Exploring Differential Geometry in Neural Implicits

Tiago Novello, Guilherme Schardong, Luiz Schirmer,
Vinicius da Silva, Helio Lopes, Luiz Velho

August 21, 2022

Abstract

We introduce a neural implicit framework that exploits the differentiable properties of neural networks and the discrete geometry of point-sampled surfaces to approximate them as the level sets of *neural implicit functions*.

To train a neural implicit function, we propose a loss functional that approximates a signed distance function, and allows terms with high-order derivatives, such as the alignment between the principal directions of curvature, to learn more geometric details. During training, we consider a non-uniform sampling strategy based on the curvatures of the point-sampled surface to prioritize points with more geometric details. This sampling implies faster learning while preserving geometric accuracy when compared with previous approaches.

We also use the analytical derivatives of a neural implicit function to estimate the differential measures of the underlying point-sampled surface.

1 Introduction

Level sets of neural networks have been used to represent implicit surfaces in $\mathbb{R}^3$ with accurate results. In this context, the *neural implicit problem* is the task of training the parameters θ of a neural network $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ such that its *zero-level set* $f_\theta^{-1}(0) = \{p \,|\, f_\theta(p) = 0\}$ approximates a desired surface in $\mathbb{R}^3$. We say that f_θ is a *neural implicit function* and that $f_\theta^{-1}(0)$ is a *neural implicit surface*.

In this work, we propose a framework to solve the neural implicit problem. The *input* is a discrete sample of points from a *ground truth* surface S, and the *output* is a neural implicit function f_θ approximating the *signed distance function* (SDF) of S. The framework explores the *differential geometry* of implicit surfaces in the learning process of f_θ. Thus, for simplicity, we consider f_θ to be a smooth function. *Sinusoidal representation networks* (SIREN) [Sitzmann et al., 2020] and *implicit geometric regularization* (IGR) [Gropp et al., 2020] are examples of smooth neural networks. We adopt as a basis the SIREN architecture which has important properties that are suitable for reconstructing signals.

Specifically, let $\{p_i, N_i\}$ be the input composed of a sample of points and normals from a possibly *unknown* surface S. We look for a set of parameters θ such that f_θ approximates the SDF of S. Since SDFs have unit gradient we ask f_θ to satisfy the *Eikonal equation* $\|\nabla f_\theta\| = 1$ in $\{p_i\}$. Moreover, it is required the conditions $f_\theta(p_i) = 0$ and $\langle \nabla f_\theta(p_i), N_i \rangle = 1$, which force the zero-level set of f_θ to interpolate $\{p_i\}$ and the gradient to be aligned to the normals N_i. Additionally, to avoid spurious components in $f_\theta^{-1}(0)$, we extend these constraints to off-surface points using a SDF approximation.

The above constraints require two degrees of differentiability of f_θ. We also explore the "alignments" between the *shape operator* $d\frac{\nabla f_\theta}{\|\nabla f_\theta\|}$ of the neural surface $f_\theta^{-1}(0)$ and the shape operator dN of S. This requires one more degree of differentiability of f_θ. As the shape operators carry the intrinsic and extrinsic geometry of their surfaces, asking their alignment would require more consistency between the geometrical features of $f_\theta^{-1}(0)$ and S.

In practice, we have a sample of points of S. Suppose that the shape operator is known at these points. During the training of f_θ, it is common to sample batches of points uniformly. However, there is a high probability of selecting points with poor geometric details on their neighborhoods, which leads to slow learning of the network.

This work proposes a sampling based on the curvature to select points with important geometric details during the training.

With a trained neural implicit function f_θ in hand, we can analytically calculate the differential geometry formulas of the corresponding neural implicit surface since we have its shape operator $d\frac{\nabla f_\theta}{\|\nabla f_\theta\|}$. We provide the formulas along with the text.

The main contribution of our work is a global geometric representation in the continuous setting using neural networks as implicit functions. Besides its compact representation that captures geometrical details, this model is robust for shape analysis and efficient for computation since we have its derivatives in closed form. The contributions can be summarized as follows:

- A method to approximate the SDF of a surface S by a network f_θ. The *input* of the framework is a sample of points from S (the *ground truth*) endowed with its normals and curvatures, and the SDF approximation f_θ is the *output*.
- A *loss functional* that allows the exploration of tools from continuous differential geometry during the training of the neural implicit function. This provides high fidelity when reconstructing geometric features of the surface and acts as an implicit regularization.
- During the training of the network, we use the discrete differential geometry of the dataset (point-sampled surface) to *sample* important regions. This provides a robust and fast training without losing geometrical details.
- We also use the derivatives, in closed form, of a neural implicit function to estimate the differential measures, like normals and curvatures, of the underlying point-sampled surface. This is possible since it lies in a neighborhood of the network zero-level set.

In this work, we will focus on implicit surfaces in $\mathbb{R}^3$. However, the definitions and techniques that we are going to describe can be easily adapted to the context of implicit n-dimensional *manifolds* embedded in $\mathbb{R}^{n+1}$. In particular, it can be extended to curves and gray-scales images (graph of 2D functions). For neural animation of surfaces, see [Novello et al., 2022].

2 Related concepts and previous works

The research topics related to our work include implicit surface representations using neural networks, discrete and continuous differential geometry, and surface reconstruction.

2.1 Surface representation

A surface $S \subset \mathbb{R}^3$ can be represented *explicitly* using a collection of *charts* (*atlas*) that covers S or *implicitly* using an implicit function that has S as its zero-level set. The *implicit function theorem* defines a bridge between these representations. Consider S to be a smooth surface, i.e. there is a smooth function f having S as its zero-level set and $\nabla f \neq 0$ on it. The normalized gradient $N = \frac{\nabla f}{\|\nabla f\|}$ is the *normal* field of S. The differential of N is the *shape operator* and gives the curvatures of S.

In practice, we usually have a point cloud $\{p_i\}$ collected from a *real-world* surface S whose representation is unknown. Thus, it is common to add a structure on $\{p_i\}$ in order to operate it as a surface, for example, to compute its normals and curvatures. The classical explicit approach is to reconstruct S as a *triangle mesh* having $\{p_i\}$ as its vertices. It will be a piecewise linear approximation of S with topological guarantees if $\{p_i\}$ satisfies a set of special properties [Amenta et al., 2000].

For simplicity, since the input of our method is a sample of points endowed with normals and curvatures, we *consider* it to be the set of vertices of a triangle mesh. Then, we can use classical algorithms to approximate the normals and curvatures at the vertices. However, we could use only point cloud data and compute its normals and curvatures using well-established techniques in the literature [Mitra and Nguyen, 2003, Alexa et al., 2003, Mederos et al., 2003, Kalogerakis et al., 2009].

2.1.1 Discrete differential geometry

Unfortunately, the geometry of the triangle mesh T cannot be studied in the classical differentiable way, since it does not admit a continuous normal field. However, we can define a *discrete* notion of this field considering it to be constant on each triangle. This implies a discontinuity on the edges and

vertices. To overcome this, we use an average of the normals of the adjacent faces [Meyer et al., 2003]. Hence, the variations of the normal field are concentrated on the edges and vertices of T.

The study of the discrete variations of the normals of triangle meshes is an important topic in *discrete* differential geometry [Meyer et al., 2003, Cohen-Steiner and Morvan, 2003, Taubin, 1995]. Again, these variations are encoded in a *discrete shape operator*. The principal directions and curvatures can be defined on the edges: one of the curvatures is zero, along the edge direction, and the other is measured across the edge and it is given by the dihedral angle between the adjacent faces. Finally, the shape operator is estimated at the vertices by averaging the shape operators of the neighboring edges. We consider the approach of Cohen-Steiner and Morvan [2003].

The existent works try to discretize operators by mimicking a certain set of properties inherent in the continuous setting. Most often, it is not possible to discretize a smooth object such that all of the natural properties are preserved, this is the *no free lunch* scenario. For instance, Wardetzky et al. [2007] proved that the existent discrete *Laplacians* do not satisfy the properties of the continuous Laplacian.

Given an (oriented) point cloud $\{p_i, N_i\}$ sampled from a surface S, we can try to reconstruct the SDF of S. For this, points outside S may be added to the point cloud $\{p_i\}$. After estimating the SDF on the resulting point cloud we obtain a set pairs $\{p_i, f_i\}$ of points and the approximated SDF values.

2.2 Classic implicit surface reconstruction

Radial basis functions (RBF) [Carr et al., 2001] is a classical technique that approximates the SDF f from $\{p_i, f_i\}$. The RBF interpolant is given by: $s(p) = \sum \lambda_i \phi(\|p - p_i\|)$, where the coefficients $\lambda_i \in \mathbb{R}$ are determined by imposing $s(p_i) = f_i$. The *radial function* $\phi : \mathbb{R}^+ \to \mathbb{R}$ is a real function, and p_i are the centers of the radial basis function. In order to consider the normals $\{N_i\}$, Macêdo et al. [2011] proposed to approximate the function f by a *Hermite* radial basis function. It is important to note that the RBF representation is directly dependent on the dataset, since its interpolant s depends on p_i.

Poisson surface reconstruction [Kazhdan et al., 2006] is another classical method widely used in computer graphics to reconstruct a surface from an oriented point cloud $\{p_i, N_i\}$.

In this work, a multilayer perceptron (MLP) network f_θ is used to overfit the unknown SDF. θ is trained using the point-sampled surface $\{p_i\}$ endowed with its normals and curvatures. A loss function is designed to fit the zero-level set of f_θ to the dataset. We use the curvatures in the loss function to enforce the learning of more geometrical detail, and during the sampling to consider minibatches biased by the curvature of the data. In Section 5.3 we show that our neural implicit representation is comparable with the RBF method making it a flexible alternative in representing implicit functions.

Both RBF and our method look for the parameters of a function such that it fits to the signed distance of a given point-sampled surface. Thus they are related to the *regression* problem. Differently from RBF, a neural network approach provides a compact representation and is not directly dependent on the dataset, only the training of its parameters. The addition of constraints is straightforward, by simply adding terms to the loss function. Compared to RBF, adding more constraints increases the number of equations to solve at inference time, thus increasing the problem's memory requirements.

2.3 Neural implicit representations

In the context of implicit surface representations using networks, we can divide the methods in three categories: 1st generation models; 2nd generation models; 3rd generation models.

The 1st generation models correspond to *global* functions of the ambient space and employ as implicit model either a *indicator function* or a *generalized SDF*. They use a fully connected MLP network architecture. The model is learned by fitting the input data to the model. The loss function is based either on the L_1 or L_2 norm. The seminal papers of this class appeared in 2019. They are: Occupancy Networks [Mescheder et al., 2018], Learned Implicit Fields [Chen and Zhang, 2019], Deep SDF [Park et al., 2019], and Deep Level Sets [Michalkiewicz, 2019].

