3D Neural Sculpting (3DNS): Editing Neural Signed Distance Functions

Petros Tzathas[1] **Petros Maragos**[1] **Anastasios Roussos**[2,3]

[1]School of Electrical & Computer Engineering, National Technical University of Athens, Greece
[2]Institute of Computer Science (ICS), Foundation for Research & Technology - Hellas (FORTH), Greece
[3]College of Engineering, Mathematics and Physical Sciences, University of Exeter, UK

Abstract

*In recent years, implicit surface representations through neural networks that encode the signed distance have gained popularity and have achieved state-of-the-art results in various tasks (*e.g. *shape representation, shape reconstruction, and learning shape priors). However, in contrast to conventional shape representations such as polygon meshes, the implicit representations cannot be easily edited and existing works that attempt to address this problem are extremely limited. In this work, we propose the first method for efficient interactive editing of signed distance functions expressed through neural networks, allowing free-form editing. Inspired by 3D sculpting software for meshes, we use a brush-based framework that is intuitive and can in the future be used by sculptors and digital artists. In order to localize the desired surface deformations, we regulate the network by using a copy of it to sample the previously expressed surface. We introduce a novel framework for simulating sculpting-style surface edits, in conjunction with interactive surface sampling and efficient adaptation of network weights. We qualitatively and quantitatively evaluate our method in various different 3D objects and under many different edits. The reported results clearly show that our method yields high accuracy, in terms of achieving the desired edits, while at the same time preserving the geometry outside the interaction areas.*

1. Introduction

Representing and manipulating surfaces and 3D shapes is a problem of paramount importance in many diverse applications, ranging from mechanical and architectural design to computer animation, augmented/virtual reality, and physical simulations. It thus comes as no surprise that the representations devised over the years are as many and diverse as the applications, each with their respective advantages and disadvantages. Bézier patches, B-splines and subdivision surfaces are only some of the choices, with the most ubiquitous being the polygon meshes [20].

Although polygon meshes offer a useful and efficient representation it is hard to model diverse topologies, as that would require the vertices or their connectivity to change. To surpass these limitations researchers have tried incorporating different geometrical representations, such as voxel grids, octrees, and implicit functions. Due to the grid (or grid-like) structure of the former two, they have been used with convolutional networks [11, 21, 40]. Nevertheless, voxel grids cannot achieve high resolution and, even though octrees address this, they, too, result in jagged models.

In the last years, given the ever-rising popularity of artificial neural networks, a new class of surface representations has been proposed, namely the Implicit Neural Representations (INRs). In this approach, the surface, which is frequently required to be closed, is represented implicitly as a level set of a neural network with one output. Several papers have presented very interesting and promising results using such representations [31, 43]. In most of the papers, the network tries to learn either the signed distance function or the occupancy function. Also, the network can learn only one surface or a class of surfaces by taking a class code along with the spatial coordinates as input. In contrast to 3D meshes, voxel grids, and other common representations, INRs are not coupled to spatial resolution and can be sampled at arbitrary spatial resolutions, since they are continuous functions.

Despite the particularly promising results of implicit representations, there are still limitations to their usage. One of the most important limitations is that the shapes cannot be easily edited. This is due to the fact that in these representations the geometric structures are not represented in a local fashion. Each weight of the corresponding network affects the geometry over an unbounded region of the output space. This means that, in order to perform a localized modification on the 3D shape, generally all weights of the network need to be modified.

