University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's thesis

Representing time-varying surfaces using neural networks

Plzeň 2021 Filip Hácha

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Filip HÁCHA**
Osobní číslo:	**A19N0112P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Počítačová grafika**
Téma práce:	**Reprezentace časově proměnných povrchů neuronovými sítěmi**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s metodou reprezentace statických povrchů pomocí neuronové sítě reprodukující znaménkovou vzdálenostní funkci.
2. Navrhněte způsoby, jak metodu rozšířit o možnost reprezentace časově proměnného povrchu.
3. Implementujte navržené způsoby a implementaci verifikujte na dodaných datech.
4. Dosažené výsledky a výhody a nevýhody jednotlivých navržených přístupů zhodnoťte a navrhněte možná další rozšíření a vylepšení.

Rozsah diplomové práce:	**doporuč. 50 s. původního textu**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování diplomové práce:	**tištěná**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce:	**Doc. Ing. Libor Váša, Ph.D.** Katedra informatiky a výpočetní techniky
Datum zadání diplomové práce:	**11. září 2020**
Termín odevzdání diplomové práce:	**20. května 2021**

L.S.

Doc. Dr. Ing. Vlasta Radová děkanka	**Doc. Ing. Přemysl Brada, MSc., Ph.D.** vedoucí katedry

V Plzni dne 24. září 2020

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Plzeň, 20th May 2021

Filip Hácha

Acknowledgement

I would like to thank doc. Ing. Libor Váša, Ph.D. for professional guidance, many valuable and inspiring advice, suggestions, recommendations, comments, and at the same time, considerable patience and willingness to provide consultations for this thesis. At the same time, I would like to thank Ing. Jan Dvořák for his valuable observations, advice, and help with implementing experiments associated with this work. Thanks also go to AXYZ design S.R.L. for the provided data.

Abstract

This work deals with the possibility of using neural networks for the representation of time-varying surfaces. The proposed method is based on the overfitting of a neural network in the regression of the signed distance function of the surface. The proposed neural representation of dynamic surfaces was tested on two different sequences of triangle meshes, and subsequently, other techniques were proposed to improve the quality of the reconstructed surface. The quality of the resulting representation was examined using a perceptual metric for comparing surfaces. The results show good compression properties of the proposed representation of dynamic surfaces and demonstrate the possibility of using this method for temporal super-resolution of the original surface.

Abstrakt

Tato práce se zabývá možností použití neuronových sítí pro reprezentaci časově proměnných povrchů. Navržená metoda je založena na přeučení neuronové sítě při regresi znaménkové vzdálenostní funkce povrchu. Navržená neuronová reprezentace dynamických povrchů byla otestována na dvou různých sekvencích trojúhelníkových sítí a následně byly navrženy další techniky pro zlepšení kvality rekonstruovaného povrchu. Kvalita výsledné reprezentace byla prozkoumána s použitím percepční metriky pro porovnávání povrchů. Výsledky práce ukazují dobré kompresní vlastnosti navržené reprezentace dynamických povrchů a demonstrují možnost použití této metody pro zvýšení snímkové frekvence původního povrchu.

Contents

1 Introduction

In computer graphics, computational geometry, and other scientific fields, we often work with representations of three-dimensional objects. These representations can be generally divided into volume representations and surface representations.

For volume representations, we are typically interested not only in the shape of the object, but also in the values inside. These values can represent the material composition of a given object, temperature, density, etc. A typical example is an output of a CT or MRI scan, where we get values of measured X-ray data for different points, and we can examine both the shape of individual organs and their internal properties. On the other hand, in the case of surface representations, we are exclusively interested in the shape of a given object or its properties on the surface.

In the surface representation of an object, we have a choice of a number of different methods. These methods could be further divided into mesh methods based on a tessellation of the given surface into basic primitives. In computer graphics, these meshes are most often made of triangles. On the contrary, mesh-free (meshless) methods typically do not require any tessellation. The advantage of mesh methods is their easy rendering and their editing, while the advantage of meshfree methods is that they lead to continuous and smooth representations, where we can easily interpolate the data and they also offer a more compact representation of the surface.

However, the whole problem of surface representation can be further complicated when we need to represent dynamic surfaces, i.e., surfaces which shape changes over time. Recent research in the field of surface representation and machine learning has shown, that we can train neural network to provide an implicit description of a static surface, so we can use this neural network as a compact surface representation. The objective of this work is to generalize the approach based on the use of neural networks to dynamic surfaces and to study the properties of this representation.

2 Surfaces and their representations

In the context of computer graphics, the term surface refers to an oriented continuous 2D manifold in the space $\mathbb{R}^3$ [BKP*10]. In the case of $\mathbb{R}^3$, a manifold is a two-dimensional variaty or area. Otherwise, we can also characterize the surface as the boundary of a three-dimensional body with a non-zero volume. In this case, the surface is the interface between the inside and outside of the body.

2.1 Parametric surfaces

Parametric surfaces are representations given by a function $\boldsymbol{f} : \Omega \rightarrow \mathcal{S}$, which assigns a specific point on the surface $\mathcal{S} = \boldsymbol{f}(\Omega) \subset \mathbb{R}^3$ to a 2D parameter from the set $\Omega \subset \mathbb{R}^2$ [BKP*10]. Parametric surfaces could be further divided according to the type of parameterization into rational surfaces that admits parameterizations by a rational function. Another group can be surfaces using spherical coordinates or surfaces using cylindrical coordinates.

An example of a parametric surface description can be a description of a spherical surface. Such a description can be constructed, for example, by rotating a circle around an axis that lies in the plane of the circle and passes through its center. Let us have a parametric description of a unit circle with a center at the beginning $k(t) = [cos(t), sin(t)]^T; t \in \langle 0, \pi \rangle$. Rotating this circle by an angle π gives a description of the surface $s(u, v) = [cos(u)cos(v), sin(u)cos(v), sin(v)]^T; u \in \langle 0, 2\pi \rangle, v \in \langle 0, \pi \rangle$, which will be a spherical surface with radius $r = 1$ with the center at the origin $[0, 0, 0]^T$ (see figure 2.1).

Equivalently, we can define a time-varying surface using a parametric representation such as $\mathcal{S}(t) = f(\Omega, t) \subset (\mathbb{R}^4)$ where the fourth component of the vector in $\mathbb{R}^4$ represents the time. One of the possible advantages of such a representation of dynamic surfaces is that the parametric function is generally continuous with respect to the time parameter t, so we are not limited by the finite frame rate when rendering the surface. The disadvantage is that finding such a representation of a dynamic surface is generally a complex task and is therefore not widely used in practice.

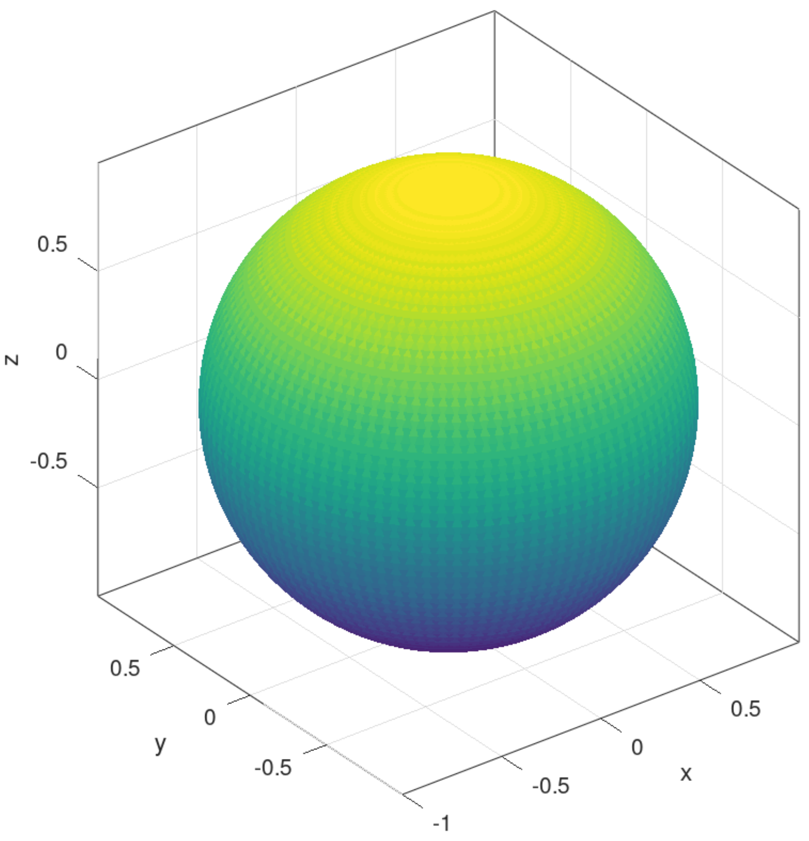

Figure 2.1: Example of spherical surface given by parametric representation.

2.2 Implicit surfaces

Another possible representation of surfaces is the so-called implicit description, which are defined as a set of points $\boldsymbol{x} \in \mathbb{R}^3$ for which function $F : \mathbb{R}^3 \to \mathbb{R}$ takes the value zero. We can descripbe a surface $\mathcal{S}$ as $\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^3 | F(\boldsymbol{x}) = 0\}$ [BKP*10]. In general, we can also describe the implicit surface as $F(\boldsymbol{x}) = h, h \in \mathbb{R}$, according to the specific choice of the associated value h, we then obtain the equations of the so-called iso-surfaces. An example of an implicit surface description can also be an equation describing the surface of a sphere. E.g. the surface of a sphere with a center at the origin and a radius $r = 1$ is described by the equation. (2.1) Such a description thus corresponds to the parametric description of the sphere shown in the figure 2.1

One of the advantages of the implicit description is the possibility of visualizing the surface section by substituting a constant value $c \in \mathbb{R}$ for one of the coordinates x, y, z. In this section, we can also easily show the relationship between the associated values of the implicit description and a given iso-surface. Section of a unit sphere obtained by fixing a coordinate $y = 0$ can be seen in the figure 2.2. The slice corresponding to the associated value $h = 0$ is highlighted with red color in the figure, the other slices are graded with step 1.

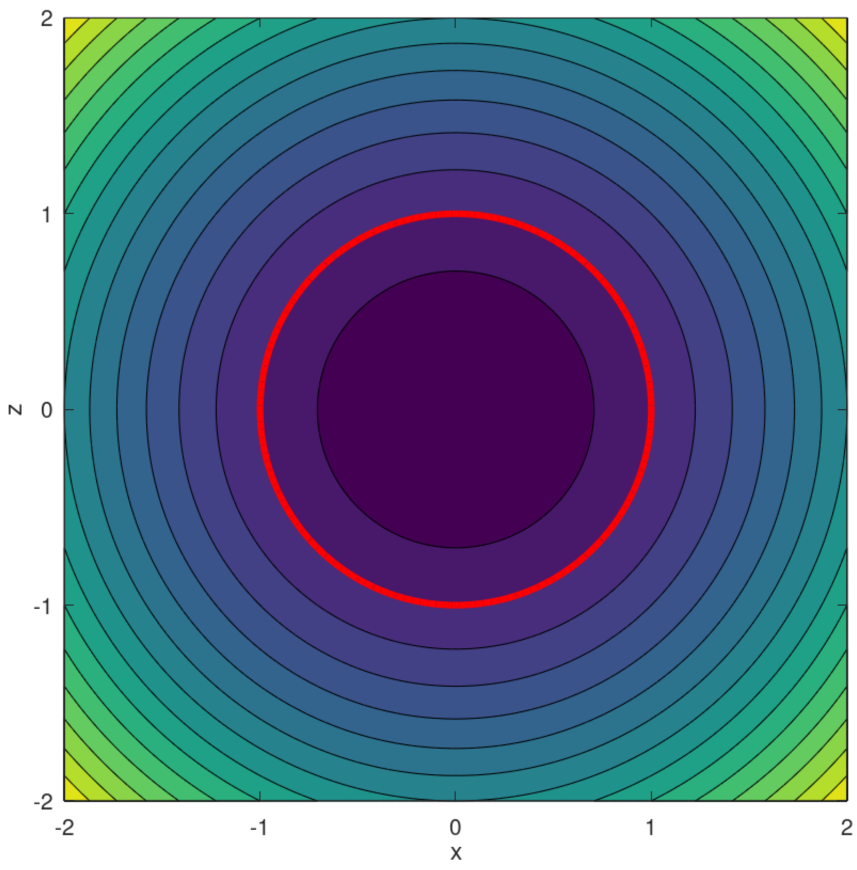

Figure 2.2: Iso-surfaces visualisation of implicit sphere description.

$$F(\boldsymbol{x}) = x^2 + y^2 + z^2 - 1 = 0 \tag{2.1}$$

Equivalently, as in the case of a parametric representation, we can define a time-varying surface in an implicit representation by adding a time parameter $t \in \mathbb{R}$ to the function F, as shows equation (2.2). Even here, however, finding a function $F(\boldsymbol{x}, t)$ representing a given surface is very difficult.

$$F(\boldsymbol{x}, t) = 0 \tag{2.2}$$

2.3 Triangle meshes

Triangle meshes are a commonly used representation of surfaces. This is a special case of so-called polygonal meshes, which is an approximation of a surface using a set of polygons. Triangle mesh $\mathcal{M}$ is given by the set of vertices $\mathcal{V}$, the set of triangle faces $\mathcal{F}$ and the set of edges $\mathcal{E}$. Any point $\boldsymbol{p}$ on the surface, which is represented by a triangle mesh can be expressed as a point within a relevant triangle face with vertices $[\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}]$ using barycentric coordinates $[\alpha, \beta, \gamma]$, as shows equation (2.3).

$$\begin{aligned}&\boldsymbol{p} = \alpha\boldsymbol{a} + \beta\boldsymbol{b} + \gamma\boldsymbol{c},\\&\alpha + \beta + \gamma = 1,\\&\alpha, \beta, \gamma \geq 0\end{aligned} \tag{2.3}$$

Furthermore, it is appropriate to introduce the concept of a manifold in the context of a triangle mesh. In order to declare a triangle mesh to be a manifold, it must satisfy the condition that each inner point on the surface is locally homeomorphic to a disc, if it is a point that lies on the surface boundary, the point must be locally homeomorphic to a half-disc. This statement is then equivalent to the statement that a triangle mesh is a manifold if it does not contain so-called non-manifold vertices (see figure 2.3 on the left and on the right), non-manifold edges (viz obr. 2.3 in the middle) and no two faces intersect.