The 2nd generation models correspond to a set of *local* functions that combined together gives a representation of a function over the whole space. These models are based either on a shape algebra, such as in *constructive solid geometry*, or convolutional operators. The main works of this category appeared in 2019-2020: Local Deep Implicit Functions [Genova et al., 2019], BSP-Net [Chen et al., 2020], CvxNet [Deng et al., 2020] and Convolutional Occupancy Networks [Peng et al., 2020].

The 3rd generation models correspond to true SDFs that are given by the *Eikonal* equation. The model exploits in the loss function the condition $\|\nabla f\| = 1$. The seminal papers of this category appeared in 2020. They are: IGR [Gropp et al., 2020] and SIREN [Sitzmann et al., 2020].

Inspired by the 3rd generation models, we explore smooth neural networks that can represent the SDFs of surfaces. That is, the Eikonal equation is considered in the loss function which adds constraints involving derivatives of first order of the underlying function. One of the main advantages of neural implicit approaches is their flexibility when defining the optimization objective. Here we use it to consider higher order derivatives (related to curvatures) of the network during its training and sampling. This strategy can be seen as an implicit regularization which favors smooth and natural zero-level set surfaces by focusing on regions with high curvatures. The network utilized in the framework is a MLP with a smooth activation function.

3 Conceptualization

3.1 Implicit surfaces

In Section 2.1 we saw that the zero-level set $f^{-1}(0)$ of a function $f : \mathbb{R}^3 \to \mathbb{R}$ represents a regular surface if $\nabla f \neq 0$ on it. However, the converse is true, i.e. for each regular surface S in $\mathbb{R}^3$, there is a function $f : \mathbb{R}^3 \to \mathbb{R}$ having S as its zero-level set [Do Carmo, 2016, Page 116]. Therefore, given a sample of points on S, we could try to construct the corresponding implicit function f.

3.1.1 Differential geometry of implicit surfaces

Let S be a surface and $f : \mathbb{R}^3 \to \mathbb{R}$ be its implicit function. The differential $dN_p : T_pS \to T_pS$ of $N = \frac{\nabla f}{\|\nabla f\|}$ at $p \in S$ is a linear map on the tangent plane T_pS. The map dN is called the *shape operator* of S and can be expressed by:

$$dN = (I - N \cdot N^\top)\frac{\mathbf{H}f}{\|\nabla f\|}, \tag{1}$$

where $\mathbf{H}f$ denotes the *Hessian* of f and I is the identity matrix. Thus, the shape operator is the product of the Hessian (scaled by the gradient norm) and a linear projection along the normal.

As dN_p is symmetric, the spectral theorem states that there is an orthogonal basis $\{e_1, e_2\}$ of T_pS (the *principal directions*) where dN_p can be expressed as a diagonal 2×2 matrix. The two elements of this diagonal are the *principal curvatures* κ_1 and κ_2, and are obtained using $dN(e_i) = -\kappa_i e_i$.

The *second fundamental form* of S can be used to interpret dN geometrically. It maps each point $p \in S$ to the quadratic form $\mathbf{II}_p(v) = \langle -dN_p(v), v\rangle$. Let α be a curve passing through p with unit tangent direction v. The number $\kappa_n(p) = \mathbf{II}_p(v)$ is the *normal curvature* of α at p. Kindlmann et al. [2003] used κ_n to control the width of the *silhouettes* of S during rendering.

Restricted to the unit circle of T_pS, $\mathbf{II}_p$ reaches a maximum and a minimum (principal curvatures). In the frame $\{e_1, e_2\}$, $\mathbf{II}_p$ can be written in the quadratic form $\mathbf{II}_p(v) = x_1^2\kappa_1 + x_2^2\kappa_2$ with $v = x_1e_1 + x_2e_2$. Points can be classified based on their form: *elliptic* if $\kappa_1\kappa_2 > 0$, *hyperbolic* if $\kappa_1\kappa_2 < 0$, *parabolic* if only one κ_i is zero, and *planar* if $\kappa_1 = \kappa_2 = 0$. This classification is related to the *Gaussian curvature* $K = \kappa_1\kappa_2$. Elliptic points have positive curvature. At these points, the surface is similar to a dome, positioning itself on one side of its tangent plane. Hyperbolic points have negative curvature. At such points, the surface is saddle-shaped. Parabolic and planar points have zero curvature.

The Gaussian curvature K of S can be calculated using the following formula [Goldman, 2005].

$$K = -\frac{1}{\|\nabla f\|^4}\det\left[\begin{array}{c|c} \mathbf{H}f & \nabla f \\ \hline \nabla f^\top & 0 \end{array}\right]. \tag{2}$$

The *mean curvature* $H = (\kappa_1 + \kappa_2)/2$, is an extrinsic measure that describes the curvature of S. It is the half of the *trace* of dN which does not depend on the choice of basis. Expanding it results in the divergence of N, i.e. $2H = \operatorname{div}\frac{\nabla f}{\|\nabla f\|}$. Thus, if f is a SDF, the mean curvature can be written using the Laplacian.

An important advantage of representing a surface using level sets is that the geometric objects, like normals and curvatures, can be computed analytically — no discretization is needed. Figure 1 illustrates the Gaussian and mean curvatures of a neural implicit surface approximating the Armadillo.

The corresponding network was trained using the method we are proposing. We use the *sphere tracing* algorithm [Hart, 1996] to ray cast the zero-level set. The image was rendered using the traditional *Phong* shading. Both normal vectors and curvatures were calculated analytically using PyTorch automatic differentiation module (`torch.autograd`) [Paszke et al., 2019]. We used a transfer function to map points with high/medium/low curvatures to the red/white/blue colors.

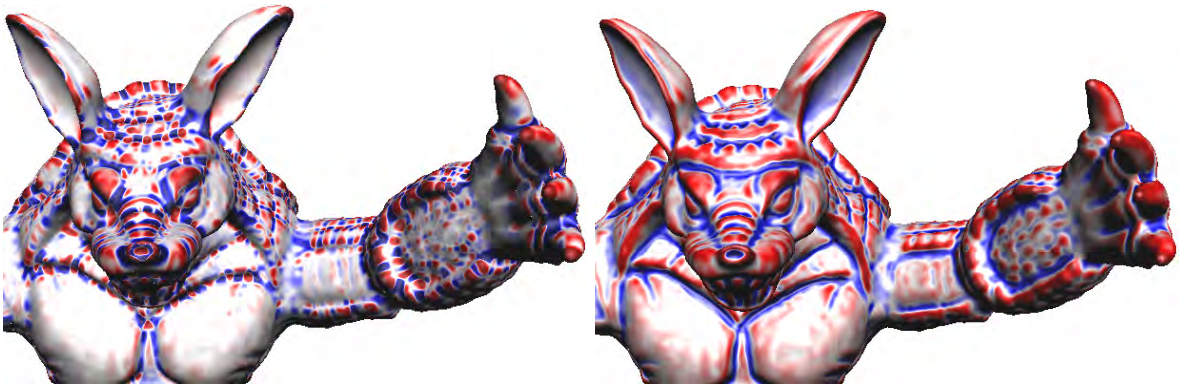


Figure 1: Gaussian and mean curvatures of the smooth Armadillo model.

There are several representations of implicit functions. For example, in *constructive solid geometry* the model is represented by combining simple objects using union, intersection, and difference. However, this approach has limitations, e.g. representing the Armadillo would require a prohibitive number of operations. RBF is another approach consisting of interpolating samples of the underlying implicit function which results in system of linear equation to be solved. A neural network is a compact option that can represent any implicit function with arbitrary precision, guaranteed by the *universal approximation theorem* [Cybenko, 1989].

3.2 Neural implicit surfaces

A *neural* implicit function $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ is an implicit function modeled by a neural network. We call the zero-level set $f_\theta^{-1}(0)$ a *neural implicit surface*. Let S be a compact surface in $\mathbb{R}^3$, to compute the parameter set of f_θ such that $f_\theta^{-1}(0)$ approximates S, it is common to consider the *Eikonal* problem:

$$\begin{cases} \|\nabla f_\theta\| = 1 & \text{in } \mathbb{R}^3, \\ f_\theta = 0 & \text{on } S. \end{cases} \tag{3}$$

The *Eikonal equation* $\|\nabla f_\theta\| = 1$ asks for f_θ to be a *SDF*. The *Dirichlet condition*, $f_\theta = 0$ on S, requires f_θ to be the signed distance of a set that contains S. These two constraints imply the *Neumann condition*, $\frac{\partial f_\theta}{\partial N} = 1$ on S. Since $\frac{\partial f_\theta}{\partial N} = \langle \nabla f_\theta, N \rangle$, Neumann constraint forces ∇f_θ to be aligned to the normal field N. These constraints require two degree of differentiability of f_θ, thus, we restrict our study to smooth networks.

There are several advantages of using neural surfaces. Besides having the entire framework of neural networks available, these functions have a high capacity of representation. We also have access to the differential geometry tools of neural surfaces, for this, we only need the Hessian and gradient operators of the network since these are the ingredients of the shape operator (Eq. 1). As a consequence, we can design loss functions using high-order differential terms computed analytically.

3.3 Learning a neural implicit surface

Let S be a compact surface in $\mathbb{R}^3$ and $f : \mathbb{R}^3 \to \mathbb{R}$ be its SDF. Let $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ be an *unknown* neural implicit function. To train θ, we seek a minimum of the following loss function, which forces f_θ to be a solution of Equation (3).

$$\mathcal{L}(\theta) = \underbrace{\int_{\mathbb{R}^3} \big|1 - \|\nabla f_\theta\|\big| dp}_{\mathcal{L}_{\text{Eikonal}}} + \underbrace{\int_S |f_\theta| dS}_{\mathcal{L}_{\text{Dirichlet}}} + \underbrace{\int_S 1 - \left\langle \frac{\nabla f_\theta}{\|\nabla f_\theta\|}, N \right\rangle dS}_{\mathcal{L}_{\text{Neumann}}}. \tag{4}$$

$\mathcal{L}_{\text{Eikonal}}$ encourages f_θ to be the SDF of a set $\mathcal{X}$ by forcing it to be a solution of $\|\nabla f_\theta\| = 1$. $\mathcal{L}_{Dirichlet}$ encourages $\mathcal{X}$ to contain S. $\mathcal{L}_{\text{Neumann}}$ asks for ∇f_θ and the normal field of S to be aligned. It is common to consider an additional term in Equation (4) penalizing points outside S, this forces f_θ to be a SDF of S, i.e. $\mathcal{X} = S$. In practice, we extended $\mathcal{L}_{Dirichlet}$ to consider points outside S, for this we used an approximation of the SDF of S.

We investigate the use of the shape operator of $f_\theta^{-1}(0)$ to improve $\mathcal{L}$, by forcing it to align with the *discrete* shape operator of the ground truth point-sampled surface. For the sampling of points used to feed a discretization of $\mathcal{L}$, the discrete curvatures access regions containing appropriate features.