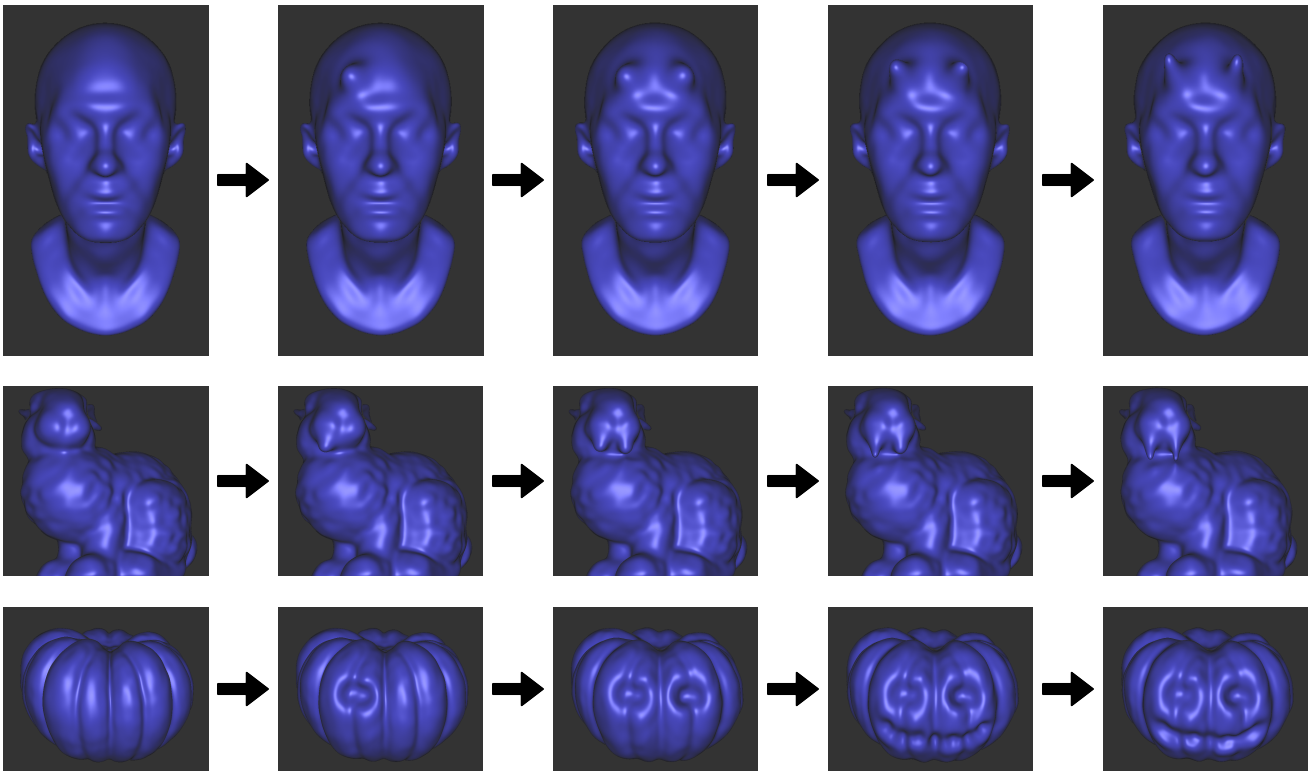


Figure 1: Examples of shape editing capabilities offered by our 3DNS, acting directly on the Neural SDF representation. First column: original shapes. Following columns: results of multiple edits using various brush settings, visualizing intermediate stages of the process. All edits and renderings are done on the implicit neural representation, avoiding completely the use of triangular meshes. Images were taken from our interactive editor, which uses Sphere Tracing [19] to render the zero-level set of the Neural SDFs. Please zoom in for details and refer to the paper's webpage (`https://pettza.github.io/3DNS/`) for a demo video.

This editability problem of INRs is an open challenge that has attracted very limited attention in the literature. Existing works allow for some form of interactive editing, by either optimizing the shape code fed to the network [13, 18] or by training the networks for articulated objects [12, 27] and changing the joints' parameters (which are also fed to the network). In either case, the editing is limited inside a learned shape space and, so, these methods do not support arbitrary modifications of the shape's 3D geometry.

To overcome the aforementioned limitations, this work introduces the first method that allows interactive editing of INRs, specifically neural Signed Distance Functions (SDFs). We approach the problem from a 3D sculpting perspective, aiming at equipping INRs with functionalities that 3D software have for the standard mesh representations. Our method, which we call 3D Neural Sculpting (3DNS), edits the surface modeled by the zero-level set of a neural network in a brush-based manner. As mentioned above, using a feedforward neural network to represent an SDF creates a problem of locality. For this, we propose using samples of the surface represented by the network to regulate the learning of the desired deformation. The source code is available at `https://github.com/pettza/3DNS`.

2. Related Work

2.1. Implicit Surface Representations

The idea to represent surfaces implicitly is by no means a new one. In fact, there have been continual attempts to use implicit representations in computer graphics and machine learning. In the shader art community, analytic implicit representations have been used to render from simple primitives to complex scenes and fractal objects [33]. On the other hand, in the machine learning community earlier approaches have relied upon radial basis functions (RBFs) [6] and octrees [15] to express SDFs. The authors of [42] use points sampled on an implicit surface to control its shape.

Recently, the use of a neural network as the function expressing the surface was proposed in by three concurrent works [8, 25, 31], which ignited interest in these implicit neural representations. DeepSDF [31] uses a network to represent the SDF for a shape (or a shape class, using ad-

ditionally a shape code as input to the network), while the other two [8, 25] express the occupancy function. Since then many more works on INRs have ensued.

The works referenced above use a regression-type loss function for training. For SDFs, this requires the computation of ground truth distances at points in space which can be difficult. Various attempts have been made to reformulate the loss function for training a neural SDF. SAL [2] neural SDFs are trained using a loss that, nevertheless, disregards the sign of the distance, which requires careful initialization. SAL++ [3] extends this method to utilize information about the normal vectors of the surface. The authors of [1] incorporate samples of the level sets of the network function to the loss. Further progress was made by IGR [17], which uses the fact that SDFs satisfy the eikonal equation to train the network as a differential equation solving network [36], thus requiring only uniform samples inside a bounding box of the surface and samples that lie on the surface, without computation of ground truth distances. Our loss is derived from this.

