Neural Implicit Surface Evolution

Tiago Novello, IMPA
Vinicius da Silva, PUC-Rio
Guilherme Schardong, U Coimbra
Luiz Schirmer, Unisinos
Helio Lopes, PUC-Rio
Luiz Velho, IMPA

Abstract

This work investigates the use of smooth neural networks for modeling dynamic variations of implicit surfaces under the ***level set equation*** *(LSE). For this, it extends the representation of neural implicit surfaces to the space-time $\mathbb{R}^3\times\mathbb{R}$, which opens up mechanisms for* ***continuous*** *geometric transformations. Examples include evolving an initial surface towards general vector fields, smoothing and sharpening using the mean curvature equation, and interpolations of initial conditions.*

The network training considers two constraints. A data term is responsible for fitting the initial condition to the corresponding time instant, usually $\mathbb{R}^3\times\{0\}$. Then, a LSE term forces the network to approximate the underlying geometric evolution given by the LSE, ***without any supervision****. The network can also be* ***initialized based on previously trained initial conditions*** *resulting in faster convergence when compared with the standard approach.*

1. Introduction

A *neural implicit function* $g:\mathbb{R}^3\to\mathbb{R}$ is a smooth neural network that represents an implicit function. Since g is smooth, objects from the *differential geometry* of its *regular level sets* can be used in closed form [35, 26].

This work investigates the extension of the domain of neural implicit functions to the *space-time* $\mathbb{R}^3\times\mathbb{R}$, encoding the evolution of the function g as a higher-dimensional function $f:\mathbb{R}^3\times\mathbb{R}\to\mathbb{R}$. The resulting animation is governed by a PDE, the *level set equation* (LSE) $\frac{\partial f}{\partial t} = v\,\|\nabla f\|$, which encodes the propagation of the level sets S_t of $f(\cdot,t)$ towards their normals with speed v. The choice of the function v depends on the underlying geometric model. LSE is an important tool for *geometry processing* applications.

We propose to use a *neural network* $f:\mathbb{R}^3\times\mathbb{R}\to\mathbb{R}$ to represent the above level set function. For this, we train f to learn the evolution S_t of an initial surface S under a given LSE. Accordingly, we add the constraint $f=g$ on $\mathbb{R}^3\times\{0\}$, to the LSE. If g is the *signed distance function* (SDF) of S, a solution f of this problem encodes an animation of S. Our method is the first neural approach in geometry processing, that does not consider numerical approximations of the LSE solution during sampling and does not discretize the LSE in the loss function.

Our strategy has two steps to train f. First, a sample $\{p_i, g(p_i)\}$ enables us to train the initial condition $f=g$ on $\mathbb{R}^3\times\{0\}$. Second, an LSE constraint is used to fit f into a solution of the LSE in $\mathbb{R}^3\times\mathbb{R}$. This term does not need any supervision, i.e. it does not consider samples of f. The constraint *only* uses samples in the form (p_i, t_i). The only requirement of our method is that f must be smooth. Given this, our main contributions can be summarized as follows:

- Extension of neural implicit surfaces to space-time without the use of numerical/discrete approximations;
- Development of a neural framework to train solutions of an LSE using only its analytical expression (Sec 4);
- The method is flexible enough to be used in a variety of applications, such as surface motion by vector fields, smoothing, sharpening, and interpolation (Sec 6);
- We propose a novel network initialization based on previously trained initial conditions (Sec 4.4).

2. Related Works

Many problems, such as shape correspondence, topology changes, and animation of deformable objects, can be posed using implicit surfaces [17, 32, 3, 4, 14, 13]. Studying their shape properties through differential geometry leads to a framework for intrinsic operations. An example is smoothing a surface using the *mean curvature equation*, an important PDE in geometry processing [6, 5, 15, 13]. Problems in this topic rely on computing derivativess – a hard task when dealing with meshes [7, 11, 12]. A practical neural implicit approach would allow us to compute such objects in closed form, and it is the objective of this work.

Several works showed that modeling surfaces as level sets of neural networks result in a compact representation [29, 25, 18, 31, 26, 10]. Most of them fit a network into data.

In the SDF case, a regularizer term forces the network to satisfy the *Eikonal equation*. The robustness of those approaches is our motivation to study the evolution of neural implicit surfaces using the level set equation [28, 27], a PDE widely used in geometry processing [13, 2, 21, 30, 23, 34].

NFGP [35] and NIE [24] are recent neural approaches to evolve implicit surfaces. Both solve the problem in discrete time steps, differently from our continuous approach. They use a network g_ϕ to fit an initial function g. Then, ϕ is updated at each time step creating a sequence of networks $g_{\phi_i}: \mathbb{R}^3 \to \mathbb{R}$. This is similar to the Euler and Runge-Kutta methods but, instead of fitting the solution to a grid, they use a network. In contrast, our method does not discretize time, learning the solution in a continuous interval using a single network with a domain in $\mathbb{R}^3 \times \mathbb{R}$.

NIE is the only approach that evolves g_ϕ using the mean curvature equation $\frac{\partial f}{\partial t} = \|\nabla f\| \kappa$. To update ϕ, it uses a finite difference scheme to approximate the discrete solutions. They compute the mean curvature κ using a discrete Laplacian, which is problematic since it depends on approximating the level sets by meshes using marching cubes. Also, this operator does not preserve the Laplacian natural properties – the *no free lunch scenario* [33]. In contrast, our approach uses the network high-order derivatives to evaluate the LSE analytically. For example, we compute the curvature using the *divergence* of $\frac{\nabla f}{\|\nabla f\|}$. Moreover, NIE cannot consider multiple initial conditions as our approach does.

Recently, there has been a growing interest by the *physical simulation* community in solving PDEs using neural networks. *Physically informed neural networks* (PINNs) are established approaches in this context. Unlike our proposal, this method [16] relies on measurements of the PDE solution at intermediate times. PINNs are extensively evaluated in surveys [20, 8]. Karniadakis et al. [20] reviews *inverse problems*, which try to infer the PDE parameters based on supervised data. This context differs from ours since we do not address the inverse problem, they also do not consider implicit surface evolution using the LSE, and we do not rely on supervised data of the evolution.

Cuomo et al. [8] surveys a broader range of problems based on the PDE type. We focus on the evolution of implicit surfaces using the LSE, which is a geometric *time-dependent PDE* not explored in their review [8, Sec 3.2.2]. Thus, to the best of our knowledge, there is no PINN-based approach to solve the LSE for implicit surface evolution.

Our proposal seeks to bridge this gap by leveraging the representation capacity of (sinusoidal coord-based) networks to solve the above geometric problem without any measurements of the PDE solution at intermediate times; i.e., we need **supervision only on the initial surfaces**. This can potentially enable new applications in computer graphics, computer-aided design, and computational geometry.

Regarding the interpolation problem, Liu et al. [22] propose the *Lipschitz MLP*, which regularizes a neural network by penalizing the upper bound of its Lipschitz constant. We use a specific PDE to interpolate SDFs, resulting in smoother and more natural transitions between shapes (see Sec 6.3). Moreover, our method manages to use smaller architectures by considering sinusoidal MLPs.

3. Background and conceptualization

3.1. Implicit surfaces

The level set $g^{-1}(0)$ of a smooth function $g: \mathbb{R}^3 \to \mathbb{R}$ is a (regular) surface if $\nabla g \neq 0$ in $g^{-1}(0)$. Conversely, given a surface S, there is a function g having it as its zero-level set. Thus, g may be reconstructed from a sample of S. For this, we parametrize g using a neural network. SIREN [31] and IGR [18] are examples of such networks.

To compute the parameters of g such that $g^{-1}(0) \approx S$, it is common to consider the *Eikonal* problem:

$$\|\nabla g\| = 1 \text{ subject to } g = 0 \text{ on } S. \tag{1}$$

Which asks for g to be the SDF of a set containing S. We can derive from Eq (1) that $\langle \nabla g, N \rangle = 1$ on S, which implies that ∇g must be aligned with the normals of S.

We are interested in using neural networks to evolve S. The level sets of g could be used, but they do not allow intersections between surfaces at different instants. To avoid this, we extend the domain of the implicit function to $\mathbb{R}^3 \times \mathbb{R}$, where the parameter $t \in \mathbb{R}$ controls the animation.

3.2. Evolving implicit surfaces

We use a function $f : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ to define the above extension. For evolving the level sets of g we require $f = g$ in $\mathbb{R}^3 \times \{0\}$ and the resulting *evolution* is given by

$$S_t = f_t^{-1}(0) = \{p \in \mathbb{R}^3 | \, f(p,t) = 0\} \tag{2}$$

We are assuming that f is negative in the interior of S_t and its *normal* vectors are given by $N = \frac{\nabla f}{\|\nabla f\|}$. Moreover, ∇f denotes the *gradient* of f with respect to the space $\mathbb{R}^3$.

The time t allows continuous navigation in S_t and represents a *transformation* $S_0 \to S_t$ of the initial surface S_0. Moreover, for $t_0, t_1 \in \mathbb{R}$, the function f provides a *smooth deformation* between S_{t_0} and S_{t_1}. These surfaces can contain singularities as their topologies may change over time. Hart [19] has studied this phenomenon using Morse theory.

Such an Eulerian approach is usually explored in geometry processing by storing f on a 4D grid [13]. Here, we take an analytical formulation by parametrizing f by a (coord-based) network. Such an approach has several advantages. First, *automatic differentiation* provides us the analytical *derivatives* of f, which may be used at the loss function.

We can also compute the normals and curvature measures of S_t in closed form [26]. Additionally, neural networks are compact representations for implicit functions, guaranteed by the *universal approximation theorem* [9]. Storing f with precision in a 4D grid could be an unfeasible task.

3.3. Level set equation

Encoding the surface evolution S_t by the time-dependent function f results in a PDE – the *level set equation* (LSE). We describe it below.