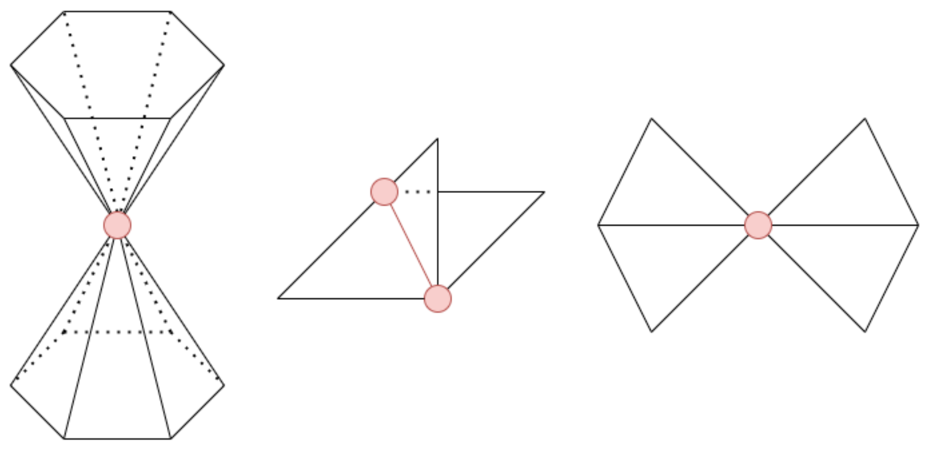

Figure 2.3: Representation of non-manifold triangle meshes.

As in the case of parametric and implicit descriptions, we can also use triangle meshes to represent time-varying surfaces. One possibility is to represent the surface as a sequence of static triangle meshes. The advantage of this representation is that such triangle meshes are generally not subject to any special requirements. The disadvantage, however, is that when using a sequence of triangles, we do not have the possibility of interpolating the surface over time. All information about the surface is only determined at times that correspond to the given frames.

The second option is to separate the geometry and connectivity of the triangle mesh. By the geometry of a triangle mesh, we mean the positions of the vertices of the mesh in space, while by connectivity we mean how the vertices of the network are connected by edges and faces. If we want to represent a time-varying surface, we can construct the connectivity of a triangle mesh that will be the same at all times and only the geometry will change. This method of representation is widely used in practice, for example in cases where we animate an originally static mesh. The animator can use the so-called skeleton of the model. It is a hierarchical structure, most often formed by a set of line segments. For each vertex of the triangle mesh, we then introduce an ordered set of weights, which determines how much the position of the given vertex is tied to the position of the given skeleton element of the model. We can then easily create the animation by determining only the way in which the skeleton of the model moves and calculating the positions of the individual vertices.

Of course, even in such case, we can sample the representation in time and obtain a representation formed by a sequence of triangle meshes. It follows, therefore, that the representation by a mesh sequence is in a sense more general, since it does not impose a condition on the same connectivity of the mesh at different times.

2.4 Signed distance function

Signed distance function (or oriented distance function) is generally a function, which we define in a metric space, which for each element from the space assignes a numerical value, whose absolute value corresponds to the distance of the given element from the boundary of the set Ω and whose sign corresponds to whether the element lies within the set Ω, or outside. We can write that the signed distance function $sdf(\boldsymbol{x})$ is given by formula (2.4), where $\partial\Omega$ is the boundary of the set Ω and $d(\boldsymbol{x_1}, \boldsymbol{x_2})$ is the distance between points $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ according to the metric in this space.

$$\begin{aligned} sdf(\boldsymbol{x}) &= \begin{cases} d(\boldsymbol{x}, \partial\Omega) & \text{if} \quad \boldsymbol{x} \in \Omega \\ -d(\boldsymbol{x}, \partial\Omega) & \text{if} \quad \boldsymbol{x} \in \Omega^C \end{cases} \\ d(\boldsymbol{x}, \partial\Omega) &= \inf_{\boldsymbol{p} \in \Omega} d(\boldsymbol{x}, \boldsymbol{p}) \end{aligned} \tag{2.4}$$

Thus, we can use the signed distance function as a representation of a surface so that for a set Ω by marking all points within the surface $\Omega = \mathcal{S}$

and distance $d(\boldsymbol{x_1}, \boldsymbol{x_2})$, where $\boldsymbol{x_1}, \boldsymbol{x_2} \in \mathbb{R}^3$, we define as Euclidean distance according to equation (2.5).

$$d(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{2.5}$$

In such a case, the signed distance function is actually only a special case of the implicit description of the surface, since the set of points $S \subset \mathbb{R}^3$, where each point $\boldsymbol{s} \in S$, for which this function takes the value 0 $sdf(\boldsymbol{s}) = 0$, gives the original surface $\mathcal{S}$. An example of the SDF function of a three-dimensional surface is shown in Figure 2.4.

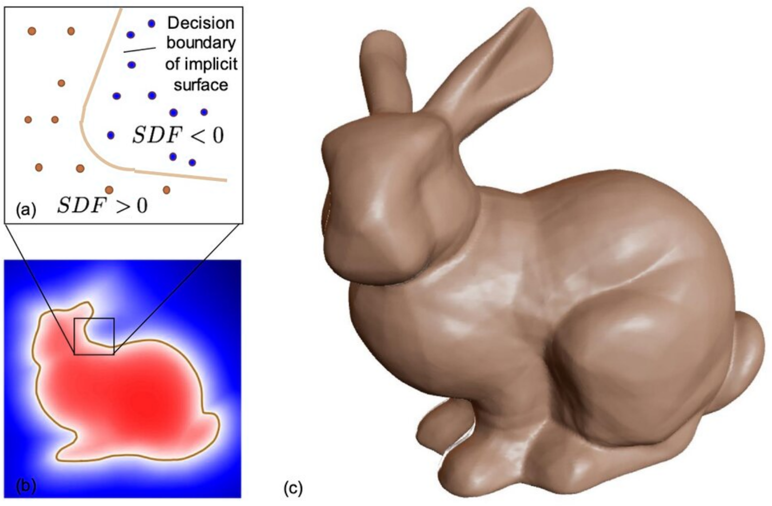

Figure 2.4: Signed distance function of the bunny surface. The signed distance function is zero on the surface of the bunny, positive inside (red region) and negative outside (blue region). [PFS*19]

As in the previous cases, in principle, there is nothing to prevent us from using a signed distance function to represent a dynamic surface by sampling the surface described by one of the above representations at different times. The possibilities of calculating signed distance function from another surface representation are further discussed in the section 5.1.

2.5 Conversions between representations

In general, of course, it is possible to represent one surface in several different ways, and there are therefore methods for converting individual representations. For example, a surface described by a parametric equation can generally be converted to an implicit surface equation. We achieve this in such a way that we try to express the parameters of the given parametric equations from $d-1$ equations, where d is the dimension of the given space, and we substitute them into the last equation. In this way, we can get a different representation of exactly the same surface, however, finding the analytical formula of the implicit function in this way can be a rather complex problem. Therefore, it is often more appropriate to only approximate the original surface, which gives a surface that differs slightly from the original, however, the whole process of conversion can be significantly simplified.

Further in this work, for the purposes of experiments, we will deal with the need to convert the implicit surface representation to a triangle mesh. There are several algorithms often used in practice for this conversion. One of the most commonly used algorithms is the Marching Cubes [LC87] method. The Marching cubes algorithm is based on a division of the space in which the surface is located in a grid. This grid is further traversed by individual cubes, and in each cube the vertices are classified based on whether they are above or below the surface to be treated. Since the number of vertices of a cube is finite (8 vertices) and only two cases can occur in each vertex, the number of ways in which the surface that we approximate in a given cube by a set of triangles passes through a given cube is also finite. Thus, this set of triangles is added whose vertices are calculated by iterpolating the associated value of the implicit function in the vertices of the cube. After passing through the whole grid, we get a set of vertices and triangle faces, including their orientation, which therefore defines the given triangle mesh.

2.6 Overfit SDF

Overfit SDF, also called a neural implicit, is a way of representing a surface by regressing the signed distance function using an artificial neural network. An artificial neural network is a type of computational model in the field of machine learning. The model for the creation of artificial neural networks are biological structures formed by neurons and dendrites. Based on this template, McCulloch and Pitts [MP88] described the artificial neuron model currently used by most artificial neural networks. This artificial neuron can be described as the transformation of the input $\boldsymbol{x}$ to the output Y by equation (2.6), where w_i are the synaptic weights, Θ is the threshold of the neuron and $S(x)$ is the activation function of the neuron.

$$Y = S(\textstyle\sum_{i=1}^{N}(w_i x_i) + \Theta) \tag{2.6}$$

By connecting artificial neurons into layers and chaining them, structures are created that we can use for solving complex regression or classification problems. Thanks to this, todays artificial neural networks are used in various fields, including computer graphics.

A forward neural network can be used to approximate the signed distance function of the surface, which means that the network contains several linearly concatenated layers, where the input of a neuron is the output of the neurons in the previous layer. Such a network usually has 3 inputs, which are formed by the coordinates of the points x, y and z. The output of the neural network then represents an approximation of the signed distance function at this point. The network contains several hidden layers, typically with ReLU activation functions, and a hyperbolic tangent is used as the output function.

The results that can be achieved by surface representation using SDF overfit are shown in the work of T. Davies et al. [DNJ21]. They introduced a suite of technical contributions to improve reconstruction accuracy, convergence, and robustness when learning the signed distance field induced by a polygon mesh and demonstrated robustness, scalability, and performance of this representation.

Based on the results obtained using neural networks in the representation of static surfaces, the question arises whether it is possible to use neural networks for the representation of time-varying surfaces. Investigation of possible approaches to the use of neural networks in the representation of dynamic surfaces is the subject of this work.

3 Related work

As in other application areas, neural networks are increasingly used in a wide range of possible applications. One of this possible application is field of shape representation in computer graphics.

An interesting use of neural networks for shape representation was introduced in the work of Park et al. [PFS*19]. In contrast to the usual representation of a static surface using DeepSDF, where the neural network is overfitted with signed distance function data of one particular surface, the model described in this work allows to extend the generalization capabilities of the representation using an auto-encoder neural network of a given shape. Using this auto-encoder, the shape is converted to a code vector. When we then train the neural representation itself, we create a prediction of the SDF function not only using the coordinates of the given box, but also on the basis of this code vector (see Fig. 3.1).

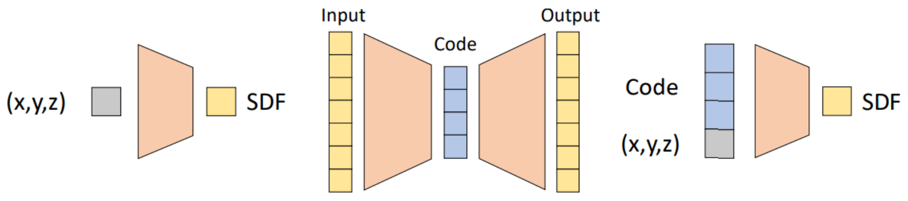

Figure 3.1: Left: Basic DeepSDF representation of single shape. Middle: Auto-encoder. Right: Coded shape DeepSDF. [PFS*19]

The improvement of the results achieved by the use of neural networks in the representation of surfaces using special periodic activation functions was recently described in the work of Sitzmann et al. [SMB*20]. In contrast to previously commonly used functions for the representation of shapes such as ReLU, TanH, or radial basis functions, the authors use SIRENE architecture. This architecture uses the sine as a periodic activation function (see Equations (3.1) and (3.2)).

$$\Phi(\mathbf{x}) = \mathbf{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ ... \circ \phi_0)(\mathbf{x}) + \mathbf{b}_n \tag{3.1}$$

$$\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i) = sin(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i) \tag{3.2}$$

Experiments in their work demonstrated, that the proposed SIREN representation enables accurate representations of natural signals, such as images, audio, and video in a deep learning framework.