Other works have experimented with the architecture of the networks. SIREN [37], which uses sines instead of the usual RELUs, presented promising results in a variety of tasks including surface reconstruction. Convolutional networks are used in [9, 10] by discretizing the input point cloud. State-of-the-art results have been attained by coupling neural networks with data structures that retain localized spatial information. Octrees, which store learnable weights are used in [7, 38] and a method called hash encoding is used by Instant-NGP [28]. Besides high detailed representations, NGLoD [38] and Instant-NGP [28] achieve interactive framerates, as well.

Besides training using raw geometric data like point clouds and meshes, there have been efforts to train neural SDFs directly from images [4, 29, 30, 44]. The authors of [45] propose an SDF variant that takes direction into account (Signed Directional Distance Function). In contrast to a neural SDF which expresses the distance approximately, they prove that their network structure ensures it expresses the SDDF of some object.

It is worth mentioning that, while we focus on neural representations of shapes, implicit representations have found success in expressing quantities other than distance functions. For example, NeRF [26] produces highly realistic images by expressing the radiance and density of a scene. A recent survey [43] explores these representations, to which the authors refer as Neural Fields, in depth.

2.2. Neural SDF Editability

The research on editability is quite limited. DualSDF [18] proposes a two-level representation, one which comprises a collection of primitives (*e.g.* spheres) and a neural SDF which share the shape space. The user is able to manipulate the fine representation of the neural SDF by specifying changes to primitives' parameters which affect the shape code. A similar process is possible with DIF [13], where instead of primitives a sparse set of points is used. The authors of [12] and those of [27] both deal with articulated objects. The joints' parameters are given as input to the network and, thus, can be used to manipulate the shape.

Since the above works deform the expressed shape by proxy of the network's inputs the space of possible shapes is limited to the one learned during training. In contrast, our method allows the user to more freely change the local 3D geometry in ways that do not necessarily lie within a learned shape space, similar to the functionalities that until now were offered by 3D software for meshes only.

3. Background

We begin by presenting some material upon which we rely to develop our method. In Section 3.1 we give a definition of SDF, in Section 3.2, we describe SIREN [37] which is the architecture that we build upon, in Section 3.3 we present weight normalization [35] and finally in Section 3.4, we describe the formulation of the adopted loss function.

3.1. Signed Distance Functions (SDFs)

Let S be a surface in $\mathbb{R}^3$, then the (unsigned) distance of a point $\boldsymbol{x}$ to the surface is:

$$UDF(\boldsymbol{x}, S) = \min_{\boldsymbol{y} \in S} \{d(\boldsymbol{x}, \boldsymbol{y})\} \tag{1}$$

where d is a metric on $\mathbb{R}^3$, typically (in our case as well) the Euclidean distance.

If S is closed the signed distance function is defined as:

$$SDF(\boldsymbol{x}, S) = \begin{cases} UDF(\boldsymbol{x}, S) & \text{if } \boldsymbol{x} \text{ inside } S \\ -UDF(\boldsymbol{x}, S) & \text{otherwise} \end{cases} \tag{2}$$

A **Neural SDF** is a neural network that takes the spatial coordinate $\boldsymbol{x}$ as input and approximates the SDF of a surface. As a consequence, Neural SDFs use a whole neural network to parameterize a single shape. They effectively use a continuous function to represent a shape and in this way they are not dependent on a specific spatial resolution.

3.2. SIREN Representation

We build upon SIREN [37], which is a multilayer perceptron (MLP) that uses sines, instead of the usual ReLUs for its non-linearities. Evidently, sines allow the network to more accurately represent finer details. The use of sines is related to Fourier features [5], for which the aforementioned property has been theoretically explained [39] using the NTK framework [22]. We chose SIREN as our architecture due to its simplicity and effectiveness. We note, however, that our method is model-independent and, therefore,

could be applied to a variety of networks. The only alteration we experiment with is adding weight normalization to each layer which we discuss next.

3.3. Weight Normalization

We propose to extend SIREN by equipping it with weight Normalization, which is a memory-efficient technique proposed by [35] for accelerating convergence. It works by reparametrizing the weights of a linear layer so that they are represented by a length parameter and a direction vector. This approximately 'whitens' the gradients during training and this makes the 3D shape represented by the Neural SDF to update across iterations in a more effective manner. As we show in Section 5.4 applying it to SIREN has a positive effect.

3.4. Loss Function for Neural SDF

We adopt the loss function introduced by SIREN [37] and briefly present it here for the sake of completeness. In more detail, many of the works on neural SDFs [7, 10, 28, 31, 38] use a regression type loss to train the neural networks. The loss is evaluated for points sampled on the surface, near the surface, and in a bounding box around it. The ground truth signed distance needs to be computed for off-the-surface points which can be difficult, especially for non-mesh data which interest us (see Section 4). The authors of [17] propose a loss function that comprises a regression loss evaluated only on points on the surface (for which the distance is 0), a term for the normal vectors at the same points, and a term that enforces the network to have unit gradients w.r.t. to the input (this is commonly referred to as the eikonal term). The latter term is evaluated for points that are sampled uniformly inside a bounding box. The authors of [37], besides some minor changes, expand the above loss with a term that penalizes small values of the neural SDF at off-surface points. In summary, the loss function $L(\theta)$ that we minimize during training is defined as:

