build the dataset $\boldsymbol{X_j}$ corresponding to the shape $\mathcal{B}_j$. A set of points $\boldsymbol{x_j^{(1)}}, ..., \boldsymbol{x_j^{(T)}}$ is sampled in the ambient space $\mathbb{R}^3$. These points are sampled close to the surface $\mathcal{B}_j$ [1] . The obtained training points constitute the data sets

$$\boldsymbol{X_j} := \{(\boldsymbol{x_j^{(t)}}, \text{SDF}_j(\boldsymbol{x_j^{(t)}})), 1 \leq t \leq T\},\ 1 \leq j \leq N. \tag{4}$$

1.3.3 Learning the latent representation

In this paragraph we describe how the latent representation of the training shapes $\mathcal{B}_1, ..., \mathcal{B}_N$ is learned by *DeepSDF* model.

DeepSDF is an auto-decoder network [ZLLM19]. In this architecture, there is no encoder network, *i.e.* there is no parametric function $\mathcal{B} \mapsto \boldsymbol{z}$ that directly encodes the shapes in the latent space.

[1] For the sake of simplicity, we do not dive into details regarding the way training points are sampled as it is the main topic of Section 3.2. For more information on this, we refer to the supplementary materials of [PFS+19].

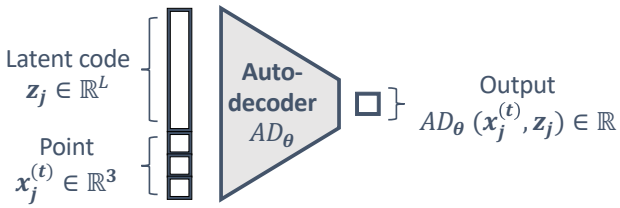


Figure 1: *DeepSDF* model architecture. Each training input is the concatenation of a point $\boldsymbol{x_j^{(t)}} \in \mathbb{R}^3$ and a latent code $\boldsymbol{z_j} \in \mathbb{R}^L$ (the one referring to the envelope $\mathcal{B}_j$). The value of L and the architecture of the network are hyperparameters of the model. The parameters optimized during the training step are both the latent codes $\boldsymbol{z_1}, ..., \boldsymbol{z_N}$ and the network's weights $\boldsymbol{\theta}$.

Let us denote the auto-decoder network AD_θ. θ refers to the weights of the network. The network architecture is depicted in Figure 1. The input of the network is a concatenation of a latent code and three point coordinates. The output of the network is a scalar value.

Given a batch of points $\boldsymbol{x_j^{(t)}} \in \mathbb{R}^3$ and latent codes $\boldsymbol{z_j} \in \mathbb{R}^L$ $(1 \leq j \leq N, 1 \leq t \leq T)$, the network is trained in order to minimize a loss which is the absolute difference between its outputs and the true SDF values:

$$(\boldsymbol{\theta^*}, \boldsymbol{z_1^*}, ..., \boldsymbol{z_L^*}) := \underset{\boldsymbol{\theta}, \boldsymbol{z_1}, ..., \boldsymbol{z_L}}{\text{argmin}} \sum_{t,j} |\ \text{AD}_\theta(\boldsymbol{x_j^{(t)}}, \boldsymbol{z_j}) - \text{SDF}_k(\boldsymbol{x_j^{(t)}})| \tag{5}$$

where $\boldsymbol{z_1}, ..., \boldsymbol{z_N} \in \mathbb{R}^L$ are the latent codes that refer to the shapes $\mathcal{B}_1, ..., \mathcal{B}_N$. They are initialized with values sampled from a Standard Gaussian law in $\mathbb{R}^L$.

A regularized term is used to force the distribution of the latent codes to keep close to a Standard Gaussian distribution [2] .

Once the network has been trained, the latent codes $\boldsymbol{z_1}, ..., \boldsymbol{z_N}$ obtained constitute the latent representation of the training shapes $\mathcal{B}_1, ..., \mathcal{B}_N$. In the context of regression or classification on the set of shapes, the latent codes are used as inputs to predict the labels associated with the shapes.

1.3.4 Latent code inference for unseen shapes

Let us consider a new surface $\mathcal{B}_{\text{new}} \notin \{\mathcal{B}_1, ..., \mathcal{B}_N\}$. In more common approaches like autoencoder [Bal87b], we would be able to compute directly its latent representation using the encoder network as the latter maps any shape to a latent code.

In the auto-decoder framework, the latent code $\boldsymbol{z_{\text{new}}}$ associated to $\mathcal{B}_{\text{new}}$ is computed using inference [3] . Practically, a latent vector is randomly initialized and then optimized so that the output of the network gets as close as possible to the SDF values of the sampled points for the given new shape. Formally, points $\boldsymbol{x_{\text{new}}^{(1)}}, ..., \boldsymbol{x_{\text{new}}^{(T)}}$ are sampled close to the surface.

[2] The main idea of the network optimization is summarized in Equation 5. For the sake of simplicity, we consider that Equation 5 summarizes well the network optimization. For more details, we refer to Section 4.2 of [PFS+19].

[3] Here the term inference refers to Bayesian inference which is the underlying theory used here. The Bayesian derivation of the auto-decoder-based DeepSDF model is described in Section H of the supplementary material of [PFS+19]. In our work, we only describe the model practically. As explained right after, the latent code associated to the new shape is optimized using gradient descent.

The procedure used to sample these points is the same as the one used to build the datasets $\boldsymbol{X_1}, ..., \boldsymbol{X_N}$ (see Paragraph 1.3.2). The obtained dataset can be defined as follows:

$$\boldsymbol{X_{new}} := \{(\boldsymbol{x_{new}^{(t)}}, \mathrm{SDF}_{\mathrm{new}}(\boldsymbol{x_{new}^{(t)}})), 1 \leq t \leq T\}. \tag{6}$$

Afterwards, the previously trained auto-decoder network is used with fixed parameter values to perform a gradient descent in $\mathbb{R}^L$ as follows:

$$\boldsymbol{z_{new}} := \underset{\boldsymbol{z} \in \mathbb{R}^L}{\mathrm{argmin}} \sum_t |\, \mathrm{AD}_{\theta^*}(\boldsymbol{x_{new}^{(t)}}, \boldsymbol{z}) - \mathrm{SDF}_{\mathrm{new}}(\boldsymbol{x_{new}^{(t)}})|. \tag{7}$$

1.3.5 Shape generation

Shape generation can be performed by picking some new point [4] in the latent space $\mathbb{R}^L$. Given points randomly sampled in $\mathbb{R}^3$ in $\mathbb{R}^3$, one can get an estimate of the SDF at these points. Neverthelss, effective methods to compute the zero-level set of a given function $\mathbb{R}^3 \to \mathbb{R}$ are discussed in [OFP04, RLR+20, LDG18]. Most of works in the field of computer vision use the Marching Cubes algorithm [LC87, LLVT03] on the Signed Distance Function. This algorithm is used to extract a 2D surface mesh from a 3D voxel grid. It can be conceptualized as a 3D generalization of isoline drawing on topographical maps. SDF values are incoded as a voxel grid and the algorithm looks for the voxels that cross the zero-level set of the SDF values.

1.4 Edge Detection

A surface edge is typically characterized as a tangential discontinuity. In [WHH10], Weber et al. describe edges as distinct features (such as peaks and valleys) where two planes meet, along with corners formed at the convergence of three or more planes. In [HWS16], Hackel et al. refer to these as "wire-frame contours", defining edges as lines where there is an abrupt change in the orientation (normals) of the underlying surface.

In this paper, we propose a method for improving the training of Neural UDFs by focusing the network's effort on surface edges. Before training, and even before building the training dataset, all types of surface edges (crests, valleys, peaks,...) need to be detected. When the surface is encoded as a point cloud, it is conventional to compute local geometric descriptors at each point of the cloud [HLP+21, DVVR07, PGK02, WJHM15, HWS16].

For example, in [PGK02], Pauly et al. perform local covariance analysis at each point in a point cloud to compute several local geometric descriptors. They aim to detect surface edges. It allows them to sample more points around surface edges than in areas in which the surface is planar or quasi-planar. By doing so, they optimize the sampling of surfaces. This is called surface simplification. In [WJHM15], Weinmann et al. use these local geometric descriptors to perform classification and shape matching.

Formally, consider a surface $\mathcal{B}$ numerically encoded by a point cloud $\boldsymbol{B}$. Let k be a positive integer and let denote $\boldsymbol{N_k}(\boldsymbol{x_0}) := (\boldsymbol{x_i})_{1 \leq i \leq k}$ the k neighborhoods of a given point $\boldsymbol{x_0}$ belonging to the surface. That is,

$$\boldsymbol{N_k}(\boldsymbol{x_0}) := \underset{\{\boldsymbol{y_1}, ..., \boldsymbol{y_k}\} \subset \boldsymbol{B}}{\mathrm{argmin}} \sum_{i=1}^{k} |\boldsymbol{y_i} - \boldsymbol{x_0}| := \{\boldsymbol{x_1}, ..., \boldsymbol{x_k}\}. \tag{8}$$

[4] The way the point is picked depends on the application. For example, if the latent representation of the shapes is used as an input for a surrogate model of numerical simulations, one can pick a point in the latent space that is mapped to good simulation outputs.

In this paper, $\boldsymbol{x_0}$ is referred to as the **centroid** of the **neighborhood** $\boldsymbol{N_k(x_0)}$. k represents the scale at which we detect edges. It is an important parameter of the method. In [HLP+21], Himeur et al. propose to compute some local descriptors at several scales *i.e.* for different values of k. In our work, k is a fixed parameter and its value is empirically chosen. We discuss further on the the value of k in Section 4.1.

Then, we can define the local covariance matrix

$$\mathcal{V} = \frac{1}{k+1}\sum_{i=0}^{k}(\boldsymbol{x_i} - \overline{\boldsymbol{x}})(\boldsymbol{x_i} - \overline{\boldsymbol{x}})^T \tag{9}$$

where

$$\overline{\boldsymbol{x}} := \frac{1}{k+1}\sum_{i=0}^{k}\boldsymbol{x_i} \tag{10}$$

is the barycenter of $\boldsymbol{N_k(x_0)} \cup \{\boldsymbol{x_0}\}$. The local covariance matrix (also called 3D structure tensor) is non negative-definite with three non-negative eigenvalues that correspond to an orthogonal system of eigenvectors. The three eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ of the local structure tensor give an idea of the local shape of the surface. In particular, **Pauly's descriptor** [5] of the surface in $\boldsymbol{x_0}$ is given by the following formula:

$$\epsilon_W(\boldsymbol{x_0}) := \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3} \tag{11}$$

This local descriptor measures the surface variation with respect to the tangent plane of the local neighborhood $\boldsymbol{N_k(x_0)}$. If the surface is locally planar around $\boldsymbol{x_0}$, then $\lambda_3 \ll \lambda_1, \lambda_2$ and thus $\epsilon_W(\boldsymbol{x_0}) \simeq 0$. If $\boldsymbol{x_0}$ is close to a surface edge, then the local 3D structure tensor is more likely to be isotropic. In this case, $\lambda_1 \simeq \lambda_2 \simeq \lambda_3$ and $\epsilon_W(\boldsymbol{x_0}) \simeq \frac{1}{3}$.

As explained in [WJHM15], Pauly's descriptor, combined with other geometric indicators, yields great performance in point cloud classification tasks. But it turns out that in some cases, this method does not identify the surface edges. For example, if $\boldsymbol{x_0}$ is located close to a very acute crest or valley, then the local 3D structure is similar to the one of a planar area [6] . Indeed, the average plane (carried by the two main axes of the eigen decomposition) can be orthogonal to the underlying surface, which invalidates the surface variation estimate made by Pauly's method.

In this paper, we propose an edge detector that is also based on neighborhood selection, but that uses statistical tools. Indeed, our local descriptor is the p-value of a statistical test and not directly a geometric descriptor. It is described in details in Sections 2.1 and 3.1.

1.5 Goodness-Of-Fit Tests For Sphericity

In this work, we propose a novel edge detection method. It is applied to a point cloud representing a surface. We compute a local descriptor at each point of the surface. This local descriptor is the p-value of a statistical test on the neighboring points after orthogonal projection on the average plane of the neighborhood (see Section 3.1). More precisely, the projected points are considered as *i.i.d* random vectors in $\mathbb{R}^2$ (projection onto the average plane). In this Section, we give a review of the statistical tests commonly used to assess the sphericity of a distribution, based on the empirical cumulative distribution.

[5] In [PGK02], this feature is called surface variation. In this paper, it is both referred as surface variation and Pauly's descriptor

[6] This limit of Pauly's descriptor is explained in more details in Appendix A.2.