Let $g : \mathbb{R}^3 \to \mathbb{R}$ be the SDF of the initial surface S. A function $f : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ encodes the evolution S_t if $f = g$ on $\mathbb{R}^3 \times \{0\}$, and for each point p, the function f is constant along its path $\alpha(t)$, i.e. $f\big(\alpha(t), t\big) = c$ iff $g(p) = c$. Thus, deriving the function $f\big(\alpha(t), t\big)$ we obtain

$$\frac{\partial f}{\partial t}\big(\alpha(t), t\big) + \big\langle \nabla f\big(\alpha(t), t\big), \alpha'(t) \big\rangle = 0. \tag{3}$$

The derivative $\alpha'(t)$ is a vector field along the path $\alpha(t)$.

As Eq (3) holds for each point p, we can drop its path α and use only α', which can be seen as a time-dependent vector field $V(p, t) = \alpha'(t)$. Then, the function f encoding the animation S_t is a solution of the LSE:

$$\begin{cases} \dfrac{\partial f}{\partial t} + \big\langle \nabla f, V \big\rangle = 0 & \text{in } \mathbb{R}^3 \times (a, b), \\ f = g & \text{on } \mathbb{R}^3 \times \{t = 0\}. \end{cases} \tag{4}$$

The time interval (a, b) contains $t = 0$ and controls the evolution S_t. A solution f of Eq (4) implicitly encodes the integration of V. Thus, defining a family of vector fields is a way to animate a given surface using the LSE.

Observe that in the same sense that we considered a neural implicit function as a solution of Eq (1), we assume that the function f is a solution of $\frac{\partial f}{\partial t} + \big\langle \nabla f, V \big\rangle = 0$ subject to $f = g$ on $\mathbb{R}^3 \times \{0\}$. Sec 5 presents examples of LSEs.

4. Method

We propose representing a surface evolution by a (coord-based) neural network $f_\theta : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$. Given an LSE with initial conditions, this section defines a machine learning framework, consisting of a loss functional, sampling strategies, and a network initialization, to train f_θ to approximate a solution to the LSE problem.

Let $g_i : \mathbb{R}^3 \to \mathbb{R}$ be the SDFs of n surfaces S_i. We train f_θ by forcing it to approximate a solution a *neural* LSE:

$$\begin{cases} \mathcal{F} := \dfrac{\partial f_\theta}{\partial t} + \big\langle \nabla f_\theta, V \big\rangle = 0 & \text{in } \mathbb{R}^3 \times (a, b), \\ f_\theta = g_i & \text{on } \mathbb{R}^3 \times \{t_i\}. \end{cases} \tag{5}$$

We employed the notation $\mathcal{F}$ to represent the LSE for brevity. The untrained network f_θ must encode the movement ruled by the vector field V. (a, b) can be used to control the resulting neural animation S_t of S.

4.1. Loss functional

We use Eq (5) to define a loss function to train f_θ.

$$\mathcal{L}(\theta) = \underbrace{\int_{\mathbb{R}^3 \times (a,b)} \big|\mathcal{F}\big| dp dt}_{\mathcal{L}_{\text{LSE}}(\theta)} + \underbrace{\sum_{i=1}^{n} \int_{\mathbb{R}^3 \times \{t_i\}} \big|f_\theta - g_i\big| dp}_{\mathcal{L}_{\text{data}}(\theta)}. \tag{6}$$

The LSE *constraint* $\mathcal{L}_{\text{LSE}}$ forces the network f_θ to satisfy $\mathcal{F} = 0$ and works like a regularization of f_θ requiring it to follow the underlying phenomenon. The *data constraint* $\mathcal{L}_{\text{data}}$ asks for f_θ to satisfies $f_\theta = g_i$ on $\mathbb{R}^3 \times \{t_i\}$.

4.2. Sampling

To approximate a solution f_θ of Eq (5), we seek a minimum θ of the loss function $\mathcal{L}$ using the *stochastic gradient descent*. For this, we enforce $\mathcal{L} = \mathcal{L}_{\text{LSE}} + \mathcal{L}_{\text{data}}$ with a sampling in $\mathbb{R}^3 \times (a, b)$ and another in $\bigcup_{i=1}^{n} \mathbb{R}^3 \times \{t_i\}$.

Sampling space-time points
During training, we sample minibatches of $l_1 \in \mathbb{N}$ *space-time points* $(p_j, t_j) \in \mathbb{R}^3 \times (a, b)$ randomly. Then, $\mathcal{L}_{\text{LSE}}$ is enforced in (p_j, t_j), yielding the approximation:

$$\widetilde{\mathcal{L}_{\text{LSE}}}(\theta) = \frac{1}{l_1} \sum_{j=1}^{l_1} |\mathcal{F}(p_j, t_j)|$$

Monte Carlo integration ensures that the approximation of $\mathcal{L}_{\text{LSE}}$ gets better as the size l_1 of the sampling grows. Note that $\mathcal{L}_{\text{LSE}}$ does not need any data supervision.

Sampling initial conditions
The data constraint $\mathcal{L}_{\text{data}}$ forces f_θ to fit the input dataset, which we consider to be the SDFs g_i of n surfaces S_i.

We use Eq (1) to define $\mathcal{L}_{\text{data}} = \sum \mathcal{L}_i$, with $\mathcal{L}_i$ managing the restriction $g_i = f_\theta$ on $\mathbb{R}^3 \times \{t_i\}$.

$$\mathcal{L}_i = \underbrace{\int_{\mathbb{R}^3 \times t_i} \Big|1 - \big|\nabla f_\theta\big|\Big| dp}_{\mathcal{L}_{\text{Eikonal}}} + \underbrace{\int_{\mathbb{R}^3 \times t_i} \big|f_\theta - g_i\big| dp}_{\mathcal{L}_{\text{Dirichlet}}} + \underbrace{\int_{S_i} \big|1 - \big\langle \nabla f_\theta, N_i \big\rangle\big| dS_i}_{\mathcal{L}_{\text{Neumann}}}.$$

Where $\mathcal{L}_{\text{Dirichlet}}$ asks for f_θ to fit g_i at time t_i, $\mathcal{L}_{\text{Neumann}}$ requires the alignment between ∇f_θ and the normals of S_i, and $\mathcal{L}_{\text{Eikonal}}$ is the Eikonal regularization. During training, these constraints are discretized, as in the PDE constraint case. Then, we sample *minibatches* with l_2 *on-surface* points ($g_i = 0$) and l_3 *off-surface* points ($g_i \neq 0$).

In practice, we use two kinds of initial conditions. First, neural networks g_i fit the SDFs of S_i, resulting in faster training since we have, for each point p_i, the values $g_i(p_i)$ and ∇g_i given by the evaluation of g_i and its derivative at p_i. Second, we consider point clouds $\{p_j, N_j\}_i$ sampled from S_i, where we have to approximate the SDF of S_i [26].

During training, we sample minibatches of size $l_1+l_2+l_3$ to feed $\mathcal{L}$. l_i are the numbers of space-time, on-surface, and off-surface points. The experiments shown good results using that l_1, l_2, l_3 have $50\%, 25\%, 25\%$ of the minibatch size. The supplementary materials give experiments varying l_i.

4.3. Neural network architecture

We consider the neural network to be a *sinusoidal MLP* $f_\theta(p) = W_{d+1} \circ f_d \circ \cdots \circ f_1(p) + b_{d+1}$, with d hidden layers $f_i(p_i) = \sin(W_i p_i + b_i)$, where $W_i \in \mathbb{R}^{N_{i+1} \times N_i}$ are the weight matrices, and $b_i \in \mathbb{R}^{N_{i+1}}$ are the biases. The sine is applied at each coordinate of $W_i p_i + b_i$. θ consists of the union of the coefficients of W_i and b_i. The integer d is the *depth* of f_θ and the dimensions N_i are the layers *widths*.

The network f_θ is smooth, and we can compute its derivatives using automatic differentiation. Therefore, we train f_θ using the loss function $\mathcal{L}$ in Eq (6).

4.4. Network initialization

We introduce a novel initialization of $f_\theta : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ based on a previously trained network $g_\phi : \mathbb{R}^3 \to \mathbb{R}$. This initialization of θ using ϕ results in faster training compared with the standard definitions [31] (see Sec 6.4).

Assume that the training of f_θ is subject to $f_\theta = g_\phi$ on $\mathbb{R}^3 \times \{0\}$. Then, we define θ in terms of ϕ such that $f_\theta(p,t) = g_\phi(p)$ for all t, i.e. f_θ will be constant and equal to g_ϕ along the time. This would allow us to train f closer to a solution of the underlying LSE model.

For this, we suppose that f_θ is wider than g_ϕ and that their depths are equal to d. Specifically, let $B_i \in \mathbb{R}^{N_{i+1} \times N_i}$, $b_i \in \mathbb{R}^{N_{i+1}}$ be the *trained* weight matrices and biases of g_ϕ, and $A_i \in \mathbb{R}^{M_{i+1} \times M_i}$, $a_i \in \mathbb{R}^{M_{i+1}}$ be the *untrained* weight matrices and biases of f_θ. Since f_θ is wider than g_ϕ, i.e. $N_i \leq M_i$, we can define A_i, b_i using

$$A_1 = \begin{pmatrix} B_1 & 0 \\ F_p & F_t \end{pmatrix}, \quad A_i = \begin{pmatrix} B_i & 0 \\ 0 & 0 \end{pmatrix} \text{ for } i = 2, \dots, d,$$
$$A_{d+1} = \begin{pmatrix} B_{d+1} & L \end{pmatrix}, \quad a_i = \begin{pmatrix} b_i \\ 0 \end{pmatrix} \text{ for } i = 1, \dots, d+1.$$

Thus, $f_\theta(p,t) = g(p)$ for all t, see supp. material for the details. F_p, F_t project the input (p,t) in the dictionary $\sin(F_p p + F_t t)$, and are initialized using the standard approach. Note that these sines are not used in the first training step, but as it advances, the new hidden weights combine them improving the training (see Sec 6.4).