$$L(\theta) = L_S(\theta) + L_{eik}(\theta) + L_{es}(\theta), \text{ where:} \tag{3}$$

$$L_S(\theta) = \mathbb{E}_{p_S}\{\lambda_1 \left|f_\theta(\boldsymbol{x})\right| + \lambda_2\, g\left(\nabla_x f_\theta(\boldsymbol{x}), n_{\boldsymbol{x}}\right)\} \tag{4}$$

$$L_{eik}(\theta) = \lambda_3\, \mathbb{E}_q\big\{\big|\|\nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x})\| - 1\big|\big\} \tag{5}$$

$$L_{es}(\theta) = \lambda_4\, \mathbb{E}_q\{e^{-\alpha|f_\theta(\boldsymbol{x})|}\} \tag{6}$$

where λ_1, λ_2, λ_3 and λ_4 are balancing weights (set to $1.5 \cdot 10^3$, 5, 2.5, 5 respectively), S is the target surface, p_S is a distribution on the surface, q is the uniform distribution in a bounding box, θ is the parameter vector, f_θ is the network function, g is the cosine distance, $n_{\boldsymbol{x}}$ is the normal vector at $\boldsymbol{x}$ and α a large positive number (set to 100). L_S encompasses the regression and normal terms, L_{eik} is the eikonal term, and L_{es} is the term described last.

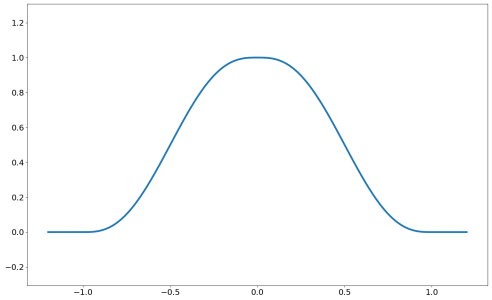


Figure 2: Brush template profile. See equation 11.

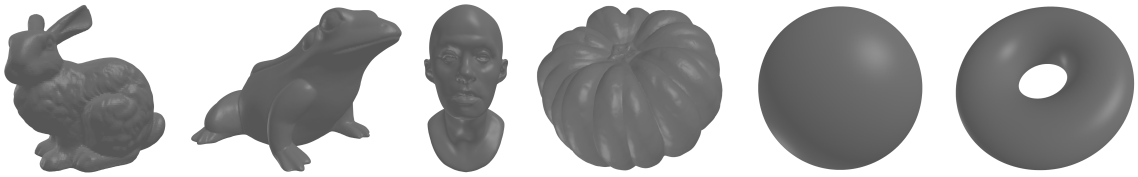
4. Proposed 3D Neural Sculpting

Our goal is to deform a surface (represented as a neural SDF) around a selected point on it in a manner similar to what is possible for meshes with 3D sculpting software. We refer to the selected point as the *interaction point*, to the area around it as the *interaction area* and to the process in general as an *interaction* or *edit*. The main problem in our case is how to enforce locality. Generally, a change to the parameters of a neural network is expected to affect its output for an unbounded region of the input space. Consequently, naively trying to train the network only where the surface needs to change will ostensibly distort the rest of the surface as well, which is what we confirm through experimentation in Section 5.6. In order to ameliorate this adverse effect as much as possible, we produce the point cloud that is to be used to evaluate the loss of equation 4 by including both samples from the surface that the network already represents (we call these model samples) and samples from the desired deformation. After discarding samples from the former set that are close to the interaction point we use the union with the latter set for training. In Section 4.1 we describe our surface sampling, then we present our formulation of sculpting brushes in Section 4.2 and then, in Section 4.3 we describe how we sample the interaction area.

4.1. Surface Sampling

Two prior works [1, 10] have proposed similar algorithms that sample the zero-level set of a neural network function. They work by sampling points uniformly inside a bounding box and then projecting them on the level set with generalized Newton-Raphson iterations. For a true SDF, this would move a point to the surface point closest to it.

A naive way to produce samples for each training iteration would be to use this algorithm. However, this approach has two major drawbacks. Firstly, it is time inefficient. Secondly, the distribution of the resulting samples can be quite non-uniform. Inspired by [10] where the sample set is extended by perturbing existing samples with gaussian noise we produce the samples for the next iterations using the



(a) Bunny (b) Frog (c) Bust (d) Pumpkin (e) Sphere (f) Torus

Figure 3: The dataset of 3D shapes that we use in our experiments. Please zoom in for details.

ones we already have. To that purpose, we add to each point a vector sampled uniformly from the tangent disk and then reproject them on the surface using the aforementioned procedure. We opt for the tangential disks, instead of gaussians, so that we explore the surface as much as possible without moving too far from it. The radius of the tangent disks is a hyperparameter we set to 0.04.