1.5.1 Notations

Let $\boldsymbol{X_1}, ..., \boldsymbol{X_k} \in \mathbb{R}^2$ be a set of *i.i.d.* observations and $(r_1, \phi_1), ..., (r_k, \phi_k)$ their polar coordinates *w.r.t.* a given axis. We aim to test the following null hypothesis:

$$H_0 : \begin{cases} H_{0,1} : \phi_1 \text{ is uniformly distributed on } [-\pi, \pi) \\ H_{0,2} : r_1 \text{ and } \phi_1 \text{ are independent.} \end{cases} \tag{12}$$

Let us denote respectively F_k^r, F_k^ϕ and $F_k^{r,\phi}$ the empirical cumulative distribution functions (CDF) built on $(r_i)_{1\leq i\leq k}$, $(\phi_i)_{1\leq i\leq k}$ and $(r_i, \phi_i)_{1\leq i\leq k}$ respectively. That is,

$$\forall t \in \mathbb{R}_+, F_n^r(t) := \sum_{i=1}^{k} \mathbb{1}_{r_i\leq t}, \tag{13}$$

$$\forall \psi \in [-\pi, \pi), F_k^\phi(\psi) := \sum_{i=1}^{k} \mathbb{1}_{\phi_i\leq \psi}, \tag{14}$$

$$\forall (t, \psi) \in \mathbb{R}_+ \times [-\pi, \pi), F_k^{r,\phi}(t, \psi) := \sum_{i=1}^{k} \mathbb{1}_{r_i\leq t \text{ and } \phi_i\leq \psi}. \tag{15}$$

1.5.2 Goodness-Of-Fit Tests For Uniformity of the Polar Angles

The classical statistical methodology to validate or invalidate the null hypothesis $H_{0,1}$ (uniformity of the polar angles) consists to compute a discrepancy between the empirical distribution of the sample $(\phi_i)_{1\leq i\leq k}$ and the uniform distribution. To begin with, let F_0 be the CDF of the uniform distribution. That is,

$$\forall \psi \in [-\pi, \pi),\ F_0(\psi) = \frac{1}{2\pi}(\psi + \pi). \tag{16}$$

We will briefly discuss the classical goodness-of-fit test procedures.

This short review is based on the book [Y+95]. Following Y. Nikitin, we only focus on the test statistics that are based on the difference $F_k^\phi - F_0$. In [Kol33], Kolmogorov proposes the now classical statistic

$$A_k := \sup_{-\pi<\psi<+\pi} \left|F_k^\phi(\psi) - F_0(\psi)\right| \tag{17}$$

In [Dar83], Darling introduced the centered variant of the Kolmogorov statistics, which is usually called Watson-Darling statistic:

$$B_k := \sup_{-\pi<\psi<+\pi} \left|F_k^\phi(\psi) - F_0(\psi) - \int_{-\pi}^{+\pi} (F_k^\phi(\rho) - F_0(\rho))dF_0(\rho)\right| \tag{18}$$

Some other statistics are based on the L^2-norm of the difference $F_k^\phi - F_0$. For example the Cramér-Von-Mises-Smirnov statistic is defined as follow:

$$C_k := \int_{-\infty}^{+\infty} (F_k^\phi(\psi) - F_0(\psi))^2 dF_0(\psi) \tag{19}$$

These statistics tend to be more efficient than the Kolmogorov-Smirnov-based ones as they need smaller samples to converge. In the other hand, they involve integrating the squared

difference between the empirical and theoretical CDFs, which can be computationally more intensive, especially for large sample sizes. Moreover, they are less sensitive to small deviations from the null hypothesis.

In [AD52], Anderson and Darling propose to generalize the previous statistic to any L^α-norm ($\alpha > 0$) and to improve its properties by using a non-negative weight function q as follows:

$$D_k^{\alpha,q} := \int_{-\pi}^{+\pi} (F_k^\phi(\psi) - F_0(\psi))^\alpha q(F_0(\psi)) dF_0(\psi). \tag{20}$$

The incorporation of the weight function allows for more flexibility in handling data with varying importance or significance. This makes it suitable for situations where certain observations may be more influential than others in assessing the goodness-of-fit.

In terms of consistency and sensitivity to tail behavior, the best known weighted statistic is the Anderson-Darling statistic (see [AD54]):

$$E_k := \int_{-\pi}^{+\pi} \frac{(F_k^\phi(\psi) - F_0(\psi))^2}{F_0(\psi)(1 - F_0(\psi))} dF_0(\psi). \tag{21}$$

All the aforementioned test statistics can be compared in terms of Bahadur efficiency. The latter measures the rate at which the test statistic converges to its expected value under H_0. For the statistics described in this Section, the expected value under H_0 is zero. One can refer to [Bah71] for more details on Bahadur efficiency.

1.5.3 Centered Cramer-Von-Mises Test On The Unit Circle

In [Smi77] the author proposes a test statistics that estimates directly and globally the spherical symmetry of the underlying distribution. Let describe his procedure. First let introduce the following quantity:

$$X_k(r, \psi) := F_k^{r,\psi}(r, \psi) - \frac{\psi}{2\pi} F_k^r(r) \tag{22}$$

In order that the test statistics does not depend on the chosen polar axis, he defines the following centered version:

$$Y_k(r, \psi) := X_k(r, \psi) - \frac{1}{2\pi} \int_{0 \le t \le 2\pi} X_k(r, t) dt \tag{23}$$

Under H_0, $Y_k(r, \psi)$ is close to 0. This last observation motivates the following test statistic proposed in [Smi77]:

$$U_k := k \iint Y_k^2(r, \psi) dF_k^{r,\psi}(r, \psi). \tag{24}$$

In our work, we use the Kolmogorov test statistic (see Equation 17) applied to the angular coordinate of 2D points (see Section 3.1). We will proceed in order to avoid the drawback of a test statistics depending on the chosen origin on the circle (*i.e.* the choice of polar axis $\phi = 0$). Practically, instead of using the statistics proposed in Equation 18, we center the data using their Fréchet mean. The centering procedure is described in Section 2.1.4.

The decision to validate or reject H_0 is based on the p-value of the test. It is defined as the probability that the test statistic A_k (considered as a random variable) is greater or equal to its realization a, under H_0. That is:

$$p\text{-value} := \mathbb{P}(A_k \geq a \mid H_0). \tag{25}$$

1.6 Distance between Point Clouds

In this Section, we recall some classical distances between two point clouds. Let $\boldsymbol{A_1} \in \mathbb{R}^{n_1 \times D}, \boldsymbol{A_2} \in \mathbb{R}^{n_2 \times D}$ be two point clouds. Here, $D \in \{2, 3\}$ is the dimension of the ambient space and n_1, n_2 are the number of points in $\boldsymbol{A_1}$ and $\boldsymbol{A_2}$. The three following distances are quite popular:

Definition 1.4 (Hausdorff distance)

$$d_H(\boldsymbol{A_1}, \boldsymbol{A_2}) := \max(\max_{\boldsymbol{a} \in \boldsymbol{A_1}} d(\boldsymbol{a}, \boldsymbol{A_2}), \max_{\boldsymbol{a} \in \boldsymbol{A_2}} d(\boldsymbol{a}, \boldsymbol{A_1})). \tag{26}$$

Definition 1.5 (Chamfer distance)

$$d_{Ch}(\boldsymbol{A_1}, \boldsymbol{A_2}) := \frac{1}{n_1} \sum_{\boldsymbol{a} \in \boldsymbol{A_1}} d(\boldsymbol{a}, \boldsymbol{A_2}) + \frac{1}{n_2} \sum_{\boldsymbol{a} \in \boldsymbol{A_2}} d(\boldsymbol{a}, \boldsymbol{A_1}). \tag{27}$$

Definition 1.6 (Wasserstein distance)

$$d_W(\boldsymbol{A_1}, \boldsymbol{A_2}) := \min_{\xi \in Bij} \sum_{\boldsymbol{a} \in \boldsymbol{A_1}} \|\boldsymbol{a} - \xi(\boldsymbol{a})\|_2. \tag{28}$$

Here, Bij denotes the set of all one to one applications from $\boldsymbol{A_1}$ onto $\boldsymbol{A_2}$.

In Geometric Deep Learning, it is a common practice to use Chamfer distance (Definition 1.5) or Wasserstein distance (Definition 1.6) to train neural networks (by incorporating them into the loss calculation) or to evaluate them (by computing them after training) [ADMG18, FSG17, GFK+18, LCL18, PFS+19]. These distances give an idea of the dissimilarity between two point clouds, averaged over all the underlying shapes. Here, we use the Hausdorff distance [HKR93] (Definition 1.4) because it measures the local dissimilarity in the area where it is the highest. In the context of Neural UDF evaluation, it measures the highest local reconstruction error. In this sense, Hausdorff distance is considered more restrictive and is preferred from other distances.

2 Contributions

In our work, we propose a new statistical method to achieve edge detection on surfaces encoded as unstructured point clouds. We show that it can be used to improve the training procedure of 3D surface Neural UDFs.

In this Section, we formalize our main contributions. In Section 2.1, we describe our statistical edge detection method. In Section 2.2, we discuss the task of UDF learning. Finally, the evaluation of the trained Neural UDFs is formalized in Section 2.3.

2.1 Statistical Edge Detection

In this Section, we discuss the problem of edge detection. Let us consider a watertight surface $\mathcal{B} \subset \mathbb{R}^3$ (see Definition 1.1) encoded as a point cloud $\boldsymbol{B} \in \mathbb{R}^{n_s \times 3}$ containing n_s points in $\mathbb{R}^3$. The points in the discrete set $\boldsymbol{B}$ are assumed to be uniformly sampled from the continuous surface $\mathcal{B}$.

Given a point $\boldsymbol{x_0} \in \mathcal{B}$ (this point is called **centroid**), we wish to know if $\boldsymbol{x_0}$ is located on an edge of the surface. Let us extract from $\boldsymbol{B}$ the k nearest neighbors of $\boldsymbol{x_0}$, as formalized in Equation 8: these points, denoted $\boldsymbol{x_1}, ..., \boldsymbol{x_k}$, constitute the **neighborhood** of $\boldsymbol{x_0}$

The intuition that led to the edge detector described in this Section is inspired from the local surface variation descriptor proposed by Pauly et al. [PGK02]. Our edge detector is based on an analysis of the projections of the points $\boldsymbol{x_0}, ..., \boldsymbol{x_k}$ on their **average plane** (defined in Section 2.1.1). If the surface $\mathcal{B}$ is planar or quasi-planar (*situation 0*) around $\boldsymbol{x_0}$, then the average plane is locally tangent to the surface. If the surface is very sharp (*situation 1*) or folded around $\boldsymbol{x_0}$, then the average plane is not tangent to the surface locally (see Figure 2). In order to differentiate between situations 0 and 1, we perform an orthogonal projection of the points $\boldsymbol{x_0}, ..., \boldsymbol{x_k}$ onto their average plane. This projection is formally described in Section 2.1.1. The projected points are denoted $\boldsymbol{x'_0}, ..., \boldsymbol{x'_k}$. It can be visualized on Figure 2 that in *situation 0*, $\boldsymbol{x'_0}$ lies in the middle of $\boldsymbol{x'_1}, ..., \boldsymbol{x'_k}$. Whereas in *situation 1*, $\boldsymbol{x'_0}$ is off-centered with respect to the other projected points (see Figure 2).

The key point of our descriptor is to discriminate between these two situations using a statistical test of central symmetry on the projections of the points, with respect to the projection of $\boldsymbol{x_0}$ (that is considered as the potential center of symmetry under null hypothesis). We use the obtained p-value to discriminate between situations 0 and 1. This step is detailed in Section 2.1.2.

The statistical test applied to the projections depends on reference frame for polar coordinates. In order to make it independent from this this reference frame, we center the angular coordinates with respect to their Fréchet mean. This is described in details in Section 2.1.4.

2.1.1 Projection of the neighboring points on the average plane

Given a set of points $\boldsymbol{x_0}, ..., \boldsymbol{x_k} \in \mathbb{R}^3$, we aim to compute their projections $\boldsymbol{x'_0}, ..., \boldsymbol{x'_k} \in \mathbb{R}^2$ on their average plane.

We define the average plane as the affine hyperplane of $\mathbb{R}^3$ containing the centroid point and orthogonal to the third eigenvector of the local covariance matrix. This matrix is computed on the points $\boldsymbol{x_0}, ..., \boldsymbol{x_k}$ according to Equation 9. Roughly speaking, the average plane is the one in which the local point cloud extends the most (we refer to [Hot33] for more details on the average plane). Let us denote $\boldsymbol{e_1}$ and $\boldsymbol{e_2}$ the first two eigenvectors (with largest eigenvalues) of the local covariance matrix. The obtained 2D points can be defined as follows:

$$\forall i \in [\![0, k]\!],\ \boldsymbol{x'_i} := \begin{pmatrix} \boldsymbol{x_i} \cdot \boldsymbol{e_1} \\ \boldsymbol{x_i} \cdot \boldsymbol{e_2} \end{pmatrix} \tag{29}$$

The next step consists in performing a central symmetry test on the set of points $\boldsymbol{x'_0}, ..., \boldsymbol{x'_k}$.