Another approach was described in the work of Mescheder et al. [MON*19]. They use occupancy neural networks to represent a given shapes. They are using neural network to approximate occupancy function f_Θ, which assignes probability of occupancy to every possible point $\mathbf{x} \in \mathbb{R}^3$ (see Eq. (3.3)). Occupancy values is then 0 or 1 depending on whether point $\mathbf{x}$ lies outside or inside of given shape. They proved that occupancy networks are very expensive, but when the position of points used for training of network is not discretized, it can be used to represent realistic high-resolution meshes.

$$f_\Theta : \mathbb{R}^3 \to [0, 1] \tag{3.3}$$

In contrast to most other representations, [AL20] et al. in their work they introduced the use of sign agnostic learning for the representation of static surfaces. The advantage of using sign agnostic learning is that it does not require information during training whether a given point is inside or outside the surface, which is represented by a sign of a given value when using SDF. For training of their neural representation they used loss function showed in Eq. (3.4).

$$loss(\boldsymbol{\Theta}) = \mathbb{E}_{x \sim D_\mathcal{S}} \tau(f(\mathbf{x}, \boldsymbol{\Theta}), h_\mathcal{S}(\mathbf{x})) \tag{3.4}$$

Where $\boldsymbol{\Theta}$ are weights of neural network, $f : \mathbb{R}^3 \times \mathbb{R}^m \to \mathbb{R}$ is the surface approximating function of neural network. Function $h_\mathcal{S}(\mathbf{x})$ measures unsigned distance of point $\mathbf{x}$ from the surface and $\tau(a, b)$ is differentiable unsigned similiarity function defined bz the properies described in Equations (3.5) and (3.6). Condition shown in Eq. (3.5) means that similiarity function must be sign agnostic and condition shown in Eq. (3.6) requires that similiarity function is monotonic. Function $\rho : \mathbb{R} \to \mathbb{R}$ is monotonically increasing function with $\rho(0) = 0$.

$$\tau(-a, b) = \tau(a, b), \forall a \in \mathbb{R}, b \in \mathbb{R}_+ \tag{3.5}$$

$$\frac{\partial \tau}{\partial a}(a, b) = \rho(a - b), \forall a, b \in \mathbb{R}_+ \tag{3.6}$$

The work of Tancik et al. [TSM*20b] showed that to achieve better results when using neural networks to represent coordinate based data, it is possible to use the transformation of feature vectors using Fourier features mapping. The experiments performed in this work show the advantages of using Fourier features in the approximation of high-frequency signals of various kinds, such as image, MRI data and shapes.

The advantages of using neural implicit surface representation are investigated in the work of Davies et al. [DNJ21] especially in terms of its compressive properties. Other techniques for improving results such as the use of SIREN activations and Fourier features mapping are also mentioned in this work. The work discusses the approach of how to sample triangle meshes into SDF. The basic ideas for generalizing the neural representation for the time-varying surface, which this work deals with, were drawn from their results.

4 Proposed time-varying representation

In this work, several possible approaches to the application of neural networks in the representation of time-varying surfaces were designed and tested for comparison. The following sections describe the neural representation for dynamic surfaces and possible approaches to further improve results of this representation. Examination of results of this representation is shown in the Chapter 8.

4.1 Dynamic neural implicit representation

As in the case of the representation of a static surface, in the case of a dynamic surface we can construct a neural network that approximates the SDF of a given surface. However, since SDF is time-varying in the case of the dynamic surface, we must increase the dimension of the feature space by the time component. Thus, the feature space consists of 4-component vectors (x, y, z, t). Furthermore, the network contains several inner dense layers and the output layer is again formed by one neuron, because the output of the neural network for a given feature vector will be an approximate value of the SDF at the given point and time.

As an activation function of the inner layers, it is possible to use the ReLU function as in the representation of static surfaces in [DNJ21]. Similarly, the hyperbolic tangent activation function is used for the output neuron.

Since the regression of the time-varying SDF is a significantly more complex task than the regression of the static SDF, it can be expected that the complexity of the neural network used will have to be greater than when representing static surfaces, since a static surface is a 2D variety in a 3D space, while a dynamic surface is a 3D variety in 4D space. The dimension of the whole problem is therefore higher. For this reason, neural networks with approximately 16 inner layers of 256 neurons per layer are used in the experiments described in Chapter 8. The structure of the neural network used is shown in figure 4.1.

The Adam algorithm [KB17], which is the first-order gradient-based optimization of stochastic objective functions algorithm, is used for neural network training.

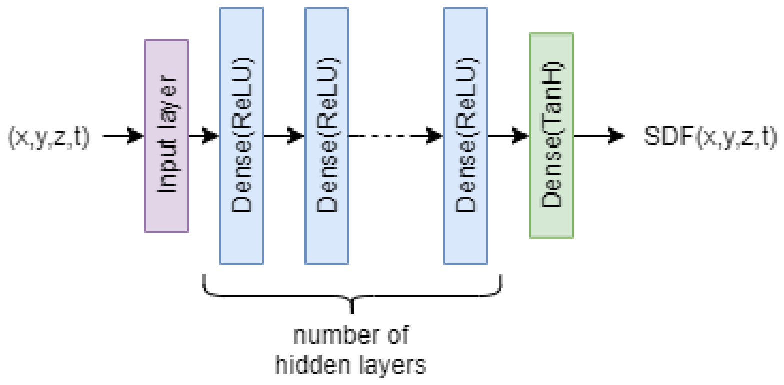

Figure 4.1: Structure of neural network used for dynamic surface representation.

Another key hyperparameter when using a neural network is the choice of the correct loss function. The aim of the loss function is to penalize the current parameters of a given neural network based on the error that the neural network makes when regressing on a training dateset. By neural network parameters, we mean the weights of individual neurons. The process of training a neural network then consists of minimizing this function by finding the optimal weights of its neurons. When using a neural network to represent a dynamic surface, it is appropriate to use Mean Absolute Error (MAE) as a loss function, as with the representation of static surfaces.

It is important to note that when using a neural network to represent a surface, unlike most other tasks, we try to overfit the neural network on data corresponding to one particular surface, while in most other applications we try to avoid overfitting the neural network.

4.2 Surface sensitive loss function

One of the possibilities that can lead to a more accurate approximation or faster convergence to the optimal setting of the neural network weights when using neural networks is to try to design a loss function that better corresponds to the problem at hand. It is therefore worth considering whether, even if a neural network is used to approximate the SDF dynamic surface, it is not possible to propose a loss function that could lead to better results than MAE.

In this work, a loss function was designed, which is based on the properties of the approximated SDF and the knowledge of how the surface is retrospectively reconstructed when using the neural network. When reconstructing the approximated dynamic surface from the implicit function

$f(\mathbf{x}, t)$, we perform iso-surface extraction at level zero. When converting the implicit surface representation to a triangle mesh, a set of vertices is created where the function $f(\mathbf{x}, t) = 0$. The idea behind the creation of the new loss function is that for this reason it would be good for the approximation of the implicit function $f(\mathbf{x}, t)$ to be as accurate as possible just near the zero iso-surface, where the approximation error causes a deformation of the resulting surface. Conversely, at points farther from the zero iso-area, we do not mind the approximation error, unless it causes a change in the sign of the approximated SDF. If the sign was changed in a place where the original surface was not, it would mean that artifacts in the form of unwanted triangle faces would be created during the reconstruction of the surface.

Based on these requirements, we can construct a function that depends on the absolute value of difference beween the true value y_{true} and predicted value y_{pred}, as well as the MAE, near the approximate surface. It will also take on which value for points where the sign of the true and predicted value differs. But at points that lie farther from the surface, and the predicted and true values here have the same sign, the loss function will yield zero. Due to the fact that the approximated SDF itself acquires values corresponding to the distance of a given point from the surface, its value can be used to find out how far a given point is from a given surface when calculating the loss function. As in the calculation of MAE, we use the arithmetic mean when comparing multiple samples. We can therefore define this function by an Eq (4.1), where $\delta > 0$ is a parameter defining the distance from the surface at which we consider points to be close enough to penalize an error in the approximation of their SDF and n is number of samples.

$$loss(y_{true}, y_{pred}) = \frac{1}{n} \sum_{(y_{true}, y_{pred})} \begin{cases} |y_{true} - y_{pred}| & \text{when } |(y_{true}| < \delta \\ |y_{true} - y_{pred}| & \text{when } |(y_{pred}| < \delta \\ |y_{true} - y_{pred}| & \text{when } y_{true} \cdot y_{pred} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

In order to be able to effectively quantify such a loss function when training a neural network, we will convert it to a more suitable form using the signum function. Thanks to this, it is possible to calculate this function for the whole set of pairs of true and predicted values of elementary operations, which will ensure a sufficiently efficient calculation. We will replace individual conditions by expressions c_1, c_2, and c_3. We negate the conditions, and say that these terms will take the value 0 if the prediction is to be penalized on the basis of the relevant condition, and 1 if there is no reason to penalize the prediction. We obtain the following Eq. (4.2).

$$loss(y_{true}, y_{pred}) = \frac{1}{n} \sum_{(y_{true}, y_{pred})} (1 - c_1 c_2 c_3) \left| y_{true} - y_{pred} \right| \tag{4.2}$$

The conditions themselves can then be written using Equations (4.3), (4.4) and (4.5).

$$c_1(y_{true}, y_{pred}) = \left| \frac{1}{2} \left(sgn(y_{true}) + sgn(y_{pred}) \right) \right| \tag{4.3}$$

$$c_2(y_{true}, \delta) = \frac{1}{2} sgn(|y_{true}| - \delta) + \frac{1}{2} \tag{4.4}$$

$$c_3(y_{pred}, \delta) = \frac{1}{2} sgn(|y_{pred}| - \delta) + \frac{1}{2} \tag{4.5}$$

Such a function can already be calculated very easily when training a neural network, however, the problem is that this function is neither differentiable nor continuous, which could cause problems in optimizing the error function.

However, we can relatively easily modify this function by using a suitable blending function $b(x)$ instead of the signum function. This function, like the signum function, should be odd, for values of the argument going to negative infinity it should take the values -1 (see Eq. (4.6)) and for values going to positive infinity it should take the values 1 (see Eq. (4.7)). This function should of course be continuous.

$$\lim_{x \to -\infty} b(x) = -1 \tag{4.6}$$

$$\lim_{x \to +\infty} b(x) = 1 \tag{4.7}$$

The hyperbolic tangent function satisfies these conditions. Therefore, we replace the signum function with the function shown in Eq. (4.8), using the s parameter, which affects the slope of the function.

$$b(x) = tanh(s \cdot x) \tag{4.8}$$

The resulting condition terms are then given in the form shown in Equations (4.9), (4.10) and (4.11). The shape of this function for one sample can then be seen in Fig. 4.2 and Fig. 4.3, where the levels of this function are shown.

$$c_1(y_{true}, y_{pred}) = \left|\frac{1}{2}\left(tanh(s \cdot y_{true}) + tanh(s \cdot y_{pred})\right)\right| \tag{4.9}$$

$$c_2(y_{true}, \delta) = \frac{1}{2}tanh(s \cdot |y_{true}| - \delta) + \frac{1}{2} \tag{4.10}$$

$$c_3(y_{pred}, \delta) = \frac{1}{2}tanh(s \cdot |y_{pred}| - \delta) + \frac{1}{2} \tag{4.11}$$

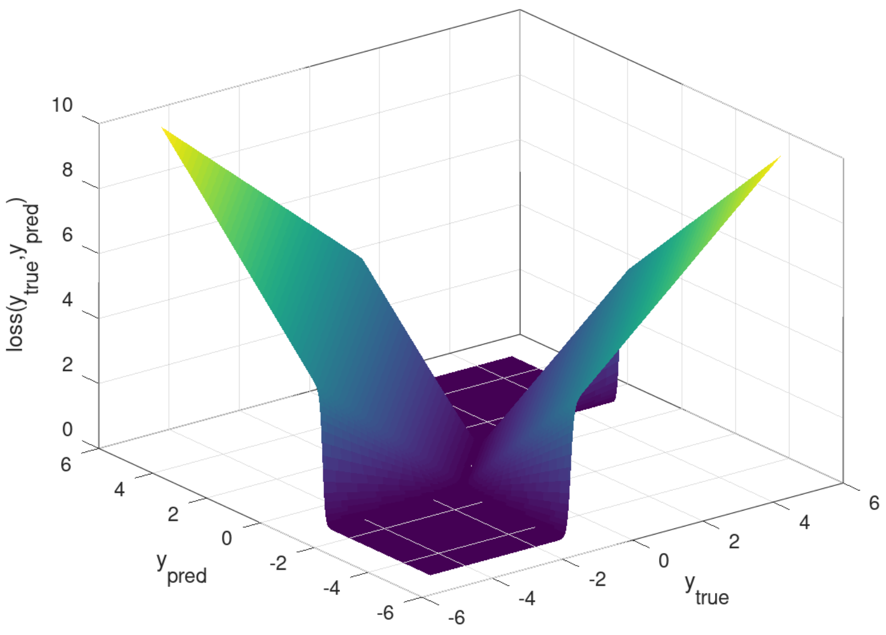

Figure 4.2: Shape of proposed surface sensitive function.