This way of sampling forms a Markov Chain [34]. Naturally, the stationary pdf distribution is of interest. It is relatively easy to see that the requirements of Theorem 1 of [14] (regarding the existence of the stationary distribution of a continuous space Markov Chain) are satisfied and, hence, the stationary distribution exists and has support over the surface. It is hard to theoretically reason about the shape of the distribution. However, we provide experimental results in Section 5.5, which demonstrate that our sampling process produces more uniformly distributed samples. Uniformness guarantees that every surface region is included equally during training.

4.2. Brushes

We define a brush template as a C^1 (or higher) positive 2D function defined over the unit disk centered at the origin which reaches a maximum value of 1 and vanishes at the unit circle (ideally its gradient and its higher derivatives vanish as well). Suppose $b_T(\boldsymbol{x})$ is a brush template, then

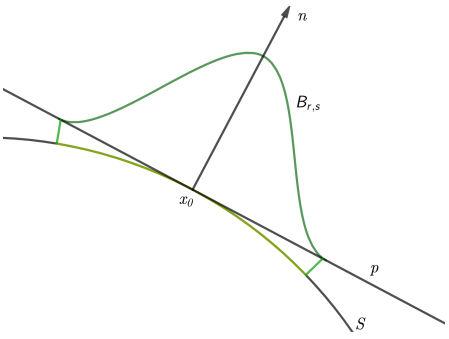


Figure 4: The brush application is demonstrated above. S is the surface, $\boldsymbol{x}_0$ is the interaction point, $\boldsymbol{n}$ is the normal vector at $\boldsymbol{x}_0$, p is the tangent plane and $B_{r,s}$ is the brush function whose graph over p is the dark green curve.

the properties above are summarized as follows:

$$b_T : \{\boldsymbol{x} \in \mathbb{R} \,|\, \|\boldsymbol{x}\| \leq 1\} \rightarrow \mathbb{R}^+ \tag{7}$$

$$\max\{b_T(\boldsymbol{x})\} = 1 \tag{8}$$

$$\|\boldsymbol{x}\| = 1 \Rightarrow b_T(\boldsymbol{x}) = 0 \,(\wedge \nabla_x b_T(\boldsymbol{x}) = \boldsymbol{0}) \tag{9}$$

We can, then, define a brush family $B_{r,s}$ parametrized over radius r and intensity s:

$$B_{r,s}(\boldsymbol{x}) = s\, b_T\left(\frac{\boldsymbol{x}}{r}\right),\ r \in \mathbb{R}^+, s \in \mathbb{R} \tag{10}$$

We show the behaviour of different radii and intensities in Section 5.3. Notice that a positive value for the intensity creates a bump on the surface, while a negative value creates a dent. In our experiments, we use the following brush template whose profile is shown in Figure 2. For other types, please refer to the supplementary material.

$$b_T(\boldsymbol{x}) = \begin{cases} P(1 - \|\boldsymbol{x}\|) & \text{if } \|\boldsymbol{x}\| < 1 \\ 0 & otherwise \end{cases} \tag{11}$$

$$P(x) = 6x^5 - 15x^4 + 10x^3 \tag{12}$$

In order to apply the brush at a point on the surface, we consider it defined on the tangent plane at that point. Due to the implicit function theorem [24] we can express the zero-level set of the network function, in a region of that point, as the graph of a 2D function over the same plane. We can, thus, apply the brush by simply adding the brush function to the latter. As we will see, even though this is not important for computing the samples' positions, we use it to compute the correct deformed normals.

4.3. Interaction Sampling

We will now describe how we produce samples on the area of the surface that is affected by the brush. We begin by placing uniform samples on a disk tangent to the interaction point whose radius is the same as the brush's radius. We can, then, project these samples on the unaltered surface and then move them perpendicular to the tangent plane (at the interaction point) a distance equal to the brush function's value. Theoretically, for what we discussed in the previous section to be applicable this needs to be a parallel projection, however, we use the same procedure that we described

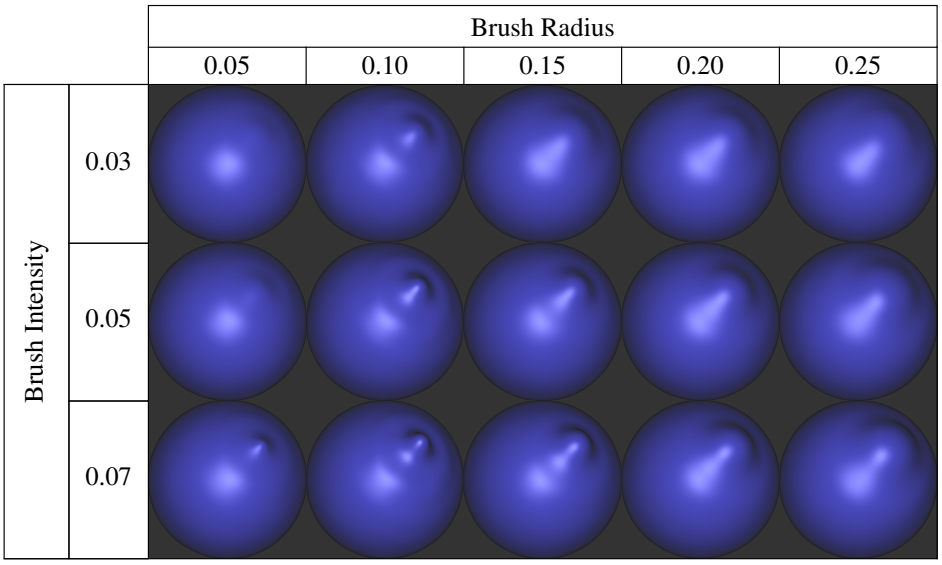


Figure 5: Example of the effect of a supported bumping brush (causing an outward local deformation) on the same interaction point on a sphere using different values for the radius and intensity. A similar effect is supported for a denting brush (causing an inward local deformation).