2.1.2 Central symmetry test on the projected points

Given a set of points $\boldsymbol{x'_0}, ..., \boldsymbol{x'_k}$ on a hyperplane of $\mathbb{R}^3$, we aim to know if the points $\boldsymbol{x'_1}, ..., \boldsymbol{x'_k}$ are evenly distributed around $\boldsymbol{x'_0}$ (see Figure 2).

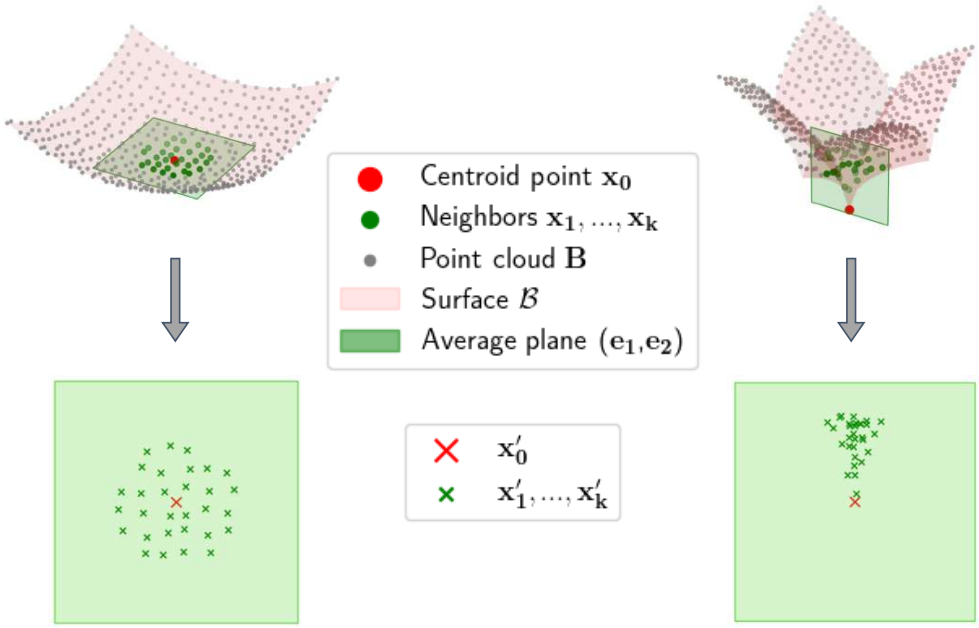


Figure 2: Projection onto the local average plane. On the left, a quasi-plane surface (*situation 0*), and on the right, a pointed surface (*situation 1*). On the top row, the surface $\mathcal{B}$ is in transparent red color. Points (in grey) are uniformly sampled from the surface. We aim to know if the surface is folded or pointed in $\boldsymbol{x_0} \in \mathcal{B}$. First, we compute its k nearest neighbors $\boldsymbol{x_1}, ..., \boldsymbol{x_k}$ (green points). Then the average plane of the neighboring points is computed (in transparent green color). On the bottom row, the plots represent the projection of $\boldsymbol{x_0}, ..., \boldsymbol{x_k}$, denoted $\boldsymbol{x'_0}, ...\boldsymbol{x'_k}$. We observe that in *situation 0*, the projection $\boldsymbol{x'_0}$ is in the middle of the neighbors' projections $\boldsymbol{x'_1}, ...\boldsymbol{x'_k}$. Whereas in *situation 1*, it is completely off-centered.

Let us consider the centered vectors $(\boldsymbol{X_i})_{1\leq i\leq k}$:

$$\forall i \in [\![1,k]\!],\ \boldsymbol{X_i} := \boldsymbol{x'_i} - \boldsymbol{x'_0}. \tag{30}$$

In our modelling we consider $\boldsymbol{X_1}, ..., \boldsymbol{X_k}$ as *i.i.d* realizations of a 2D random vector. Our approach is to assess the central symmetry of this random vector around the origin. We rely on the method described in Section 1.5. We use Kolmogorov-Smirnov's test statistics A_k defined in Equation 17 on the empirical CDF defined in Equation 14, to validate or reject the null hypothesis $H_{0,1}$ (see (12)). The p-value of the test is computed as in Equation 25. Note that the polar angular coordinates are first centered with respect to their Fréchet mean. Kolmogorov-Smirnov's test is performed on the centered angles. Fréchet centering is described in details in Section 2.1.4.

2.1.3 Kolmogorov-Smirnov's descriptor

If the p-value is higher than a given threshold p_0, we retain the null hypothesis, and consider that the angles are uniformly distributed. This supports the conclusion that the surface is locally quasi-planar. On the contrary, if the p-value is smaller than p_0, we conclude that the surface is pointed or folded (see Figure 2). The computed p-value is thresholded using a hyperparameter $p_0 \in [0,1]$. This allows to define our statistical edge detector, also called **Kolmogorov-Smirnov's descriptor**:

$$\forall \boldsymbol{x} \in \boldsymbol{B},\ \epsilon_{\mathrm{KS}}(\boldsymbol{x}) := \begin{cases} 1 \text{ if } p\text{-value} \leq p_0. \\ 0 \text{ if } p\text{-value} > p_0. \end{cases} \tag{31}$$

The threshold value p_0 is chosen empirically. We discuss the choice of p_0 in Section 4.1.

2.1.4 Fréchet Centering

In our edge detection method, we calculate the p-value associated with a central symmetry test on the underlying distribution of the projected points in the average plane, seen as a k-sample $\boldsymbol{X_1}, ..., \boldsymbol{X_k} \in \mathbb{R}^2$. In order to do so, we use the polar coordinates of the observations. In particular, we perform a goodness-of-fit test on the empirical law of the polar angle (coordinate ϕ) against the uniform law (as mentioned in the previous Section 2.1.2, and formalized in Section 1.5).

The projected points $\boldsymbol{X_1}, ..., \boldsymbol{X_k}$ are initially written in the Cartesian reference system $(O, \boldsymbol{e_1}, \boldsymbol{e_2})$ (see Equations 29 and 30). It is therefore natural to compute the polar coordinates $(r_1, \phi_1), ..., (r_k, \phi_k)$ as follows:

$$\forall i \in [\![1,k]\!],\ \begin{cases} r_k & := \|X_k\|_2 \\ \phi_k & := \widehat{\boldsymbol{e_1}, \boldsymbol{X_k}} \end{cases} \tag{32}$$

where $\widehat{\boldsymbol{e_1}, \boldsymbol{X}}$ is the oriented angle between $\boldsymbol{e_1}$ and $\boldsymbol{X}$, taken from the interval $[-\pi, \pi)$.

Thus, the polar angle (coordinate ϕ) is defined with respect to a reference axis, and it turns out that the p-value of the aforementioned goodness-of-fit is highly dependent on this reference axis. Figure 3 shows four different situations. They all correspond to a surface edge but the projected points are distributed in a different way on the average hyperplane (in the green rectangles). In the context of edge detection, we would expect to obtain roughly the same p-value for the four situations, but the reference axes $\boldsymbol{e_1}$ and $\boldsymbol{e_2}$ have different orientations, resulting in different angle distributions (on the histograms).

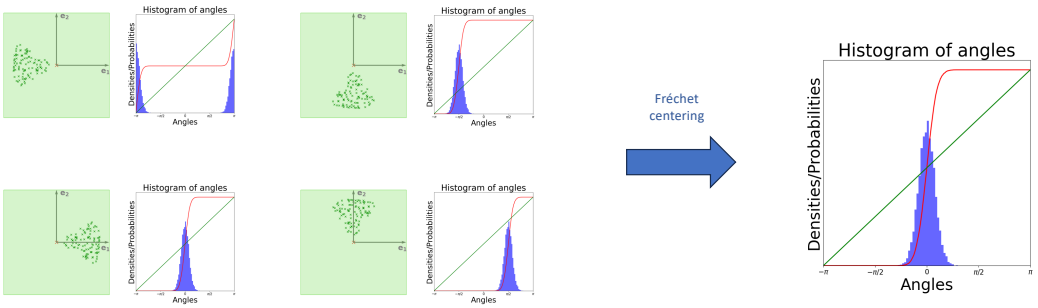


Figure 3: Fréchet centering of polar angles in the average hyperplane. The four green plots on the left correspond to the projections of the points of four point clouds onto their mean hyperplanes. In each situation, the projection of the centroid point is represented by a red cross in the center. The green crosses represent the projections of the neighboring points. In all four situations, the projection of the centroid point (red cross) is off-centered with respect to the projections of the neighboring points (green crosses). These are therefore four surface edges and the obtained p-values are expected to be roughly the same. However, the four point clouds are positioned differently with respect to the polar reference axis $\boldsymbol{e_1}$ (corresponding to $\phi = 0$ in polar coordinates). The distribution of polar angles $\phi_1, ..., \phi_k$ is therefore different in the four situations (see the four associated histograms). This results in different p-values for each situation. By centering the angles around their Fréchet mean, we obtain the same distribution centered at 0, as in the resulting histogram on the right. Thus, we obtain the same p-value in all four situations, which makes our method agnostic to any polar reference axis.

To overcome this problem, we center the angles $\phi_1, ..., \phi_k$ with respect to their Fréchet mean [Fré48] before testing the central symmetry of the observations. For the sake of formalization, we define the angular distance

$$\forall \psi_1, \psi_2 \in [-\pi, \pi),\ d_{\text{ang}}(\psi_1, \psi_2) := \min(|\psi_1 - \psi_2|, 2\pi - |\psi_1 - \psi_2|). \tag{33}$$

This distance can be conceptualized as the arclength distance between points along the unit circle. We use a generalization of the Euclidean mean, called Fréchet mean [Fré48], on the angular distance:

$$\overline{\phi}_F := \underset{\psi}{\text{argmin}} \sum_{i=1}^{n} d_{\text{ang}}(\phi_i, \psi)^2. \tag{34}$$

All angles are centered with respect to the Fréchet mean:

$$\forall i \in [\![1, k]\!],\ \phi_i' := \phi_i - \overline{\phi}_F \tag{35}$$

Visually, this amounts to using a reference axis that depends on the angles $\phi_1, ..., \phi_k$. More precisely, they are centered with respect to the average direction of the projected point cloud.

Note that the uniqueness of the Fréchet mean in not guaranteed for any set of observations. For example, if the angles $\phi_1, ...\phi_k$ are distributed in a perfectly even way on $[-\pi, \pi)$, then their Fréchet mean is not well defined. In our work, we assume that this does not happen and that the Fréchet mean is always well defined. One can refer to [Cha13] for a more detailed work on the uniqueness of the Frechet mean on the unit circle.

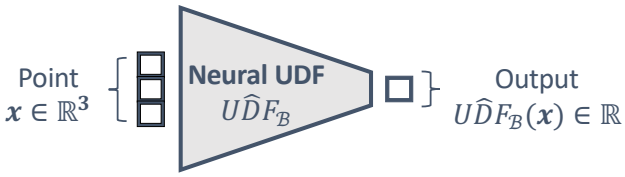


Figure 4: Neural UDF scheme for a watertight surface $\mathcal{B}$. The inputs of the network are tridimensional: they represents points in $\mathbb{R}^3$. The outputs are real values and the network is trained to fit the true UDF of the surface $\mathcal{B}$.

2.2 UDF Learning

In this Section, we discuss the UDF learning task. It can be formulated as follows: given a watertight surface $\mathcal{B} \subset \mathbb{R}^3$ (Definition 1.1), we aim to train a neural network to reproduce its UDF (Definition 1.3).

The function $\mathrm{UDF}_{\mathcal{B}}$ is an implicit representation of the surface $\mathcal{B}$ in the sense that it can not directly allow to draw the surface. Nevertheless, the *geometry* of the surface is contained in its UDF as its zero-level set:

$$\mathcal{B} = \mathrm{UDF}_{\mathcal{B}}^{-1}(\{0\}). \tag{36}$$

Let us consider a fixed network architecture considered as a family of functions $\mathcal{F}$. The architecture is roughly described in Figure 4. $\mathcal{F}$ is parameterized by the weights of the network. The **Neural UDF** training consists in the following optimization:

$$\widehat{\mathrm{UDF}}_{\mathcal{B}} := \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{\boldsymbol{x} \in \boldsymbol{X}} (f(x) - \mathrm{UDF}_{\mathcal{B}}(\boldsymbol{x}))^2. \tag{37}$$

where $\mathcal{B}$ and $\boldsymbol{X} \in \mathbb{R}^{n \times 3}$ is the training dataset. The procedure used to sample the training points $\boldsymbol{X}$ is discussed in details in Section 3.2. $\widehat{\mathrm{UDF}}_{\mathcal{B}}$ is called the Neural UDF of the surface $\mathcal{B}$.

