Neural Representation of 3D Data with Level Sets

Daniel Jiménez

Abstract

The use of neural representation of 2D and 3D data has increased in recent years due to its importance in areas such as rendering, reconstruction, memory optimization, collision detection, etc. Furthermore, neural networks give us the ability to transform a discrete image or 3D model into a continuous space with its respective applications. Nowadays, level sets are one of the most common techniques used by current algorithms to generate 3D models, so we will make an overview of algorithms that use level sets as their main prediction method, focusing on the SIREN algorithm.

Keywords— siren, sdf, level set, iso-surface

1 Introduction

For 3D representation neural networks should be compact, expressive and efficient. Most common representations are:

- A **voxel** is a 3D base cubical unit that can be used to represent more complex figures by joining many of them. Its 2D analogy is the pixel. Due to their simplicity, voxels are commonly used for 3D representation. They an represent arbitrary typology but it has cubically growing compute and memory requirements.
- A **point cloud** is a set of data points in a three dimensional space. It represents the shape of an object filling all the object volume with points. These points are not connected among them. This representation do not describe surface, and other post-processing techniques have to be applied to generate the final 3D mesh.
- A **mesh** is a geometric data structure that represents objects by a set of polygons. It is made up of vertices connected by edges making objects of a polygonal shape. The most common way for meshes is the triangular meshing, in which all polygons that describe an object are triangles. However, these methods still have some problems as high memory requirements.

These three previous techniques are the most basic methods for generating 3D models, however, new algorithms are emerging that improve the performance of these previous ones. We will explain some algorithms for generating 3D models and also take a deeper look at SIREN for both 2D and 3D data.

2 Algorithms for 3D data

One of the most common ways to recognize shape boundaries for implicit neural representation of 3D data is through the use of level sets and iso-surfaces. Given a shape as an object of study, the iso-surface is the outline of the shape itself, its limits. The iso-surface of an object is usually in the zero-level set, that is, when the function becomes 0. In the DeepSDF algorithm we will provide a problem in which the iso-surface technique is better explained.

2.1 DeepSDF

A signed distance function is a continuous function that, for a given spatial point, outputs the point's distance to the closest surface, whose sign encodes whether the point is inside (negative) or outside (positive) of the 3D model:

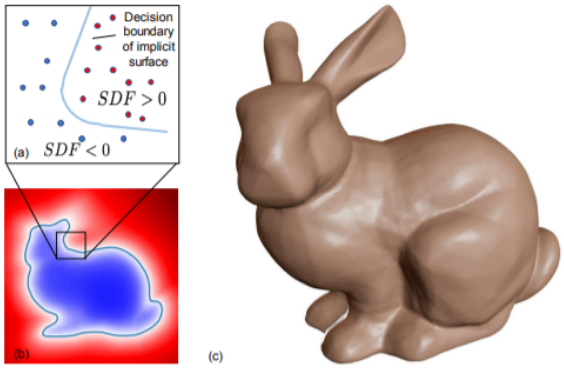
$$SDF(x) = s : x \in R^3, s \in R$$

The SDF model trains a single deep network to predict a given target shape. For a given target shape, we prepare a set of pairs X that contain the point samples and their SDF values of that shape.

$$X = (x, s) : SDF(x) = s$$

$$f_\theta(x) \sim SDF(x), \forall x \in X$$

They describe modeling shapes as the zero iso-surface decision boundaries of feed-forward networks trained to represent SDFs, i.e. the boundaries of a shape are the points where the trained network that replicates the SDF gives 0 as output $SDF(\cdot) = 0$.



The training is done by minimizing the difference between the predicted and real SDF values, using L_1 loss function:

$$L(f_\theta(x), s) = |min(\delta, max(-\delta, f_\theta(x))) - min(\delta, max(-\delta, s))|$$

The parameter δ helps us ignore the values of SDF that are too far from the shape surface. Larger values of δ make the network converge faster, and smaller values of δ improve the details on the predicted surface. The purpose of the part $min(\delta, max(-\delta, f_\theta(x)))$ is to return the δ value when the absolute value of $f_\theta(x)$ is greater than the δ value, $|f_\theta(x) > \delta| \rightarrow \delta$.