in Section 4.1 so that we don't affect the surface in a greater area than intended. The above process is shown in Figure 4 Next we compute the normals at the sampled position.

Suppose that f is a 2D function defined over a 3D plane (its gradient ∇f then lies on the plane), and $\boldsymbol{n}$ is the normal vector to that plane (analogous to the z axis), then the following is an unnormalized vector, perpendicular to the graph of the function:

$$-\nabla f + \boldsymbol{n} \tag{13}$$

Accordingly, what we need in order to compute the normals of the samples is the gradient of the 2D function whose graph is the deformed surface. As we explained in the previous section, this function is the sum of the brush function and the function defined implicitly by the network. We can directly calculate the gradient of the brush function. By the implicit function theorem, the gradient of the implicit function is given by:

$$-\frac{(\nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x}))_{\parallel}}{(\nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x}))_{\perp}} \tag{14}$$

where $\parallel$ denotes the component parallel to the tangent plane and $\perp$ the component perpendicular to it.

We, firstly, sample the model and discard the samples that lie inside a sphere centered at the interaction point with a radius equal to the brush's radius. Then, we sample the interaction. The number of samples we take is the number of discarded samples multiplied by an integer factor bigger than 1. We utilize this factor in order to balance the contributions of the model and interaction samples. We want the influence of the interaction samples to be larger since it is where the surface must change, while adapting to the size of the affected area. We use a factor of 10 for our experiments. Finally, the set of samples for the training is the union of the non-discarded model samples and the interaction samples.

Shape	Chamfer Distance $\times 10^3$ (↓)	
	Without WN	With WN
Bunny	9.021	**8.995**
Frog	8.095	**7.921**
Bust	7.167	**7.139**
Pumpkin	8.646	**8.506**
Sphere	7.087	**6.861**
Torus	7.198	**6.882**
Average	7.869	**7.717**

Table 1: Chamfer distances computed with 100000 points for weight normalization ablation.

5. Experiments

In this section, we qualitatively and quantitatively evaluate our proposed 3DNS in various 3D objects and under several different surface edits. For additional results please refer to the supplementary material.

5.1. Dataset

For our experiments we have created a small dataset of six 3D shapes, see Fig. 3. For four of them, we start from

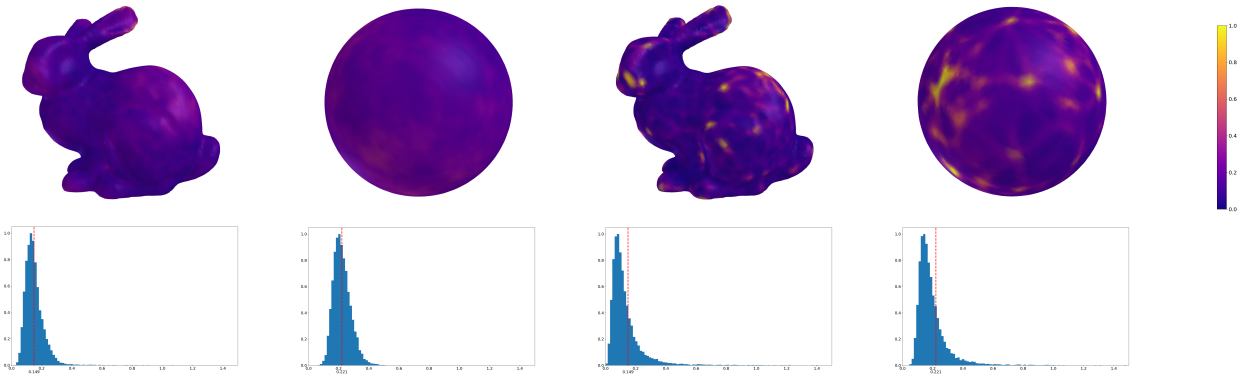


Figure 6: On the top row the estimated pdf is visualized with face colors. On the bottom row, the respective histograms of the estimated pdf values are shown. The dashed vertical red line indicates the value of the uniform pdf. For the two columns on the left, our proposed algorithm was used, while for the ones on the right, the naive approach.

a mesh representation: The Stanford Bunny, which comes from [41] and the frog, pumpkin, and bust, which come from TurboSquid [1]. The other two shapes are a sphere and a torus which are usually provided by 3D modeling software as starting shapes. We start from an analytical representation of these shapes. Specifically, we use a sphere with a radius of 0.6 and a torus with a major radius of 0.45 and a minor radius of 0.25. For our implementation, we use PyTorch [32]. As a pre-processing step, we normalize the coordinates of the four meshes of our dataset by translating and scaling them uniformly so that they lie inside $[-(1-b), 1-b]^3$, where b is a positive number. The latter parameter is used so that there is space around the models where we can edit them. We set b to 0.15. We train networks to represent the shapes of the dataset by sampling them uniformly. The architecture of the networks is SIREN with 2 hidden layers and 128 neurons each, and weight normalization (except in Section 5.4). We use 120000 samples for the loss of equation 4 and another 120000 for the losses of equations 5 and 6 and train for 10^6 iterations.