2.3 Neural UDF Evaluation

In order to evaluate the Neural UDF of a given surface, we measure its ability to represent the surface. Consider a neural UDF $\widehat{\mathrm{UDF}}_{\mathcal{B}}$ that has been trained on a given surface $\mathcal{B}$. The evaluation task consists in quantifying the difference between the zero-level set of the Neural UDF and the true surface $\mathcal{B}$. In this Section, we explain how we estimate the Neural UDF's zero-level set and define the metric used to evaluate the neural UDF's accuracy.

For the sake of formalization, we define the zero-level set of the Neural UDF $\widehat{\mathrm{UDF}}_{\mathcal{B}}$ as follows:

$$\widehat{\mathcal{B}} := \widehat{\mathrm{UDF}}_{\mathcal{B}}^{-1}(\{0\}). \tag{38}$$

Effective methods to compute the zero-level set of a given function $\mathbb{R}^3 \to \mathbb{R}$ are discussed in Section 1.3.5. The mentioned methods require fine tuning to yield good results in a reasonable running time in our application. Moreover, they are commonly based on the Signed Distance Functions but here we only have access to the Unsigned Distance Functions.

In this work, we use another method: our idea is based on the fact that the Neural UDF is differentiable with respect to its inputs (spatial coordinates). A set $\boldsymbol{B_r}$ of n_r points is

sampled from the true surface $\mathcal{B}$ and is used as initial guess in the following optimization problem:

$$\widehat{\boldsymbol{B}} := \underset{(\boldsymbol{b_1},\dots,\boldsymbol{b_{n_r}})\in\mathbb{R}^{n_r\times 3}}{\operatorname{argmin}} \sum_{i=1}^{n_r} |\widehat{\mathrm{UDF}}_{\mathcal{B}}(\boldsymbol{b_i})|. \tag{39}$$

The optimization method is a gradient descent. The obtained set of points $\widehat{\boldsymbol{B}}$ is then compared to the initial set of points $\boldsymbol{B_r}$ sampled from the true surface. The difference between $\widehat{\mathcal{B}}$ and the true surface $\mathcal{B}$ is approximated using the Hausdorff distance between the point clouds $\boldsymbol{B_r}$ and $\widehat{\boldsymbol{B}}$. This metric is defined in Section 1.6. Therefore, we define the **reconstruction error** of the Neural UDF as

$$\delta(\widehat{\mathrm{UDF}}_{\mathcal{B}}) := d_H(\boldsymbol{B_r}, \widehat{\boldsymbol{B}}). \tag{40}$$

The reconstruction method we describe here has also some limitations. It highly depends on the initialization step *i.e.* on the way $\boldsymbol{B_r}$ is built. Indeed, we initialize the points on the true surface $\mathcal{B}$. If the Neural UDF is accurate enough, its zero-level set can be assumed to be close to the true surface $\mathcal{B}$. This means that the estimated zero-level set $\widehat{\boldsymbol{B}}$ is potentially searched without exploring a large part of the ambient space $\mathbb{R}^3$. Furthermore, a large number of points are needed to cover the entire zero-level set of the Neural UDF.

3 Methodology

Neural UDF training and its evaluation consist of several steps. The steps that constitute our main contributions are discussed in the previous Section. In this Section, we present the step-by-step computational implementation of our method and and its evaluation.

Recall that, given a surface $\mathcal{B}$, our aim is to train its Neural UDF. To measure the accuracy of the estimated Neural UDF, we need to compute the distance between the Neural UDF's zero-level set and the initial surface $\mathcal{B}$. Neural UDF training can be broken down into several sub-steps. First, we implement a function to detect pointed or folded areas (edges) on the surface (see Section 3.1). We then use this function to build a set of training points whose distribution is more concentrated around these edges (see Section 3.2).

Afterwards, we compute the true UDF of the surface at any point in space, using the mesh of $\mathcal{B}$ (see Section 3.3). Finally, we train a neural network to learn the true UDF. Once trained, this neural network is called a Neural UDF (see Section 3.4).

To measure the performances of our learning method, we use the trained Neural UDF to reconstruct the true surface and compare the obtained output with the true surface. This evaluation procedure is described in Section 3.5. To perform this validation step, we generate points on the zero-level set of the Neural UDF. Next, we measure the distance between these points from the original surface.

The whole pipeline for both training and evaluation of the Neural UDF is inherently random. Therefore, the results can vary from a run to another. In order to measure statistically the improvement due to our method, we repeat the experiments several times and compute the median value of the series of improvement measurements obtained. And we achieve this for several subsets of shapes from ShapeNet dataset [CFG+15]. This is explained in Section 3.6.

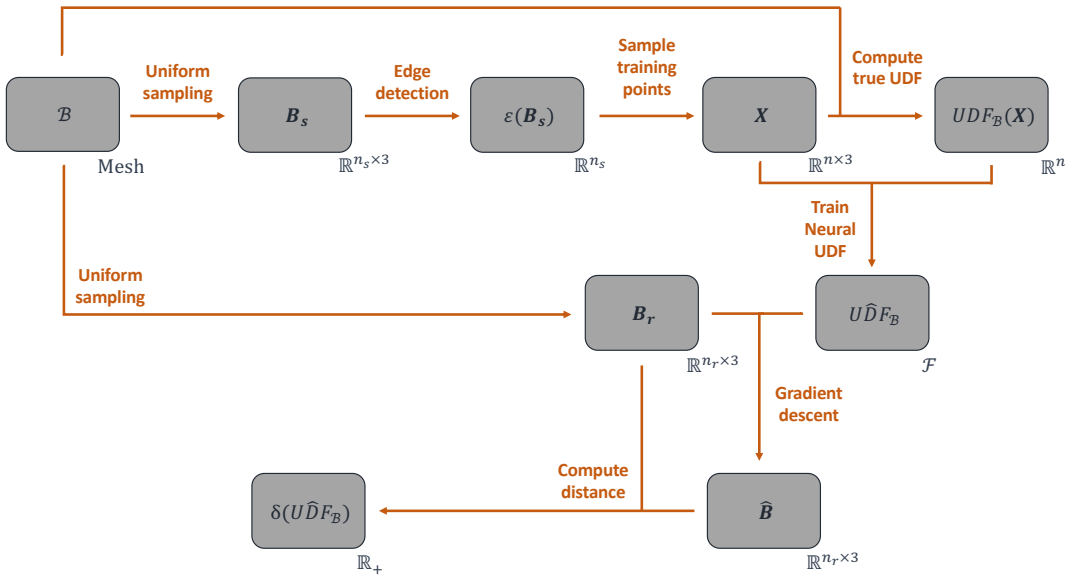


Figure 5: Neural UDF training and evaluation: methodology scheme. Given a 3D surface $\mathcal{B}$ encoded as a mesh, we aim to train its Neural UDF and evaluate its precision.

3.1 Detecting edges

In this Section, we describe the method used to detect edges on a surface and also explain how we choose the values of the involved parameters. Given a surface $\mathcal{B}$, the problem amounts in looking for the mapping:

$$\forall \boldsymbol{x} \in \mathcal{B},\ \epsilon(\boldsymbol{x}) := \begin{cases} 1 \text{ if } \boldsymbol{x} \text{ is close to an edge of } \mathcal{B}. \\ 0 \text{ if } \mathcal{B} \text{ is locally planar or quasi-planar in } \boldsymbol{x}. \end{cases} \tag{41}$$

First, we sample points uniformly from the surface. We denote $\boldsymbol{B_s} \in \mathbb{R}^{n_s \times 3}$ the point cloud thus generated. Given $\boldsymbol{x} \in \boldsymbol{B_s}$, we select the k nearest neighbors of $\boldsymbol{x}$. The resulting points are then projected onto their average plane. A statistical test of central symmetry is performed on the resulting projections. The computed p-value is thresholded using a parameter $p_0 \in [0, 1]$. This procedure is described in detail in Section 2.1.

The three parameters n_s, k and p_0 must be adjusted for each type of surface. For surfaces in ShapeNet dataset, we set $n_s = 2000$, $k = 40$ and $p_0 = 0.2$. The choice of these values is discussed in Section 4.1.

3.2 Sampling the Training Points

In this Section, we describe the procedure used to sample the points used for Neural UDF training. We will also introduce and discuss the parameters used to configure this sampling procedure. We consider a watertight surface $\mathcal{B}$, and the same set of points $\boldsymbol{B_s} \in \mathbb{R}^{n_s \times 3}$ uniformly sampled on $\mathcal{B}$, as introduced in the previous section. It is assumed that the surface has been previously normalized in order to be a subset of the unit ball

$$\mathcal{B}_{\|\cdot\|_2} = \{\boldsymbol{x} \in \mathbb{R}^3,\ \|\boldsymbol{x}\| \leq 1\}. \tag{42}$$

Let denote the training dataset $\boldsymbol{X_0} \in \mathbb{R}^{n \times 3}$. It is an n-sized sample with probability distribution $\mathcal{L}(\mathbb{R}^3)$. Hereafter we define this probability distribution.

The training points need to be sampled everywhere in the unit ball $\mathcal{B}_{\|\cdot\|_2}$ and also on the surface $\mathcal{B}$. Hence, we build $\mathcal{L}(\mathbb{R}^3)$ as the mixture of two sub-distributions:

$$\mathcal{L}(\mathbb{R}^3) = (1-\nu)\mathcal{U}(\mathcal{B}_{\|\cdot\|_2}) + \nu\mathcal{L}(\mathcal{B}). \tag{43}$$

Here, $\mathcal{U}(\mathcal{B}_{\|\cdot\|_2})$ is the uniform distribution on the unit ball, $\mathcal{L}(\mathcal{B})$ is a probability distribution on the surface $\mathcal{B}$ that is defined below (see Equation 45), and $\nu \in [0,1]$ is a parameter that quantifies how the training effort of the Neural UDF is concentrated on the surface $\mathcal{B}$. If $\nu = 1$, the training points are all sampled on the surface. On the contrary, if $\nu = 0$, the training points are all uniformly sampled from the unit ball $\mathcal{B}_{\|\cdot\|_2}$.

The points that are sampled from the surface can be either located near a surface edge or in a planar area. In order to improve the accuracy of the Neural UDF, we concentrate the training effort around surface edges, *i.e.* around points $\boldsymbol{b} \in \boldsymbol{B_s}$ such that $\epsilon_{\text{KS}}(\boldsymbol{b}) = 1$. For the sake of formalization, let define the following sets:

$$\boldsymbol{B_{s,i}} := \{\boldsymbol{b} \in \boldsymbol{B_s}, \epsilon_{\text{KS}}(\boldsymbol{b}) = i\}, i \in \{0,1\}. \tag{44}$$

In order to simplify the sampling procedure implementation, the points sampled from the surface are picked from the sets $\boldsymbol{B_{s,0}}$ and $\boldsymbol{B_{s,1}}$. Therefore, we can write

$$\mathcal{L}(\mathcal{B}) = (1-\nu_1(\mathcal{B}))\mathcal{U}(\boldsymbol{B_{s,0}}) + \nu_1(\mathcal{B})\mathcal{U}(\boldsymbol{B_{s,1}}) \tag{45}$$

where $\mathcal{U}(\boldsymbol{B_{s,0}})$ (*resp.* $\mathcal{U}(\boldsymbol{B_{s,1}})$) is the uniform distribution on the point cloud $\boldsymbol{B_{s,0}}$ (*resp.* $\boldsymbol{B_{s,1}}$) and $\nu_1(\mathcal{B}) \in [0,1]$ is a scalar that measures the proportion of the surface training points located close to surface edges.

In our experiments, we train Neural UDFs for several surfaces that can have different geometries. Some may have more edges than others. In other words, surfaces can have varying degrees of complexity. We formally define the complexity of the surface $\mathcal{B}$ as the value

$$\tau(\mathcal{B}) := \frac{1}{n_s}\sum_{\boldsymbol{b}\in\boldsymbol{B_s}} \epsilon_{\text{KS}}(\boldsymbol{b}) = \frac{\text{Card}(\boldsymbol{B_{s,1}})}{\text{Card}(\boldsymbol{B_s})} \tag{46}$$

To measure the influence of our surface edge detection method on the accuracy of Neural UDFs on a set of surfaces, we need to set a fixed oversampling parameter across all the surfaces. This oversampling parameter must locate the value $\nu_1(\mathcal{B})$ between the degree of complexity $\tau(\mathcal{B})$ (uniform sampling on the whole surface) and 1 (sampling on surface edges only). This amounts in setting a parameter $\xi \in [0,1]$ such that for any surface $\mathcal{B}$,

$$\nu_1(\mathcal{B}) = \xi + (1-\xi)\tau(\mathcal{B}). \tag{47}$$

One can see that if $\xi = 0$, then $\nu_1(\mathcal{B}) = \tau(\mathcal{B})$, which means that the training points sampled on the surface are uniformly sampled on the whole surface (without oversampling the edges). On the contrary, if $\xi = 1$, then $\nu_1(\mathcal{B}) = 1$ and all the points sampled on the surface are sampled on surface edges.