3.4 Discrete surfaces

Let $T = (V, E, F)$ be a triangle mesh approximating S. $V = \{p_i\}$ are the vertices, E denotes the edge set, and F denotes the faces. The discrete curvatures at an edge e can be estimated using $\beta(e)\,\bar{e}\cdot\bar{e}^\top$, where $\beta(e)$ is the signed dihedral angle between the two faces adjacent to e and $\bar{e}$ is a unit vector aligned to e. Then, the *discrete shape operator* can be defined on a vertex p_i by averaging the shape operators of the neighboring edges [Cohen-Steiner and Morvan, 2003].

$$\mathcal{S}(p_i) = \frac{1}{\text{area}(B)} \sum_{e \in E} \beta(e)\,|e \cap B|\,\bar{e}\cdot\bar{e}^\top. \tag{5}$$

Where B is a neighborhood of p_i and $|e \cap B|$ is the length of $e \cap B$. Figure 2 shows a schematic illustration of this operator. It is common to consider B being the dual face of p_i. This operator is the discrete analogous to the shape operator (Eq. (1)).

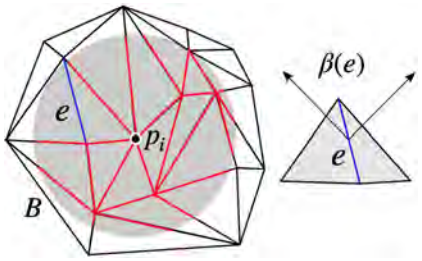


Figure 2: Discrete shape operator setting.

Again, the discrete shape operator is a 3×3 matrix, and in this case it is symmetric: there are exactly three eigenvalues and their respective eigenvectors. The normal N_i is the eigenvector associated with the smaller (in absolute) eigenvalue. The remaining eigenvalues are the principal curvatures, and their associated eigenvectors are the principal directions. The principal curvatures and principal directions are permuted. The *discrete Gaussian* curvature K_i and *mean* curvature H_i at a vertex p_i are the product and the average of the principal discrete curvatures.

4 Differentiable Neural Implicits

This section explores the differential geometry of the level sets of networks during their training. For the sampling, we use the curvatures of the dataset to prioritize important features.

4.1 Neural implicit function architecture

We consider the neural function $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ to be defined by

$$f_\theta(p) = W_n \circ f_{n-1} \circ f_{n-2} \circ \cdots \circ f_0(p) + b_n \tag{6}$$

where $f_i(p_i) = \varphi(W_i p_i + b_i)$ is the ith layer, and p_i is the output of f_{i-1}, i.e. $p_i = f_{i-1} \circ \cdots \circ f_0(p)$. The smooth *activation function* $\varphi : \mathbb{R} \to \mathbb{R}$ is applied to each coordinate of the affine map given by the linear map $W_i : \mathbb{R}^{N_i} \to \mathbb{R}^{N_{i+1}}$ translated by the *bias* $b_i \in \mathbb{R}^{N_{i+1}}$. The linear operators W_i can be represented as matrices and b_i as vectors. Therefore, the union of their coefficients corresponds to the parameters θ of f_θ.

We consider φ to be the sine function since the resulting network is suitable for reconstructing signals and can approximate all continuous functions in the cube $[-1,1]^3$ [Cybenko, 1989]. Recently, Sitzmann et al. [2020] proposed an initialization scheme for training the parameters of the network in the general context of signal reconstruction — implicit surfaces being a particular case. Here, we explore the main properties of this definition in implicit surface reconstruction. For example, its first layer is related to a *Fourier feature mapping* [Benbarka et al., 2021], which allows us to represent high-frequency three-dimensional implicit functions.

Another property of this network is its smoothness, which enables the use of differential geometry in the framework. For example, by Equation 1, the shape operator of a neural surface can be computed using its gradient and Hessian. These operators are also helpful during training and shading. As a matter of completeness we present their formulas below.

4.1.1 Gradient of a neural implicit function

The neural implicit function f_θ is smooth since its partial derivatives (of all orders) exist and are continuous. Indeed, each function f_i has all the partial derivatives because, by definition, it is an affine map with the smooth activation function φ applied to each coordinate. Therefore, the chain rule implies the smoothness of f_θ. We can compute the gradient of f_θ explicitly:

$$\begin{aligned}\nabla f_\theta(p) &= \mathbf{J}W_n\big(f_{n-1}\circ\cdots\circ f_0(p)\big)\cdot\cdots\cdot\mathbf{J}f_1\big(f_0(p)\big)\cdot\mathbf{J}f_0(p)\\ &= W_n\cdot\mathbf{J}f_{n-1}(p_{n-1})\cdot\cdots\cdot\mathbf{J}f_1(p_1)\cdot\mathbf{J}f_0(p),\end{aligned}\tag{7}$$

where $\mathbf{J}$ is the *Jacobian* and $p_i = f_{i-1}\circ\cdots\circ f_0(p)$. Calculations lead us to an explicit formula for the Jacobian of f_i at p_i.

$$\mathbf{J}f_i(p_i) = W_i\odot\varphi'\big[a_i|\cdots|a_i\big].\tag{8}$$

$\odot$ is the *Hadamard* product, and the matrix $\big[a_i|\cdots|a_i\big]$ has N_i copies of $a_i = W_i(p_i)+b_i\in\mathbb{R}^{N_{i+1}}$.

4.1.2 Hessian of a neural implicit function

Recall the *chain rule* formula for the *Hessian* operator of the composition of two maps $f:\mathbb{R}^m\to\mathbb{R}^n$ and $g:\mathbb{R}^n\to\mathbb{R}$:

$$\mathbf{H}(g\circ f)(p)=\mathbf{J}f(p)^\top\cdot\mathbf{H}g\big(f(p)\big)\cdot\mathbf{J}f(p)+\mathbf{J}g\big(f(p)\big)\cdot\mathbf{H}f(p)\tag{9}$$

We use this formula to compute the hessian $\mathbf{H}f_\theta$ of the network f_θ using induction on its layers.

Let $f = f_{i-1}\circ\cdots\circ f_0$ be the composition of the first i layers of f_θ, and g be the l-coordinate of the i-layer f_i. Suppose we have the hessian $\mathbf{H}f(p)$ and jacobian $\mathbf{J}f(p)$, from the previous steps of the induction. Then we only have to compute the hessian $\mathbf{H}g\big(f(p)\big)$ and jacobian $\mathbf{J}g\big(f(p)\big)$ to obtain $\mathbf{H}(g\circ f)(p)$. Equation (8) gives the formula of the jacobian of a hidden layer.

Expanding the Hessian $\mathbf{H}g(p)$ of the layer $g(p)=\varphi(w_lp+b_l)$ gives us the following formula.

$$\mathbf{H}g(p) = w_l^\top w_l\cdot\varphi''(w_lp+b_l).\tag{10}$$

Where w_l is the l-line of W and b_l is the l-coordinate of the bias b. When using $\varphi=\sin$, we have $\mathbf{H}g(p) = -w_l^\top w_l\cdot g(p)$.

4.2 Loss functional

Let S be a compact surface and $f:\mathbb{R}^3\to\mathbb{R}$ be its SDF. Here, we explore the loss functional $\mathcal{L}=\mathcal{L}_{\text{Eikonal}}+\mathcal{L}_{\text{Dirichlet}}+\mathcal{L}_{\text{Neumann}}$ used to train neural implicit functions. The training consists of seeking a minimum of $\mathcal{L}$ using the *gradient descent*. We present ways of improving the Dirichlet and Neumann constraints.

4.2.1 Signed distance constraint

In practice we have a sample of points $\{p_i\}_{i=1}^n$ being the vertices of a triangulation T of S. Then we replace $\mathcal{L}_{\text{Dirichlet}}$ by

$$\widetilde{\mathcal{L}}_{\text{Dirichlet}}(\theta) = \frac{1}{n}\sum_{i=1}^{n}|f_\theta(p_i)|. \tag{11}$$

Equation 11 forces $f_\theta = f$ on $\{p_i\}$, i.e. it asks for $\{p_i\} \subset f_\theta^{-1}(0)$. However, the neural surface $f_\theta^{-1}(0)$ could contain undesired spurious components. To avoid this, we improve $\widetilde{\mathcal{L}}_{\text{Dirichlet}}$ by including off-surface points. For this, consider the point cloud $\{p_i\}_{i=1}^{n+k}$ to be the union of the n vertices of T and a sample of k points in $\mathbb{R}^3 - S$. The constraint can be extended as follows.

$$\widetilde{\mathcal{L}}_{\text{Dirichlet}}(\theta) = \frac{1}{n+k}\sum_{i=1}^{n+k}|f_\theta(p_i) - f(p_i)| \tag{12}$$

The algorithm in Section 4.2.2 approximates f in $\{p_i\}_{i=n+1}^{n+k}$.

Sitzmann et al. [2020] uses an additional term $\int e^{-100|f_\theta|}dp$, to penalize off-surface points. However, this constraint takes a while to remove the spurious components in $f_\theta^{-1}(0)$. Gropp et al. [2020] uses a pre-training with off-surface points. Here, we use an approximation of the SDF during the sampling to reduce the error outside the surface. This strategy is part of our framework using computational/differential geometry.

4.2.2 Signed distance function

Here we describe an approximation of the SDF f of S for use during the training of the network f_θ. For this, we simply use the point-sampled surface consisting of n points $\{p_i\}$ and their normals $\{N_i\}$ to approximate the absolute of f:

$$|f(p)| \approx \min_{i\leq n}\|p - p_i\| \tag{13}$$

The sign of $f(p)$ at a point p is negative if p is inside S and positive otherwise. Observe that for each vertex p_i with a normal vector N_i, the sign of $\langle p - p_i, N_i\rangle$ indicates the side of the tangent plane that p belongs to. Therefore, we approximate the sign of $f(p)$ by adopting the dominant signs of the numbers $\langle p - p_j, N_j\rangle$, where $\{p_j\} \subset V$ is a set of vertices close to p. This set can be estimated using a spatial-indexing structure such as Octrees or KD-trees, to store the points $\{p_i\}$. Alternatively, we can employ *winding numbers* to calculate the sign of $f(p)$. Recent techniques enable a fast calculation of this function and extend it to point clouds [Barill et al., 2018].