The difference between classical SDF and DeepSDF is that the network can learn an entire class of shapes instead of only a single shape.

Training a specific neural network for each shape is neither feasible nor very useful. Instead, they propose a model that can represent a wide variety of shapes. The network will embed all the common properties from a class of shapes. Then, a latent vector z, can be thought as an encoder of the desired shape. The coordinates and this vector are passed as input to the network. We repeat this process for all the coordinates of the 3D space and we get the predicted shape.

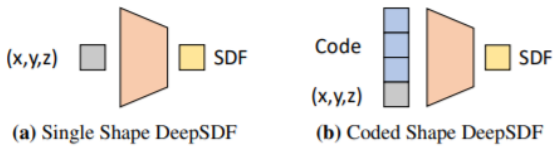


Figure 1: a) In the single-shape DeepSDF, the shape properties are stored in the network. b) In the DeepSDF, the network contains common information about the shapes class and the shape information is encoded in a vector that is concatenated with the 3D sample location. In both cases, DeepSDF outputs the SDF value at the 3D location

For constructing the network we have two valid approaches, the auto-encoder or the auto-decoder, both explained in the figure 2. For this problem, the auto-decoder option is selected due to the auto-encoder does not give

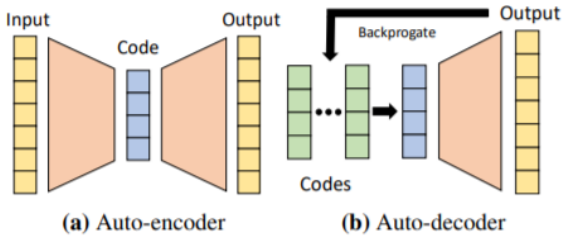


Figure 2: a) This model use two networks, the first has as input the shape and produces a vector that encodes that shape. Then this code is passed as input to the network that recreates the shape using SDF. b) A random code that encodes the shape is provided at the beginning and passed through the network as the correct code of the shape. Then back propagation updates both the weights and code in the input for optimization

information about how it is working in the test phase since its use is restricted to the training phase. We want to test the shape codes as a part of the whole network.

Given all data points from a class of shapes X, we construct a set of point samples and their signed distance values.

$$X_i = (x_j, s_k) : s_j = SDF^i(x_j)$$

$$DeepSDF \rightarrow (x_j, s_k) \in X_i : f_\theta(z_i, x_j) \sim s_j$$

For an auto-decoder, each shape code z_i is paired with training shape X_i. We have optimize both the latent code z_i and the network weights θ.

$$p_\theta(z_i|X_i) = p(z_i)\Pi_{(x_j,s_j)\in X_i} p_\theta(s_j|z_i; x_j)$$

The probability of having that shape code z_i as the correct shape code of the shape is the probability of having that shape code multiplied by the probability of output the correct SDF values given z_i and the 3D coordinate for all data points of that shape. They assume that the prior distribution for $p(z_i)$ is a zero-mean multivariate Gaussian with spherical covariance σ^2.

The probability of the second part is

$$p_\theta(s_j|z_i; x_j) = exp(-L(f_\theta(x), s_j))$$

2.2 Differentiable volumetric rendering

Most models for implicit 3D representations do not take into account the texture component of the models and only focus on the shape. The differentiable volumetric rendering algorithm builds the 3D model of a 2D image and its texture without 3D supervision.