In the dataset $\boldsymbol{X_0}$ constructed using the aforementioned sampling procedure, a large proportion of the points are sampled from the surface $\mathcal{B}$. Indeed, the laws $\mathcal{U}(\boldsymbol{B_{s,0}})$ and $\mathcal{U}(\boldsymbol{B_{s,1}})$ are distributions on the surface $\mathcal{B}$, since $\boldsymbol{B_{s,0}} \subset \mathcal{B}$ and $\boldsymbol{B_{s,1}} \subset \mathcal{B}$. Thus, the target output for these points is zero, as their UDF is equal to zero. To prevent the network from converging towards the trivial null solution, we perturb the points with a Gaussian noise. Hence the dataset rewrites as

$$\boldsymbol{X} := \boldsymbol{X_0} + \boldsymbol{e} \tag{48}$$

where $\boldsymbol{e} \in \mathbb{R}^{n\times 3}$ is a $(n,3)$-sample of the distribution $\mathcal{N}(0, 0.025^2)$. This Gaussian perturbation and the value of the standard deviation 0.025^2 has been suggested in the supplementary material of [PFS+19].

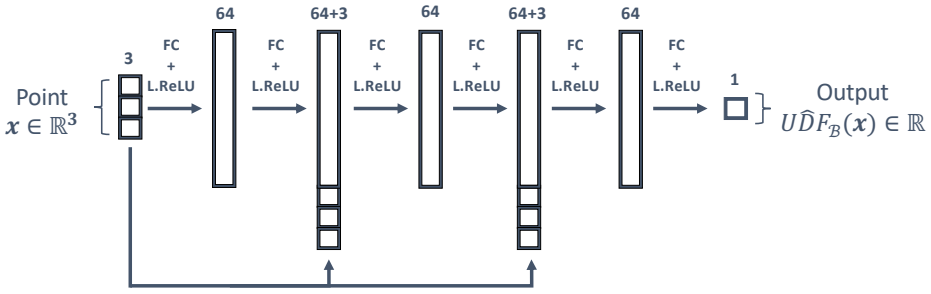


Figure 6: Neural UDF architecture. The network is composed of three blocks. Each of the three is composed of two fully-connected layers (`Leaky ReLU` activation functions [MHN+13]). The blocks are separated with two skip connections [HZRS16] to avoid gradient vanishing issues.

3.3 Computing the True UDF

In this Section, we describe the method used to calculate the true Unsigned Distance Function of a watertight surface $\mathcal{B}$ at training points $\boldsymbol{X}$. Formally, we seek to compute

$$\forall \boldsymbol{x} \in \boldsymbol{X},\ \mathrm{UDF}_{\mathcal{B}}(\boldsymbol{x}) = \min_{\boldsymbol{b} \in \mathcal{B}} \|\boldsymbol{x} - \boldsymbol{b}\| \tag{49}$$

To do this, we use the function `find_closest_cell` from framework `PyVista` [SK19]. For a query point $\boldsymbol{x} \in \mathbb{R}^3$ and a surface $\mathcal{B}$ encoded as a triangular surface mesh, this function returns the point $\boldsymbol{b_x} \in \mathcal{B}$ closest to $\boldsymbol{x}$. $\boldsymbol{b_x}$ can be seen as the orthogonal projection of $\boldsymbol{b}$ in $\mathcal{B}$. Once $\boldsymbol{b_x}$ has been found, we compute the distance $\|\boldsymbol{x} - \boldsymbol{b_x}\|$ to find $\mathrm{UDF}_{\mathcal{B}}(\boldsymbol{x})$.

The function `find_closest_cell` used here is not optimally implemented. This is not a problem for us, as we calculate the value of $\mathrm{UDF}_{\mathcal{B}}$ for sets of points of size $n \simeq 1000$. To reduce the time needed to calculate $\mathrm{UDF}_{\mathcal{B}}$, we can sample points uniformly over $\mathcal{B}$ and index these points using a k-dimensional tree [Ben75].

3.4 Training the Neural UDF

In our work, Neural UDFs are Multi Layer Perceptron neural networks with 3 input features (input point coordinates) and 1 output feature (input point UDF). Their architecture is described in Figure 6.

During network training, weights are optimized with the *Adam* gradient descent [KB14]. The training inputs are the points in the dataset $\boldsymbol{X}$ (see Equation 48) and the target outputs are their true UDF values (see Equation 49). The minimization is formally written in Equation 37.

3.5 Evaluating the Reconstruction Capacity of a Neural UDF

Consider a surface $\mathcal{B}$ and its trained neural UDF $\widehat{\mathrm{UDF}}_{\mathcal{B}}$. In this Section, we quickly review the computational implementation of the evaluation of the neural UDF's accuracy. The evaluation procedure and the resulting metric are described in detail in Section 2.3.

First, we construct a point cloud representing the surface $\mathcal{B}$. To do this, we sample points uniformly on $\mathcal{B}$ and gather them in the set $\boldsymbol{B_r} \in \mathbb{R}^{n_r \times 3}$, with $n_r = 2000$. These points are used in the initialization step to reconstruct the zero-level set of the Neural UDF $\widehat{\mathrm{UDF}}_{\mathcal{B}}$, according to Equation 39. We denote $\widehat{\boldsymbol{B}}$ the reconstructed points.

The ability of the Neural UDF to reconstruct the true surface $\mathcal{B}$ is given by the distance between $\boldsymbol{B_r}$ and $\widehat{\boldsymbol{B}}$ (see Equation 40).

3.6 Dealing with Randomness

Neural UDF training is an inherently random procedure. First, to detect surface edges, points are randomly generated on the surface (see Section 3.1). Second, the Neural UDF training dataset is an n-sample of a mixture of probability distributions (see Section 3.2). The initialization of the Neural UDF's weights is also randomized.

Furthermore, Adam gradient descent [KB14] and mini-batch learning [Ber96] are used for both the Neural UDF's weights optimization (in the training procedure) and the surface reconstruction (in the evaluation procedure). In this optimization method, each iteration is randomized.

To take into account the randomness in the modelling, the experiments are ran on a set surfaces. And for each surface, Neural UDF training and evaluation are repeated several times and we calculate the median of the reconstruction errors. Formally, consider a set of surfaces $\mathcal{B}_1, ..., \mathcal{B}_N$, an integer n representing the number of training points and $\xi \in [0, 1]$ the surface edges oversampling parameter, as defined in Section 3.2. For each of the surfaces, we train 5 Neural UDFs with different random seeds. The 5 computed values of the metrics are summarized in their median.

We denote $\left\{\widehat{\mathrm{UDF}}^{l}_{\mathcal{B}_j}(n,\xi)\right\}_{\substack{1\le j\le N\\1\le l\le 5}}$ the Neural UDFs obtained, and their reconstruction errors $\left\{\delta^{l}_{\mathcal{B}_j}(n,\xi)\right\}_{\substack{1\le j\le N\\1\le l\le 5}}$ (as defined in Equation 40).

As we aim to measure the influence of surface edge oversampling on the reconstruction error, we repeat the same operation with $\xi = 0$. Thus, we obtain a set of Neural UDFs $\left\{\widehat{\mathrm{UDF}}^{l}_{\mathcal{B}_j}(n,0)\right\}_{\substack{1\le j\le N\\1\le l\le 5}}$ and their reconstruction errors $\left\{\delta^{l}_{\mathcal{B}_j}(n,0)\right\}_{\substack{1\le j\le N\\1\le l\le 5}}$. For each of the Neural UDFs and for a fixed number of training points n, we compute the **relative improvement** due to ξ :

$$\forall j \in [\![1, N]\!],\ \mathcal{I}(\mathcal{B}_j, n, \xi) := 1 - \frac{\delta^{m}_{\mathcal{B}_j}(n,\xi)}{\delta^{m}_{\mathcal{B}_j}(n,0)} \tag{50}$$

where $\delta^{m}_{\mathcal{B}_j}(n,\xi)$ is the median reconstruction error of the five Neural UDFs of shape $\mathcal{B}_j$ trained with parameter values n and ξ, that is:

$$\delta^{m}_{\mathcal{B}_j}(n,\xi) := \mathrm{median}\left(\left\{\delta^{l}_{\mathcal{B}_j}(n,\xi)\right\}_{1\le l\le 5}\right) \tag{51}$$

and naturally,

$$\delta^{m}_{\mathcal{B}_j}(n,0) := \mathrm{median}\left(\left\{\delta^{l}_{\mathcal{B}_j}(n,0)\right\}_{1\le l\le 5}\right). \tag{52}$$

4 Results

In this Section, we present the results of our numerical experiments. In Section 4.1, we analyze the influence of the scaling parameter k (see Section 2.1) and the thresholding parameter p_0 on edge detection accuracy (see Section 3.1). In Section 4.2 we show that our descriptor detects surface edges more accurately than Pauly's descriptor [PGK02] (see

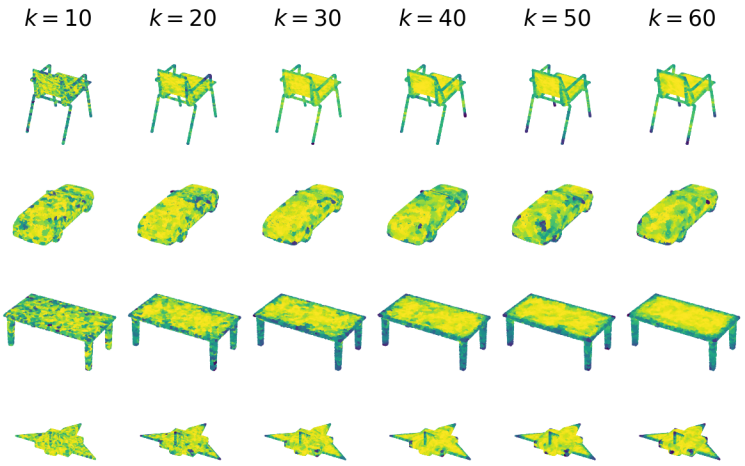


Figure 7: Setting the scale parameter k. Kolmogorov-Smirnov's descriptor on a few shapes on the dataset ShapeNet (`chair`, `car`, `table` and `airplane`) for several values of k. The local descriptor that is depicted here is the log-p-value of the local central symmetry test (see Equation 25). It can be seen on the plots that edges are detected for high values of k (higher than 30). All the edge detectors are computed with $n_s = 2000$ points. The log-p-value is not thresholded here.

Section 1.4 for a detailed explanation of this descriptor). In Section 4.3.1, we show that oversampling surface edges improves the accuracy of Neural UDFs locally around surface edges. In Section 4.3.2, we show that improving the accuracy of Neural UDFs around surface edges improves the global reconstruction capability of Neural UDFs.

4.1 Discussion on the Parameters for Edge Detection

Our edge detector is described in details in Section 2.1. It depends on the values of three parameters:

- n_s is the number of points sampled on the surface.
- k is the number of nearest neighbors in neighborhood selection. It can be considered as a scale parameter.
- p_0 is the decision threshold for the p-value

Of course, the choice of n_s should depend on the complexity of the considered shapes and the available computational resources. Indeed, for simple shapes containing few folded or pointed areas, a few points seem enough to depict the shape. For more complex surfaces with numerous edges, we need to sample more surface points. However, our edge detection method requires to perform a test and compute a p-value for each sampled point. Therefore, it can be computationally costly to compute our descriptor for a large number of surface points. In our work, and in particular on ShapeNet dataset, we empirically observe that $n_s = 2000$ points are generally sufficient to describe accurately the surfaces and require a reasonable computation time.

The parameter k corresponds to the scale at which the surface edges are detected. The choice of the scale is important because a point cloud obtained through an acquisition process consists of points that sample real-world objects, where edges can be rounded or damaged (we refer to *rounding* and *damage* as defined in [HLP+21]). The scale at which edges can be detected depends of their degree of rounding and damage. For example, in a

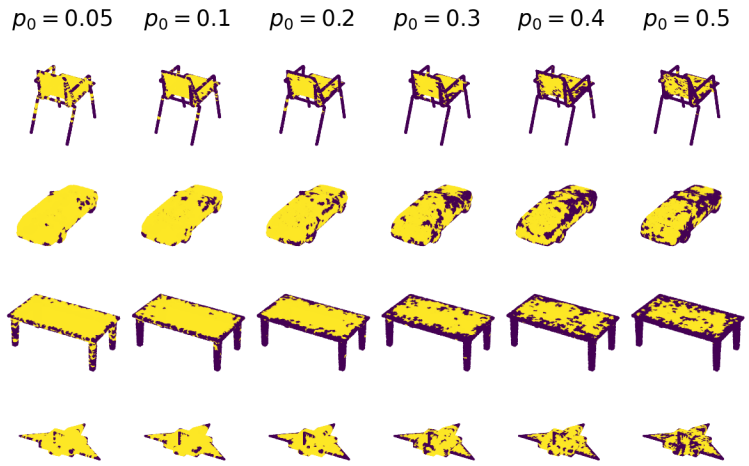


Figure 8: Setting the thresholding parameter p_0. Kolmogorov-Smirnov's descriptor on a few shapes on the dataset ShapeNet (`chair`, `car`, `table` and `airplane`) for several values of p_0. The local descriptor that is depicted here is the thresholded p-value ϵ_{n_s,k,p_0} (see Equation 31). It can be seen on the plots that edges are detected when $0.4 \leq p_0 \leq 0.6$. All the edge detectors are computed with $n_s = 2000$ and $k = 40$.

building, two façades may be joined by a smoothly curved surface, perceived as an edge at the building scale, which might not be readily detectable at finer scales, such as the scale of individual bricks [HLP+21]. In our work, the scale parameter k is the same for all the shapes of a dataset. We choose it empirically by comparing visually its the best value on a few examples. In Figure 7, one can observe that edges are detected for values of k higher than 30. In all our experiments, we set $k = 40$.