5.2. Performance Metric

For quantitative comparisons, we use the Chamfer distance, which is a common metric used for geometric tasks. If A, B are two point clouds, their Chamfer distance is defined as follows:

$$CD(A,B) = \frac{1}{|A|}\sum_{a\in A}\min_{b\in B} d(a,b) + \frac{1}{|B|}\sum_{b\in B}\min_{a\in A} d(a,b) \tag{15}$$

5.3. Brush Parameters

We demonstrate how the brush parameters allow the user to control the edit in Figure 5. Specifically, we use different radii and intensities on the same interaction point on the sphere. For each row, the intensity is the same and is shown on the left. Similarly, for each column, the radius is the same and is shown on the top.

[1] https://www.turbosquid.com

5.4. Weight Normalization Ablation Study

In order to prove the improvements provided by weight normalization, we train six models for the shapes we presented above in the first case with weight normalization and in the second case without. The Chamfer distance between a point cloud sampled from the ground truth models and one sampled from the trained network by our sampling algorithm is given in Table 1. It can be seen that, even though not by a large margin, the models trained with weight normalization perform better in every case.

5.5. PDF Estimation

We want to study the uniformness of the stationary distribution of our sampling algorithm. We do this by estimating the pdf over the surface. Firstly, we create a triangle mesh of the surface using the Marching Cubes algorithm [23]. We, then, generate samples with our algorithm for N iterations, with M samples per iteration. In effect, we are simulating M Markov chains. For each triangle of the mesh, we count the number of samples c that are closest to it. The estimated mean value of the pdf over the triangle is then:

$$pdf = \frac{c}{N \cdot M \cdot A} \tag{16}$$

where A is the area of the triangle.

We visualize the estimated pdf for the bunny and the sphere with the face colors in Figure 6. In the same figure, we show histograms of the pdf estimations, as well. The value of the uniform pdf, which is equal to the inverse of the surface area, is shown in the histograms with a dashed

Shape	Mean Chamfer Distance $\times 10^3$ (↓)					
	Over whole surface			Inside interaction area		
	Ours	Naive	Simple Mesh	Ours	Naive	Simple Mesh
Bunny	**9.407**	14.106	11.127	**5.527**	12.919	17.707
Frog	**8.172**	12.865	8.756	**4.750**	9.805	17.051
Bust	**7.279**	11.486	7.901	**3.818**	8.779	14.926
Pumpkin	**8.774**	13.558	11.489	**4.315**	5.910	20.693
Sphere	**7.209**	12.550	7.399	**3.555**	6.117	12.982
Torus	**7.142**	13.516	7.415	**3.574**	5.980	13.402
Average	**7.997**	13.014	9.015	**4.257**	8.252	16.127

Table 2: Comparison of our editing method with and without model samples (Ours and Naive, respectively) and direct mesh editing on a mesh with equivalent size (Simple Mesh). Chamfer distances are computed with 100000 points. The mean for each shape is taken over 10 independent edits.

vertical red line. We can see that the histograms are centered tightly around this value, indicating that the stationary distributions of our sampling process are quite uniform. For comparison, we give the corresponding results for the naive sampling outlined in Section 4.1, in the same Figure. Here, we notice the bright areas which are sampled more frequently than the rest of the surface, as well as the longer tails of the histograms and the fact that they are off-centered.

5.6. Mesh Editing Comparison

We compare our proposed method with the editing of meshes which is by far the most popular representation for 3D modeling and sculpting and, also, the naive approach of using only interaction samples. We edit a mesh by changing the positions of the vertices that lie inside the interaction area (the sphere that was used to discard samples). We follow the process described in Section 4.3, the only difference being that, instead of projecting samples from the tangent plane onto the surface, we compute a vertex's corresponding position on that plane as the intersection of a ray starting from the vertex with direction along its normal. In order to have a fair comparison, we use a mesh with approximately the same size as the network. We begin with a high-resolution mesh as the ground truth (this is the mesh the network was trained on for the four meshes and one constructed via Marching Cubes [23] for the sphere and torus) and, following [38], use quadratic decimation [16] to get the smallest mesh with size larger or equal to the network's size. Afterward, we perform the same ten edits on these three representations. Each edit is performed on the unedited models. We compute the mean Chamfer distance of the network and the simplified mesh to the ground truth mesh, over the whole surface, as well as only inside the interaction areas. We set the brush's radius to 0.08 and its intensity to 0.06 for all the edits. The results are summarized in Table 2, where it is shown that our method outperforms the other approaches. We, also, provide an example in Figure 7.