5. Examples

Here, we present examples of neural implicit evolution using LSE. Sec 5.1 shows examples using time-independent vector fields. Sec 5.2 considers the mean curvature equation, which is intrinsically related to the surface and results in smoothing/sharpening applications for implicit neural surfaces. Sec 5.3 investigates interpolations between implicit neural surfaces using an LSE.

Recently, these problems have been addressed by different neural methods [24, 35, 22]. Comparisons are made in Sec 6. Hereafter, we give each application's conceptualization and corresponding loss function.

5.1. Time-independent vector fields

Moving a surface S towards a vector field $V : \mathbb{R}^3 \to \mathbb{R}^3$ results in a simple LSE. Specifically, let g be the SDF of S. Since V does not change over time, it may be defined and customized beforehand. For example, sources, sinks, saddles, and constant vectors may be used to generate vector fields based on specific applications.

We train a neural network $f_\theta : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ to implicitly encode the evolution of S by V using the resulting LSE:

$$\begin{cases} \dfrac{\partial f_\theta}{\partial t} - v \left\| \nabla f_\theta \right\| = 0 & \text{in } \mathbb{R}^3 \times (a,b), \\ f_\theta = g & \text{on } \mathbb{R}^3 \times \{0\}. \end{cases} \tag{7}$$

Here v denotes the size of the normal component of V, i.e. $v(p,t) = \langle V(p), N_t(p) \rangle$. The minus in Eq (7) is because we need the inverse of the resulting flow to compose with g.

Sec 6.1 gives two experiments using time-independent vector fields as a proof of concept. Other examples are presented in the video supplementary material.

5.2. The mean curvature equation

The *mean curvature equation* evolves the level sets with velocity given by the negative of their mean curvature, resulting in a smoothing along the time [6, 1].

Let $V(p,t) = -\kappa(p,t) N(p,t)$ be the *mean curvature vector*, where N is the normal field of the level sets and $\kappa = \operatorname{div} N$ is the *mean curvature*; div is the *divergence* operator. Replacing V in Eq (5) results in:

$$\begin{cases} \dfrac{\partial f}{\partial t} - \alpha \left\| \nabla f \right\| \kappa_\theta = 0 & \text{in } \mathbb{R}^3 \times (a,b), \\ f = g & \text{on } \mathbb{R}^3 \times \{t = 0\}. \end{cases} \tag{8}$$

Intuitively, the zero-level set moves toward the mean curvature vector $-\kappa N$, contracting regions with positive curvature and expanding regions with negative curvature. Thus, such procedure *smooths* (*sharpens*) the surface if $t > 0$ ($t < 0$). α controls the level set evolution, which has relations with *minimal surfaces* (Sec 1 of supp. material).

Analogous to Eq (5), we define $\mathcal{L}_{\text{LSE}} + \mathcal{L}_{\text{data}}$ to fit a solution of Eq (8) using $\mathcal{F} := \frac{\partial f}{\partial t} - \alpha \left\| \nabla f \right\| \kappa_\theta$. Sec 6.2 presents smoothing and sharpening using this technique.

5.3. Interpolation between implicit surfaces

Let g_i be the SDFs of two surfaces S_i. We present a LSE approach to interpolate g_i. A vector field V for Eq (5), such that its solution interpolates between S_i, has the form:

$$V(p,t) = -\big(g_2(p) - f(p,t)\big) \frac{\nabla f(t,p)}{\left\| \nabla f(t,p) \right\|}, \tag{9}$$

with $f(p,0) = g_1(p)$. Note that the evolution towards V forces each c-level set of g_1 to match the c-level set of g_2. The resulting LSE is given by substituting Eq (9) in Eq (5)

$$\begin{cases} \dfrac{\partial f}{\partial t} - \|\nabla f\|\,(g_2 - f) = 0 & \text{in } \mathbb{R}^3 \times \mathbb{R}, \\ f = g_i & \text{on } \mathbb{R}^3 \times \{t_i\}. \end{cases} \tag{10}$$

A solution f of Eq (10) will locally inflate S_1 if inside S_2, and deflate it if outside so that S_1 will always try to fit into S_2 [13]. Again, we define a loss function $\mathcal{L}_{\text{LSE}} + \mathcal{L}_{\text{data}}$ to fit a solution of Eq (10) using $\mathcal{F} := \frac{\partial f}{\partial t} - \|\nabla f\|\,(g_2 - f)$.

Theoretically, we could use the mean curvature equation to minimize deformations along the interpolation. This LSE has the property of minimizing area distortions of the resulting evolution; see Sec 1 of the supplementary material.

6. Experiments

Here, we present the experiments of the examples given in Sec 5. See the supplementary material for an ablation study of the training, sampling, and initialization.

6.1. Deformation driven by vector fields

We use the the definitions in Sec 5.1 to train an animation based on vector fields spatially related to the initial surface.

First, consider the twist $V(x,y,z) = y(-z,0,x)$ of $\mathbb{R}^3$ along the y-axis. Substituting it in Eq (5) results in a level set equation, which we use to derive a loss function.

Let g be the SDF of the Armadillo and f_θ be a network with 2 hidden layers $f_i : \mathbb{R}^{256} \to \mathbb{R}^{256}$. We trained f_θ during 48000 epochs using minibatches of 25000 on-surface points ($g = 0$), 25000 off-surface points ($g \neq 0$), and 8000 points in $\mathbb{R}^3 \times [-1,1]$. Fig 1 shows 3 reconstructions of the zero-level sets of f_θ at times $t_i = 0, 0.25, 0.5$.

Figure 1. Evolving the level sets of the Armadillo's SDF using the vector field that represents a twist of $\mathbb{R}^3$ along the y-axis.

Although we do not provide data in $\mathbb{R}^3 \times \{t \neq 0\}$, the solution is well approximated (see Fig 1). The vertical axis is the y-axis, and the origin of $\mathbb{R}^3$ is at the ground.

For the next experiment, let g be the SDF of the Spot (Fig 2, center). Define $V = V_1 - V_2$ as the sum of a *source* V_1 and a *sink* $-V_2$, with $V_i(p) = e^{-\frac{|p-p_i|^2}{0.18}}(p - p_i)$. The points p_1, and p_2 are the centers of Spot's body and head. Again, we use V to derive a loss function to train f_θ. We parameterize f_θ with one hidden layer $f_i : \mathbb{R}^{128} \to \mathbb{R}^{128}$ and train it for 46000 epochs. As expected, it reduces the Spot's head while it increases the body size, see Fig 2.

Figure 2. Evolving the zero-level sets of a network according to a vector field with a source and a sink. We set the SDF of the Spot as the initial condition at $t = 0$ (middle). The sink/source are inside the head/body of the Spot.

Table 1 shows that the above networks are close to satisfying the LSE problems. We compare $f_\theta(\cdot, 0)$ with the initial condition g and measure how close f_θ is from satisfying $\mathcal{F} = 0$. For this we use the following measures: 1) The absolute difference $|f_\theta(\cdot,0) - g|$ in $\mathbb{R}^3 \times \{0\}$; 2) The evaluation of f_θ in $|\mathcal{F}|$ in $\mathbb{R}^3 \times \mathbb{R}$. We use a sample of 1000 points in $\mathbb{R}^3 \times \{0\}$ and $\mathbb{R}^3 \times \mathbb{R}$, not included in the training process, to evaluate the mean/maximum of each measure.

Vector field	on-surface constraint		off-surface constraint		PDE constraint	
	mean	max	mean	max	mean	max
twist	0.0008	0.003	0.002	0.028	1e-5	0.0004
source-sink	0.0009	0.005	0.001	0.015	2.7e-6	0.0005

Table 1. Comparisons between the ground truth initial conditions at $t = 0$ and the measures of how close the trained networks are to satisfy the underlying LSEs.

6.2. The mean curvature equation

The next set of experiments seek to solve the mean curvature equation. First, we consider simple initial conditions such as the cube and the dumbbell. Then we use the intrinsic properties of this LSE to provide smoothing/sharpening of detailed surfaces.

6.2.1 Simple initial conditions for validation

Let g be the SDF of the cube, and $f_\theta : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ be a network with 3 hidden layers $f_i : \mathbb{R}^{256} \to \mathbb{R}^{256}$. We set $\alpha = 0.1$ and optimized f_θ for 8000 epochs using the loss function resulting from Eq (8). We used an oriented point cloud of size 40000, sampled from the cube. During training, we consider minibatches of 5000 on-surface points, 5000 off-surface points, and 10000 in $\mathbb{R}^3 \times [-1,1]$.

Fig 3 shows the level sets of f_θ at $t = \frac{i}{5}$, $i = 0, \ldots, 5$. As expected, regions with positive mean curvature, such as the cube corners, contract. This LSE evolves the surface toward the normals times the negative of the mean curvature.

Therefore, the cube will at some instant of time collapse to a point, but right before it will be very close to a sphere [6].

Figure 3. Mean curvature equation of cube surface.

The *dumbbell* is a classical example. Let g be its SDF and f_θ be a network with 2 layers $f_i : \mathbb{R}^{256} \to \mathbb{R}^{256}$. We set $\alpha = 0.05$ and sample a point cloud of size 80000 from the dumbbell. The training took 2800 epochs using mini-batches of 5000 on-surface points, 5000 off-surface points, and 10000 points in $\mathbb{R}^3 \times [-1, 1]$.

Fig 4 shows the level sets of f_θ at times $t_i = \frac{i}{10}$ for $i = 0, \ldots, 7$. As expected, since the neck region has higher mean curvature, it pinches off first creating two connected components. Later, each component collapses to a point, becoming small spheres right before that. The resulting flow has critical points in different instances of time.