The architecture is divided in two steps. The first one consists on representing the 3D shape of an object implicitly using the occupancy network (Mescheder et al. [2018]). The weights of the function are θ, and it needs as input the point coordinates in the 3d space and the 2d image. They encode the input image with an ResNet-18 encoder network g_θ which outputs a 256-dimensional latent code z. The output of the function returns the probability of occupancy to the point $p \in \mathbb{R}^3$ in 3D space.

$$f_\theta : \mathbb{R} \times Z \rightarrow [0, 1]$$

The second step consists on selecting the RGB value of the point p. The texture of a 3D model using a texture field is as follows. The output of the function is the RGB values.

$$t_\theta : \mathbb{R}^3 \times Z \rightarrow \mathbb{R}^3$$

The goal is to learn f_θ and t_θ only with the 2D image. The loss is defined as

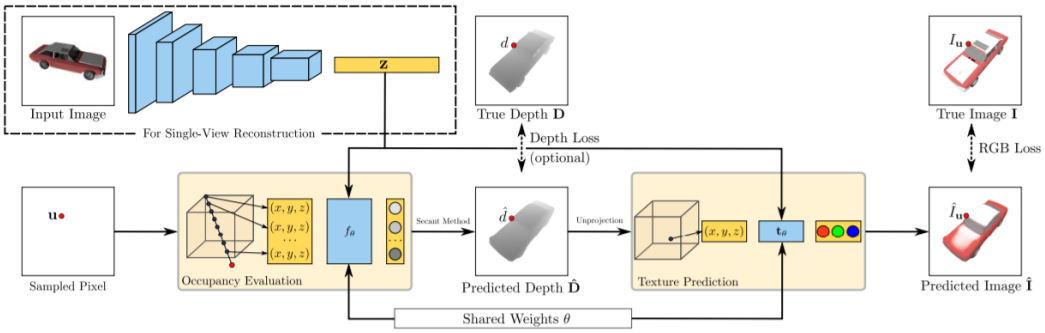


Figure 3: Differentiable volumetric rendering architecture

$$L(I, \widehat{I}_u) = \sum_u \parallel \widehat{I}_u - I_u \parallel$$

where I is the original image and $\widehat{I}$ is the rendered image, I_u is the RGB value at location u and $\parallel \cdot \parallel$ is a measure such as the l_1-norm.

To compute the gradients we apply the chain rule:

$$\frac{\partial L}{\partial \theta} = \sum_u \frac{\partial L}{\partial \widehat{I}_u} \cdot \frac{\partial \widehat{I}_u}{\partial \theta}$$

Given f_θ we can render a 3D model from any camera viewpoint using a ray. Consider an image I and any camera viewpoint. We cast a ray from the camera point that crosses I in the pixel u and determines the intersection of point $\widehat{p}$ with the surface of the 3D model. The 3D surface of an object is implicitly determined by the level set $f_\theta = \tau$ for a threshold parameter $\tau \in [0, 1]$. We can determine the depth $\widehat{d}$ by finding the first occupancy change on the ray r. To evaluate the occupancy we create points of equally distance using a step size of Δs through all the ray and we evaluate them with the function f_θ. All the points are expressed as

$$\widehat{p}_j = r(\Delta s) + s_0$$

where s_0 is the closest possible surface point. As we go close to $\widehat{p}_n$ we are more sure that the point is inside the 3D model, but we are far away of the surface of the model.

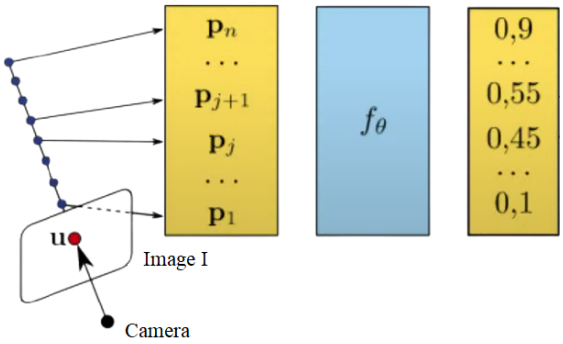


Figure 4:

Consider $\tau = 0.5$, the two points that delimit the surface are $\widehat{p}_j$ and $\widehat{p}_{j+1}$.

$$j = \arg\min j'(f_\theta(\widehat{p}_{j'+1}) \geq \tau > f_\theta(\widehat{p}_{j'}))$$

The color value I_u is given by $I_u = t_\theta(\widehat{p}_j)$. Now we can continue with the gradients of the loss function which is divided into the pixel depth and pixel color loss.

$$\frac{\partial \widehat{I}_u}{\partial \theta} = \frac{\partial t_\theta(\widehat{p})}{\partial \theta} + \frac{\partial t_\theta(\widehat{p})}{\partial \widehat{p}} \cdot \frac{\partial \widehat{p}}{\partial \theta}$$

since both t_θ as well as $\widehat{p}$ depend on θ.