4.2.3 Loss function using curvatures

We observe that instead of using a simple loss function, with the eikonal approach, our strategy using model curvatures leads to an implicit regularization. The on-surface constraint $\int 1 - \langle \nabla f_\theta, N\rangle\, dS$ requires the gradient of f_θ to be aligned to the normals of S. We extend this constraint by asking for the matching between the shape operators of $f_\theta^{-1}(0)$ and S. This can be achieved by requiring the alignment between their eigenvectors and the matching of their eigenvalues:

$$\int_S \sum_{i=1,2,3}\Big(1 - \langle (e_i)_\theta, e_i\rangle^2 + |(\kappa_i)_\theta - \kappa_i|\Big)dS, \tag{14}$$

where $(e_i)_\theta$ and $(\kappa_i)_\theta$ are the eigenvectors and eigenvalues of the shape operator of $f_\theta^{-1}(0)$, and e_i and κ_i are the eigenvectors and eigenvalues of the shape operator of S. We opt for the square of the dot product because the principal directions do not consider vector orientation. As the normal is, for both $f_\theta^{-1}(0)$ and S, one of the shape operator eigenvectors associated to the zero eigenvalue, Equation (14) is equivalent to:

$$\int_S 1 - \left\langle \frac{\nabla f_\theta}{\|\nabla f_\theta\|}, N\right\rangle dS + \int_S \sum_{i=1,2}\Big(1 - \langle (e_i)_\theta, e_i\rangle^2 + |(\kappa_i)_\theta - \kappa_i|\Big)dS \tag{15}$$

The first integral in Equation (15) coincides with $\mathcal{L}_{Neumann}$. In the second integral, the term $1 - \langle (e_i)_\theta, e_i \rangle^2$ requires the alignment between the principal directions, and $|(\kappa_i)_\theta - \kappa_i|$ asks for the matching of the principal curvatures. Asking for the alignment between $(e_1)_\theta$ and e_1 already forces the alignment between $(e_2)_\theta$ and e_2, since the principal directions are orthogonal.

We can weaken the second integral of Equation (15) by considering the difference between the mean curvatures $|H_\theta - H|$ instead of $|(\kappa_1)_\theta - \kappa_1| + |(\kappa_2)_\theta - \kappa_2|$. This is a weaker restriction because $|H_\theta - H| \leq \frac{1}{2}|(\kappa_1)_\theta - \kappa_1| + \frac{1}{2}|(\kappa_2)_\theta - \kappa_2|$. However, it reduces the computations during optimization, since the mean curvature H_θ is calculated through the divergence of $\frac{\nabla f_\theta}{\|\nabla f_\theta\|}$.

Next, we present the sampling strategies mentioned above for use in the training process of the neural implicit function f_θ.

4.3 Sampling

Let $\{p_i, N_i, \mathcal{S}_i\}$ be a sample from an *unknown* surface S, where $\{p_i\}$ are points on S, $\{N_i\}$ are their normals, and $\{\mathcal{S}_i\}$ are samples of the shape operator. $\{p_i\}$ could be the vertices of a triangle mesh and the normals and curvatures be computed using the formulas given in Section 3.4. Let $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ be a neural implicit function, as we saw in Section 4.2, its training consists of defining a loss functional $\mathcal{L}$ to force f_θ to be the SDF of S.

In practice, $\mathcal{L}$ is evaluated on a dataset of points dynamically sampled at training time. This consists of a sampling of on-surface points in $\{p_i\}$ and a sampling of off-surface points in $\mathbb{R}^3 - S$. For the off-surface points, we opted for an uniform sampling in the domain of f_θ. Additionally, we could bias the off-surface sampling by including points in the *tubular neighborhood* of S — a region around the surface given by a disjoint union of segments along the normals.

The shape operator encodes important geometric features of the data. For example, regions containing points with higher principal curvatures κ_1 and κ_2 in absolute codify more details than points with lower absolute curvatures. These are the elliptic ($\kappa_1\kappa_2 > 0$), hyperbolic ($\kappa_1\kappa_2 < 0$), or parabolic points (when only one κ_i is zero). Regions consisting of points close to planar, where $|\kappa_1|$ and $|\kappa_2|$ are small, contain less geometric information, thus, we do not need to visit all of them during sampling. Also, planar points are abundant, see Figure 1.

We propose a non-uniform strategy to select the on-surface samples $\{p_i\}$ using their curvatures to obtain faster learning while maintaining the quality of the end result. Specifically, we divide $\{p_i\}$ in three sets V_1, V_2, and V_3 corresponding to *low*, *medium*, and *high* feature points. For this, choosing $n = n_1 + n_2 + n_3$, with $n_i > 0$ integer, and sorting $\{p_i\}$ using the *feature* function $\kappa = |\kappa_1| + |\kappa_2|$, we define $V_1 = \{p_i \,|\, i \leq n_1\}$, $V_2 = \{p_i \,|\, i > n_1 \text{ and } i \leq n_1 + n_2\}$, and $V_3 = \{p_i \,|\, i > n_1 + n_2\}$. Thus, V_1 is related to the planar points, and $V_2 \cup V_3$ relates to the parabolic, hyperbolic and parabolic points.

Therefore, during the training of f_θ, we can prioritize points with more geometrical features, those in $V_2 \cup V_3$, to accelerate the learning. For this, we sample less points in V_1, which contains data redundancy, and increase the sampling in V_2 and V_3.

The partition $V = V_1 \sqcup V_2 \sqcup V_3$ resembles the decomposition of the graph of an image in *planar*, *edge*, and *corner* regions, the *Harris corner detector* [Harris et al., 1988]. Here, $V_2 \cup V_3$ coincides with the union of the edge and corner regions.

We chose this partition because it showed good empirical results (see Sec 5.2), however, this is one of the possibilities. Using the absolute of the Gaussian or the mean curvature as the feature function has also improved the training. In the case of the mean curvature, the low feature set contains regions close to a *minimal* surface. In future works, we intend to use the regions contained in the neighborhood of extremes of the principal curvatures, the so-called *ridges* and *ravines*.

5 Experiments

We first consider the point-sampled surface of the Armadillo, with n=172974 vertices $\{p_i\}$, to explore the loss function and sampling schemes given in Section 4. We chose this mesh because it is a classic model with well-distributed curvatures. We approximate its SDF using a network f_θ with three hidden layers $f_i : \mathbb{R}^{256} \to \mathbb{R}^{256}$, each one followed by a sine activation.

We train f_θ using the loss functional $\mathcal{L}$ discussed in Section 4.2. We seek a minimum of $\mathcal{L}$ by employing the ADAM *algorithm* [Kingma and Ba, 2014] with a *learning rate* $1e-4$ using minibatches of size $2m$, with m on-surface points sampled in the dataset $\{p_i\}$ and m off-surface points uniformly sampled in $\mathbb{R}^3 - \{p_i\}$. After $\lceil \frac{n}{m} \rceil$ iterations of the algorithm, we have one *epoch* of training, which is equivalent to passing through the whole dataset once. We use the initialization of parameters of Sitzmann et al. [2020].

The proposed model can represent geometric details with precision. Figure 3 shows the original Armadillo and its reconstructed neural implicit surface after 1000 epochs of training.

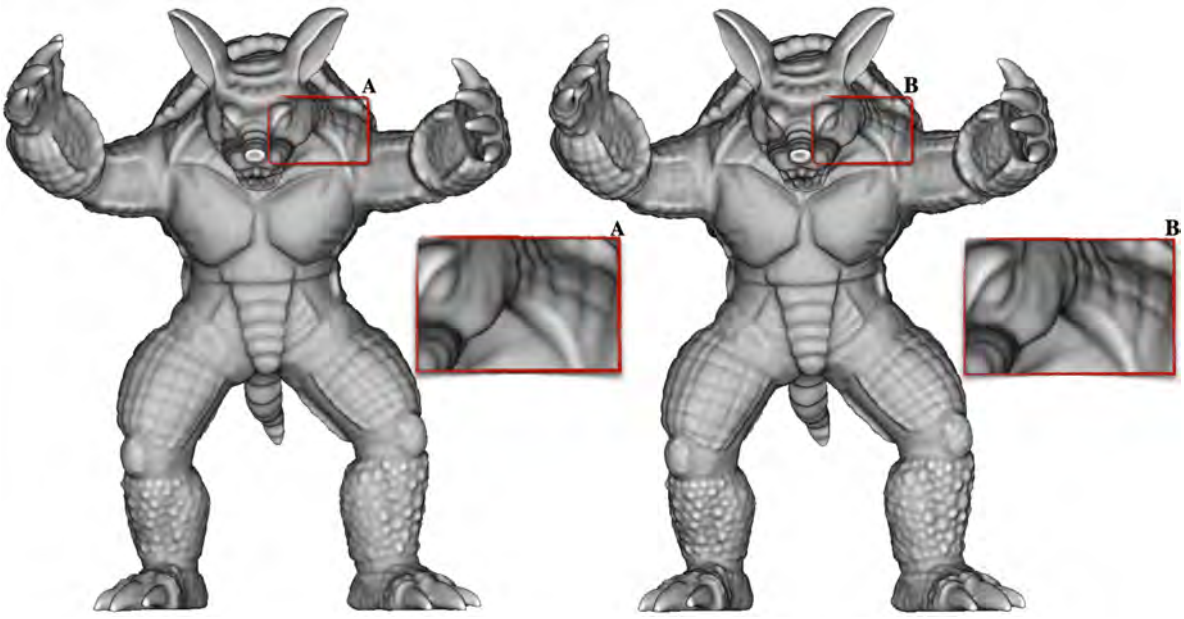


Figure 3: Comparison between the ground truth Armadillo model (right) and its reconstructed neural implicit surface (left) after 1000 epochs of training.

Next, we use the differential geometry of the point-sampled surface to improve the training of f_θ by adding curvature constraints in $\mathcal{L}$ and changing the sampling of minibatches in order to prioritize the points with more geometrical information.

5.1 Loss functional

As we saw in Section 4.2, we can improve the loss function $\mathcal{L} = \mathcal{L}_{\text{Eikonal}} + \mathcal{L}_{\text{Dirichlet}} + \mathcal{L}_{\text{Neumann}}$ by adding curvature terms. Here, we use the alignment between the direction of maximum curvature $(e_1)_\theta$ of $f_\theta^{-1}(0)$ and the principal direction e_1 of the (ground truth) surface S, which leads to a regularization term.

$$\mathcal{L}_{\text{Dir}}(\theta) = \int_E 1 - \langle e_1, (e_1)_\theta \rangle^2 \, dS \tag{16}$$

To evaluate $\mathcal{L}_{\text{Dir}}$ in $\{p_i\}$, we calculate e_1 in a pre-processing step considering $\{p_i\}$ be the vertices of a mesh. Due to possible numerical errors, we restrict $\mathcal{L}_{\text{Dir}}$ to a region $E \subset S$ where $|\kappa_1 - \kappa_2|$ is high. A point with $|\kappa_1 - \kappa_2|$ small is close to be *umbilical*, where the principal directions are not defined.

Figure 4 compares the training of f_θ using the loss function $\mathcal{L}$ (line 1) with the improved loss function $\mathcal{L} + \mathcal{L}_{\text{Dir}}$ (line 2).