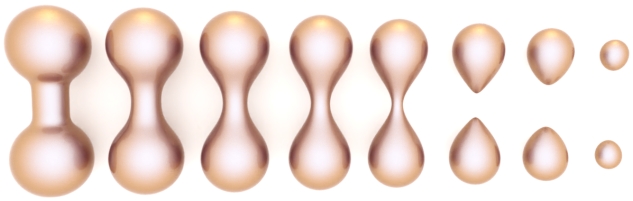

Figure 4. Mean curvature equation of Dumbbell surface.

6.2.2 Smoothing and sharpening

The mean curvature equation evolves the level sets by contracting (expanding) regions with positive (negative) mean curvature. As a consequence, its solution smooths (sharpens) the level sets when $t > 0$ ($t < 0$).

Let g be the SDF of the Armadillo, and f_θ be a network with 2 hidden layers $f_i : \mathbb{R}^{256} \to \mathbb{R}^{256}$. We set $\alpha = 0.001$ in Eq (8). The network f_θ was trained during 33000 epochs using an oriented point cloud of size 80000. During training, we used minibatches of 5000 on-surface points, 5000 off-surface points, and 10000 in $\mathbb{R}^3 \times [-1, 1]$.

Fig 5 presents three reconstructions of the zero-level sets of f_θ at times $t = 0, 0.1, 0.2$. As expected the surface of the Armadillo was properly reconstructed at $t = 0$ and as time progressed it became smoother. Regions with positive mean curvature, such as the fingers, contracted.

For the sharpening we reconstruct the zero-level sets at $t = 0, -0.1, -0.2$ (see Fig 6). As expected, regions with positive curvature have expanded, resulting in an enhancement of the surface geometrical features.

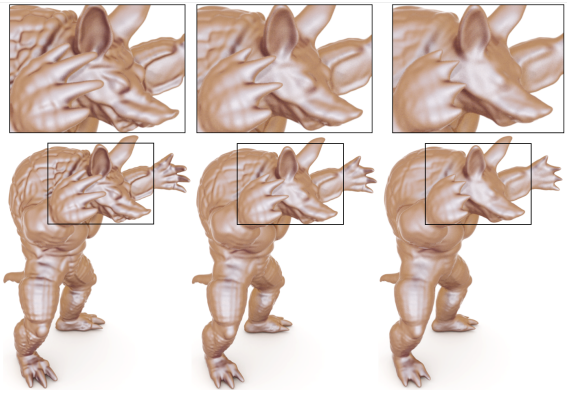

Figure 5. Armadillo smoothing using the mean curvature equation.

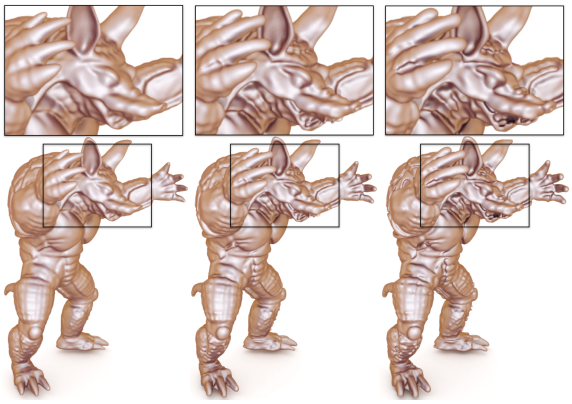

Figure 6. Using the mean curvature equation to enhance the geometrical details of the Armadillo surface.

Numerical evaluation

Table 2 shows that the above networks are close to satisfying Eq (8). We compare the networks at $t = 0$ with the initial surfaces. We also measure how close the networks are to satisfying the LSE. We used a sample of 1000 points in $\mathbb{R}^3 \times \{0\}$ and $\mathbb{R}^3 \times \mathbb{R}$, not included in the training process, to evaluate the mean and maximum values of each measure.

Model	on-surface constraint		off-surface constraint		PDE constraint	
	mean	max	mean	max	mean	max
Cube	0.0006	0.007	0.0013	0.024	0.0015	0.009
Dumbbell	0.0003	0.002	0.0010	0.013	0.0009	0.017
Armadillo	0.0008	0.004	0.0022	0.016	0.0019	0.013

Table 2. Quantitative evaluation of our method in the problem of approximating solutions of the mean curvature equation.

Comparisons

We compare our technique with NFGP [35] and NIE [24] for smoothing and sharpening of neural implicit surfaces.

NFGP evolves a network $g_\theta : \mathbb{R}^3 \to \mathbb{R}$ such that the level set of the resulting network g_ϕ smooths/sharpens $g_\theta^{-1}(0)$.

The training optimizes, $(\kappa_\phi - \beta\kappa_\theta)^2$, the difference between the mean curvatures of the level sets of g_ϕ and g_θ. Then, using $\beta < 1$ ($\beta > 1$), it would force a smoothing (sharpening) of the initial surface. However, NFGP trains a network g_ϕ for each β, thus, it cannot represent a continuous evolution along time. In contrast, our framework directly evolves over time using a single network. Although the NFGP approach does not use the mean curvature equation model, we can still perform a qualitative analysis as a means of comparison since a numerical comparison is not feasible. Fig 7 shows this comparison for sharpening. The artifacts in the Armadillo's ears are probably due to the inconsistencies in the loss function of NFGP which asks for $(g_\phi - g_\theta)^2$ and $(\kappa_\phi - \beta\kappa_\theta)^2$, thus the level sets would try to evolve but $(g_\phi - g_\theta)^2$ forces it to be constant.

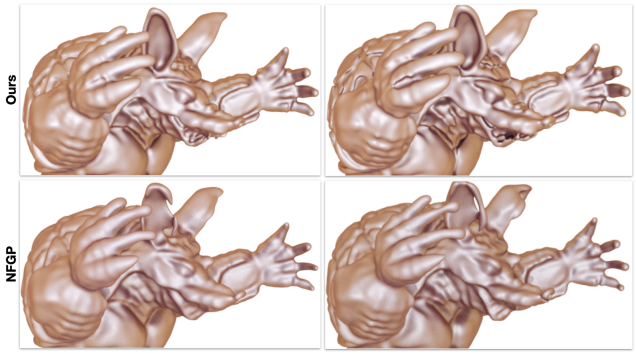

Figure 7. Sharpening comparison of our approach with NFGP [35]. We use the same Armadillos in Fig 6, and $\beta = 2, 2.5$ for the NFGP. The experiments used the same initial condition. Notice that NFGP may produce artifacts while sharpening, as can be seen in the Armadillo's ears.

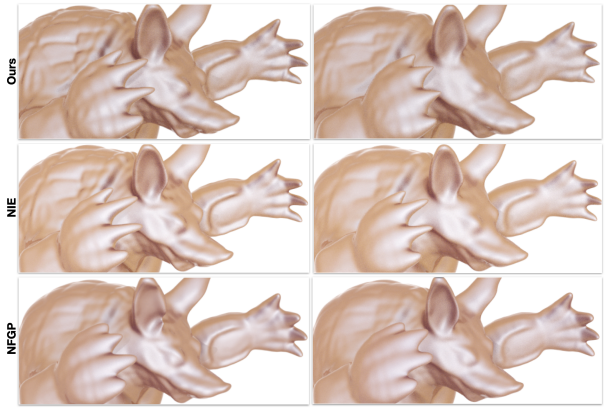

Figure 8. Smoothing comparison. Line 1 repeats Fig 5. Line 2 presents steps 4 and 7 of smoothing using NIE [24]. Line 3 shows the results using NFGP [35] with $\beta = 0.95, 0.8$. All experiments used the same initial condition. As expected, our approach is comparable with NIE because they are based on the mean curvature equation. While the overall result of NFGP is smoothing, it may produce artifacts, as can be noticed in the fingers.

We also compare with NIE [24]. Given a *time step* Δt, it fits the solution of Eq (8) at times $t_i = i\Delta t$ using the approximation $f_{t_{i+1}} = f_{t_i} - \Delta t\alpha\langle\nabla f_{t_i}, \kappa_\theta N\rangle$. Thus, for a network $g_{\phi_i} \approx f_{t_i}$, NIE trains the next state $g_{\phi_{i+1}}$ by minimizing $(g_{\phi_{i+1}} - f_{t_{i+1}})^2$. Moreover, a *discrete Laplacian* estimates $\kappa_\theta N$ at the vertices of a mesh approximating $g_{\phi_i}^{-1}(0)$. The resulting networks g_{ϕ_i} have the domain in $\mathbb{R}^3$ while we use a single network with a domain in $\mathbb{R}^3 \times \mathbb{R}$.

Fig 8 presents two smoothings of the Armadillo using our method, NIE, and NFGP. Line 1 repeats the results of Fig 5. Line 2 shows results for NIE, considering 4 and 7 steps with $\Delta t = 1$. Line 3 presents the results using NFGP to train a network with $\beta = 0.95, 0.8$ defined empirically. To present a fair comparison, we considered the initial network to be $f_\theta(\cdot, 0)$; f_θ is the network from our experiment. We used the procedure in Sec 3 of the supplementary material to extract $f_\theta(\cdot, 0)$ from f_θ.

6.3. Interpolation between implicit surfaces

Suppose g_i are the SDFs of the Bob and Spot (Fig 9, left-right) and that f_θ has 1 hidden layer $f_i : \mathbb{R}^{128} \to \mathbb{R}^{128}$. We train f_θ using $\mathcal{L}_{\text{LSE}} + \mathcal{L}_{\text{data}}$, as described in Sec 5.3. Line 1 in Fig 9 shows the reconstructions of the level sets.