3 SIREN

3.1 2D data

SIREN is a neural network that instead of working with activation functions as ReLU or tahn, it uses periodic activation functions, in this case, the activation function sin. The easier part of this network is to train a function ϕ that maps a 2D coordinate to a RGB value $\phi : \mathbb{R}^2 \to \mathbb{R}^3$

We will train this network with an image in which each input will correspond to each of the pixels of the image and the output will be the color of that pixel. To do that training, the pixels are given to the network in mini batches that are composed of random pixel locations. Once trained, we can discard the image and keep only the neural network. The architecture of the network is as follows:

$$\phi(x) = W_n(\phi_{n-1} \circ \phi_{n-2} \circ ... \circ \phi_0)(x) + b_n$$

It has n layers and the last one does not use any activation function. For each of the intermediate layers they use a sin activation function.

$$x_i \to \phi_i(x_i) = sin(W_i x_i + b_i)$$

The image is composed for a dataset $D = \{(x_i, f(x_i))\}_i$ of pixel coordinates $x_i = (x_i', y_i')$ associated with their RGB colors $f(x_i)$. This network has only one constraint $C(f(x_i), \phi(x)) = \phi(x_i) - f(x_i)$ which translates to the loss L which is just the L_2 loss

$$L = \sum_i ||\phi(x_i) - f(x_i)||^2$$

Since the sin function is periodic we have to be careful during the training. If the global optima is close to 1 in the output of the sin function and we take an step too large in the modification of weights during back propagation , it is possible that we end up going down in the sin function. To solve this problem they propose an specific initialization of weights. We have to initialize the weights uniformly from this uniform distribution

$$w_i \sim U(-c/\sqrt{n}, c/\sqrt{n}), c \in \mathbb{R}$$

They propose $c = 6$. This ensures that the input to each sine activation is normal distributed with a standard deviation of 1.

3.2 Derivatives

However, SIREN has other properties that make it very useful. Because its activation function is a sine function, its first derivative is the cosine function, which we can reformulate as the phase-shifter sine. Therefore, the derivatives of a SIREN inherit the properties of SIRENs.

Let generalize the function ϕ of the beginning.

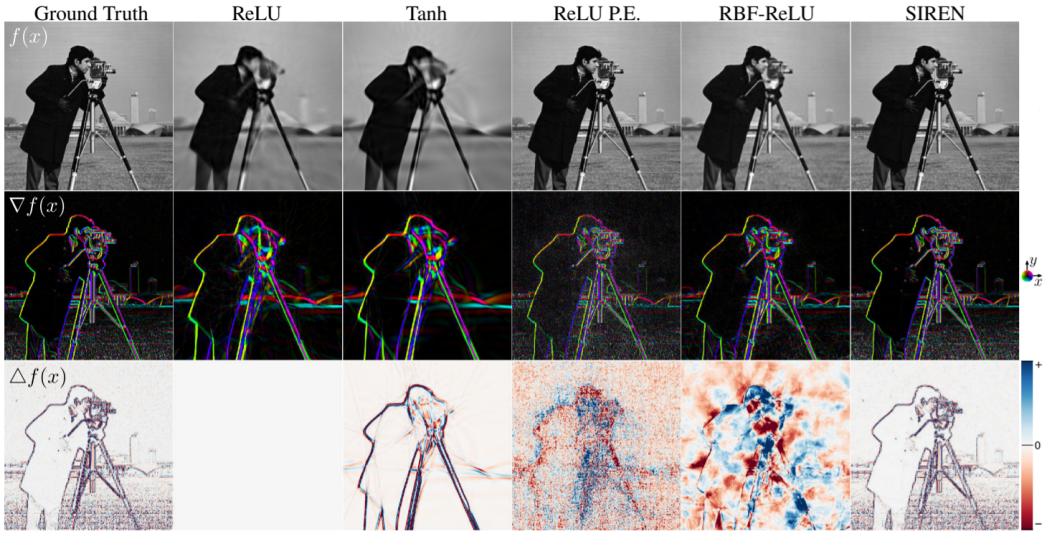
$$F(x, \phi, \nabla_x \phi, \nabla_x^2 \phi, ...) = 0, \phi : x \to \phi(x)$$