4.3 Weights compression

By using neural data alone, it is possible to achieve a compact representation of the dynamic surface compared to a sequence of triangle meshes. Furthermore, we can consider whether it would be possible to further compress the trained neural network and thus achieve even greater data savings.

Consider a sequence of triangle meshes of n frames. Each frame is formed by a mesh described by a set of vertices $\mathcal{V}_i$ and a set of triangle faces $\mathcal{F}_i$. If we store these meshes using an indexed face list, where we use 32-bit float point numbers to describe the coordinates of individual vertices and use a

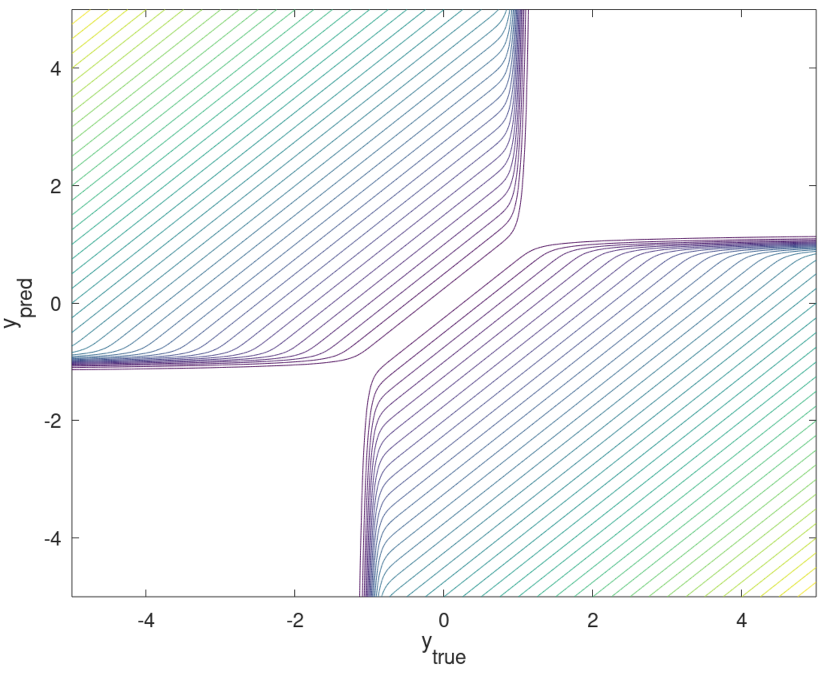

Figure 4.3: Levels of proposed surface sensitive function.

32-bit integers to store vertex indices, we can calculate the total size of this sequence using an Eq. (4.12).

$$\text{size} = \sum_{i=1}^{n} 3 \cdot 32 \cdot |\mathcal{V}_i| + 3 \cdot 32 \cdot |\mathcal{F}_i| \tag{4.12}$$

In the case where we convert the dynamic surface to a neural representation, we need to store the neural network weights and the constants used in normalization. If we store all sequences using a model with the same hyperparameters, we can consider other network settings as implicit knowledge and we do not have to store it. The normalization constants are then significantly smaller than the weights of the neural network, so we can neglect them when calculating the compression. For example, a neural network with 16 hidden layers with 265 neurons and one output layer with one neuron then contains 988,417 weights. These weights can then be stored as 32-bit floating point numbers.

In the case of triangle meshes compression, quantization is often used. Quantization is a process in which real numbers represented by floating point numbers are divided into a number of quantization levels, which rounds the values. These values are then represented by integers, the magnitude of which depends on the number of quantization levels used. When quantizing a generally arbitrary signal, due to the rounding of the value we distort the original signal. This distortion is called quantization noise. Thus, signal quantization represents a lossy compression. In the experiments presented in the Chapter 8, it is investigated whether it is possible to use quantization for the compression of neural network weights and what impact this quantization has on the resulting dynamic surface. Furthermore, the possibilities of using a lossless compression algorithm for further compression of already quantized data are investigated within these experiments. For this purpose, compression by the ZIP algorithm was chosen.

When quantizing, we start from how many bits we want to represent the resulting weights. If we want to compress the original 32-bit weights w_i into k-bit numbers using quantization, we get 2^k quantization layers. Then we can compute size of quantization layers q using Eq. (4.13). The quantization itself is then performed using the Eq. (4.14).

$$\mathrm{q} = \frac{max_i(w_i) - min_i(w_i)}{2^k} \tag{4.13}$$

$$\hat{w}_i = round(\frac{w_i - min_j(w_j)}{q}) \tag{4.14}$$

5 Data preprocessing

This chapter deals with surface sampling and calculation of the signed distance function and data preprocessing methods, which were used in the experiments. It is mainly the normalization of inputs and the transformation of features into feature spaces of higher dimension.

5.1 Triangle mesh sampling

As part of training neural networks for surface representation, it is necessary to first sample the original surface. It is therefore necessary to generate the positions of points in space and calculate the value of the signed distance function at these points with respect to the given surface. Specifically, in our case, we will deal with sampling of triangle meshes.

When using the naïve approach for triangle mesh sampling, we have a relatively simple algorithm. First, we generate the coordinates of the point x as a random vector in the bounding box of a triangle mesh extended by a sufficiently large edge in each direction. Next, at each of these points, we calculate the distance from all the triangle faces that make up the triangle mesh and keep the minimum. Then we have to determine on which side our generated point with respect to the surface, in order to determine whether the resulting signed distance should have a positive or a negative sign. If we are working with a closed surface, we only need to go through all the triangle faces again and calculate whether the number of intersections with a ray leading from the point x is even or odd. This approach gives correct results, however, its computational complexity is too great and triangle mesh sampling would take too long. In addition, the generation of samples from the uniform distribution given by the bounding box mesh is usually not very efficient due to the fact that when extracting the iso-surface of the signed distance approximation function when rendering the surface, we need high accuracy close to the surface.

In order to achieve a high accuracy of the approximation close to the surface, we need to generate more samples there and with increasing distance from the surface we want the number of samples to decrease. This is achieved by generating samples from the even distribution given by the bounding box mesh, as in the naive approach, however, immediately after generating the sample and calculating the distance form the surface, we calculate the probability whether this point should be used or not. The probability of preserving the sample is calculated based on the distance of the point from the surface $d(\mathbf{x}, \mathcal{S})$ and the coefficient k according to the formula (5.1). This formula describes a radial function that determines how the probability of maintaining a sample with increasing distance from the surface will decrease. The shape of the function is shown in the figure 5.1.

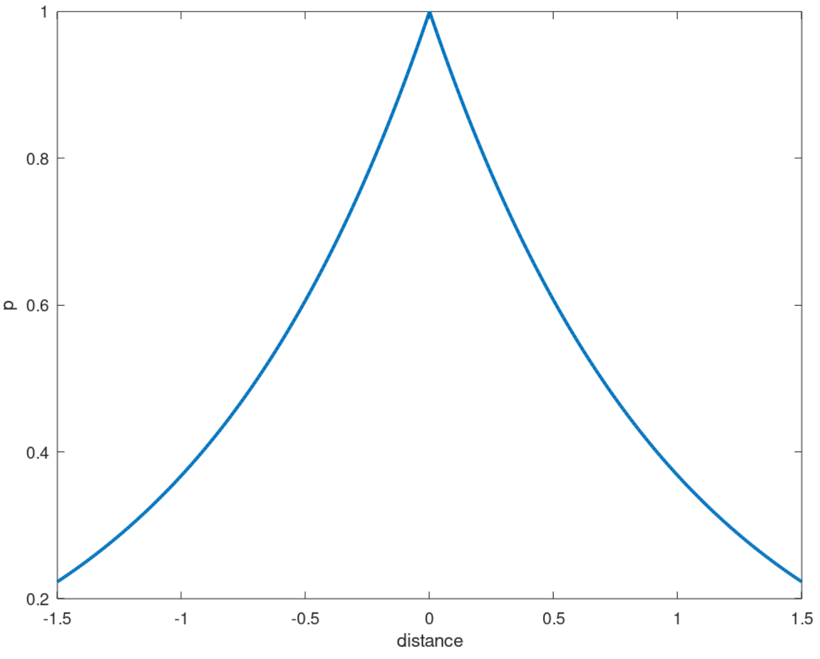

Figure 5.1: Example of sample preservation probability function for $k = 1$.

$$p = exp(-k \cdot d(\mathbf{x}, \mathcal{S})) \tag{5.1}$$

For an easier setting of the parameter k, we express k depending on the distance $d_{0.5}$ in which we want the probability of receiving the sample to be equal to the value 0.5. (5.2)

$$k = -\frac{ln(0.5)}{d_{0.5}} \tag{5.2}$$

To speed up the calculation of the distance from the surface, a structure based on octree bounding volumes hierarchy of triangle faces is used, which allows us to avoid testing all triangle faces. Before sampling the triangle mesh, we create an octree, where each of its nodes contains a set of triangle faces and at the same time information about the bounding box of these faces. The construction of the octree takes place from top to bottom. At the beginning all, the triangle faces are placed in the root of the tree. Next, the bounding box aligned with the axes of these faces is calculated. This bounding box is divided into eight identical cells, and faces are assigned to these cells cells that it at least partially intersects. The bounding box of the cell is then enlarged to such dimensions that all the faces assigned to this cell are completely inside the cell. These eight cells form the children nodes of the root in the tree structure. This procedure is recursively repeated for all descendants until the number of triangle faces in the cell is zero or the maximum depth of the tree is exceeded.

Thanks to the tree created in this way, we can gradually update the upper estimate of the minimum distance of the point to the surface and thus avoid calculating the distance from triangle faces, for which we know that the distance to them cannot be lower, because the minimum distance to their bounding box is greater than the current upper estimate.

When calculating the minimum distance between a point in space and a triangle face, there can be three cases, which can be distinguished according to whether the nearest point on the triangle is at its vertex, on an edge, or inside the face. If we project a given point into the plane of the triangle, we can easily recognize these cases (see figure 5.2).

Depending on whether the nearest point on the triangle face lies at the vertex, at its edge, or inside the face, we use different methods to determine whether the point is inside or outside the surface, so we use these methods to determine the sign of the resulting SDF value.

If the nearest point $\mathbf{y}$ on the surface of the triangle mesh to the point $\mathbf{x}$ where the sampling takes place is located inside the triangle face, we can determine the sign based on the normal of the given face. Let $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ be the vertices of a given triangle face. The unit length normal $\mathbf{n}$ of this face can then be determined using a cross product, as you can see in the Equation (5.3).

$$\mathbf{n} = \frac{(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)}{\|(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)\|} \tag{5.3}$$

If the condition that all trianke faces have the same counter clock wise orientation is met, then this normal vector always points outwards from the surface. The sign of the SDF itself can then be determined using the dot product of the normal vector $\mathbf{n}$ and the vector from point $\mathbf{y}$ to point $\mathbf{x}$ (see Eq. (5.4)).

$$\text{sign} = -sgn(n \cdot (\mathbf{x} - \mathbf{y})) \tag{5.4}$$

If the nearest point $\mathbf{y}$ lies on the edge of triangle face $\mathbf{F}$, we can determine the sign of the SDF by whether the edge is convex or concave. We find the convexity of this edge on the basis of the dihedral angle, which is the angle between two incident faces. The opossite face $\mathbf{F}_{opposite}$ is thus obtained by using the EF query, this is a query that finds the faces incidend with give edge. In order to determine the sign, we do not need to know the exact size of the dihedral angle, but only whether it is greater or less than 180 degrees. This can be found, for example, by calculating the barycenter $\mathbf{b}$ of the opposite face $\mathbf{F}_{opposite}$ with vertices $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$ (see Eq. (5.5)) and substituting into the normal equation the plane given by the triangle face $\mathbf{F}$ (see Eq. (5.6)). Component d in the normal equation of the plane can be computed at point $\mathbf{y}$, since we know it lies in that plane (see Eq. (5.7)). If this result is positive, then the point $\mathbf{x}$ lies inside and the sign of the SDF will be positive. When the result is negative, then the point $\mathbf{x}$ lies outside the given surface and the sign of the SDF will be negative.

$$\mathbf{b} = \frac{\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3}{3} \tag{5.5}$$

$$n_x b_x + n_y b_y + n_z b_z + d \tag{5.6}$$

$$d = -n_x y_x - n_y y_y - n_z y_z \tag{5.7}$$

In the last case, if the point $\mathbf{y}$ lies in the vertex of the triangle face $\mathbf{F}$, we can estimate the vertex normal direction and use dot product to determine the sign of the SDF. Since here we estimate the normal at a vertex, this approach may not be 100% accurate. However, in the course of the experiments performed, it turned out that the number of cases in which the sign of SDF is incorrectly determined in this way is negligible. We can compute mean curvature vector of given vertex using the dictrete Laplace operator. Specifically, cotan Laplacian is used in this work [MDSB01]. For an overview of the sampling algorithm see Algorithm 1.