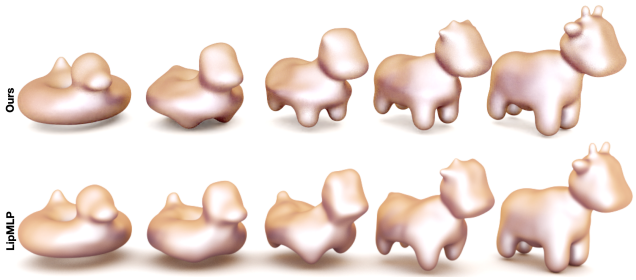

Figure 9. Interpolation between Bob and Spot. Line 1 shows the result using our method, and line 2 the Lipschitz MLP. Notice that our method results in smoother transitions between the images.

We compare our method with *Lipschitz MLP* [22] which considers the $tanh$ activation. We use a Lipschitz MLP with 5 hidden layers of 256 neurons. Each layer is followed by a Lipschitz regularization. The network was trained during 100000 epochs. See the resulting interpolation in Line 2 of Fig 9. Our network is significantly smaller than the Lipschitz MLP but results in natural interpolation. This is due to the high representation capacity of sinusoidal MLPs.

Fig 10 presents the reconstructions of interpolation between the Witch and Falcon (from the Thingi10K dataset [36]). The first line is the result of our method using a network with 2 hidden layers $\mathbb{R}^{128} \to \mathbb{R}^{128}$ trained during 20000 epochs. For the Lipschitz MLP, we have to consider a larger network with 5 hidden layers of 512 neurons and train it for 100000 epochs. Even with the added capacity and training iterations, the Lipschitz MLP cannot approximate the initial conditions properly.

Figure 10. Interpolation between Witch and Falcon. Line 1 (20000 epochs) is the result of our method and Line 2 (100000 epochs) considers the Lipschitz MLP network. Notice that our approach results in a better approximation of the initial conditions, as can be seen in the Witch's hood and sword, and Falcon's beak and talons.

6.4. Initialization based on trained networks

Let $g_\phi : \mathbb{R}^3 \to \mathbb{R}$ be a trained network with 2 hidden layers $g_i : \mathbb{R}^{256} \to \mathbb{R}^{256}$ fitting the SDF of the Bunny. Let $f_\theta : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ be a network with 2 hidden layers $f_i : \mathbb{R}^{256} \to \mathbb{R}^{256}$. We train f_θ to approximate a solution of the mean curvature equation subject to $f_\theta = g_\phi$ on $\mathbb{R}^3 \times \{0\}$.

Here we use the scheme of Sec 4.4 to define θ in terms of ϕ such that $f_\theta(p, t) = g_\phi(p)$. We compare it with the standard initialization of sinusoidal MLPs [31].

Figure 11. Network initialization comparison. Line 1 shows the results of using the initialization is given in [31] and Line 2 considers the proposed initialization. Notice how the new network initialization results in a model that can represent higher frequencies, as shown by the increased surface details in Line 2.

Fig 11 gives qualitative comparisons between the initializations, after training f_θ during 500 epochs. We empirically observed that our approach speeds up learning. For example, Line 1 shows the bunnies at $t = -0.2, 0.0, 0.6$. Line 2 gives the analogous results using our initialization which results in faster convergence. Note the preservation of surface details using the proposed initialization at $t = -0.2, 0.0$ (Line 2), compared to the standard initialization in Line 1. Fig 12 shows the plots of the constraints considering 1500 epochs. Note that, using our scheme, the training starts closer to a minimum of the loss function.

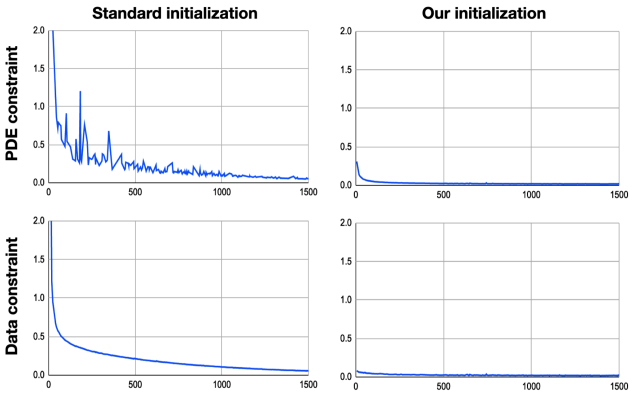

Figure 12. Loss function comparisons. Column 1 shows the plots of the LSE/data constraint using the standard initialization. Column 2 shows the corresponding plots applying our initialization scheme. The horizontal axis represents the number of epochs.

7. Conclusions and Future Work

We introduced a framework to explore the differentiable properties of smooth networks in the problem of evolving level sets of neural SDFs. For this, we extended their domain to space-time which opens up possibilities to control geometric animation and modeling using LSEs.

The method allows evolving neural implicit surfaces under LSEs without the use of additional data, only the initial conditions are used. Note that other methods compute an approximation of the solution using numerical simulations and then fit it into the neural network. However, our framework was capable of learning the animation only considering the LSE constraint. This is powerful because the models are expressed in compact LSEs that are used to define constraints. This approach enables learning the corresponding animations *without any supervision*. Most techniques in geometry processing use differential equations to model various kinds of phenomena which, in general, are written in terms of their energy formulation.

The resulting networks are smooth approximations of LSE solutions. Traditional numerical solutions are discrete, making non-trivial the task of introducing more conditions. This is a quite simple task in our method, however. We believe that the development of such methods in graphics would enable the community to use the robustness of classical continuous theories without the need for discretizations.

References

[1] Giovanni Bellettini. *Lecture notes on mean curvature flow: barriers and singular perturbations*, volume 12. Springer, 2014. 4

[2] Marcelo Bertalmio, Guillermo Sapiro, Li-Tien Cheng, and Stanley Osher. Variational problems and pdes on implicit surfaces. In *Proceedings IEEE Workshop on Variational and Level Set Methods in Computer Vision*, pages 186–193. IEEE, 2001. 2

[3] D.E. Breen and R.T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):173–192, 2001. 1

[4] M. Cani-Gascuel and M. Desbrun. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):39–50, 1997. 1

[5] Ulrich Clarenz, Udo Diewald, and Martin Rumpf. *Anisotropic geometric diffusion in surface processing*. IEEE, 2000. 1

[6] Tobias Holck Colding, William P. Minicozzi II, and Erik Kjær Pedersen. Mean curvature flow. *Bulletin of the American Mathematical Society*, 52(2):297–333, 2015. 1, 4, 6

[7] Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses*, SIGGRAPH '13, New York, NY, USA, 2013. ACM. 1

[8] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022. 2

[9] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 3

[10] Vinícius da Silva, Tiago Novello, Guilherme G. Schardong, Luiz Schirmer, Hélio Lopes, and Luiz Velho. Neural implicit mapping via nested neighborhoods. *arXiv*, abs/2201.09147, 2022. 1

[11] Fernando De Goes, Andrew Butts, and Mathieu Desbrun. Discrete differential operators on polygonal meshes. *ACM Trans. Graph.*, 39(4), July 2020. 1

[12] Fernando de Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose. Subdivision exterior calculus for geometry processing. *ACM Trans. Graph.*, 35(4), July 2016. 1

[13] Mathieu Desbrun and Marie-Paule Cani-Gascuel. Active implicit surface for animation. In *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada*, pages 143–150, June 1998. 1, 2, 5

[14] Mathieu Desbrun and Marie-Paule Gascuel. Animating soft substances with implicit surfaces. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 287–290, New York, NY, USA, 1995. Association for Computing Machinery. 1

[15] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, 1999. 1

[16] Raissi et al. Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations. *arXiv*, 2017. 2

[17] Jonas Gomes, Lucia Darsa, Bruno Costa, and Luiz Velho. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann Publishers, 1998. 1

[18] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020. 1, 2

[19] John C Hart. Morse theory for implicit surface modeling. In *Mathematical Visualization*, pages 257–268. Springer, 1998. 2

[20] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. 2

[21] Ron Kimmel and James A Sethian. Computing geodesic paths on manifolds. *Proceedings of the national academy of Sciences*, 95(15):8431–8435, 1998. 2

[22] Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. Learning smooth neural functions via lipschitz regularization. In *ACM SIGGRAPH 2022 Conference Proceedings*, SIGGRAPH '22. Association for Computing Machinery, 2022. 2, 4, 7

[23] Ravi Malladi, James A Sethian, and Baba C Vemuri. Shape modeling with front propagation: A level set approach. *IEEE transactions on pattern analysis and machine intelligence*, 17(2):158–175, 1995. 2

[24] Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. A level set theory for neural implicit evolution under explicit flows. *arXiv preprint arXiv:2204.07159*, 2022. 2, 4, 6, 7

[25] Mateusz Michalkiewicz, Jhony Kaesemodel Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4742–4751, 2019. 1

[26] Tiago Novello, Guilherme Schardong, Luiz Schirmer, Vinícius da Silva, Hélio Lopes, and Luiz Velho. Exploring differential geometry in neural implicits. *Computers & Graphics*, 108, 2022. 1, 3

[27] Stanley Osher, Ronald Fedkiw, and K Piechor. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.*, 57(3):B15–B15, 2004. 2

[28] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988. 2

[29] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[30] James Albert Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge university press, 1999. 2

[31] 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. 1, 2, 4, 8

[32] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, page 335–342, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 1

[33] 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. 2

[34] Ross T Whitaker. Algorithms for implicit deformable models. In *Proceedings of IEEE International Conference on Computer Vision*, pages 822–827. IEEE, 1995. 2

[35] Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. *Advances in Neural Information Processing Systems*, 34, 2021. 1, 2, 4, 6, 7

[36] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016. 7

Neural Implicit Surface Evolution
– Supplementary Material –

Tiago Novello, IMPA
Vinicius da Silva, PUC-Rio
Guilherme Schardong, U Coimbra
Luiz Schirmer, Unisinos
Helio Lopes, PUC-Rio
Luiz Velho, IMPA

1. Minimal surfaces

The evolution of a surface S governed by the *mean curvature equation* (MCE) leads to a family of surfaces that reduce their area over time. Let $f:\mathbb{R}^3\times\mathbb{R}\to\mathbb{R}$ be a solution of MCE and S_t be its corresponding surface evolution.

$$\begin{cases} \dfrac{\partial f}{\partial t} - \alpha\,\|\nabla f\|\,\kappa = 0 & \text{in } \mathbb{R}^3\times(a,b), \\ f = g & \text{on } \mathbb{R}^3\times\{t=0\}. \end{cases} \tag{1}$$

The area of S_t can be measured using $\text{Area}(S_t) = \int_{S_t} dS_t$, where dS_t is the area form of S_t. It can be proved that the *first variation of area* of the family S_t is given by

$$\frac{d}{dt}\text{Area}(S_t)\Big|_{t=0} = -\int_{S_0} \kappa^2 dS_0, \tag{2}$$

The proof can be found in [1, Sec. 3.5], [2, Cor. 6.2]. Thus, if the mean curvature satisfies $\kappa \neq 0$, the area of S_t initially decreases because its derivatives are negative at $t=0$.