What we want now is a function that can find a relationship between an input x and an output $\phi(x)$ or between an input x and a first derivative $\nabla_x \phi(x)$, etc. SIRENs allow us to train the network not only with the image itself, but also with its derivatives (first, second, ...).

We can train the network given as input the derivative of the image $\nabla_x f(x)$ instead of the image itself $f(x)$ and minimizing the difference between the first derivative of the function $\nabla_x \phi(x)$ and the first derivative of the image. This can be applied also for the second derivative. With this network we can predict the gradients and the laplacians of an image, where Ω is the domain of the image.

$$L_{grad.} = \int_{\Omega} ||\nabla_x \phi(x) - \nabla_x f(x)||dx$$

$$L_{lapl.} = \int_{\Omega} ||\Delta \phi(x) - \Delta f(x)||dx$$



As we see in the figure [4], this is a property that not all activation functions have since, for example, the ReLU function does not have laplacians as its first derivative its constant. We have some ways of improving this by the use of positional encoding.

Furthermore, the network trained with only the derivatives can predict also the image from previous derivatives and the image itself.

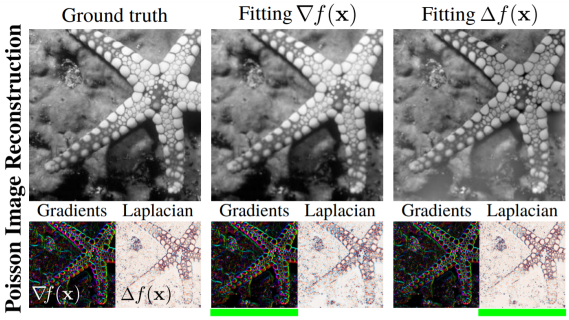


Figure 5: The first three images are the ground truth images. For the second we have trained our SIREN only with gradients of the image and the laplacians and the image are predictions. In the last, we have trained our network only with the laplacians and we predict both the image and its gradients.

3.3 3D data

For 3D data, what we want is a function that instead of giving the RGB value on a coordinate, gives us the distance of that coordinate to the boundary of the shape. It has to recreate the signed distance functions (SDFs), $\phi(x) = SDF(x)$.

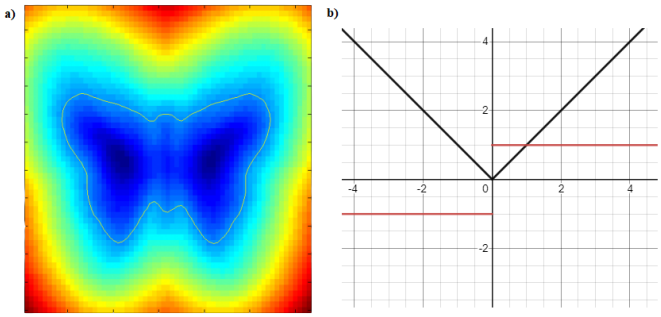
They fit SDFs directly on oriented point clouds using ReLU-based implicit neural networks and SIRENs. We are going to explain how the loss function for training the SIREN with point clouds work.

$$L_{sdf} = \int_{\Omega} \parallel |\nabla_x \phi(x)| - 1 \parallel dx + \int_{\Omega_0} \parallel \phi(x) \parallel + (1 - \langle \nabla_x \phi(x), n(x) \rangle) dx + \int_{\Omega \setminus \Omega_0} \psi(\phi(x)) dx$$