5.2 Data normalization

In the experiments using a neural network to represent dynamic surfaces, it has turned out that normalization of the data during the experiment stages is extremely important in terms of the quality of the resulting representation. Therefore, the whole process of creating a neural representation includes three stages of normalization, namely normalization of the geometry of the input sequence of triangle meshes, normalization of the SDF values, and normalization of time.

Algorithm 1: Triangle mesh SDF sampling algorithm.

```
In: Mesh sequence SEQ[]
Result: SDF samples SDF[]
for $\mathcal{M}$ in SEQ do
    aabb := BoundingBox($\mathcal{M}$)
    oct := Octree($\mathcal{M}$)
    for i=1, ..., samples per frame do
        $\mathbf{x}$ := generatePointInBoundingBox(aabb)
        $\mathbf{y}$, face, edge, case := oct.nearestPoint($\mathbf{x}$)
        dist := d($\mathbf{y}$, $\mathbf{x}$)
        probability := p(dist)
        if probability > random(0,1) then
            continue
        end
        if case == Vertex then
            $\mathbf{c}$ := meanCurvatureNormal($\mathbf{y}$)
            inside := dot($\mathbf{c}$, face.normal) > 0
        end
        if case == Edge then
            neighbor := findOppositeFace($\mathcal{M}$, face, edge)
            $\mathbf{b}$ := barycenter(neighbor.face)
            inside := pointIsAbovePlane($\mathbf{b}$, face)
        end
        if case == Face then
            inside := !pointIsAbovePlate($\mathbf{y}$, face)
        end
        if inside then
            dist := -dist
        end
        SEQ.add($\mathbf{x}$, i, dist)
    end
end
```

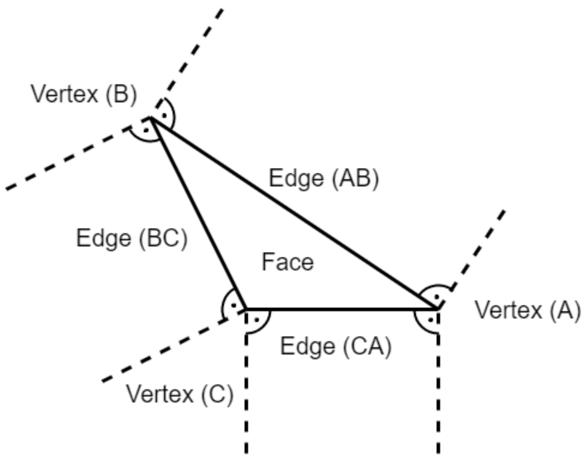

Figure 5.2: Visualisation of areas corresponding to their nearest point.

5.2.1 Mesh normalization

The first level of normalization takes place after loading the input sequence of triangle meshes. A bounding box is constructed from the loaded sequence as a set of values $x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$ that are calculated as the minima and maxima of the coordinates of the individual vertices of the triangle meshes across the entire sequence. From this bounding box, the length d of its diagonal is calculated using Eq. (5.8).

$$d = \sqrt{(x_{max} - x_{min}^2 + (y_{max} - y_{min})^2 + (z_{max} - z_{min})^2} \tag{5.8}$$

The goal of this normalization is to center the entire sequence of triangle meshes and to proportionaly scale it according to the size of the bounding box diagonal. If we centered each mesh separately, we would create discontinuities with respect to the time variable of the dynamic surface and it would therefore not be possible to represent it with a continuous function. The position of each vertex in the sequence of triangle meshes is normalized using the Eq. (5.9), normalization of y and z coordinates is the equivalent.

$$x_{norm} = \frac{x - x_{min}}{d} - \frac{x_{max} - x_{min}}{2d} \tag{5.9}$$

5.2.2 SDF normalization

After sampling the input sequence of triangle meshes, we get a set of 5-component vectors $(x, y, z, t, sdf(x, z, y, t))$ that contain the SDF value for a specific point at a specific time. These SDF values are further normalized to the interval $[-1, 1]$ using Eq. (5.10).

$$sdf_{norm} = 2\frac{sdf - sdf_{min}}{sdf_{max} - sdf_{min}} - 1 \tag{5.10}$$

5.2.3 Time normalization

The last step of the normalization is the normalization of time. In the case of time normalization, two different approaches were used in the experiments, normalization of time values to the interval $[0, 1]$, which is suitable in the case of using Fourier features mapping, and an approach based on estimating the speed of movement of a given surface. Assume that the time values of the individual frames of the processed sequence with m frames before normalization correspond to the integer indices of the given frames, so time t takes values $0, 1, ..., m - 1$.

In the case of normalization to the interval $[0, 1]$, the calculation is simple, just divide the time component by the number of frames minus one (see Eq. (5.11)).

$$t_{norm} = \frac{t}{m - 1} \tag{5.11}$$

The second way to normalize time values is based on the idea that if two different surfaces perform the same motion at different speed, then the time values of the faster surface should be normalized to a smaller interval. Although it is obvious that in reality the velocity of movement of the surface depends on the position of a given point and also on a given time, since the surface can move at different speeds at different times and at different positions, we can estimate its maximum, which we then use to normalize time values.

We estimate this maximum velocity based on the SDF values, as the largest SDF difference at one point in two consecutive frames. Altought the change in SDF at a given point will generally not correspond to the actual change in position of the point on the surface, it will serve well enough for estimation purposes.

Another problem is that since sampling of triangle meshes is performed at points with random coordinates, we will not have information about the SDF value in two consecutive frames for the same point. However, we can subtract the SDF value for point $\mathbf{x}$ from i-th frame and point $\mathbf{x}'$ from $i+1$-th frame, which is closest from all points of this frame to point $\mathbf{x}$. As the number of samples per image increases, the error caused by replacing point $\mathbf{x}$ with point $\mathbf{x}'$ will decrease to zero. A KD tree is used to effectively find the nearest point in the following frame. So we finally estimate the maximum velocity estimate with Eq. (5.12) and then we can use Eq. (5.13) to normalize time values.

$$v_{max} = max_t(max_{\mathbf{x}}(|sdf(\mathbf{x}, t) - sdf(\mathbf{x}', t+1)|)) \tag{5.12}$$

$$t_{norm} = \frac{t}{v_{max}(m-1)} \tag{5.13}$$

5.3 Fourier features

To achieve better results when using neural networks for the representation of time-varying surfaces, the use of so-called Fourier feature mapping was tested. Fourier feature mapping is a technique used to improve the results of neural networks, usually multi-layer perceptron (MLP), in cases where the size of the feature space is low. This technique provides interesting results in 2D image regression, 3D surfaces, MRI reconstruction and other areas [TSM*20a].

The Fourier feature mapping method is based on a transformation of the original feature vector into a higher dimensional space using trigonometric sine and cosine functions of different frequencies, which allows the neural network to respond better to an input signal when it consists of a mixture of several non-negligible signals of different frequencies. To calculate Fourier features $\gamma(\mathbf{v})$ from original features vector $\mathbf{v}$, we need to generate a random Gaussian matrix $\mathbf{B}$, where each entry is drawn independently of the normal distribution $N(0, \sigma^2)$. The transformation of the feature vector is then given by the formula (5.14).

$$\gamma(\mathbf{v}) = [cos(2\pi\mathbf{B}\mathbf{v}), sin(2\pi\mathbf{B}\mathbf{v})] \tag{5.14}$$

A great benefit of using Fourier features mapping can be expected in the case when the dimension of the input vector of the neural network is low. Coordinate-based multi layer perceptrons are therefore ideal cases for use of Fourier features, because their dimension of the feature space is typically low. I.e. for images, we have only two coordinates, for static surfaces, videos and generally all static scalar fields there are three coordinates, for dynamic surfaces and other time-varying scalar fields we have four coordinates.

Another possibility that can lead to an even better result when using Fourier features mapping is not to generate the coefficients of the matrix $\mathbf{B}$ randomly, but like other parameters of the neural network, to train them using the same optimization algorithm. For the purposes of this work, this method was not used.

6 Surfaces comparison and error measurement

In order to be able to evaluate the quality of the representation with respect to the original surface, it is necessary to introduce some metrics that will allow us to compare surfaces. In our case, we will compare surfaces represented by triangle meshes, however, each of these meshes will have a different connectivity.

A commonly used metric for surface comparison is the Hausdorff distance. The problem with the Hausdorff distance, however, is that its value typically depends on one single point on a given surface and therefore often does not provide results with a sufficiently large telling value about how the resulting surface visually resembles the original surface.

A commonly used metric for surface comparison is the Hausdorff distance. The problem with the Hausdorff distance, however, is that its value typically depends on one single point on a given surface and therefore often does not provide sufficiently correlated results with a visual distortion of resulting surface.

For this reason, the so-called perceptual metrics are constructed, the aim of which is to better describe the difference between two surfaces so that their result better corresponds with the humen visual perception. One such metric is the Fast Mesh Perceptual Distance.

6.1 Hausdorff distance

Hausdorff distance is one of the simplest methods for measuring the error when comparing surfaces represented by triangle meshes. Hausdorff distance can compare generally any surface using different representations and can therefore be used to compare triangle meshes even if they do not have shared connectivity. First, define the distance of the point $\boldsymbol{p}$ from the surface $\mathcal{S}$ as the minimum distance from any point lying on the surface $\mathcal{S}$ with equation. (6.1)

$$d(\boldsymbol{p}, \mathcal{S}) = \min_{\boldsymbol{p}' \in \mathcal{S}} \|p - p'\| \tag{6.1}$$

Subsequently, we can establish a relation for calculating the Hausdorff distance from the surface $\mathcal{S}$ to the surface $\mathcal{S}'$ with equation (6.2) as the maximum distance of any point $p \in \mathcal{S}$ from the surface $\mathcal{S}'$.

$$d(\mathcal{S}, \mathcal{S}') = \max_{p \in \mathcal{S}} d(\boldsymbol{p}, \mathcal{S}') \tag{6.2}$$

However, since this distance from the surface $\mathcal{S}$ to the surface $\mathcal{S}'$ is not generally the same as the distance from the surface $\mathcal{S}'$ to the surface $\mathcal{S}$, as shown in Figure 6.1, Hausdorff distance d_S between surfaces $\mathcal{S}$ and $\mathcal{S}'$ is defined as their maximum, see equation (6.3) [AScE02].

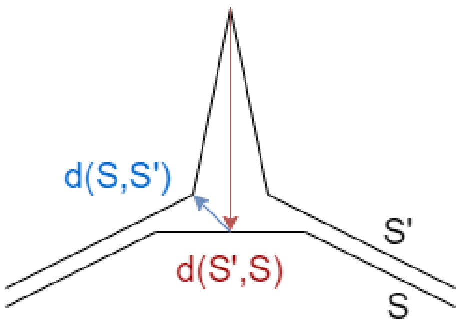

Figure 6.1: Hausforff distance visualisation in 2D.

$$d_S(\mathcal{S}, \mathcal{S}') = \max[d(\mathcal{S}, \mathcal{S}'), d(\mathcal{S}', \mathcal{S})] \tag{6.3}$$

Since we do not compare static surfaces, but time-varying surfaces, represented by a series of T static images, similarly to the comparison of surface contents, we use the maximum (6.4) and average (6.5) Hausdorff distance.

$$d_{S-max}(\mathcal{S}(t), \mathcal{S}'(t)) = \max_{t \in 1,\dots,T}(d_S(\mathcal{S}(t), \mathcal{S}'(t))) \tag{6.4}$$

$$d_{S-avg}(\mathcal{S}(t), \mathcal{S}'(t)) = \frac{1}{T}\sum_{t=1}^{T}(d_S(\mathcal{S}(t), \mathcal{S}'(t))) \tag{6.5}$$

6.2 Fast mesh perceptual distance

Fast mesh perceptual distance (FMPD) is a perceptual metric for comparing two triangle meshes proposed by Kai Wang [WTM12]. The advantage of FMPD is that it can compare meshes with different connectivity, its calculation is very fast compared to the Haudorff distance, for example, and offers very good results in terms of correlation of the metric values with subjective evaluation of surface distortion by humans.