A surface S_{t_0} is *critical* if $\frac{d}{dt}\text{Area}(S_t)\Big|_{t=t_0} = 0$, that is, if κ is constant equal to zero. This surface is called *minimal*. Examples of minimal surfaces include the plane, catenoids, helicoids, Enneper surface, Costa's minimal surfaces, etc.

We can fix a region of the initial surface S in the MCE. If S has a boundary curve, fixing it during the evolution leads to a surface of minimal area. This problem is related to the physical shapes of soap films at equilibrium under the surface tension [4].

2. Network initialization

Proposition 2.1. *Let $g_\phi:\mathbb{R}^3\to\mathbb{R}$ and $f_\theta:\mathbb{R}^3\times\mathbb{R}\to\mathbb{R}$ be networks with depth d. If f_θ is wider than g_ϕ, we can define θ in terms of ϕ such that $f_\theta(p,t) = g_\phi(p)$ for all (p,t).*

Proof. Recall that $g_\phi(p) = B_{d+1}\circ g_d\circ\cdots\circ g_1(p) + b_{d+1}$, where $g_i(p_i) = \sin(B_i p_i + b_i)$ is the i-layer, B_i is a matrix in $\mathbb{R}^{N_{i+1}\times N_i}$, and $b_i\in\mathbb{R}^{N_{i+1}}$ is the i-bias. Analogously, $f_\theta(p,t) = A_{d+1}\circ f_d\circ\cdots\circ f_1(p,t) + a_{d+1}$, with its i-layer $f_i:\mathbb{R}^{M_i}\to\mathbb{R}^{M_{i+1}}$ given by $f_i(p_i) = \sin(A_i p_i + a_i)$.

By hypothesis, f_θ, and g_ϕ have the same depth d, and the width of each layer of g_ϕ is less than or equal to the width of the respective layer of f_θ, i.e., $N_i\leq M_i$. Thus, we define the hidden layers of f_θ using $A_i = \left(\begin{smallmatrix} B_i & 0 \\ 0 & 0 \end{smallmatrix}\right)$, $a_i = \left(\begin{smallmatrix} b_i \\ 0 \end{smallmatrix}\right)$. Evaluating $(p,c)\in\mathbb{R}^{N_i}\times\mathbb{R}^{M_i-N_i}$ in f_i results in

$$f_i(p,c) = \sin\left(\left(\begin{smallmatrix} B_i & 0 \\ 0 & 0 \end{smallmatrix}\right)\left(\begin{smallmatrix} p \\ c \end{smallmatrix}\right) + \left(\begin{smallmatrix} b_i \\ 0 \end{smallmatrix}\right)\right) = \left(\begin{smallmatrix} g_i(p) \\ 0 \end{smallmatrix}\right).$$

Thus, defining $A_1 = \left(\begin{smallmatrix} B_1 & 0 \\ F_p & F_t \end{smallmatrix}\right)$, $a_1 = \left(\begin{smallmatrix} b_1 \\ 0 \end{smallmatrix}\right)$, we obtain the desired result because

$$f_1(p,c) = \sin\left(\left(\begin{smallmatrix} B_1 & 0 \\ F_p & F_t \end{smallmatrix}\right)\left(\begin{smallmatrix} p \\ t \end{smallmatrix}\right) + \left(\begin{smallmatrix} b_i \\ 0 \end{smallmatrix}\right)\right) = \left(\begin{smallmatrix} g_1(p) \\ F_p c + F_t t \end{smallmatrix}\right).$$

In other words, the neurons $g_i(p)$ of the network g_ϕ remain intact along the layers. □

The blocks F_p and F_t project the entry points (p,t) to a dictionary of sines waves, which are not considered in the following layers because they are fed to zeros blocks. However, new hidden weights can activate such features as the training advances. In Sec 4.4, we present experiments varying the width of f_θ to explore such initialization in the problem of solving the MCE.

To reproduce Prop 2.1 with f_θ deeper than g_ϕ, we must be able to add a hidden layer $f(p_i) = \sin(Ap + a)$ to f_θ which do not exist in g_ϕ. Thus, it would be desirable to initialize f as an identity layer. Following the above approach, we could define $A = I$ and $a = 0$ obtaining $f(p) = \sin(p)$, however, in general, $\sin(p)\neq p$. This can be fixed using that $\sin(p)\approx p$ when $\|p\|$ is close to zero. Therefore, we define $A = \lambda I$, with λ being a small number, and multiply the resulting output of f by $\frac{1}{\lambda}$ to keep it close to p.

3. Extracting a network at a given time instant

Here, for a given time instant t, we extract the network $g_\phi = f(\cdot,t):\mathbb{R}^3\to\mathbb{R}$ from a neural network $f_\theta:\mathbb{R}^3\times\mathbb{R}\to\mathbb{R}$. Suppose $f_\theta(p,t) = A_{d+1}\circ f_d\circ\cdots\circ f_1(p,t) + a_{d+1}$, with each i-layer defined by $f_i(p_i) = \sin(A_i p_i + a_i)$.

To define g_ϕ such that $g_\phi(p) = f_\theta(p,t)$ for all (p,t), we modify the first layer $f_1(p,t) = \sin(A_1(p,t) + a_1)$ of f_θ. The matrix A_1 has 4 column vectors $\{w_1, w_2, w_3, u\}$ in

$\mathbb{R}^{M_2}$, where M_2 is the dimension of the codomain of f_1. Denoting p by (x, y, z), we obtain

$$A_1(p, t) = x \cdot w_1 + y \cdot w_2 + z \cdot w_3 + t \cdot u,$$

We use the matrix B_1 consisting of the columns w_1, w_2, w_3, and the bias $b_1 = a_1 + t \cdot u$ to set the first layer g_1 of g_ϕ. Specifically, we define g_ϕ through:

$$g_\phi(p) = A_{d+1} \circ f_d \circ \cdots \circ f_2 \circ g_1(p) + a_{d+1}.$$

Note that g_ϕ equals f_θ, except for its first layer $f_1(p, t)$, which is replaced by $g_1(p)$. We define it as

$$g_1(p) = \sin\Big(\underbrace{x \cdot w_1 + y \cdot w_2 + z \cdot w_3}_{B_1 p} + \underbrace{t \cdot u + a_1}_{b_1}\Big).$$

From the definition of g_ϕ, we have $g_\phi(p) = f_\theta(p, t)$, which implies a kind of opposite direction of Prop 2.1.

Proposition 3.1. *Let $f_\theta : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ be a neural network, and $t \in \mathbb{R}$. There is a network $g_\phi : \mathbb{R}^3 \to \mathbb{R}$ with the same hidden layers of f_θ such that $f_\theta(p, t) = g_\phi(p)$ for all $p \in \mathbb{R}^3$.*

4. Ablation studies

The ablation studies detailed below were performed using the MCE (Eq 1) under two settings: With our initialization scheme, presented in Prop 2.1, and the standard initialization [3]. The goal is to compare the training convergence of *data constraint* $\mathcal{L}_{\text{data}}$ and *LSE constraint* $\mathcal{L}_{\text{LSE}}$ for both initialization schemes under different circumstances.

We will visualize the graphs of $\mathcal{L}_{\text{data}}$ and $\mathcal{L}_{\text{LSE}}$ during the training of a neural network $f_\theta : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ to satisfy the MCE within a time interval (a, b). In the upcoming experiments, we will use the SDF $g : \mathbb{R}^3 \to \mathbb{R}$ of the Bunny as the initial condition, $f = g$ on $\mathbb{R}^3 \times \{0\}$. To initialize f_θ using our method, we approximate g by a network g_ϕ.

4.1. Varying time interval

Here, we vary (a, b) in the MCE with scale $\alpha = 0.001$. We recall that the initial condition is at $0 \in (a, b)$ and on the positive (negative) part, the MCE smooths (sharpens) it. Thus, the positive part should be easier to train since no higher frequencies would arise. In contrast, training the negative part creates new higher frequencies, which could take longer to learn. We validate it in the following intervals:

$$\begin{aligned}(a, b) = &(0, 0.25), (0, 0.5), (0, 1),\\ &(-0.1, 0.25), (-0.1, 0.5), (-0.1, 1),\\ &(-0.25, 0.25), (-0.25, 0.5), (-0.25, 1).\end{aligned}$$

Fig 1, 2 present the data/LSE constraint convergences for these intervals. As expected, our initialization (top images) provides a better training convergence. For $\mathcal{L}_{\text{data}}$, this is due to the fact that $f_\theta = g_\phi$ at $t = 0$, thus, $\mathcal{L}_{\text{data}}$ only have to maintain this restriction.