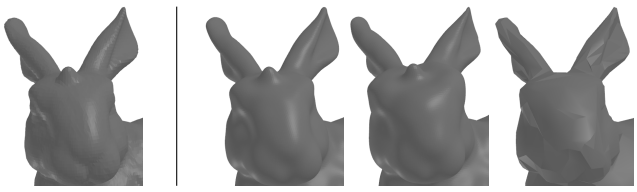


Figure 7: Example of an edit using different methods. From left to right, ground truth (ideal edit result), ours, naive and simplified mesh.

6. Limitations and Future Work

Despite the successes of neural SDFs and the results we present above, there are also drawbacks. One problem with this way of editing is that the shape cannot easily be edited outside the bounding box where the eikonal loss has been applied because there the network does not approximate an SDF. Also, since the neural network function is smooth it is difficult to model hard edges and corners using a neural SDF. However, there is research on neural SDFs that use auxiliary data structures to represent the surface locally [7, 28, 38] that can be potentially utilized to address these shortcomings, which is a direction we aim to pursue. Furthermore, our framework could in the future be extended in editing neural scene representations such as NeRFs [26].

7. Conclusion

We have presented 3DNS, a method for editing neural SDF representations in an interactive fashion inspired by existing 3D software. We believe that research towards the editability of these representations can help render them viable for more applications, either scientific or artistic in nature. Nevertheless, in order to compete with the existing tools and capabilities of meshes, more research is required.

References

[1] Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, and Yaron Lipman. Controlling neural level sets, 2019.

[2] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data, 2020.

[3] Matan Atzmon and Yaron Lipman. Sald: Sign agnostic learning with derivatives, 2020.

[4] Sai Praveen Bangaru, Michaël Gharbi, Tzu-Mao Li, Fujun Luan, Kalyan Sunkavalli, Miloš Hašan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Frédo Durand. Differentiable rendering of neural sdfs through reparameterization, 2022.

[5] Nuri Benbarka, Timon Hofer, Hamd ul-moqeet Riaz, and Andreas Zell. Seeing implicit neural representations as fourier series. *arXiv preprint arXiv:2109.00249*, 2021.

[6] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 67–76, New York, NY, USA, 2001. Association for Computing Machinery.

[7] Rohan Chabra, Jan Eric Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction, 2020.

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

[9] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion, 2020.

[10] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning, 2020.

[11] Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction, 2016.

[12] Boyang Deng, JP Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. Neural articulated shape approximation. In *The European Conference on Computer Vision (ECCV)*. Springer, August 2020.

[13] Yu Deng, Jiaolong Yang, and Xin Tong. Deformed implicit field: Modeling 3d shapes with learned dense correspondence. In *IEEE Computer Vision and Pattern Recognition*, 2021.

[14] Persi Diaconis and David Freedman. On markov chains with continuous state space. *Technical Report no. 501*, 1997.

[15] Sarah Frisken, Ronald Perry, Alyn Rockwood, and Thouis Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *ACM SIGGRAPH*, 07 2000.

[16] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997.

[17] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes, 2020.

[18] Zekun Hao, Hadar Averbuch-Elor, Noah Snavely, and Serge Belongie. Dualsdf: Semantic shape manipulation using a two-level representation. pages 7631–7641, 2020.

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

[20] Donald Hearn, M Pauline Baker, and M Pauline Baker. *Computer graphics with OpenGL*, volume 3. Pearson Prentice Hall Upper Saddle River, NJ:, 2004.

[21] Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3d object reconstruction, 2017.

[22] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020.

[23] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *COMPUTER GRAPHICS*, 21(4):163–169, 1987.

[24] Jerrold E. Marsden and Anthony Tromba. *Vector Calculus*. W. H. Freeman, 2011.

[25] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2019.

[26] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[27] Jiteng Mu, Weichao Qiu, Adam Kortylewski, Alan Yuille, Nuno Vasconcelos, and Xiaolong Wang. A-sdf: Learning disentangled signed distance functions for articulated shape representation, 2021.

[28] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.

[29] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision, 2019.

[30] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *International Conference on Computer Vision (ICCV)*, 2021.

[31] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[33] Inigo Quilez. Inigo quilez's site, 2022. https://iquilezles.org.

[34] Sheldon Ross. *Introduction to Probability Models*. Elsevier, 2017.

[35] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks, 2016.

[36] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 12 2018.

[37] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020.

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

[39] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020.

[40] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs, 2017.

[41] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. SIGGRAPH '94, page 311–318, New York, NY, USA, 1994. Association for Computing Machinery.

[42] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. computer graphics and interactive techniques, 1994.

[43] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond, 2021.

[44] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33, 2020.

[45] Ehsan Zobeidi and Nikolay Atanasov. A deep signed directional distance function for object shape representation, 2021.