The FMPD metric is based on a local mesh roughness measure. When calculating the FMPD, the local roughness in all vertices of both meshes is

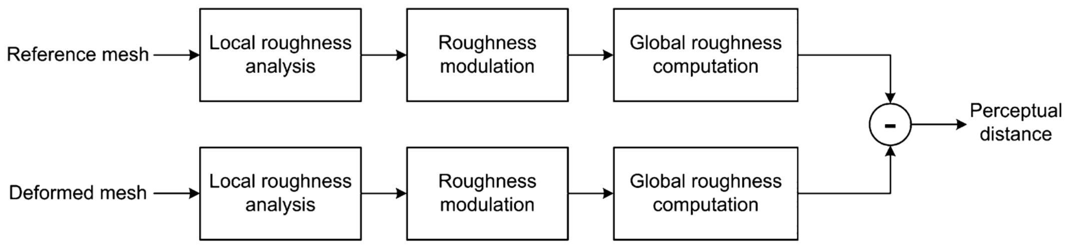

Figure 6.2: Block diagram illustrating the pipeline of the FMPD [WTM12].

determined first. Local roughness values are derived from Gaussian curvature G. For local roughness computation we also need to compute the discrete Cotan Laplacian matrix [MDSB01]. This matrix is computed according to formula shown in Eq. (6.6), where $\mathcal{N}_i^{(v)}$ is the set of all the neighboring vertices of v_i and α and β are the angles inside the triangle face opposite the edge between vertices v_i and v_j (see Figure 6.3).

$$D_{i,j} = \begin{cases} \frac{cot(\alpha_{i,j})+cot(\beta_{i,j})}{2} & \text{for } j \in \mathcal{N}_i^{(v)}, \\ -\sum_j D_{i,j} \end{cases} \tag{6.6}$$

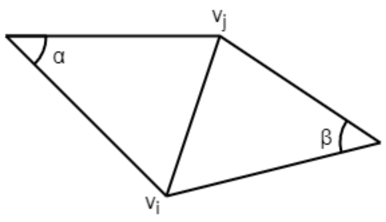

Figure 6.3: Angles used in the calculation of the discrete Laplacian matrix.

Local roughness LR_i in vertex v_i is then computed by the Eq. (6.7).

$$LR_i = \left| G_i - \frac{\sum_{j\in\mathcal{N}_i^{(v)}} D_{i,j}G_j}{\sum_{j\in\mathcal{N}_i^{(v)}} D_{i,j}} \right| = \left| G_i + \frac{\sum_{j\in\mathcal{N}_i^{(v)}} D_{i,j}G_j}{D_{i,i}} \right| \tag{6.7}$$

The local roughness comptutation is thresholded in the interval $[Th_l, Th_h]$ and modulated using power function shown in the Eq. (6.8) where a is a parameter that controls the shape of the power function and is fixed for all meshes.

$$LRM_i = f(LR_i) = (LR_i)^a - (Th_l)^a \tag{6.8}$$

After that, the average values of local roughness $\overline{LR}$ for the original mesh and $\overline{LR'}$ for the deformed mesh are computed using formula shown in Eq.

(6.9) where s_i is one third of the total area of the incident facets of v_i. These average values are then modulated with same modulation function (see Eq. (6.8)) and we get modulated average values $\overline{LRM}$ and $\overline{LRM'}$.

$$\overline{LR} = \frac{\sum_i LR_i s_i}{\sum_i s_i} \tag{6.9}$$

As a final step of the local roughness computation, we perform second modulation using Eq. (6.10). Threshold Th_{LRM} is obtained as minimum from $\overline{LRM}$ and $\overline{LRM'}$ values. The value $0 < b < 1$ is a parameter that controls the magnitude of the reduction.

$$LRF_i = Th_{LRM} + b(LRM_i - Th_{LRM}) \text{ for } LRM_i > Th_{LRM} \tag{6.10}$$

Once we have the final local roughness values LRF_i, global roughness is computed by Eq. (6.11) and after that we can compute FMPD by Eq. (6.12) where c is scaling factor that brings the perceptual distance into the $[0, 1]$ interval. The block diagram of the FMPD calculation is shown in figure 6.2.

$$GR = \frac{\sum_i LRF_i s_i}{\sum_i s_i} \tag{6.11}$$

$$FMPD = c\,|GR - GR'| \tag{6.12}$$

7 Implementation

The implementation part of this work is realized as a framework enabling easy testing of neural implicit dynamic surface representation using different data, different parameters, different methods of normalization, etc. The main goal of this application is to allow the execution of experiments with different hyperparameters and the subsequent evaluation of the results.

This framework is implemented in .NET environment using C# language and works exclusively as a Windows console application without a graphical user interface, because due to the nature of the application, which is to serve only as experimental software to perform the necessary experiments, a GUI is not required.

To work with neural networks, the application uses the Keras [C*15] library, which is available in the .NET environment using the Keras.NET package. The application also contains the Hausdorff distance evaluator based on metro.exe software [CRS98]. The framework also includes an FMPD evaluator, which uses the original implementation provided by Kai Wang [WTM12]. The framework also implements a mechanism that allows running Octave and Python scripts. The source code contains full programmer documentation directly in the source files.

7.1 Framwework use case

The implemented framework actually contains a single use case, which is used to run an experiment to test the neural representation of a dynamic surface with selected hyperparameters and input data. The application does not contain any tools for displaying processed surfaces. Instead, it saves them in the wavefront OBJ, which can be opened in most software for working with 3D models.

How to start the application is described in the user manual, which is part of the appendix. However, before the desired experiment can be started, the framework must be configured correctly (see Section 7.3) and the appropriate experiment definition (see Section 7.4) must be prepared.

After running the experiment, the framework tests all possible combinations of provided implementations of individual components such as SDF samplers, normalization algorithms, neural representations, triangle meshes comparison metrics, etc.

First, the input sequence of triangle meshes is loaded. After loading, the geometry of the given sequence is normalized. Next, the sequence is sampled into SDF samples. After sampling the triangle mesh sequence, the SDF function values and time values of these samples are normalized. Next, a neural representation of the given dynamic surface is created and trained. Subsequently, the created neural representation is used to reconstruct the surface by predicting the approximated SDF on a volumetric lattice in the bounding box of the original sequence of triangle meshes. The weights of the trained neural network are compressed and decompressed again, so that in the case of lossy compression, its impact on the resulting prediction is reflected. The corresponding iso-surface is then extracted from this volumetric lattice again as a sequence of triangle meshes. The resulting sequence is then compared to the original sequence using selected metrics. Finally, the results recording the values of the individual metrics for the tested configuration are saved in a CSV file. Along with these results, the weights of the neural network used, and the original and the resulting sequence of triangle meshes in the wavefront OBJ format are also stored in the output folder. A flow diagram of the whole process is shown in the Fig. 7.1

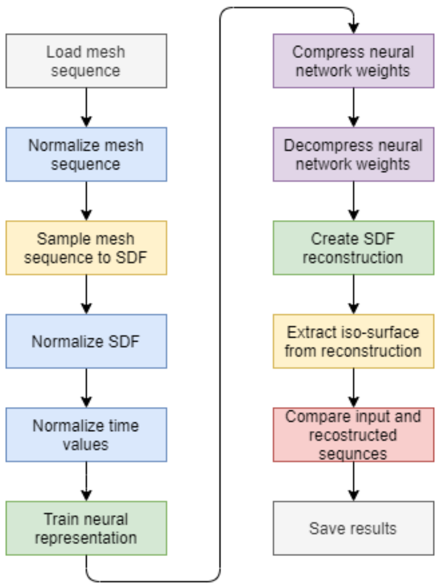
Load mesh sequence
Normalize mesh sequence
Sample mesh sequence to SDF
Normalize SDF
Normalize time values
Train neural representation
Compress neural network weights
Decompress neural network weights
Create SDF reconstruction
Extract iso-surface from reconstruction
Compare input and recostructed sequnces
Save results

Figure 7.1: Flow diagram of experiment process.

7.2 Project structure

The program is divided into the following modules:

- NeuralImplicits - The main project, which is used to run the entire application and contains most of the calculations. It is further structured into smaller folders.
- NeuralImplicits.OctaveProcessor - A project that contains an interface that allows running Octave scripts. This is used for FMPD evaluation.
- NeuralImplicits.PythonProcessor - A project that contains an interface that allows running Python scripts. This can be used if we want to make an adjustment to the neural network that is not supported by Keras.NET wrapper.
- NeuralImplicits.Tests - Contains several unit tests that were created during the development of the application.

The structure of the main NeuralImplicits project is further divided into following folders:

- Compression - Implementation of the neural network compression methods.
- DataSources - Triangle mesh data source usable in experiments.
- Dependencies - Contains a file with a list of project dependencies.
- Evaluators - Implementation of surface comparison metrics for triangle mesh sequences.
- Experiments - Definition of experiments and logic for experiment results export.
- Extensions - Extension classes.
- Geometry - Implementation of geometric calculations.
- IO - Working with files such as reading and saving triangle meshes into wavefront OBJ files, plotting charts, etc.
- Maths - Implementation of mathematic calculations.
- Neural - Contains an implementation of the proposed neural representation of dynamic surfaces.
- Normalizers - Mesh, SDF and time normalizers.
- Samplers - Triangle mesh samplers.

- Structures - Basic structures for working with triangle meshes, volume grids, vectors, etc.

- SurfaceExtraction - Surface extraction algorithms - marching cubes implementation.

7.3 Configuration

A runnable project *NeuralImplicits* contains an XML configuration file *App.config*, the contents of which must be set correctly before starting the application. It mainly deals with about setting the path to the folder with input data, setting the path to the Python and Octave executables and configuration of Serilog library, which is used for logging. This file contains following settings located inside *appSettings* section:

- DataDirectoryPath - Directory with input OBJ mesh sequences. Each sequence is placed further in a separate folder.

- Python - Path to python.exe file.

- Octave - Path to octave.bat file.

- serilog:minimum-level - Minimum level of logging messages.

- serilog:using:Console - Enables logging to the console.

- serilog:write-to:Console.outputTemplate - Console logging template.

- serilog:using:File - Enables logging to a file.

- serilog:write-to:File.path - File log path.

- serilog:write-to:File.outputTemplate - File logging template

Example of *appSettings* section is shown in Code 7.1.

Listing 7.1: Example of appSettings section of App.config file.

```
<appSettings>
  <add key="DataDirectoryPath" value="D:\neural-implicit
  \Data" />
  <add key="Python" value="C:\Python\python.exe" />
  <add key="Octave" value="D:\Octave-5.2.0\mingw64\bin
  \octave.bat" />
  <!-- Serilog -->
  <add key="serilog:minimum-level" value="Verbose" />
  <add key="serilog:using:Console" value="Serilog.Sinks
  .Console" />
  <add key="serilog:write-to:Console.outputTemplate" value=
  "[{Timestamp:HH:mm:ss} {Level}] {Message:lj}{NewLine}
  {Exception}" />
  <add key="serilog:using:File" value="Serilog.Sinks.File" />
  <add key="serilog:write-to:File.path" value="log.txt" />
  <add key="serilog:write-to:File.outputTemplate" value="
  [{Timestamp:HH:mm:ss} {Level}] {Message:lj}{NewLine}
  {Exception}" />
</appSettings>
```

7.4 Experiments setup

The individual experiments are defined directly in the source code within the created framework. To create a new experiment, there must be created a class in the *NeuralImplicits* project, ideally in the *Experiments* folder. This class must implement the *IExperiment* interface, which forces the implementation of the properties that are required to perform the experiment. For array type properties, the experiment then tests all possible combinations of the provided instances. The individual properties are described in the list below:

- string Name - Name of the experiment.
- bool SwapFrames - Framework swaps frames of SDF sequence to hard drive in order to save memory.
- IAnimationDataSource[] DataSources - Array of input mesh sequence sources.

- IMeshSDFSampler[] Samplers - Array of triangle mesh to SDF samplers.
- IMeshNormalizer[] MeshNormalizers - Array of triangle mesh normalizers.
- ISDFNormalizer[] SDFNormalizers - Array of SDF value normalizers.
- Func<ISDFAnimationRepresentation>[] Representations - Array of functions, which produces instances of neural representations. To avoid the need to keep instances of all tested representations in memory for the duration of the experiment, the array does not directly contain the representations themselves but only the parameterless methods that return them.
- IWeightsCompression[] WeightsCompressions - Compression algorithm for neural network weights array.
- ISurfaceExtraction SurfaceExtraction - Surface extraction algorithm implementation.
- IEvaluator[] Evaluators - Array of triangle mesh comparison metrics.
- float MarginX - Margin of the triangle mesh bounding box on the x-axis.
- float MarginY - Margin of the triangle mesh bounding box on the y-axis.
- float MarginZ - Margin of the triangle mesh bounding box on the z-axis.
- int OutputResolution - Max resolution of volumetric grid used for surface reconstruction.

8 Experiment results

To verify the applicability of the neural representation of dynamic surfaces and at the same time to investigate their properties, several experiments were designed and implemented. These experiments were performed on two sequences of triangle meshes. Both of these datasets capture the animation of a moving human. First of these sequences is the *jump* dataset [SMP03], which contains 222 triangle mesh frames and each of these mesh frames has 15,860 vertices and 31,660 faces. Second sequence is the *samba* dataset [VBMP08], which contains 175 frames with 9,971 vertices and 19,938 faces. Both of theese triangle mesh sequences have constant connectivity, which means we know vertex correspondence between frames. However, this information was not used in the experiments and we could perform the same experiments on sequences with varying connectivity.

The quality of the resulting representations in the following experiments was measured using Hausdorff distance and FMPD. Since these metrics are used to compare static surfaces, the maximum values and average values of these metrics over the whole sequence were calculated when measuring the distortion of dynamic surfaces.