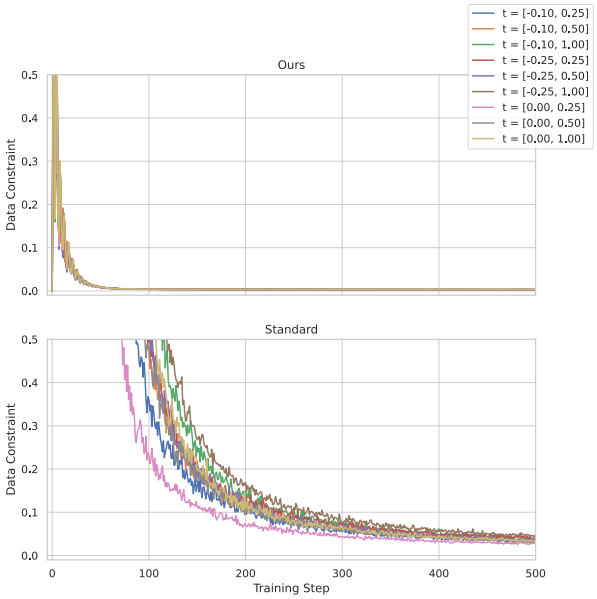

Figure 1. Data constraint values for different training intervals.

The convergence of the constraint $\mathcal{L}_{\text{data}}$ and $\mathcal{L}_{\text{LSE}}$ are faster, for both initializations, when using smaller intervals, as can be seen in the case of $(0, 0.25)$ (in purple). They also take longer to train on intervals with a negative part. This is likely because the solutions in such regions are sharper, requiring, thus, more frequencies for accurate representation, if a solution exists at all.

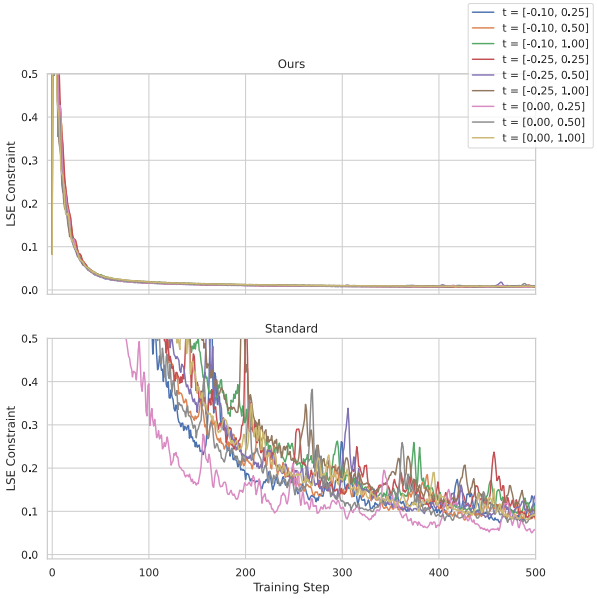

Figure 2. LSE constraint values for different training intervals.

4.2. Varying MCE scale

Here, we use the interval $(a, b) = (-0.1, 1)$ and vary the scales $\alpha = i \times 10^{-3}$ for $i = 1, 2, 3, 4, 5, 10, 100$. In theory, increasing (a, b) while fixing α is equivalent to the previous experiment. However, in practice, the representation capacity of f_θ may not be enough to learn large variations in a short time period. This is evident in Figs 3-4, where the convergence of $\mathcal{L}_{\text{data}}$ and $\mathcal{L}_{\text{LSE}}$ is sorted by α. In general, our initialization results in a better convergence, but we observed that using a high scale, $\mathcal{L}_{\text{data}}$ first diverges since $\mathcal{L}_{\text{LSE}}$ dominates the training. See the case $\alpha = 0.1$ (in purple).

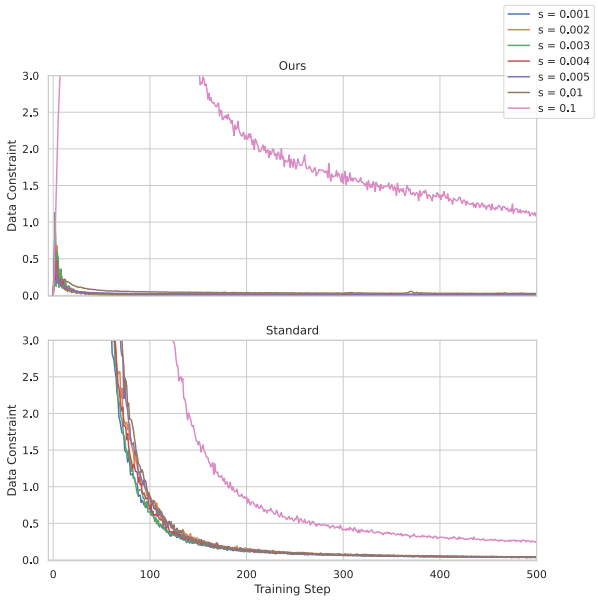

Figure 3. Data constraint values for different MCE scale values.

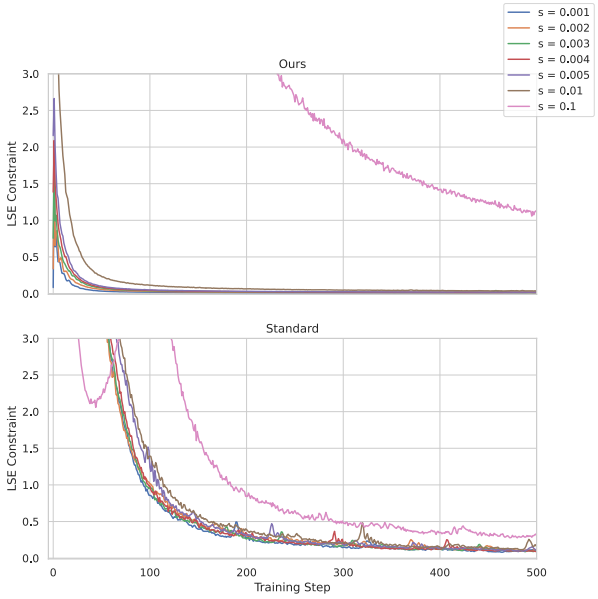

Figure 4. LSE constraint values for different MCE scale values.

4.3. Varying the point-sampling proportions

This experiment aimed to evaluate how the point-sampling of initial and intermediate conditions impacts the training convergence of $\mathcal{L}_{\text{data}}$ and $\mathcal{L}_{\text{LSE}}$. We used the default point-sampling proportions of $\{l_1, l_2, l_3\} = \{0.25, 0.25, 0.5\}$, as well as $\{0.1, 0.1, 0.8\}$ and $\{0.4, 0.4, 0.2\}$. Here, l_1, l_2, and l_3 are the numbers of space-time, on-surface, and off-surface points sampled at each training step (see Sec 4.2 of the main paper).

Figs 5-6 present the convergences of the resulting constraints during training. It can be observed that sampling fewer points at $t = 0$ results in a better convergence.

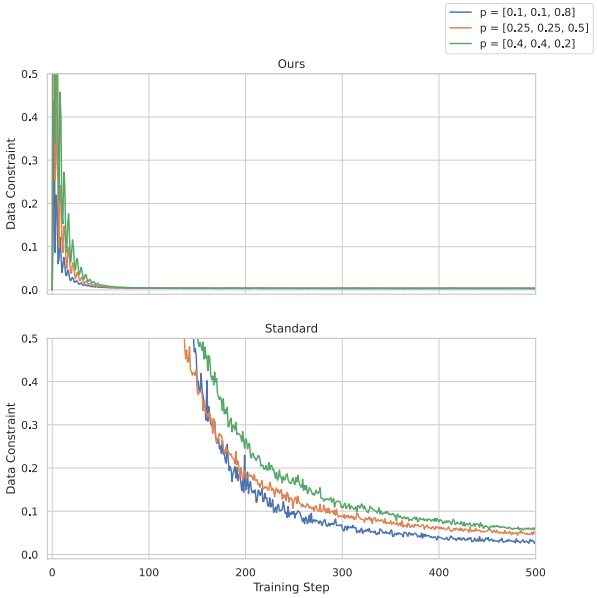

Figure 5. $\mathcal{L}_{\text{data}}$ values for different sampling proportions.

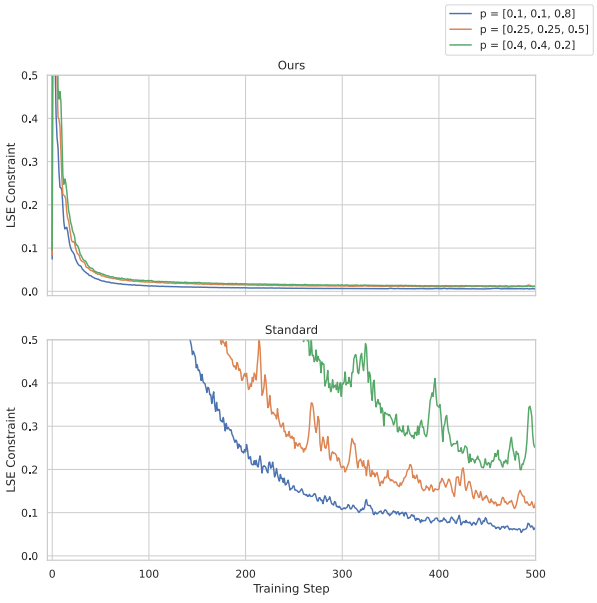

Figure 6. $\mathcal{L}_{\text{LSE}}$ values for different sampling proportions.

However, when using different sampling proportions, we obtain new constraints $\mathcal{L}_{\text{data}}$ and $\mathcal{L}_{\text{LSE}}$. Also, sampling fewer points at $t = 0$ can result in a longer convergence time for $\mathcal{L}_{\text{data}}$, as shown in the initial condition (Bunny) for each proportion in Fig 7. This is probably due to the *spectral bias* phenomenon: lower frequencies are learned first. As a result, $\mathcal{L}_{\text{LSE}}$ benefits from having a smoother initial condition and prevents the fitting at $t = 0$.