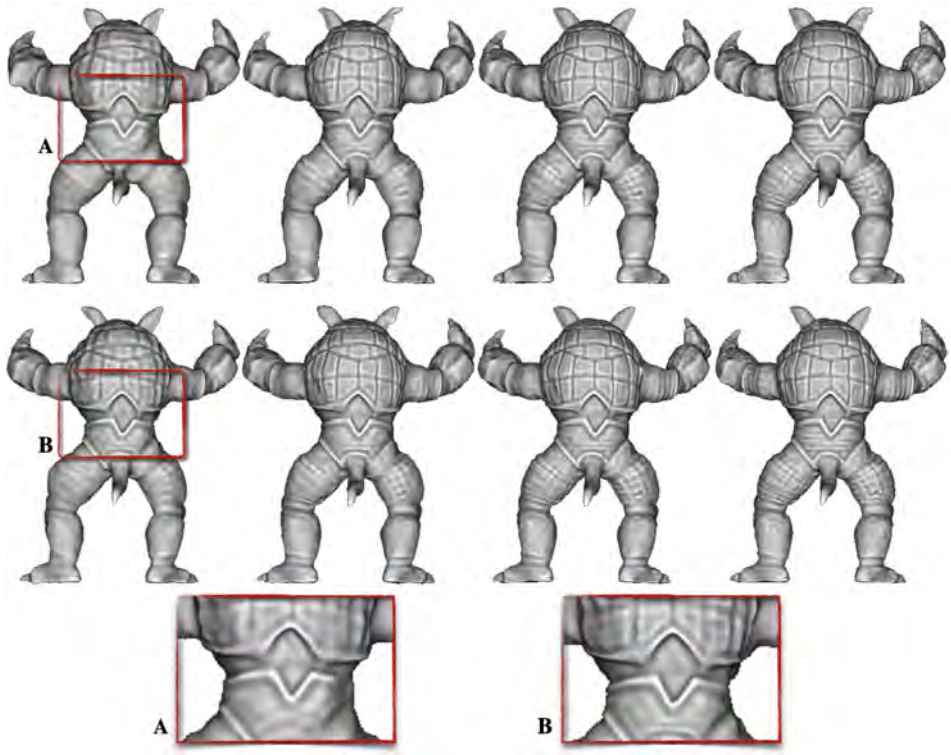


Figure 4: Neural implicit surfaces trained to approximate the Armadillo. The columns indicate the neural surfaces after 100, 200, 300, and 500 epochs of training. Line 1 shows the results using the SDF functional. Line 2 also consider the alignment between the principal directions in the loss functional.

Asking for the alignment between the principal directions during training adds a certain redundancy since we are already asking for the alignment between the normals N: the principal directions are extremes of dN. However, as we can see in Figure 4 it may reinforce the learning. Furthermore, this strategy can lead to applications that rely on adding curvature terms in the loss functional. For example, we could choose regions of a neural surface and ask for an enhancement of its geometrical features. Another application could be deforming regions of a neural surface [Yang et al., 2021].

5.2 Sampling

This section presents experiments using the sampling discussed in Section 4.3. This prioritizes points with important features during the training implying a fast convergence while preserving quality (see Fig. 9). Also, this analysis allows finding data redundancy, i.e., regions with similar geometry.

During training it is common to sample minibatches uniformly. Here, we use the curvatures to prioritize the important features. Choosing $n = n_1 + n_2 + n_3$, with $n_i \in \mathbb{N}$, we define the sets V_1, V_2, and V_3 of low, medium, and high feature points.

We sample minibatches of size $m = p_1 m + p_2 m + p_3 m = 10000$, with $p_i m$ points on V_i. If this sampling is uniform, we would have $p_i = \frac{n_i}{n}$. Thus, to prioritize points with more geometrical features, those in V_2 and V_3, we reduce p_1 and increase p_2 and p_3. Figure 5 gives a comparison between the uniform sampling (first line) and the adaptive sampling (line 2) that consider $p_i = 2\frac{n_i}{n}$ for $i = 2, 3$, i.e. it duplicates the proportion of points with medium and high features. Clearly, these new proportions depend on n_i. In this experiment, we use $n_1 = \frac{n}{2}$, $n_2 = \frac{4n}{10}$, and $n_3 = \frac{n}{10}$, thus V_1 contains half of V. This sampling strategy improved the rate convergence significantly.

Returning to minibatch sampling. In the last experiment, we were sampling more points with medium and high features in $V_2 \cup V_3$ than points with low features in V_1. Thus the training visits $V_2 \cup V_3$ more than once per epoch. We propose to reduce the number of points sampled per epoch, prioritizing the most important ones. For this, we reduce the size of the minibatch in order to sample each point of $V_2 \cup V_3$ once per epoch.

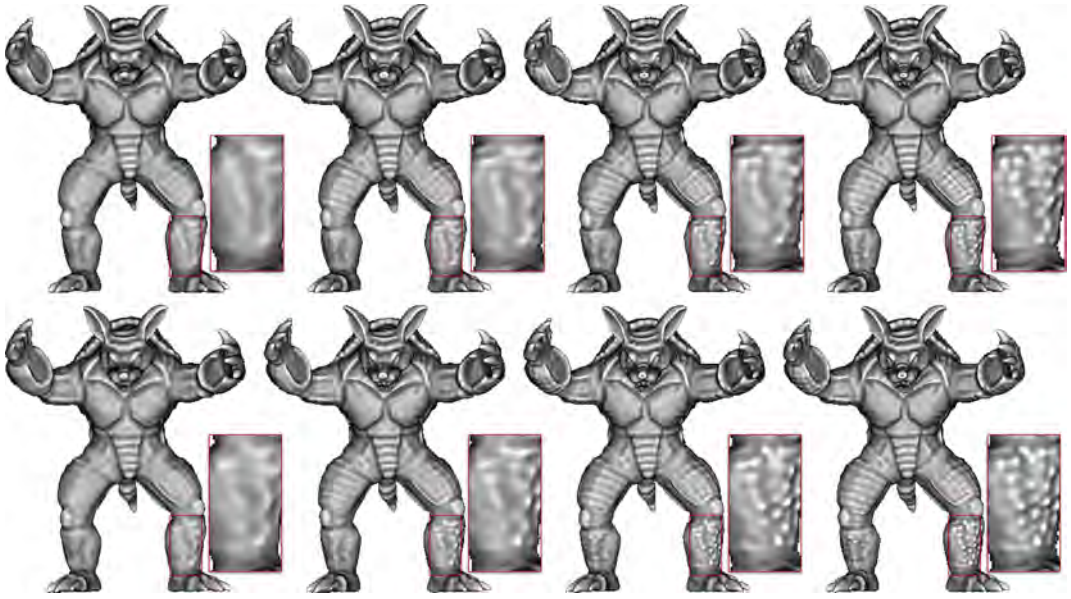


Figure 5: Neural implicit surfaces approximating the Armadillo model. The columns indicate the zero-level sets of the neural implicit functions after 29, 52, 76, and 100 epochs of training. Line 1 shows the results using minibatches sampled uniformly in V. Line 2 presents the results using the adapted sampling of minibatches with 10% / 70% / 20% of points with low/medium/high features.

Figure 6 provides a comparison between the two training strategies. The first line coincides with the experiment presented in the second line of Figure 5. It uses minibatches of size m, and $\lceil \frac{n}{m} \rceil$ iterations of the gradient descent per epoch. In the second line, we sample minibatches of size $\frac{m}{2}$ and use $\lceil \frac{n}{m} \rceil$ iterations of the gradient descent. Then the second line visits half of the dataset per epoch. We are using the same minibatch proportions p_i and sets V_i, as in the previous experiment.

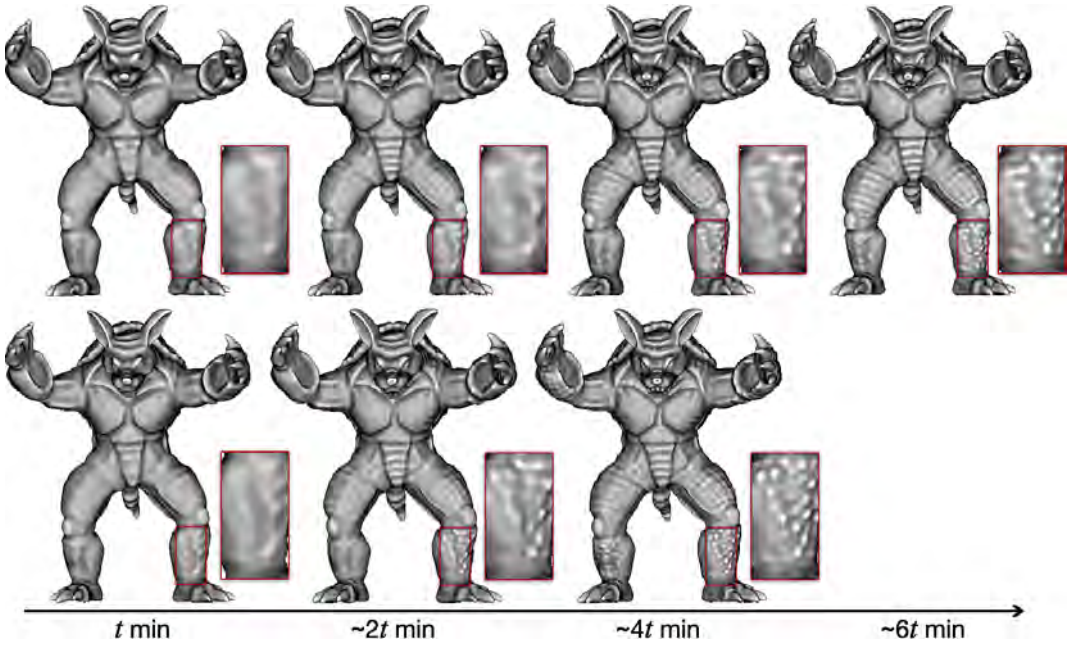


Figure 6: The columns indicate the zero-level sets of the neural implicit functions after t, $2t$, $4t$, and $6t$ minutes of training. Line 1 shows the results using the n points of the dataset per epoch and minibatches of size m containing 20% /60%/20% of points with low/medium/high features. The results in line 2 use $\frac{n}{2}$ points per epoch and minibatches of $\frac{m}{2}$ points with 20% /60%/20% of low/medium/high features. Both experiments use $\lceil \frac{n}{m} \rceil$ steps per epoch.

As we can see in Figure 6, going through all the points with medium and higher features once per epoch, while reducing the number of points with low features, resulted in faster training with better quality results.