As highlighted in Equation 31, Kolmogorov-Smirnov's descriptor is strongly dependent on a decision threshold that we call p_0. It corresponds to the minimal p-value for which we retain the null hypothesis *i.e.* consider that the surface is locally planar or quasi-planar. The value of this threshold parameter is set empirically by comparing the plots of a few shapes. For the sake of automation, it remains the same for all the shapes of the dataset. It turns out that for categories `chair`, `car`, `table` and `airplane` of ShapeNet dataset, edges are well detected for $0.05 \leq p_0 \leq 0.3$.

4.2 Edge Detection On ShapeNet Data

In order to evaluate the accuracy of our edge detector, we compare it with the reference descriptor proposed in [PGK02] (see Section 1.4). The plots show that our edge detector is more accurate on chairs, cars, tables and airplanes in the ShapeNet dataset (see Figure 8). This difference is mainly due to two limitations of Pauly's descriptor. First, for very sharp or locally folded surfaces, Pauly's descriptor approaches zero because the local neighborhood of points can be almost as flat as for quasi-planar areas (this issue is illustrated in Figures 15a and 15b). Secondly, thin plates are often detected as edges by Pauly's descriptor because the neighborhood can contain both faces of the plate (see Figure 15c). These two limitations are explained with more details and illustrated on toy examples in Appendix A.

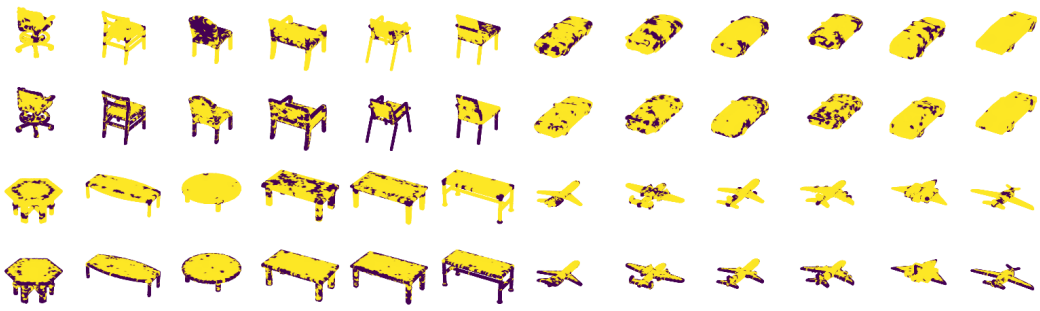


Figure 9: Edge detection using Pauly's descriptor (rows 1 and 3) *vs.* Kolmogorov-Smirnov's descriptor (rows 2 and 4). Surfaces are sampled with $n_s = 2000$ points. Each point on the surface is associated with $k = 40$ neighboring points. Points with a p-value lower than $p_0 = 0.4$ are considered as close to an edge. On the plots, the latter are colored in purple. The undetected areas are colored in yellow.

4.3 Application to UDF Learning

4.3.1 Precision on Edges

In this Section, we show that our method improves the accuracy of Neural UDFs locally around the surface edges.

Let us consider a surface $\mathcal{B}$ and a Neural UDF $\widehat{\text{UDF}}_{\mathcal{B}}$ trained with parameters n and ξ defined in Section 3.2. Points are uniformly sampled on the surface and Kolmogorov-Smirnov's descriptor is computed for each of them. Doing this, we are repeating the same procedure than the one described in Section 3.2. Among the set of points uniformly sampled, we call $\boldsymbol{B}_{s,0}$ (*resp.* $\boldsymbol{B}_{s,1}$) the set of points that are not detected (*resp.* detected) as edges (see Equation 44).

In order to measure the accuracy of Neural UDFs around these surface edges, we compute the average magnitude of the Neural UDF's output over the edges:

$$\overline{|\widehat{\text{UDF}}_{\mathcal{B}}(\boldsymbol{B}_{s,1})|} := \frac{1}{\text{Card}(\boldsymbol{B}_{s,1})} \sum_{\boldsymbol{b} \in \boldsymbol{B}_{s,1}} |\widehat{\text{UDF}}_{\mathcal{B}}(\boldsymbol{b})| \tag{53}$$

This metric can be considered as the average error of the Neural UDF on the edges, because the target output for these points is zero as they lie on the surface $\mathcal{B}$.

Figure 10 illustrates that the Neural UDF error on surface edges decreases as ξ increases, i.e. as surface edges are oversampled. This shows that oversampling improves the Neural UDF locally. On the other hand, we observe that this local improvement in accuracy decreases as n increases. This is because, when a very large number of training points are sampled, the folds and peaks are already well sampled and oversampling them has less impact.

4.3.2 Global precision

In this Section, we show that our edge detection method can be used to improve Neural UDF training. The main idea is to sample more training points around the surface edges, as formalized in Section 3.2. As a result, the accuracy of the network is locally better around these areas (see Section 4.3.1.

Nevertheless, in the frame of UDF learning, the aim is to accurately reconstruct the entire surface. So that, the distance between initial and reconstructed surfaces is an interesting

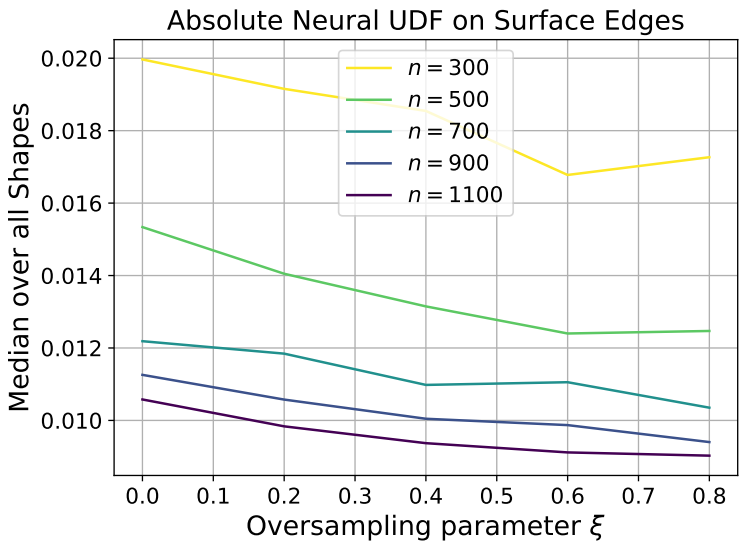


Figure 10: Evolution of the average Neural UDF error near edges (see Equation 53) with respect to the oversampling parameter ξ, for several values of the number of training points n. The plotted metric corresponds to the median of the values computed for each of the Neural UDFs, over $N = 25$ chairs from ShapeNet dataset.

metric. This reconstruction error is defined in Section 3.5 (see Equation 40).

As UDF learning and its evaluation are subject to a high degree of randomness, the precision metrics may vary from one run to another. We show that the metrics are statistically improved by repeating the experiments several times and keeping the median of the computed metrics. This approach is described in detail in Section 3.6.

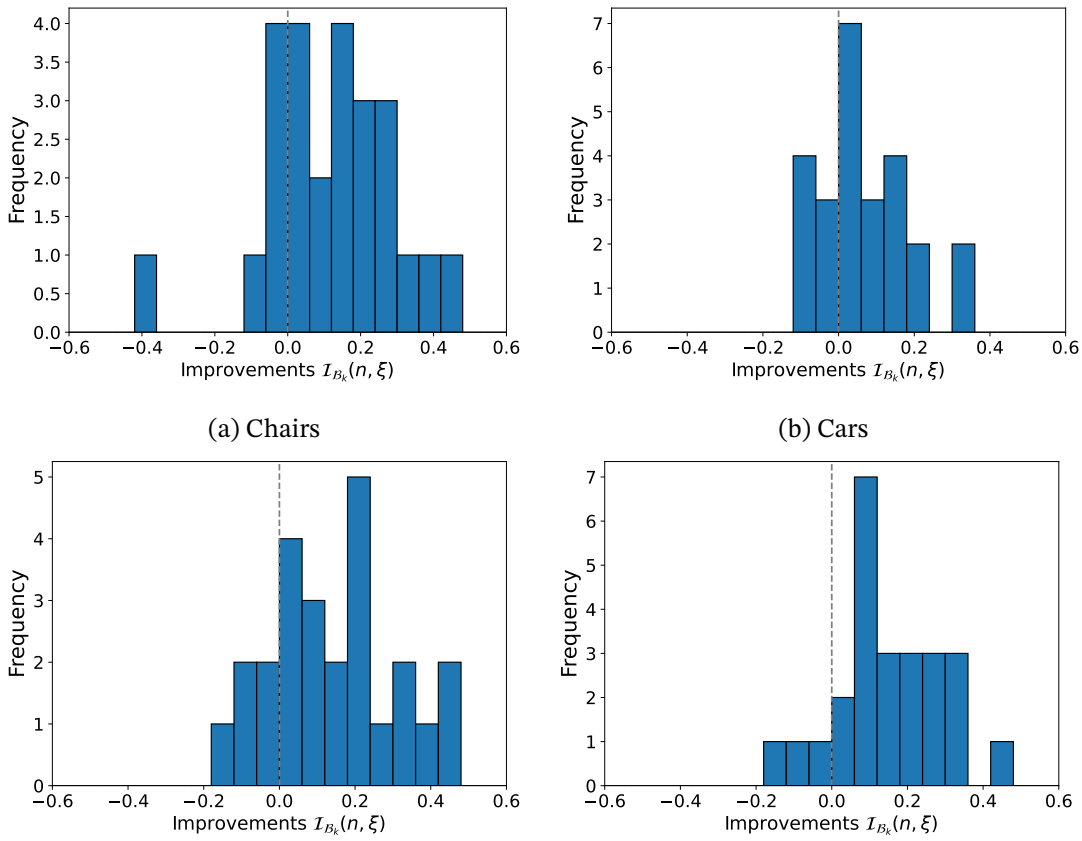
We apply our method to ShapeNet dataset, for 4 categories (chair, car, table and airplane). For each category, we train Neural UDFs for $N = 25$ different shapes, randomly taken from the dataset. For each shape, we compute the accuracy improvement due to the use of Edge Detection to build the training dataset (see Equation 50).

Histograms of the improvements are plotted in Figure 11. For this experiment, we set $n = 600$ and $\xi = 0.6$. Looking carefully at the charts, we observe that for some shapes our method does not improve the precision of Neural UDFs. However, this precision is improved for 76% of the chairs, 72% of the cars, 80% of the tables and 88% of the airplanes. The average improvement is around 15% for the four categories.

5 Conclusion

In this paper, we propose a new statistical method for edge detection on unstructured point clouds. This method is based on two key ideas. First, we use locally a statistical test of symmetry. The idea is to quantify the position of points with respect to their nearest neighbors. Second, we use a generalization of the notion of average to center circular data. This makes the circular data independent from any choice of starting point on the circle. Given observations distributed on the unit circle, it allows to compute a central symmetry test p-value that does not depend on any reference axis for polar coordinates. This allowed us to build our method on a basic goodness-of-fit test: Kolomorog-Smirnov's test.

Our edge detection method shows better results than commonly used geometric descriptors on surfaces of varying complexities. Moreover, this method is original since it is based



(a) Chairs

(b) Cars

(c) Tables

(d) Airplanes

Figure 11: Improvement of the Neural UDF precision for ShapeNet surfaces.

on a local statistical test of central symmetry. This idea could inspire new hybrid approaches. On the one hand, Pauly's descriptor does not detect sharpest edges, which are well captured by Kolmogorov-Smirnov's descriptor. On the other hand, Kolomogorov-Smirnov's descriptor does not capture obtuse edges, which can be detected by Pauly's descriptor. Further work will be done to combine both descriptors and achieve better accuracy in edge detection.

We show that this method can be used to improve the learning of 3D object representations. In particular, when learning the distance functions of 3D surfaces using neural networks, the training effort can be concentrated around surface edges. This paper explains in details the sampling procedure used to build training datasets that are denser around surface edges. This sampling procedure can be used for any application in which one needs to concentrate the training effort of a machine learning model in some areas of interest in the input space.