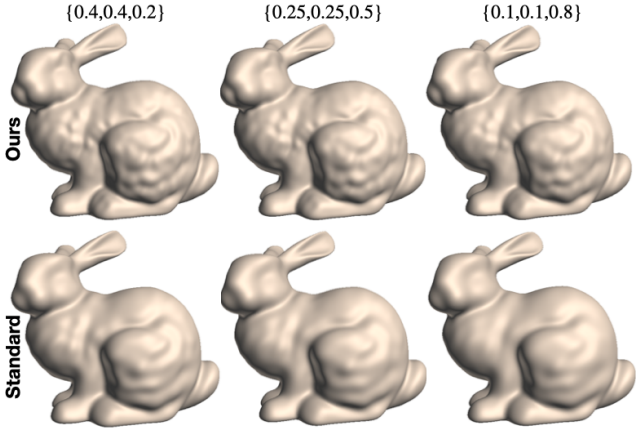

Figure 7. The zero-level sets of f_θ at $t = 0$ trained using the proportions $\{0.1, 0.1, 0.8\}$, $\{0.25, 0.25, 0.5\}$, and $\{0.4, 0.4, 0.2\}$.

4.4. Varying the network width

This experiment evaluates the impact of the network width on the training convergence using our standard initializations. We began with a width of 128 neurons and increased it by 16 neurons up to a limit of 256. The remaining parameters are set to $(a, b) = (-0.25, 1)$, $\alpha = 1e-3$, $\{l_1, l_2, l_3\} = \{0.25, 0.25, 0.5\}$. As expected, increasing the width leads to better convergence; see Figs 8-9.

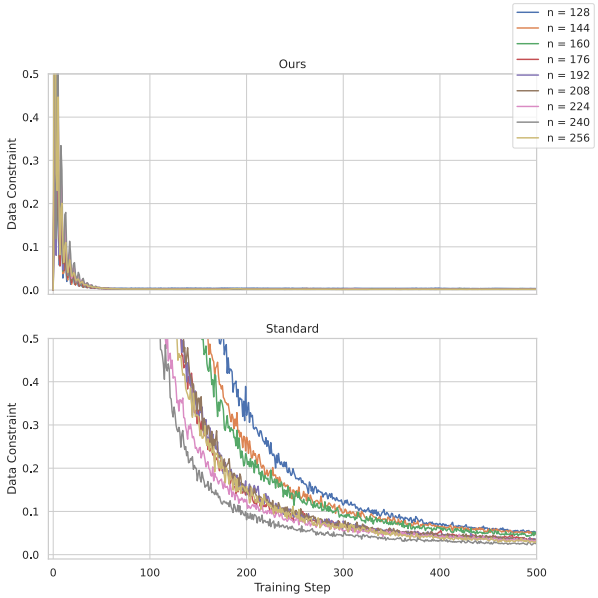

Figure 8. Data constraint values for different network widths.

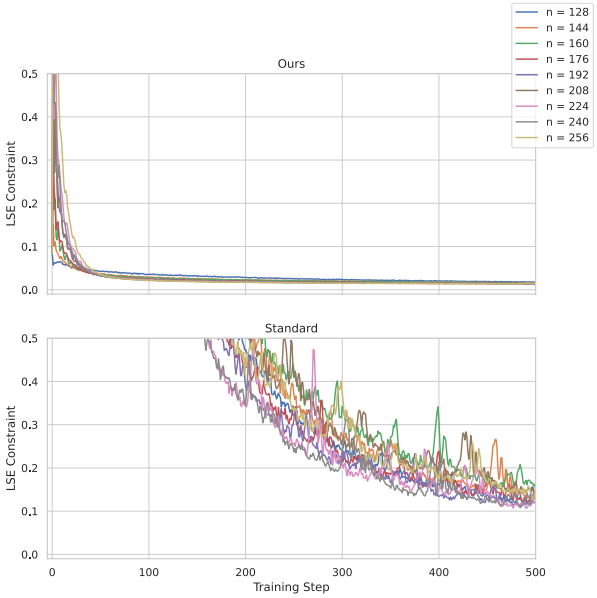

Figure 9. LSE constraint values for different network widths.

4.5. Varying the initial condition

Finally, we vary the initial condition of the MCE to evaluate the convergence of the network f_θ on different models: Bob, Max Planck, Falcon, Witch, and Neptune. During the training of each model, we fix the sampling proportions to $\{l_1, l_2, l_3\} = \{0.25, 0.25, 0.5\}$. Table 1 presents the network architectures (the width of the hidden layers), the time required for training 500 epochs, the animation interval (a, b), and the MCE scale α parameters.

Model	Network arch.	Interval	Scale	Time (s)
Bob	$[64, 64]$	$(-0.5, 1)$	$1e-2$	5.04
Max	$[128, 128, 128]$	$(-0.5, 1)$	$2e-3$	7.07
Falcon	$[160, 160, 160]$	$(-0.1, 1)$	$1e-3$	112.17
Witch	$[256, 256, 256]$	$(-0.5, 1)$	$1e-3$	143.34
Neptune	$[300, 300, 300]$	$(-0.1, 1)$	$2e-4$	162.42

Table 1. The network architectures and the time spent in their training to learn the evolution of the Bob, Max Planck, Falcon, Witch, and Neptune surfaces under the MCE.

For the sampling of on-surface points, we used different point clouds sampled from the original models. This affects the time needed to train our networks, as each epoch is defined as a complete iteration over the point-cloud. The Bob, Max Planck, Falcon, Witch, Neptune have 5344, 5002, 72466, 77553, 72668 points, respectively.

Fig 10 illustrates the zero-level sets of the resulting evolutions of Bob, Max Plank, Falcon, Witch and Neptune models (middle). The sharpened models are on the left column. Notice that their geometric features are enhanced. Particularly, Max Planck's nose, mouth and ears are noticeably more prominent. The same occurs for the Wizard's sword and cape, and Neptune's hands and spear tip.

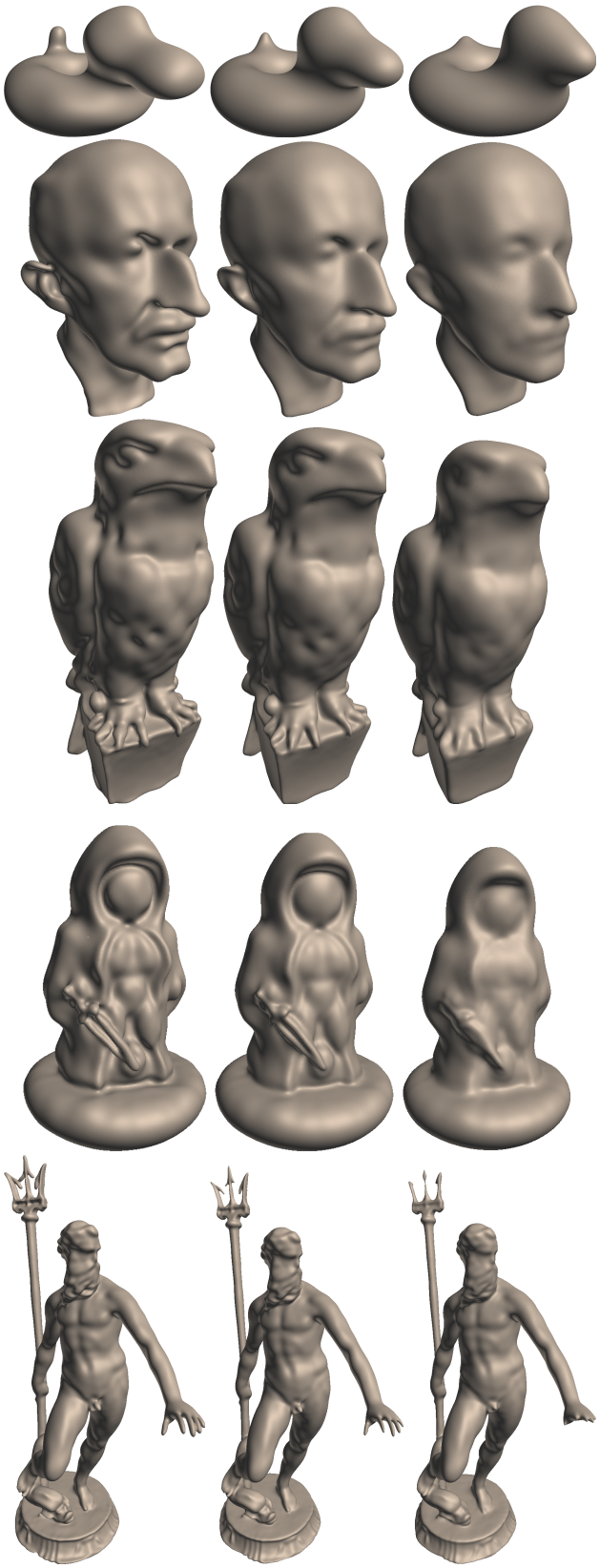

Figure 10. The zero-level sets of f_θ for Bob, Max Plank, Falcon, Witch, and Neptune models (middle). The left and right columns give the sharpening and smoothing of the models.

References

[1] Manfredo P Do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016. 1

[2] Francisco Martín and Jesús Pérez. An introduction to the mean curvature flow. In *XXIII International Fall Workshop on Geometry and Physics, held in Granada. https://www. ugr. es/jpgarcia/investigacion. html*, 2014. 1

[3] 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. 2

[4] Stephanie Wang and Albert Chern. Computing minimal surfaces with differential forms. *ACM Transactions on Graphics (TOG)*, 40(4):1–14, 2021. 1