To keep reducing the size of the minibatches such that we visit important points once per epoch, we consider smaller medium and high feature sets. That is, to visit $V_2 \cup V_3$ once per epoch it is necessary to update the sizes n_i of the feature sets V_i. For this, we present three experiments that use $n_1 = \frac{6n}{10}, \frac{75n}{100}, \frac{85n}{100}$, $n_2 = \frac{3n}{10}, \frac{2n}{10}, \frac{n}{10}$, and $n_3 = \frac{n}{10}, \frac{5n}{100}, \frac{5n}{100}$, respectively. Then we can consider minibatches of size $\frac{m}{2}$, $\frac{3m}{10}$, and $\frac{m}{10}$, i.e. 50%, 30% and 10% of the original minibatch of size m. Figure 7 shows the results of the experiments. They are all using $\lceil \frac{n}{m} \rceil$ iterations per epoch, then, we visit $\frac{n}{2}$, $\frac{3n}{10}$, and $\frac{n}{10}$ points of the dataset on each epoch, respectively. Thus, as we reduce the minibatches sizes, we remove points from $V_2 \cup V_3$, which implies that we are going to learn fewer intermediate features. This can be visualized in Figure 7. Observe that points with higher features, such as the shin, thighs and abdomen, are learned faster than points with lower features, mainly around the Armadillo chest.

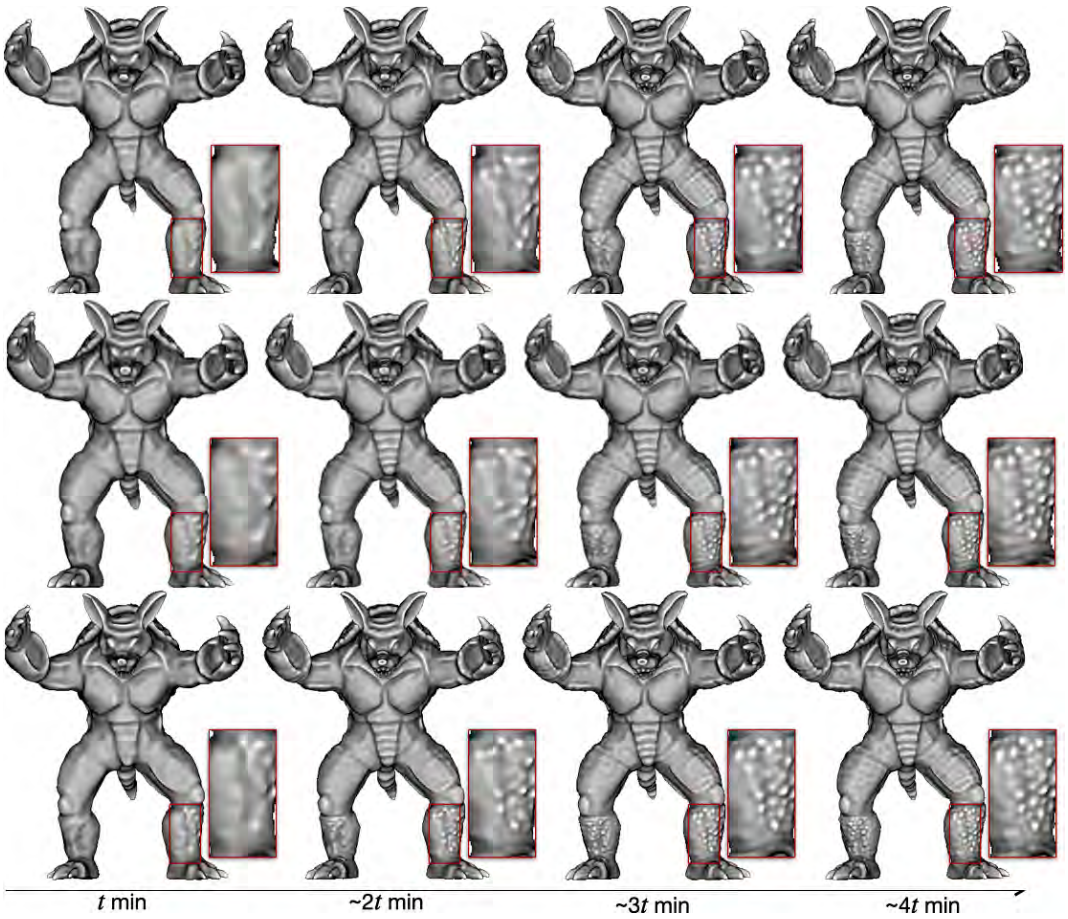


Figure 7: The columns present the zero-level sets of the model after t, $2t$, $3t$, and $4t$ minutes of training. The results in line 1 use $\frac{n}{2}$ points per epoch and minibatches of $\frac{m}{2}$ points with 20% /60%/20% of low/medium/high features. Line 2 shows the results using $\frac{3n}{10}$ points per epoch and minibatches of $\frac{3m}{10}$ points with 17% /67%/17% of low/medium/high features. The results in line 3 use $\frac{n}{10}$ points per epoch and minibatches of $\frac{m}{10}$ points with 10% /70%/20% of low/medium/high features. The experiments use $\lceil \frac{n}{m} \rceil$ steps per epoch.

Finding the optimal parameters of the proposed curvature-based sampling can be a hard task for a general surface. The experiments above show empirically that visiting the "important" points once per epoch implies good geometrical results. Another way to select important points is using the concept of *ridge* and *ravine* curves [Ohtake et al., 2004]. These are extremes of the principal curvatures along the principal directions and indicate where the surface bends sharply [Belyaev et al., 1998] .

5.3 Additional experiments and comparisons

This section presents a comparison of our framework with RBF [Carr et al., 2001] and SIREN [Sitzmann et al., 2020]. For this, we introduce two analytical surfaces with ground truth in closed form: sphere and torus; and four discrete surfaces: Bunny, Dragon, Buddha and Lucy. We choose these models because they have distinct characteristics: Bunny has a simple topology/geometry, Dragon and Lucy have a simple topology with complex geometry, and Buddha has a complex topology/geometry. It is important to mention that the goal is not showing that our method outperforms RBF, but is comparable in the function approximation task.

For method comparison, we consider RBF since it is a well-established method to estimate an implicit function from a point-sampled surface. Although both RBF and our approach can approximate implicit functions, their nature is fundamentally different. RBF solves a system of equations to weigh the influence of pairs $\{p_i, f_i\}$ on neighboring points. If we wish to include normal alignment (Hermite data) in this process [Macêdo et al., 2011], it demands profound changes in the interpolant estimation. However, including Hermite data in neural implicit models demands only additional terms in the loss function, the model architecture and training process remains unchanged.

To reconstruct the sphere and the torus models we consider a network f_θ consisting of two hidden layers $f_i : \mathbb{R}^{80} \to \mathbb{R}^{80}$ and train its parameters for each model using the basic configuration given in Section 4.2. We trained f_θ for 500 epochs considering batches of $m = 2500$ on-surface points and m off-surface points. For SIREN, we use the same network architecture and uniform sampling scheme, only the loss function was replaced by the original presented in [Sitzmann et al., 2020]. For the RBF interpolant, we use a dataset of $m = 2500$ on-surface points and m off-surface points.

We reconstruct the other models using a neural function f_θ consisting of three hidden layers $f_i : \mathbb{R}^{256} \to \mathbb{R}^{256}$. Again, we train θ using the basic training framework. We consider minibatches of $m = 10000$ on-surface points and m off-surface points. For SIREN, we use the same network architecture and the loss function and sampling scheme given in [Sitzmann et al., 2020]. For the RBF interpolant, we used a dataset of 10000 on-surface points and 10000 off-surface points.

Table 1 presents the quantitative comparisons of the above experiments. We compare the resulting SDF approximations with the ground truth SDFs using the following measures:

- The absolute difference $|\bar{f} - f|$, in the domain $\mathbb{R}^3 - S$ and on the surface, between the function approximation $\bar{f}$ and the ground truth SDF f;
- The normal alignment $1 - \left\langle \frac{\nabla \bar{f}}{\|\nabla \bar{f}\|}, \nabla f \right\rangle$ between the gradients $\nabla \bar{f}$ and ∇f on the surface.

We used a sample of 2500 on-surface points, not included in the training process, to evaluate the mean and maximum values of each measure. We also ran the experiments 100 times and took the average of the measures. Note that, for the RBF interpolation, we did not calculate the analytical derivatives because we are using a framework without support for this feature, a numerical method was employed in this case.

Our method provides a robust SDF approximation even compared with RBF. Figure 8 gives a visual evaluation presenting a sphere tracing of the zero-level sets of the SIREN and our method. In both cases, we used an image resolution of 1024×1024 and 80 sphere tracing iterations. Since we obtain a better SDF approximation the algorithm is able to ray cast the surface with precision avoiding spurious components.

We did not visualize RBF approximations because the algorithm implemented in SciPy [Virtanen et al., 2020], which is employed in this work, is not fully optimized, making the ray tracing unfeasible.

Table 2 shows the average training and inference time for RBF, SIREN, and our method. For this experiment, we train SIREN and our method for 50 epochs using 20000 points per batch, only on CPU, to provide a fair comparison. As for RBF, we used a single batch of points to build the interpolant, with each point weighting the 300 nearest points, to diminish the algorithm's memory requirements. Training time for RBF consists mostly of creating the matrices used to solve the interpolation problem. This is a relatively simple step, thus as expected, takes only a fraction of time compared to other methods. Still regarding training time, SIREN and our method are in the same magnitude, with SIREN being slightly faster in all experiments. This is mainly due to our method performing SDF querying at each training step. Even with efficient algorithms, this step impacts measurably in the training performance.

	Method	$\lvert\bar{f} - f\rvert$ in the domain		$\lvert\bar{f} - f\rvert$ on the surface		Normal alignment	
		mean	max	mean	max	mean	max
Sphere	RBF	4e-5	0.021	5e-8	1e-4	1.81e-6	1.36e-5
	SIREN	0.129	1.042	0.0031	0.013	6e-4	0.005
	Ours	0.001	0.015	0.0018	0.007	6e-5	6e-4
Torus	RBF	6e-4	0.055	2e-5	0.001	1.61e-5	3.17e-4
	SIREN	0.254	1.006	0.0034	0.013	0.0007	0.005
	Ours	0.003	0.036	0.0029	0.011	0.0002	0.002
Bunny	RBF	0.002	0.024	0.0002	0.004	0.0007	0.453
	SIREN	0.145	0.974	0.0010	0.004	0.0006	0.019
	Ours	0.003	0.081	0.0015	0.005	0.0005	0.017
Dragon	RBF	0.002	0.035	0.0006	0.009	0.0160	1.459
	SIREN	0.106	1.080	0.0010	0.006	0.0063	0.866
	Ours	0.003	0.104	0.0010	0.005	0.0034	0.234
Armadillo	RBF	0.003	0.008	0.0030	0.056	0.0134	1.234
	SIREN	0.126	0.941	0.0010	0.005	0.0021	0.168
	Ours	0.009	0.136	0.0012	0.006	0.0016	0.164
Lucy	RBF	0.002	0.048	0.0003	0.011	0.1581	1.998
	SIREN	0.384	1.048	0.0007	0.003	0.0070	0.313
	Ours	0.013	0.155	0.0009	0.006	0.0056	0.170
Buddha	RBF	0.002	0.050	0.0004	0.010	0.0687	1.988
	SIREN	0.337	1.124	0.0007	0.008	0.0141	1.889
	Ours	0.096	0.405	0.0069	0.024	0.0524	1.967

Table 1: Comparison between RBF, SIREN, and our framework. We consider two analytical models, the sphere and the torus, and five classical computer graphics models, *Bunny, Dragon, Armadillo, Happy Buddha*, and *Lucy*.