One of the things we can examine when creating a neural representation of a dynamic surface is how many samples of the SDF function we should generate for a given sequence of triangle meshes. It can be expected that this amount should naturally depend on how many frames the sequence contains. The longer the sequence, the more SDF samples will be needed to create the same quality neural representation. We can therefore examine the optimal number of samples per frame of the sequence. It can be expected that as the number of samples per frame increases, the quality of the resulting dynamic surface representation will increase. However, it is also possible to expect that as the number of samples per image increases, their contribution to the quality of the representation will decrease from a certain number of samples. An experiment measuring the dependence of the error of the resulting neural representation on the number of generated samples per frame was performed on both datasets using 16 inner layers of 256 neurons with basic configuration without the use of Fourier features mapping and with the MAE loss function. It is important to mention that the numbers of samples are measured as the numbers of generated samples, these samples are then created on the basis of the probability depending on the distance of a given point from the surface, which is described in Section 5.1. Figure 8.1 shows a graph of the dependence of the average FMPD on the number of generated frames per frame.

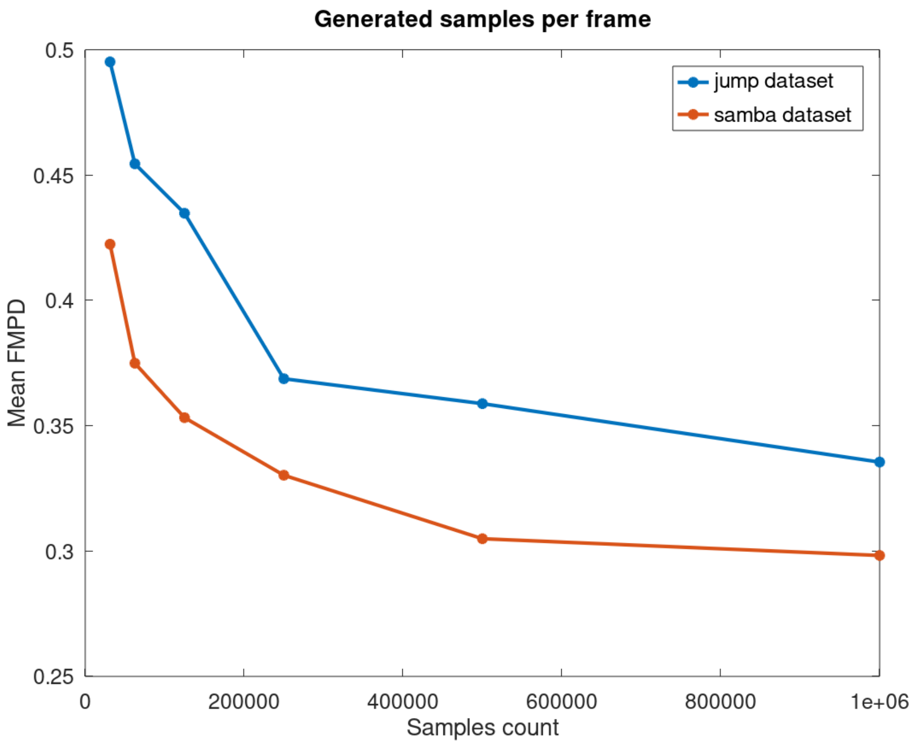

Figure 8.1: Dependence of the average FMPD on the number of sam-ples per frame.

Another performed experiment aims at investigating the relationship between the compression rate of the neural representation by quantizing the weights of the neural network and the resulting distortion of the represented surface. The second goal of this experiment is to determine whether it makes sense to compress the quantized weight further, e.g., using the ZIP compression algorithm. In Figure 8.2 you can see a graph of the dependence of the average FMPD on the compression ratio, which was achieved by quantizing the neural network weights using different numbers of quantization levels. The graph shows the results for jump dataset. You can see the same graph for the samba dataset in Figure 8.3.

From the above graphs, it is clear that for both datasets, it is possible to find such a compression ratio, which in terms of the perceptual metric used will be reflected only in negligible distortion of the resulting surface. Furthermore, it can be seen from these graphs that for different data, the results may be different in terms of FMPD using the same compression ratio. Furthermore, from these results, we can conclude that the application of ZIP compression on already quantized weights does not lead to a further

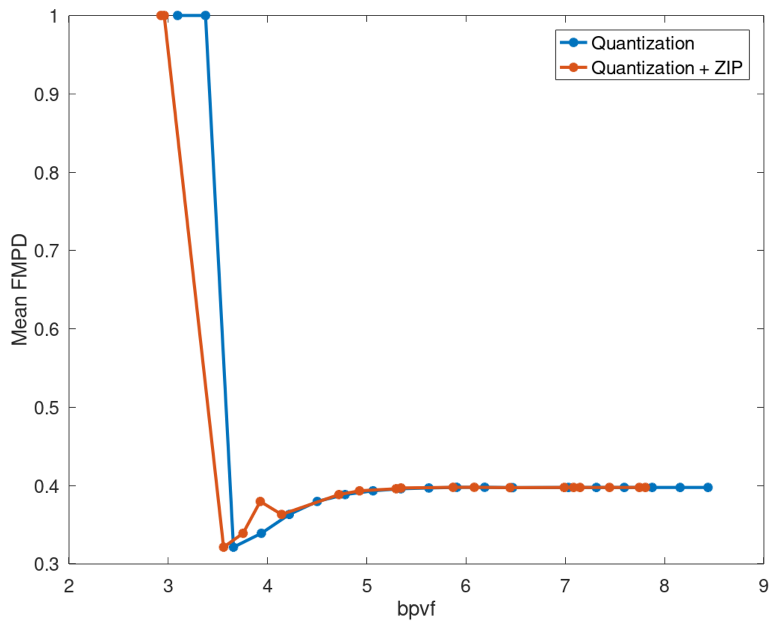

Figure 8.2: Dependence of the average FMPD on the compressionrate for the jump dataset.

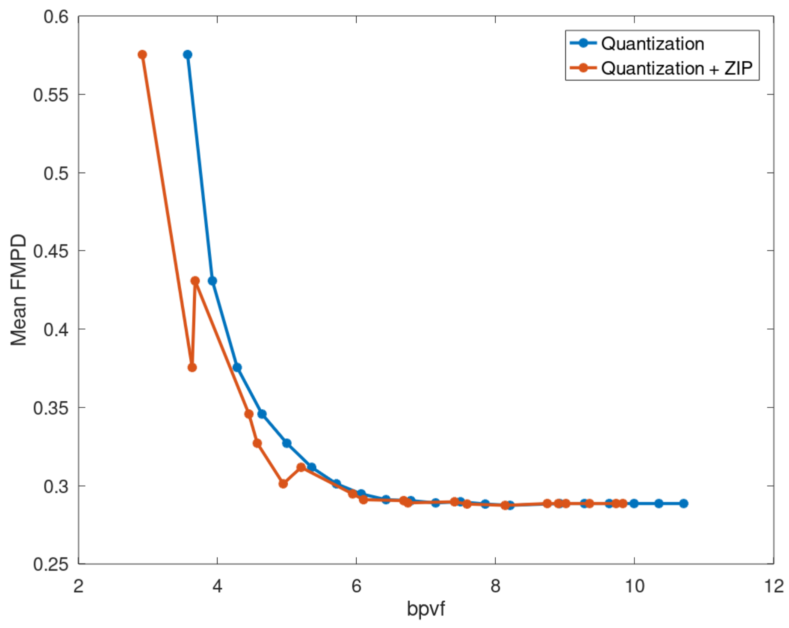

Figure 8.3: Dependence of the average FMPD on the compressionrate for the samba dataset.

Dataset	Sampling freq.	Max. Haus.	Avg. Haus.	Max. FMPD	Avg. FMPD
jump	$222 \cdot T^{-1}$	23.9540	8.3863	0.6271	0.3149
jump	$111 \cdot T^{-1}$	26.6664	7.9144	0.6709	0.3768
samba	$175 \cdot T^{-1}$	0.1027	0.0508	0.4243	0.2834
samba	$88 \cdot T^{-1}$	0.1481	0.0704	0.4271	0.2817

Table 8.1: Table with time super-resolution results.

significant improvement in the compression ratio. Even in some cases, it may lead to a small degradation of the compression ratio.

Another experiment is aimed at examining whether it is possible to exploit the fact that the neural representation of a dynamic surface is naturally continuous in the spatial axes and in time to perform super-resolution of the original surfaces. By interpolating the original frames of a sequence of triangle meshes in time, we can compute additional frames at times for which we do not have the original data. This could increase the frame rate of a given dynamic surface represented by a sequence of triangle meshes. In Table 8.1, you can see the results of using a neural representation for using two different frame rates to generate data on which neural network training is performed. In one case, the neural network is trained on data from all frames of the original sequence of triangle meshes, while in the other case, the neural network is trained only on data from every other frame. In both cases, a prediction is made for the full frame rate, so in the second case, a prediction is made even in frames that the neural network has not seen.

The results of this experiment show that using the half frame rate did not significantly degrade the resulting surface in terms of Hausdorff distance or FMPD, confirming the original hypothesis that the continuity of the neural representation over time can be used for super-resolution of the original sequence of triangle meshes.

In another experiment, a comparison of the use of the basic neural representation using the original feature space and the use of Fourier features mapping is performed. For each original feature, five Fourier features were calculated using randomly generated coefficients. The dimension of the feature vector is thus five times larger. Of course, this is also related to an increase in the number of weights in the input layer of the neural network. However, this increase is negligible compared to the total number of weights in other layers, and we can compare the Fourier feature mapping model with the original feature vectors. The coefficients for calculating the Fourier features were generated from a normal distribution with a variance $\sigma^2 = 3$. Furthermore, the same experiment contains an evaluation of the use of the original MAE loss function in comparison with the proposed surface-sensitive

Dataset	Loss func.	Fourier f.	Max. Haus.	Avg. Haus.	Max. FMPD	Avg. FMPD
jump	MAE	No	20.8320	7.3274	0.7364	0.4107
jump	MAE	Yes	9.8464	6.8367	0.7079	0.4013
jump	Surf. sensitive	No	23.8126	8.0914	0.6394	0.3274
jump	Surf. sensitive	Yes	23.9541	8.3862	0.6271	0.3149
samba	MAE	No	0.0850	0.0457	0.4943	0.3281
samba	MAE	Yes	0.1025	0.0523	0.4367	0.2864
samba	Surf. sensitive	No	0.0854	0.0455	0.4847	0.3147
samba	Surf. sensitive	Yes	0.1027	0.0508	0.4243	0.2834

Table 8.2: Table comparing results using Fourier features mapping and surface-sensitive loss function against results without them.

loss function. The results of this experiment are recorded in table 8.2.

From these results, we can see that the use of Fourier features mapping led to an improvement of the representation from the point of view of FMPD, while from the point of view of the Hausorff distance, the results deteriorated. We can assume that the results of Fourier features mapping could be further improved by generating the coefficients from a different normal distribution with a different variance for each feature. Another way to optimize the values of these coefficients is to include them among the variables whose value is optimized as well as the neural network weights in the learning process. However, the very observation of the inconsistency between Hausdorff distance and FMPD is interesting.

It is necessary to emphasize that in the search for optimal hyperparameters of the model of neural representation, emphasis was placed on these two metrics used. Since FMPD better describes surface distortion in terms of human perception, we can give more weight to its result. However, both of these metrics are primarily intended for comparing static surfaces, so it is possible that using the average and maximum values of these metrics across all images of a sequence of triangle meshes will not be the best possible solution. For this reason, it would certainly be worthwhile to address the issue of perceptualy comparing time-varying surfaces.

Furthermore, in the results shown in Table 8.2, we can notice that better results were obtained from the point of view of both metrics using the surface-sensitive loss function. The experiment is, of course, limited by the amount of data used. However, we can conclude that the use of this loss function can lead to improved results of the neural representation of the dynamic surface and therefore it makes sense to address the issue of choosing a suitable loss function.

The results of the best achieved neural representation compared to the original surface of the jump dataset can be seen in Figure 8.4. The same comparison for the samba dataset is shown in Figure 8.5.

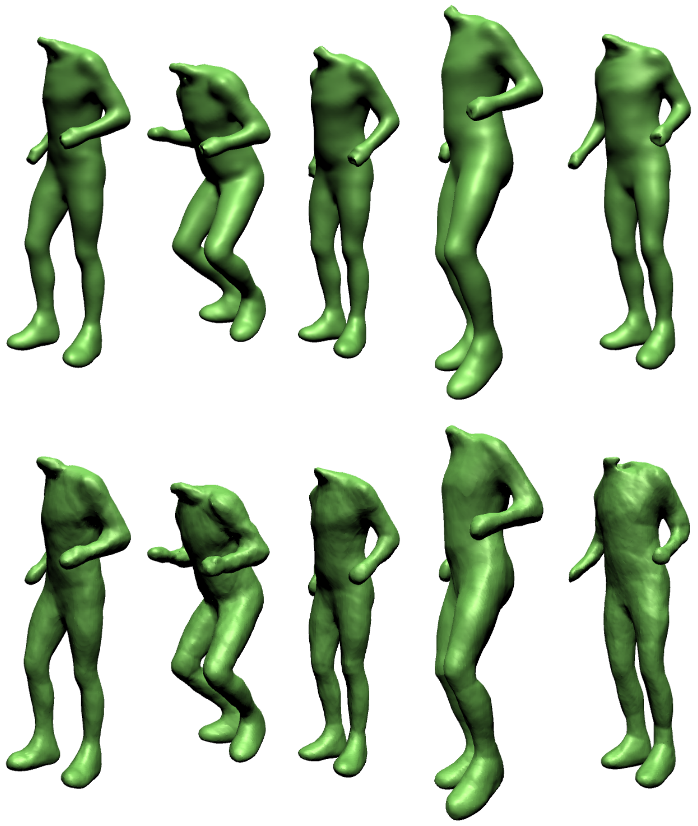

Figure 8.4: A comparison of the input surface and the results of neural representation on the jump dataset. Top row: original. Bottom row: neural representation.