Ω is the domain of the whole 3D model and Ω_0 is the zero-level set of the object, its boundary. Let split the loss into multiple formulas:

$$\int_{\Omega} \parallel |\nabla_x \phi(x)| - 1 \parallel dx$$

What we want is that all absolute gradients in the whole 3D model become 1, so to minimize the loss we subtract 1 to the gradient on that coordinate. The SDF function returns the distance from the boundary to each coordinate, so as far as we go from the boundary the distance increments uniformly. Since the SDF increment is constant, the value of the gradients on that coordinates will be 1 or -1 depending whether the coordinate is inside or outside the shape. See figure 3.3.



$$\int_{\Omega \setminus \Omega_0} \psi(\phi(x)) dx$$

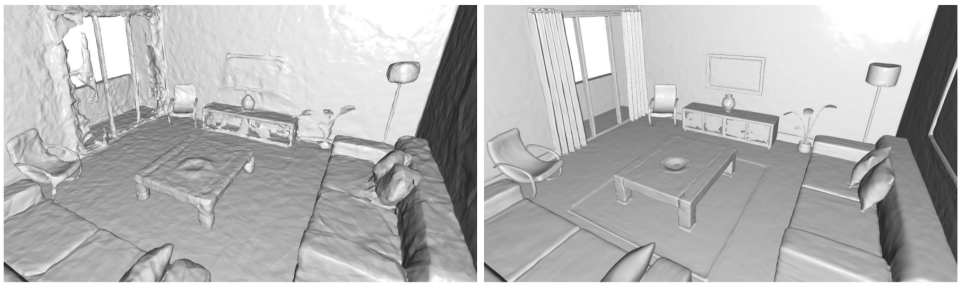
$$\psi(x) = -exp(-\alpha \cdot |\phi(x)|), \alpha \gg 1$$

$\Omega \setminus \Omega_0$ are the points that are not in the boundary. What we want is that values that are far from the boundary have a larger value. In this case, since the exponent of e is negative, as higher the value of $\phi(x)$ is, the lower value of ψ we get.

$$\int_{\Omega_0} \parallel \phi(x) \parallel + (1 - \langle \nabla_x \phi(x), n(x) \rangle) dx$$

In the last part, we take care of the coordinates in the boundary (Ω_0). First, we want to minimize the values of $\phi(x)$ until they got close to 0 since $SDF(x) = 0, x \in \Omega_0$. For the second part of the sum we have multiple functions. $n(x)$ is the normal vector at point x, which is a vector that is perpendicular to the surface at point x. The surface is the vector formed by the closest point clouds. The operator $\langle \cdot \rangle$ is the inner product of two vectors ($\langle \nabla_x \phi(x), n(x) \rangle = |\nabla_x \phi(x)| \cdot |n(x)| \cdot cos(\alpha)$), with alpha as the angle between both vectors. The length $|\cdot|$ of both vectors is 1 and because we want to minimize the loss, the cosine has to be 1, so $\alpha = 0$. With this constraint what we achieve is that both vectors have to be parallel, so the direction of the gradient is the same as the direction of the normal vector.

ReLU (baseline) SIREN



4 Results

Since this paper is only a review of other algorithms, the results of each algorithm can be found at [Park et al. [2019], Niemeyer et al. [2019], Sitzmann et al. [2020]].

5 Conclusions

In this paper we have reviewed some algorithms that use level sets, which is one of the most powerful techniques nowadays for implicit 3D neural representations. In DeepSDF and SIREN we can see the use of zero-level sets to create the shape boundary, while in the differentiable volumetric rendering algorithm we use the τ-level set for our shape. The first two methods give good results for the representation of shapes and in the second one also for the texture, however, the improvements made in the literature by SIRENs are much greater due the possibility of matching the derivatives of the image. Furthermore, the possibility of using gradients for loss functions, as we have seen for SDF reproduction, makes these networks more capable. We believe that for future work, periodic activation functions will be more used for this task due to the applications of their derivatives.

References

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

Michael Niemeyer, Lars M. Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. *CoRR*, abs/1912.07372, 2019. URL http://arxiv.org/abs/1912.07372.

Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *CoRR*, abs/1901.05103, 2019. URL http://arxiv.org/abs/1901.05103.

Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *CoRR*, abs/2006.09661, 2020. URL https://arxiv.org/abs/2006.09661.