	Method	Training Time (s)	Inference Time (s)
Bunny	RBF	0.0055	417.3928
	SIREN	173.6430	0.5773
	Ours	199.3146	0.6460
Dragon	RBF	0.0046	411.1710
	SIREN	319.8439	0.5565
	Ours	391.4102	0.5885
Armadillo	RBF	0.0045	392.0836
	SIREN	380.5361	0.9522
	Ours	443.3634	0.9290
Buddha	RBF	0.0044	410.6234
	SIREN	1297.0681	0.9158
	Ours	1646.2311	0.9689
Lucy	RBF	0.0077	358.7987
	SIREN	560.1297	0.8888
	Ours	654.1596	0.8023

Table 2: Comparison between RBF, SIREN, and our framework. We use the same models as in Table 1, except for the sphere and torus.

Regarding inference time, both our method and SIREN take less than a second for all models in a 64^3 grid. As for RBF, the inference time is close to 400 seconds for all tested cases. It is affected by the size of the interpolant, which explains the proximity in inference performance even for complex

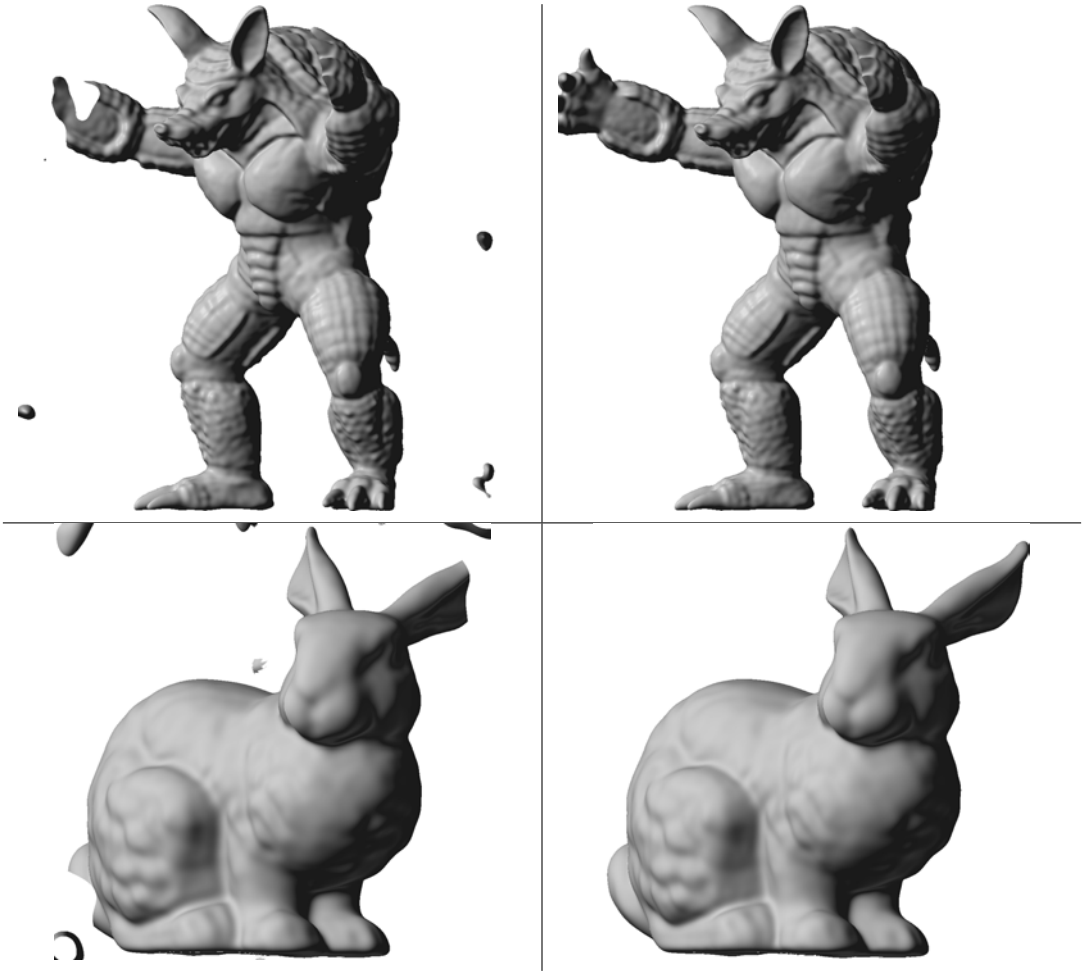


Figure 8: Sphere tracing of neural surfaces representing the Armadillo and Bunny models. On the left, the network was trained using the SIREN framework. On the right, the results using our method. Both networks have the same architecture and were trained on the same data during 500 epochs.

models (Buddha and Dragon). The RBF inference could be improved using *partition of unity* [Ohtake et al., 2003, 2006] or *fast multipole method* [Greengard and Rokhlin, 1997]. Here, we opt for the implementation in Scipy [Virtanen et al., 2020] of the RBF approach since it is widely available.

Figure 9 shows the training loss per epoch for each considered model. We did not include the Dragon because its loss function behavior is similar to the Bunny. Note that the Dirichlet condition for on-surface points (sdf_on_surf) quickly converges and approaches zero at the first 5 epochs. In all tested cases, the off-surface Dirichlet condition (sdf_off_surf) converges quickly as well, usually by the first 20 epochs. The Eikonal/Neumann constraints take longer to converge, with the notable example of Buddha, where the Neumann constraint remains at a high level, albeit still decreasing, after 100 epochs.

5.4 Curvature estimation

Another application of our work is the use of a neural network $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ to estimate differential properties of a triangle mesh T. We train f_θ to approximate the SDF of T. Since the vertices of T lie in a neighborhood of the zero-level set of f_θ we use the network to *map* properties of its level sets to T. Afterwards, we can exploit the differentiability of f_θ to estimate curvature measures on T. Figure 10 shows an example of this application. We trained two neural implicit functions to approximate the SDF of the Bunny and Dragon models. We then analytically calculate the mean curvature on each vertex by evaluating $\operatorname{div} \frac{\nabla f_\theta}{\|\nabla f_\theta\|}$. Compared to classical discrete methods, the curvature calculated using f_θ is smoother and still respects the global distribution of curvature of the original mesh. We computed

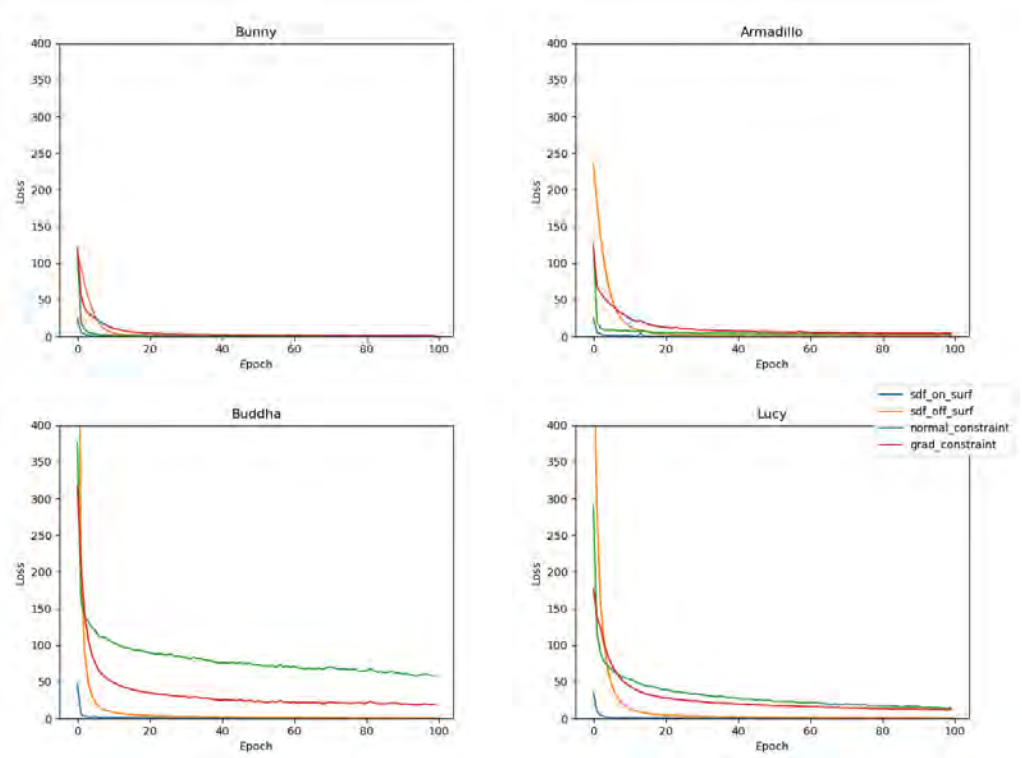
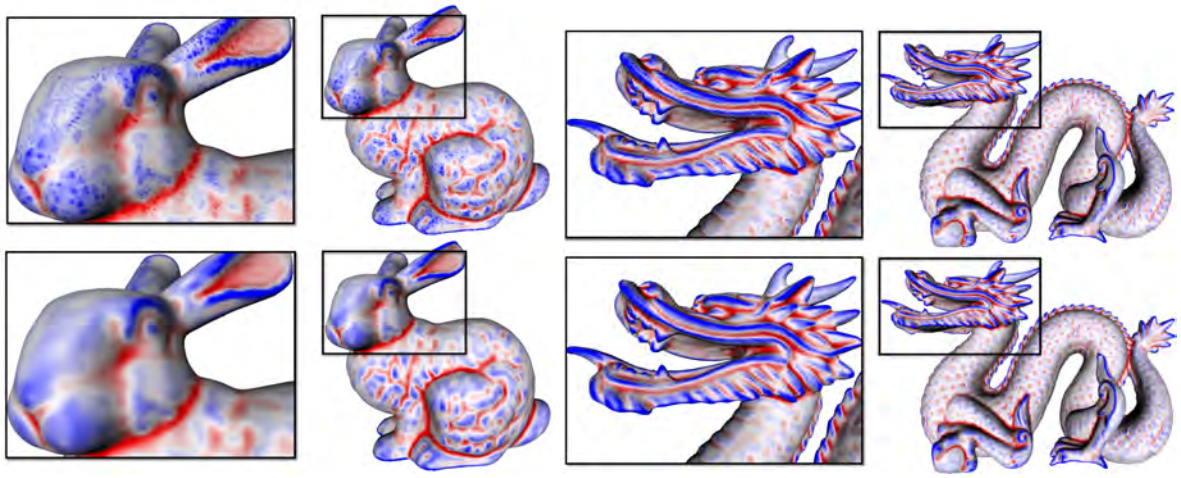


Figure 9: Training loss per epoch for all constraints for the Bunny (top-left), Armadillo (top-right), Buddha (bottom-left), and Lucy (bottom-right).

the discrete mean curvatures using the method proposed by Meyer et al. [2003]. For our method, we used PyTorch's automatic differentiation module (`autograd`) [Paszke et al., 2019].