From a visual examination of the created neural representations, it is evident that the reconstructed surfaces suffer from distortion in the form of surface undulations. This is one of the issues that should be addressed in future work. Suppression of this problem could possibly be achieved by modifying the loss function in such a way that, in addition to penalizing the error in the SDF prediction, it also penalizes the gradient error. This concept was proposed in the work Yuzhe Lu et al. [LJLB21], where the properties of neural representations of general scalar fields across various scientific applications were investigated. Another possible approach could be to force the neural network to maintain the curvature of the original surface.

The last experiment examines the use of a neural network for interpolation over time using different numbers of surrounding frames. The neural network was trained on the data of the previous and following frames, and subsequently the surface was reconstructed at the time of the center frame,

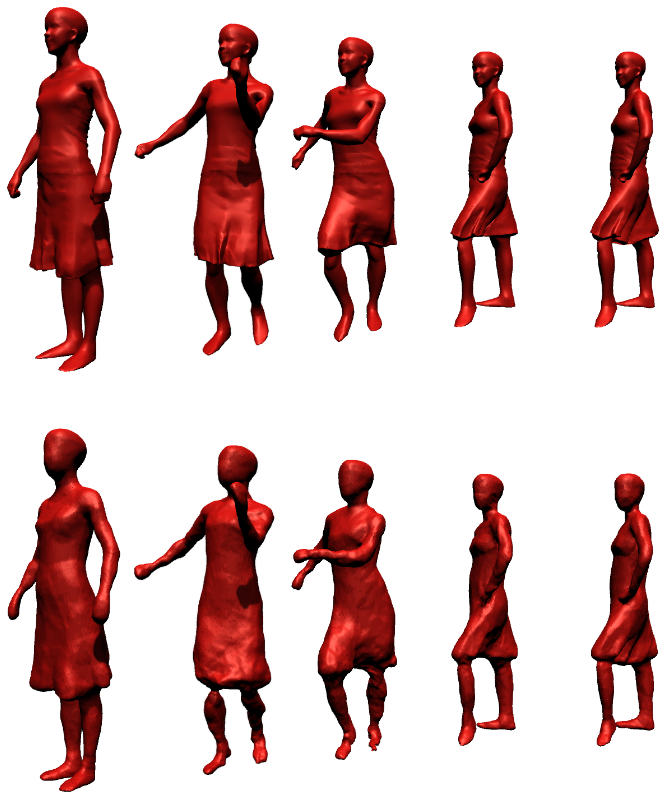

Figure 8.5: A comparison of the input surface and the results of neural representation on the samba dataset. Top row: original. Bottom row: neural representation.

where the Hausdorff distance and FMPD were evaluated. The same experiment was then repeated with the two preceding and two following frames and with the three preceding and three following frames. The results of these experiments were then compared. A comparison of the results according to the Hausdorff distance is described in the Table 8.3 and results according to the FMPD is described in the Table 8.4.

From the point of view of Hausdorff distance, it is evident that the best results were obtained using the largest number of frames. From the point of view of FMPD, the result is no longer so clear-cut, but even here, it seems that the biggest error is obtained for interpolation using data from one frame on each side and the best. However smallest error was achieved using two frames on each side. It should be also noted that in this experiment, representations using the wider neighborhood of the interpolated frame are partially disadvantaged by having to approximate the SDF over a longer time interval. Interpolating the sequence of triangle meshes with different

Dataset	Frame	Hausdorff (3)	Hausdorff (5)	Hausdorff (7)
jump	25	84,1548	9,7619	6,2016
jump	50	68,9221	7,5051	9,3070
jump	75	80,4351	10,3243	9,2510
jump	100	152,6963	12,4239	6,1047
jump	125	154,7011	15,5541	5,8171
samba	25	0,5726	0,3876	0,2333
samba	50	0,8947	0,4086	0,0661
samba	75	0,8817	0,4076	0,1044
samba	100	0,7757	0,2578	0,0777
samba	125	0,8915	0,1456	0,2466
axyz mobil	25	0,8314	0,1747	0,0943
axyz mobil	50	2,0986	0,3029	0,0877
axyz mobil	75	0,7720	0,1574	0,3248
axyz mobil	100	0,6219	0,1592	0,0899
axyz mobil	125	1,3644	0,2124	0,0936

Table 8.3: Table comparing the Haussdorf distance of the interpolated frames for different numbers of neighboring frames.

connectivity is a complicated problem. The advantage of this approach is that it uses data from the wider environment of the interpolated frame, unlike other approaches where interpolation is performed from only two adjacent frames.

Dataset	**Frame**	**FMPD (3)**	**FMPD (5)**	**FMPD (7)**
jump	25	0,5592	0,3155	0,3902
jump	50	0,3118	0,2651	0,3288
jump	75	1,0000	0,5709	0,6349
jump	100	0,7109	0,1701	0,1584
jump	125	1,0000	0,2960	0,3053
samba	25	0,5052	0,5158	0,4092
samba	50	1,0000	0,4648	0,2994
samba	75	1,0000	0,0032	0,1016
samba	100	0,0402	0,3496	0,1700
samba	125	0,1920	0,5577	0,4883
axyz mobil	25	0,6924	0,0648	0,2320
axyz mobil	50	0,6726	0,4929	0,5642
axyz mobil	75	0,4811	0,1874	0,2531
axyz mobil	100	0,3661	0,0977	0,1875
axyz mobil	125	1,0000	0,0802	0,2661

Table 8.4: Table comparing the FMPD of the interpolated frame for different numbers of neighboring frames.

9 Conclusion

This work investigated the area of static surface representation using neural networks. Based on the existing neural representations of static surfaces, a representation for dynamic surfaces was designed using neural networks. This proposed representation was tested by creating a framework implementing this method, including an algorithm for sampling the sequence of triangle meshes and metrics for evaluating the quality of the resulting representation. Several experiments were performed to verify the functionality of the proposed neural representation of dynamic surfaces using this framework. These experiments showed that the proposed representation could be used to represent dynamic surfaces. Furthermore, the properties of this representation were investigated. Experiments showed that the neural representation can serve as a compact representation of dynamic surfaces due to the high compression ratio. Another useful feature of the proposed neural representation is the fact that it creates a naturally continuous surface, even in the time axis. This feature can be used for super-resolution of the original surface by resampling the neural representation with a higher frame rate. The experiments also showed the benefits of using the Fourier features mapping acceleration method.

It was further demonstrated that when using a neural network to represent a dynamic surface, it is important to careful choose the loss function, which is optimized during the neural network. In this context, a surface-sensitive loss function was proposed, which tries to better describe the error of the neural representation from the point of view of distortion of the resulting surface. In the performed experiments, it was possible to achieve better results with the use of this proposed loss function, instead of the MAE loss function.

Given the limited accuracy of the representation of surfaces, which was achieved in the experiments in this work, it is possible to further address the possibilities for improving this representation. In this regard, it would be useful to examine the benefits of including a gradient error penalty member in the loss function. Experiments have also shown that it is very difficult to assess the quality of the proposed representation due to the absence of appropriate metrics for comparing dynamic surfaces.

The results achieved in this work prove to be relevant with respect to the current state of the art in the field of representation of dynamic surfaces. Therefore, the results of this work were also summarized in a paper and submited to the conference Symposium on Geometry Processing 2021.

Bibliography

[AL20] Atzmon M., Lipman Y.: Sal: Sign agnostic learning of shapes from raw data, 2020. `arXiv:1911.10414`.

[AScE02] Aspert N., Santa-cruz D., Ebrahimi T.: Mesh: Measuring errors between surfaces using the hausdorff distance. *Proceedings of the IEEE International Conference in Multimedia and Expo (ICME) 1* (02 2002), 705 – 708 vol.1. `doi:10.1109/ICME.2002.1035879`.

[BKP*10] Botsch M., Kobbelt L., Pauly M., Alliez P., Levy B.: *Polygon Mesh Processing.* CRC Press, 2010.

[C*15] Chollet F., et al.: Keras. `https://keras.io`, 2015.

[CRS98] Cignoni P., Rocchini C., Scopigno R.: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17* (06 1998), 167 – 174. `doi:10.1111/1467-8659.00236`.

[DNJ21] Davies T., Nowrouzezahrai D., Jacobson A.: On the effectiveness of weight-encoded neural implicit 3d shapes, 2021. `arXiv:2009.09808`.

[KB17] Kingma D. P., Ba J.: Adam: A method for stochastic optimization, 2017. `arXiv:1412.6980`.

[LC87] Lorensen W. E., Cline H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 163–169. `doi:10.1145/37402.37422`.

[LJLB21] Lu Y., Jiang K., Levine J. A., Berger M.: Compressive neural representations of volumetric scalar fields, 2021. `arXiv:2104.04523`.

[MDSB01] Meyer M., Desbrun M., Schr P., Barr A.: Discrete differential-geometry operators for triangulated 2-manifolds. *Proceedings of Visualization and Mathematics 3* (11 2001). `doi:10.1007/978-3-662-05105-4_2`.

[MON*19] Mescheder L., Oechsle M., Niemeyer M., Nowozin S., Geiger A.: Occupancy networks: Learning 3d reconstruction in function space, 2019. `arXiv:1812.03828`.

[MP88] McCulloch W. S., Pitts W.: *A Logical Calculus of the Ideas Immanent in Nervous Activity.* MIT Press, Cambridge, MA, USA, 1988, p. 15–27.

[PFS*19] Park J. J., Florence P., Straub J., Newcombe R., Lovegrove S.: Deepsdf: Learning continuous signed distance functions for shape representation, 2019. arXiv:1901.05103.

[SMB*20] Sitzmann V., Martel J. N. P., Bergman A. W., Lindell D. B., Wetzstein G.: Implicit neural representations with periodic activation functions, 2020. arXiv:2006.09661.

[SMP03] Sand P., McMillan L., Popović J.: Continuous capture of skin deformation. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, Association for Computing Machinery, p. 578–586. doi:10.1145/1201775.882310.

[TSM*20a] Tancik M., Srinivasan P., Mildenhall B., Fridovich-Keil S., Raghavan N., Singhal U., Ramamoorthi R., Barron J., Ng R.: Fourier features let networks learn high frequency functions in low dimensional domains, 06 2020.

[TSM*20b] Tancik M., Srinivasan P. P., Mildenhall B., Fridovich-Keil S., Raghavan N., Singhal U., Ramamoorthi R., Barron J. T., Ng R.: Fourier features let networks learn high frequency functions in low dimensional domains, 2020. arXiv:2006.10739.

[VBMP08] Vlasic D., Baran I., Matusik W., Popović J.: Articulated mesh animation from multi-view silhouettes. In *ACM SIGGRAPH 2008 Papers* (New York, NY, USA, 2008), SIGGRAPH '08, Association for Computing Machinery. doi:10.1145/1399504.1360696.

[WTM12] Wang K., Torkhani F., Montanvert A.: A fast roughness-based approach to the assessment of 3d mesh visual quality. *Computers & Graphics 36* (11 2012), 808–818. doi:10.1016/j.cag.2012.06.004.

A User's guide

Before starting the application, it is necessary to install the necessary dependencies that are not installed automatically using Nuget Package Manager and perform the appropriate configuration of the environment (see Section 7.3). Next, it is possible to configure experiments and run them. The following list describes the steps required to run the application:

1. Install Python environment, recommended version is 3.8.
2. Install Python packages tensorflow and keras using *pip* package installer.
3. Install GNU Octave, recommended version is 6.2.0.
4. Install Microsoft Visual Studio 2019 including .NET Framework SDK version 4.7.2.
5. Open *NeuralImplicits.sln* solution file using Visual Studio.
6. Open configuration file *App.config* in NeuralImplicits project a configure the environment (see Section 7.3).
7. To run the application using Visual studio, set project NeuralImplicits as a startup project.
8. Create an experiment class in Experiments folder inside NauralImplicitsProject and set up an experiment (see Section 7.4).
9. Add created experiment class to Experiments array property in a Program class.
10. Set target platform to x64.
11. Build solution.

After building the project, it is possible to run the application either directly in Visual Studio or from the command line. When starting the application, it is necessary to provide an argument with the name of the required experiment, which is listed in the *Name* property in the class with the definition of the experiment (see Code A.1).

Listing A.1: Example of running an application from the command line.

```
NeuralImplicits.exe --experiment=experiment_name
```