We propose a method to measure the influence of our edge detection method on the trained neural UDF accuracy. In particular, we use an optimization method that can be used for any application in which one needs to compute the iso-level set of a differentiable function.

On the one hand, our edge detection method improves the reconstruction capacity of neural networks by around 15% on the ShapeNet dataset. On the other hand, when training distance functions with a limited number of training points, oversampling can achieve satisfactory accuracy with fewer training points. Therefore, we show that the use of edge detection allows to learn more accurately the shape UDFs. As a result, we obtain a more expressive representation of the shapes with a more efficient procedure.

However, our edge detection method is subject to scaling and decision parameters that need to be determined beforehand for each dataset. A multiscale approach and a more granular classification of points according to edge sharpness will be performed in further work.

Our method improves the accuracy of Neural UDFs locally, close to surface edges. This improves the worst areas, since the reconstruction error measured in Hausdorff distance is reduced. However, when the error is averaged over the entire surface (e.g. Chamfer or Wasserstein distances), it is not improved in a consistent way all over the surfaces. To further improve the method, we will implement the calculation of a trade-off between edge and non-edge sampling.

Also, in this paper we show that our novel statistical method of surface edge detection allows to improve the accuracy of Neural UDFs. The latter are simple neural networks trained to fit the surface UDFs separately. In further work, we will use our statistical descriptor to improve the accuracy of more complex models like DeepSDF, that learn the UDFs of a whole set of surfaces in one single neural network. This will allow to build lower-dimensional representations of 3D shapes and potentially yield better performance in many geometric machine learning problems.

Acknowledgements

We are grateful to Sebastien Da Veiga, Xavier Roynard, Brian Staber, Thomas Pellegrini, Mathis Deronzier, Lucas De Lara, Simon Bartels for the fruitful discussions. Our work has been motivated and supported by Safran Tech company. They provided us with a dataset containing 3D turbine blade meshes and the results of numerical simulations. This data

was crucial for running our experiments and build the intuitions that led to our contributions. This work has been done with the support of the Artificial and Natural Intelligence Toulouse Institute (ANITI).

References

[AD52] T. W. Anderson and D. A. Darling. Asymptotic Theory of Certain "Goodness of Fit" Criteria Based on Stochastic Processes. *The Annals of Mathematical Statistics*, 23(2):193 – 212, 1952.

[AD54] T. W. Anderson and D. A. Darling. A test of goodness of fit. *Journal of the American Statistical Association*, 49(268):765–769, 1954.

[ADMG18] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018.

[AL20] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2565–2574, 2020.

[Bah71] Raghu Raj Bahadur. *Some limit theorems in statistics*. SIAM, 1971.

[Bal87a] Dana H. Ballard. Modular learning in neural networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, AAAI'87, page 279–284. AAAI Press, 1987.

[Bal87b] Dana H Ballard. Modular learning in neural networks. In *Proceedings of the sixth National conference on Artificial intelligence-Volume 1*, pages 279–284, 1987.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[Ber96] Dimitri P Bertsekas. Incremental least squares methods and the extended kalman filter. *SIAM Journal on Optimization*, 6(3):807–822, 1996.

[CFG+15] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015.

[Cha13] Benjamin Charlier. Necessary and sufficient condition for the existence of a fréchet mean on the circle. *ESAIM: Probability and Statistics*, 17:635–649, 2013.

[CPM+20] Julian Chibane, Gerard Pons-Moll, et al. Neural unsigned distance fields for implicit function learning. *Advances in Neural Information Processing Systems*, 33:21638–21652, 2020.

[CSR24] Fabien Casenave, Brian Staber, and Xavier Roynard. Mmgp: a mesh morphing gaussian process-based machine learning method for regression of physical problems under nonparametrized geometrical variability. *Advances in Neural Information Processing Systems*, 36, 2024.

[CZ19] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5939–5948, 2019.

[Dar83] Donald A. Darling. On the asymptotic distribution of watson's statistic. *The Annals of Statistics*, 11(4):1263–1266, 1983.

[DVVR07] Kris Demarsin, Denis Vanderstraeten, Tim Volodine, and Dirk Roose. Detection of closed sharp edges in point clouds using normal estimation and graph theory. *Computer-Aided Design*, 39(4):276–283, 2007.

[Fré48] Maurice Fréchet. Les éléments aléatoires de nature quelconque dans un espace distancié. In *Annales de l'institut Henri Poincaré*, volume 10, pages 215–310, 1948.

[FSG17] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[GFK+18] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018.

[GFZ+20] Xiaodong Gu, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan, and Ping Tan. Cascade cost volume for high-resolution multi-view stereo and stereo matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2495–2504, 2020.

[GYLB+19] Sambit Ghadai, Xian Yeow Lee, Aditya Balu, Soumik Sarkar, and Adarsh Krishnamurthy. Multi-level 3d cnn for learning multi-scale spatial features. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2019.

[HKR93] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9):850–863, 1993.

[HLP+21] Chems-Eddine Himeur, Thibault Lejemble, Thomas Pellegrini, Mathias Paulin, Loic Barthe, and Nicolas Mellado. Pcednet: A lightweight neural network for fast and interactive edge detection in 3d point clouds. *ACM Transactions on Graphics (TOG)*, 41(1):1–21, 2021.

[Hot33] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[HWS16] Timo Hackel, Jan D Wegner, and Konrad Schindler. Contour detection in unstructured 3d point clouds. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1610–1618, 2016.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kol33] Andrey Kolmogorov. Sulla determinazione empirica di una lgge di distribuzione. *Inst. Ital. Attuari, Giorn.*, 4:83–91, 1933.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high res-

olution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 8 1987.

[LCBF21] Mario Lino, Chris Cantwell, Anil A Bharath, and Stathi Fotiadis. Simulating continuum mechanics with multi-scale graph neural networks. *arXiv preprint arXiv:2106.04900*, 2021.

[LCL18] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018.

[LDG18] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018.

[LFBC22] Mario Lino, Stathi Fotiadis, Anil A Bharath, and Chris D Cantwell. Multi-scale rotation-equivariant graph neural networks for unsteady eulerian fluid dynamics. *Physics of Fluids*, 34(8), 2022.

[LLVT03] Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient implementation of marching cubes' cases with topological guarantees. *Journal of graphics tools*, 8(2):1–15, 2003.

[MHN+13] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[MON+19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.

[OFP04] Stanley Osher, Ronald Fedkiw, and K Piechor. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.*, 57(3):B15–B15, 2004.

[PFS+19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

[PGK02] Mark Pauly, Markus Gross, and Leif P Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization, 2002. VIS 2002.*, pages 163–170. IEEE, 2002.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[RLR+20] Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoit Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. Meshsdf: Differentiable iso-surface extraction. *Advances in Neural Information Processing Systems*, 33:22468–22478, 2020.

[Roy96] Rolls Royce. The jet engine, 5th edition. Technical report, Rolls Royce, 1996.

[SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and

Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[SK19] C. Bane Sullivan and Alexander Kaszynski. PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, 5 2019.

[Smi77] Paul J. Smith. A nonparametric test for bivariate circular symmetry based on the empirical cdf. *Communications in Statistics - Theory and Methods*, 6(3):209–220, 1977.

[WHH10] Christopher Weber, Stefanie Hahmann, and Hans Hagen. Sharp feature detection in point clouds. In *2010 shape modeling international conference*, pages 175–186. IEEE, 2010.

[WJHM15] Martin Weinmann, Boris Jutzi, Stefan Hinz, and Clément Mallet. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS Journal of Photogrammetry and Remote Sensing*, 105:286–304, 2015.

[Y+95] Nikitin Yakov et al. *Asymptotic efficiency of nonparametric tests*. Cambridge University Press, 1995.

[ZLLM19] Amir Zadeh, Yao-Chong Lim, Paul Pu Liang, and Louis-Philippe Morency. Variational Auto-Decoder: A Method for Neural Generative Modeling from Incomplete Data, 2019.

A Comparing Our Local Descriptor with Pauly's on 3D Toy Problems

In this Section, we compare our edge detection method (called Kolmogorov-Smirnov descriptor in reference to the test statistic used) with the standard geometric descriptor proposed by Pauly et al. in [PGK02]. This geometric descriptor is defined in Section 1.4. It is hereafter referred to as Pauly's descriptor. For our comparison, we generated toy data illustrating all types of surface edges. This toy data is described in Section A.1. The results obtained with Kolmogorov-Smirnov's and Pauly's descriptors are compared in Section A.2.

A.1 3D toy reconstruction problem

To reproduce the different types of surface edges commonly found on 3D objects, we generate surface portions corresponding to cones (Figure 12) and folds (Figure 13).

It turns out that with Pauly's method, areas corresponding to thin plates (such as the back of a chair or the top of a table, for example) are erroneously detected as surface edges. We have therefore also generated surface portions corresponding to thin plates (Figure 14).

For the cones, we begin by uniformly sampling points on a disk in xy plane, which we denote as $\boldsymbol{S_0}$. For each point $\boldsymbol{x} \in \boldsymbol{S_0}$, we define the radial plane of $\boldsymbol{x}$ as the plane containing the z-axis and the radial axis of $\boldsymbol{x}$. We define the tangent axis $T(\boldsymbol{x})$ as the axis that contains the centroid point $\boldsymbol{x_0}$ and that is orthogonal to the radial plane of $\boldsymbol{x}$. Next, for each angle ψ between 0 and $\frac{\pi}{2}$, we can define a new set of points $S(\psi)$ as the set of the images of each point $\boldsymbol{x}$ in $\boldsymbol{S_0}$ by the rotation of angle ψ around the tangent axis $T(\boldsymbol{x})$. This allows to generate a series of surfaces $S(\psi)$ (ψ between 0 and $\frac{\pi}{2}$) that can be thought of as images of a cone gradually closing. The centroid point for all the surfaces is the origin of the reference frame xyz. We define the range of values of ψ such that $\psi = 0$ corresponds to the initial disk (an fully open cone: $S(\psi = 0) = \boldsymbol{S_0}$), and $\psi = \frac{\pi}{2}$ corresponds to a set of points aligned on the z-axis (a fully closed cone). On Figure 12, cones are depicted for three values of ψ.

For the folds, we start with the same set of points $\boldsymbol{S_0}$ in xy plane. However, here we rotate each point around the y-axis. The resulting series of surfaces can be seen as pictures of a fold that is gradually folded in half. On Figure 13, folds are depicted for three values of ψ.

For the thin plates, 500 points are sampled uniformly on the unit disk in 2D. These points are assumed to lie in the plane ($z = 0$). We then repeat the same operation, except that the

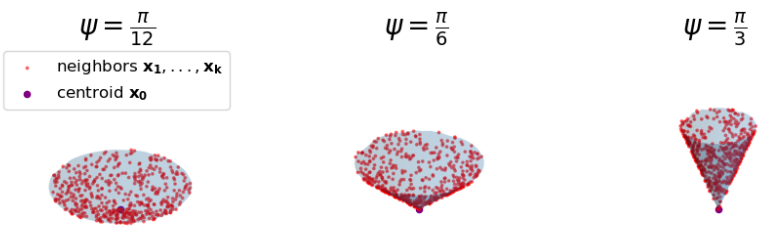


Figure 12: Different cases for local spikes. The parameter ψ represents the rotation angle. When $\psi = 0$, the cone is fully open. The higher ψ the more closed the cone. These plots were generated with 500 samples each.

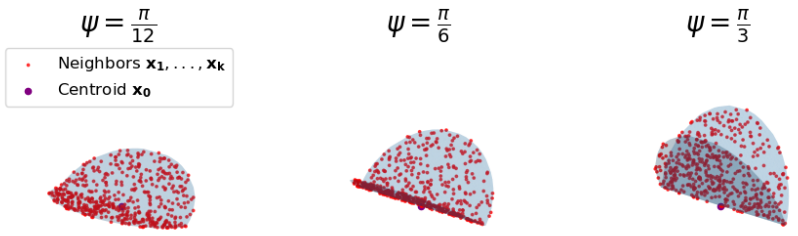


Figure 13: Different cases for local folds. The parameter ψ represents the rotation angle. When $\psi = 0$, the fold is fully plane. The higher ψ the more folded the fold. These plots were generated with 500 samples each.

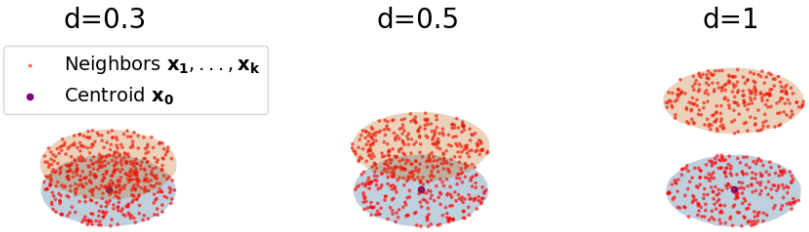


Figure 14: Different cases for fine plates. The parameter d represents the thickness of the plate. The plots were generated with 500 points: each face of the plate is sampled with 250 points from the uniform distribution on unit circle.