(a) Bunny (b) Dragon

Figure 10: Estimating mean curvature using neural implicit functions. On both images, we present the curvature at each vertex of the Bunny (10a) and Dragon (10b) overlaid as color on the mesh. On the top of each figure, the discrete mean curvature was calculated using the method in [Meyer et al., 2003], while on the bottom of each figure, we used $\operatorname{div} \frac{\nabla f_\theta}{\|\nabla f_\theta\|}$, where f_θ approximates an SDF of meshe.

5.5 Anisotropic shading

Another application is the use of the principal directions of curvatures in the rendering.

Let f_θ be a network such that its zero-level set S_θ approximates the Armadillo. We present a sphere tracing visualization of S_θ using its intrinsic geometry. For this, we consider `PyTorch` to compute the shape operator of S_θ. We use its principal directions v_1 and v_2 to compute an *anisotropic* shading based on the *Ward reflectance model* [Ward, 1992]. It consists of using the following specular coefficient at each point $p \in S_\theta$.

$$k_{spec} = \frac{1}{4\pi\alpha_1\alpha_2\sqrt{\langle N,l\rangle\,\langle N,v\rangle}} \cdot \exp\left(-2\frac{\left(\frac{\langle H,v_1\rangle}{\alpha_1}\right)^2 + \left(\frac{\langle H,v_2\rangle}{\alpha_2}\right)^2}{1+\langle N,H\rangle}\right).$$

Where N is the normal at p, v is the unit direction from p to the observer, l is the unit direction from p to the light source, $H = \frac{v+l}{\|v+l\|}$, and α_i are two parameters to control the anisotropy along the principal directions v_i. Figure 11 presents two anisotropic shadings of S_θ. The first considers $\alpha_1 = 0.2$ and $\alpha_2 = 0.5$, and the second uses $\alpha_1 = 0.5$ and $\alpha_2 = 0.2$. We used white as the specular color and gray as the diffuse color.

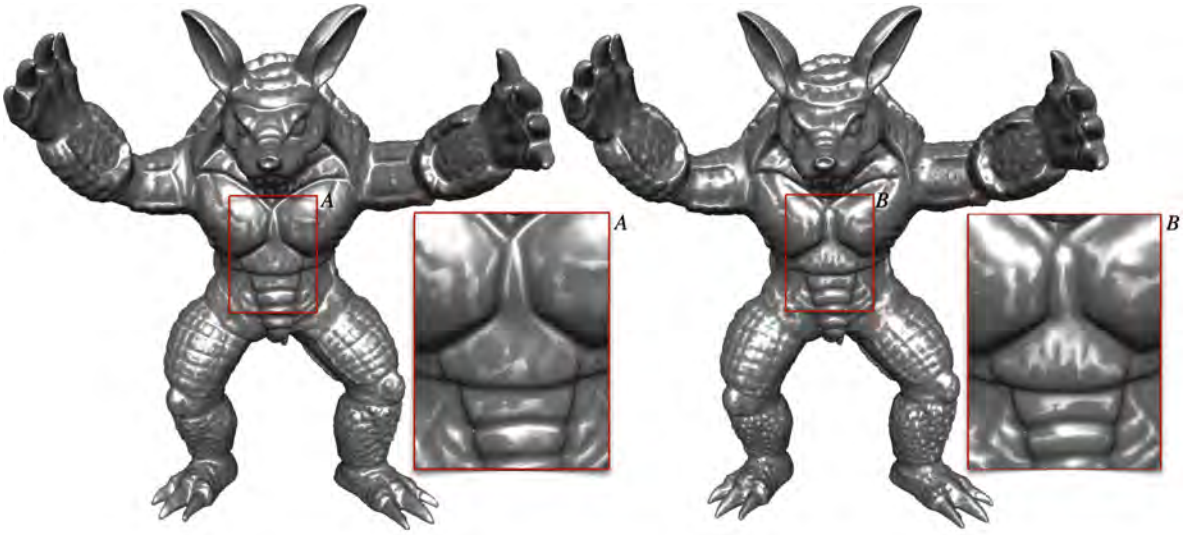


Figure 11: Ward anisotropic specular reflectance on the Armadillo neural surface. On the left, we consider a high deviation towards the maximum curvature directions. On the right, it presents the analogous for the minimum curvature directions. The principal directions were computed analytically using `PyTorch`.

5.6 Limitations

The main bottleneck in our work is the SDF estimation for off-surface points. We use an algorithm implemented in the `Open3D` library [Zhou et al., 2018]. Even with parallelization, this method still runs on the CPU, thus greatly increasing the time needed to train our neural networks. Also our method is designed to run with implicit visualization schemes, such as sphere tracing. However the inference time still does not allow for interactive frame-rates using traditional computer graphics pipelines. Besides recent advances in real-time visualizations of neural implicits [da Silva et al., 2022, Takikawa et al., 2021], this is still a challenge for future works. Finally, surfaces with sharp edges can not be accurately represented using smooth networks. Thus, trying to approximate them using smooth functions may lead to inconsistencies.

5.7 Hardware

To run all of those comparisons and tests, we used a computer with an i7-9700F with 128GiB of memory and an NVIDIA RTX 3080 with 10GiB of memory. Even to run our model in modest

hardware, our strategy is lightweight with 198.657K parameters and 197.632K multiply-accumulate (MAC) operations. Running on another computer with an AMD Ryzen 7 5700G processor, 16GiB of memory, and an NVIDIA GeForce RTX 2060 with 6GiB of memory, our model took 1.32 seconds to process 172974 point samples of the Armadillo mesh.

6 Conclusions and future works

We introduced a neural network framework that exploits the differentiable properties of neural networks and the discrete geometry of point-sampled surfaces to represent them as neural surfaces. The proposed loss function can consider terms with high order derivatives, such as the alignment between the principal directions. As a result, we obtained reinforcement in the training, gaining more geometric details. This strategy can lead to modeling applications that require curvature terms in the loss function. For example, we could choose regions of a surface and ask for an enhancement of its features.

We also present a sampling strategy based on the discrete curvatures of the data. This allowed us to access points with more geometric information during the sampling of minibatches. As a result, this optimization trains faster and has better geometric accuracy, since we were able to reduce the number of points in each minibatch by prioritizing the important points.

This work emphasized the sampling of on-surface points during the training. Future work includes a sampling of off-surface points. Using the *tubular neighborhood* of the surface can be a direction to improve the sampling of off-surface points.

Acknowledgments

We are very grateful to the anonymous reviewers for their careful and detailed comments and suggestions. We gratefully acknowledge the support from CNPq.

References

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics*, 9(1):3–15, 2003.

Nina Amenta, Sunghee Choi, Tamal K Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 213–222, 2000.

Gavin Barill, Neil Dickson, Ryan Schmidt, David I.W. Levin, and Alec Jacobson. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics*, 2018.

Alexander G Belyaev, Alexander A Pasko, and Tosiyasu L Kunii. Ridges and ravines on implicit surfaces. In *Proceedings. Computer Graphics International (Cat. No. 98EX149)*, pages 530–535. IEEE, 1998.

Nuri Benbarka, Timon Höfer, Andreas Zell, et al. Seeing implicit neural representations as fourier series. *arXiv preprint arXiv:2109.00249*, 2021.

Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, 2001.

Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling, 2019.

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

David Cohen-Steiner and Jean-Marie Morvan. Restricted delaunay triangulations and normal cycle. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 312–321, 2003.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Vinícius da Silva, Tiago Novello, Guilherme Schardong, Luiz Schirmer, Hélio Lopes, and Luiz Velho. Mip-plicits: Level of detail factorization of neural implicits sphere tracing. *arXiv preprint*, 2022.

Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. June 2020.

Manfredo P Do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.

Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas A. Funkhouser. Deep structured implicit functions. *CoRR*, abs/1912.06126, 2019. URL http://arxiv.org/abs/1912.06126.

Ron Goldman. Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design*, 22(7):632–658, 2005.

Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 135(2):280–292, 1997.

Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020.

Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

Evangelos Kalogerakis, Derek Nowrouzezahrai, Patricio Simari, and Karan Singh. Extracting lines of curvature from noisy point clouds. *Computer-Aided Design*, 41(4):282–292, 2009.

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Moller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization, 2003. VIS 2003.*, pages 513–520. IEEE, 2003.

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

Ives Macêdo, Joao Paulo Gois, and Luiz Velho. Hermite radial basis functions implicits. In *Computer Graphics Forum*, volume 30, pages 27–42. Wiley Online Library, 2011.

Boris Mederos, Luiz Velho, and Luiz Henrique de Figueiredo. Robust smoothing of noisy point clouds. In *Proc. SIAM Conference on Geometric Design and Computing*, volume 2004, page 2. SIAM Philadelphia, PA, USA, 2003.

Lars M. Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. *CoRR*, abs/1812.03828, 2018. URL http://arxiv.org/abs/1812.03828.

Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*, pages 35–57. Springer, 2003.

Mateusz Michalkiewicz. Implicit surface representations as layers in neural networks. In *International Conference on Computer Vision (ICCV)*. IEEE, 2019.

Niloy J Mitra and An Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 322–328, 2003.

Tiago Novello, Vinícius da Silva, Guilherme Schardong, Luiz Schirmer, Hélio Lopes, and Luiz Velho. Neural implicit surfaces in higher dimension. *arXiv preprint*, 2022.

Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, jul 2003. ISSN 0730-0301. doi: 10.1145/882262.882293. URL https://doi.org/10.1145/882262.882293.

Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. In *ACM SIGGRAPH 2004 Papers*, pages 609–612. 2004.

Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Sparse surface reconstruction with adaptive partition of unity and radial basis functions. *Graphical Models*, 68(1):15–24, 2006. ISSN 1524-0703. doi: https://doi.org/10.1016/j.gmod.2005.08.001. URL https://www.sciencedirect.com/science/article/pii/S1524070305000548. Special Issue on SMI 2004.

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.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Songyou Peng, Michael Niemeyer, Lars M. Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. *CoRR*, abs/2003.04618, 2020. URL https://arxiv.org/abs/2003.04618.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. *arXiv preprint arXiv:2101.10994*, 2021.

Gabriel Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings of IEEE International Conference on Computer Vision*, pages 902–907. IEEE, 1995.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Gregory J Ward. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 265–272, 1992.

Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete laplace operators: no free lunch. In *Symposium on Geometry processing*, pages 33–37. Aire-la-Ville, Switzerland, 2007.

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. *Advances in Neural Information Processing Systems*, 34:22483–22497, 2021.

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.