500 new points obtained are considered to be in the plane ($z = d$) (where d - the parameter of interest - is the thickness of the platter). The centroid point for the local descriptor is the origin (0,0,0). On Figure 14, thin plates are depicted for three values of d.

A.2 Comparison of the descriptors

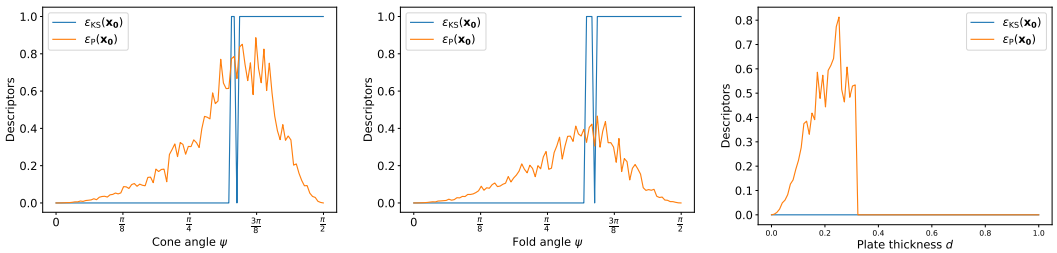
In this Section, we compare the values of Pauly's and Kolmogorov-Smirnov's local descriptors on toy surface portions (the cones, folds and thin plates described in Section A.1). The aim here is to assess the ability of each of the descriptors to detect surface edges.

To compute the descriptor over the entire surface, we must first select the neighborhood of each point on the surface. Neighborhood selection is formalized in Section 2.1.

Figure 15 supports the conclusion that Pauly's descriptor is not effective in detecting all the edges. Specifically, for large values of ψ, Pauly's descriptor decreases as the local sharpness of the surface increases. This descriptor measures the local variation of the surface and is smaller for flatter surfaces. However, for very sharp or locally folded surfaces, Pauly's descriptor also approaches zero. These areas represent sharp edges and should be detected as such. They correspond to folds with an angle close to $\frac{\pi}{2}$. Kolmogorov-Smirnov's descriptor is a better alternative as it can clearly differentiate between locally quasi-planar surfaces and sharp edges.

B Building an Intuition in 2D

In this Section, we describe the thought process that led to our statistical edge detection method. In particular, the first 2D version of the method is described. Here, surfaces are



(a) Cones (b) Folds (c) Thin plates

Figure 15: Figures 15a and 15b: Estimator values for cones and folds with different angles (descriptor values are normalized between 0 and 1). When $\psi = 0$, the cone (*resp.* fold) is fully open and the surface is locally planar and both Pauly and Kolmogorov-Smirnov descriptors are equal to zero. The discontinuity on the Kolmogorov-Smirnov curve indicates a change of principal plane. For lower values of theta, the principal plane is tangent to the surface, while for higher values of theta, it is orthogonal to the surface. Pauly's descriptor is not suitable because for large values of theta it is not increasing, whereas the larger theta, the more pointed the cone (and therefore we would expect the descriptor to be larger). Figure 15c: For thin plates ($d < 0.3$), some points on the other side of the plateau are included in the neighborhood and this induces an error in edge detection: Pauly's descriptor appears to be very sensitive to this artifact, taking on very large values despite the fact that the surface is completely flat. Whereas Kolmogorov-Smirnov's descriptor is robust to this difficulty (its values remain low whatever the thickness of the plateau).

replaced by contours, and the aim is to detect contour edges. In Section B.1, we describe some synthetic data corresponding to contour portions. In Section B.2, we explain our 2D edge detector based on a test of odds.

B.1 Toy problem in 2D

We aim to compare different local descriptors for edge detection. Hence we generated contour edges with different angles. More precisely, we sample $k = 50$ points evenly distributed on a straight line, along axis $\boldsymbol{e_x}$ in $\mathbb{R}^2$. Their x-coordinates are uniformly distributed between -1 and 1. Their y-coordinates are equal to zero. The obtained point cloud is denoted $\boldsymbol{C_0}$ corresponds to a straight contour portion. The centroid point is the origin of the Cartesian system. For any $\psi \in [0, \frac{\pi}{2})$, we apply a rotation to each point in $\boldsymbol{C_0}$ and generate the new contour portion

$$\boldsymbol{C_\psi} = \{\mathcal{R}(\text{sgn}(\boldsymbol{x} \cdot \boldsymbol{e_x})\psi)\boldsymbol{x},\ \boldsymbol{x} \in \boldsymbol{C_0}\} \tag{54}$$

where

$$\forall \psi \in [-\pi, \pi), \mathcal{R}(\psi) = \begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix}. \tag{55}$$

Figure 16 illustrates how we generated the aforementioned contours portions.

B.2 Odds-ratio descriptor

In this Paragraph, we describe the local statistical descriptor that we use to detect edges on 2D contours. We call it the *odds-ratio descriptor*. In Section 2.1, we define the Kolomogorov-Smirnov descriptor as a novel statistical descriptor used to detect edges on 3D surfaces. This descriptor can be conceptualized as a 3-dimensional generalization of the odds-ratio descriptor defined here.

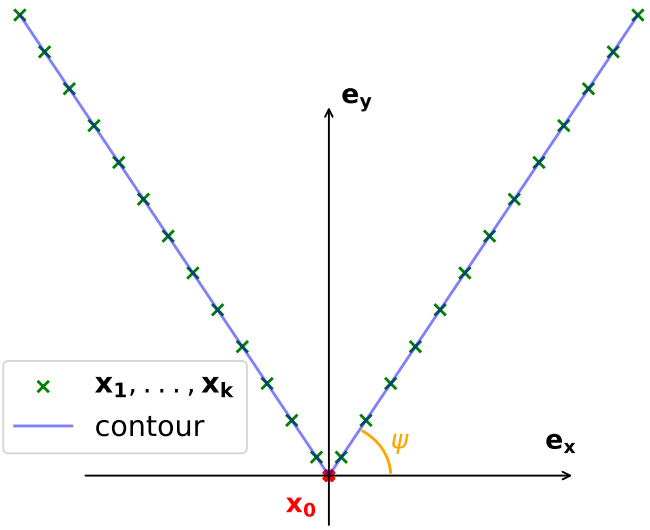


Figure 16: Illustration of the procedure used to generate the contour portions.

Let us explain the intuition that led us to the odds-ratio descriptor. We illustrate our intuition on the toy contour portions described in Paragraph B.1. These contour portions are depicted in Figure 16. A good descriptor is expected to discriminate obtuse contour portions (small values of ψ) and acute contour portions (large values of ψ).

The odds-ratio descriptor relies on a statistical approach to differentiate between obtuse (small ψ) and acute (large ψ) contour edges. This approach is based on two key ideas: (1) projecting the neighboring points on their *average axis*. Then (2) modeling the projections as observations of a probability distribution and assess the symmetry of the latter around the centroid point. Indeed, for obtuse edges (small ψ), the average axis is tangent to the contour, and the projections of the points on this axis are evenly distributed around the centroid. Whereas for acute edges (large ψ), the average axis is orthogonal to the contour, and the centroid is off-centered with respect to the projections of its neighbors.

In order to formalize the method, let introduce $\boldsymbol{x_0}$ a centroid point and $\boldsymbol{x_1}, ..., \boldsymbol{x_k}$ its k-nearest-neighbors.

The average axis of the neighboring points is defined as the first eigen axis of their covariance matrix

$$\mathcal{V} := \frac{1}{k+1} \sum_{i=0}^{k} (\boldsymbol{x_i} - \overline{\boldsymbol{x}})(\boldsymbol{x_i} - \overline{\boldsymbol{x}})^T \tag{56}$$

where

$$\overline{\boldsymbol{x}} := \frac{1}{k+1} \sum_{i=0}^{k} \boldsymbol{x_i} \tag{57}$$

is the neighborhood's mean.

For the sake of formalization, we call $\boldsymbol{e_1}$ the first eigenvector of $\mathcal{V}$. Its direction is the average axis of the neighboring points.

We project the neighboring points on their average axis and center them with respect to

the centroid point, that is:

$$\forall i \in [\![1, k]\!],\ X_i := (\boldsymbol{x_i} - \boldsymbol{x_0}) \cdot \boldsymbol{e_1} \tag{58}$$

We aim to assess if the scalar values $X_1, ..., X_k$ are evenly distributed around 0. In order to do so, we consider these values as i.i.d. realizations of a random variable X and assess the symmtery of X around 0. It amounts in testing the null hypothesis

$$H_0 : \text{"the distribution of } X \text{ is symmetric around 0"}.$$

Let introduce the number of positive realizations:

$$k_+ = \sum_{i=1}^{k} \mathbb{1}_{X_i \geq 0} \tag{59}$$

It can be shown that under the null hypothesis H_0, the quantity

$$V_k := \frac{2k_+ - k}{\sqrt{k}} \tag{60}$$

can be approximated by a Standard Gaussian distribution.

The p-value is then computed as the probability of observing a test statistic at least as extreme as the one observed here or more extreme, given the null hypothesis. Specifically, if we denote by F the cumulative distribution function of the Standard Gaussian distribution, then

$$p\text{-value} := 2 \times (1 - F^{-1}(|v|)) \tag{61}$$

where v is the observed value of the test statistic V_k.

In order to validate or reject the null hypothesis, we use a decision threshold p_0. Here we empirically set $p_0 = 0.01$. In particular, if the p-value is lower than p_0, we reject the null hypothesis, which means that the distribution of X is not symmetric. In this case, we consider that the contour edge is acute. On the contrary, the p-value is higher than p_0, we validate the null hypothesis, which means that the distribution of X is symmetric. In this case, we consider that the contour edge is obtuse *i.e.* that the contour is planar or quasi-planar. We define the odds-ratio descriptor as follows:

$$\epsilon_{OR}(\boldsymbol{x_0}) := \begin{cases} 1 \text{ if } p\text{-value} \leq p_0. \\ 0 \text{ if } p\text{-value} > p_0. \end{cases} \tag{62}$$

This method thus allows to remedy the two limitations of Pauly's approach:

1. This method clearly distinguishes situations 1) and 2) from situations 3) and 4).
2. The descriptor corresponds to the p-value of a statistical test, we can therefore interpret it and quantify its uncertainty as a function of the size of the neighborhood and the number of points sampled in it.

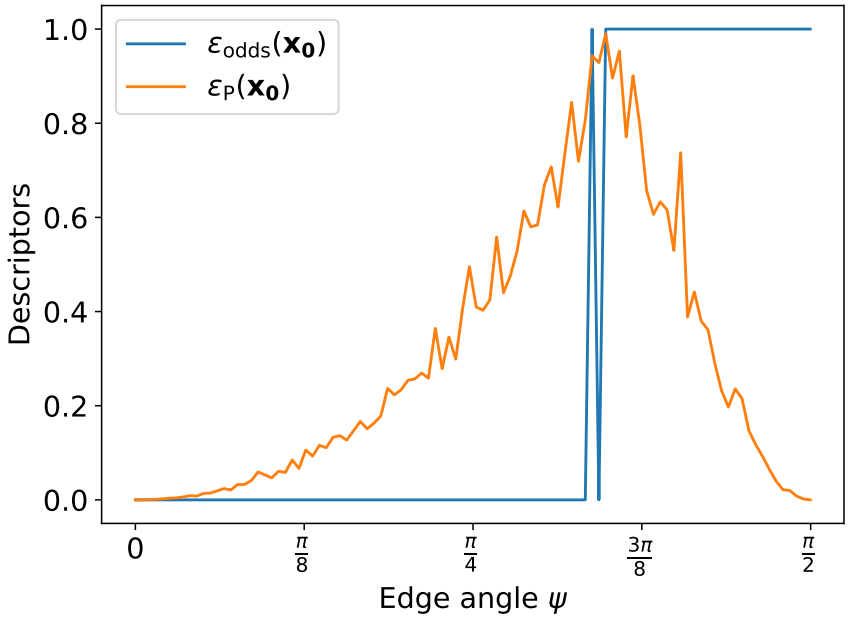


Figure 17: Descriptors computed on 2D toy contour portions as a function of the contour edge angle ψ. For 100 values of ψ between 0 and $\frac{\pi}{2}$, we generate the contour portion $\boldsymbol{C}_\psi$ as defined in Equation 54. For each contour portion, we compute odds-ratio and Pauly's descriptors. It can be seen that Pauly's descriptor does not allow to discriminate between quasi-planar and acute contours. Indeed, it takes small values in both extreme cases ($\psi \simeq 0$ and $\psi \simeq \frac{\pi}{2}$).