Constructive Solid Geometry on Neural Signed Distance Fields

Zoë Marschner
zoem@cmu.edu
Massachusetts Institute of Technology
Carnegie Mellon University
United States

Silvia Sellán
sgsellan@cs.toronto.edu
University of Toronto
Canada

Hsueh-Ti Derek Liu
hsuehtil@gmail.com
Roblox Research
University of Toronto
Canada

Alec Jacobson
jacobson@cs.toronto.edu
Adobe Research
Canada

Figure 1: Our method allows for the computation of exact neural SDFs of CSG operations. Here, we train one network to learn the swept volume of a stellated dodecahedron shape, parametric over the control points of the cubic Bézier path it is swept along. Specific swept volumes within this parameter space are then unioned together and with cylinders, resulting in a neural implicit which thanks to our regularization term forms an exact SDF of the word “SDF.”

ABSTRACT
Signed Distance Fields (SDFs) parameterized by neural networks have recently gained popularity as a fundamental geometric representation. However, editing the shape encoded by a neural SDF remains an open challenge. A tempting approach is to leverage common geometric operators (e.g., boolean operations), but such edits often lead to incorrect non-SDF outputs (which we call Pseudo-SDFs), preventing them from being used for downstream tasks. In this paper, we characterize the space of Pseudo-SDFs, which are eikonal yet not true distance functions, and derive the closest point loss, a novel regularizer that encourages the output to be an exact SDF. We demonstrate the applicability of our regularization to many operations in which traditional methods cause a Pseudo-SDF to arise, such as CSG and swept volumes, and produce a true (neural) SDF for the result of these operations.

CCS CONCEPTS
• Computing methodologies → Shape modeling. Shape representations.

KEYWORDS
signed distance field, neural implicit, CSG, swept volumes

ACM Reference Format:

1 INTRODUCTION
Neural implicit functions have gained attention as a fundamental representation of 3D objects due to their state-of-the-art performance in tasks like compression and reconstruction, as well as their generative power. They describe the boundary of a solid shape as
the zero level set of a function—for example, the Signed Distance Function (SDF) of a given surface—\( \varphi \) parameterized by a large set of network weights \( \theta \).

Direct manipulation of the network weights \( \theta \) does not necessarily correspond to an intuitive change in the encoded shape. Therefore, Constructive Solid Geometry (CSG) is an especially attractive alternative for modelling with neural implicit functions. CSG defines geometric forms as boolean operations (unions, intersections and subtractions) on primitive shapes represented as implicit functions, such as SDFs. This naturally extends to editing neural implicit functions, by simply replacing the analytical SDFs with neural SDFs.

Unfortunately, boolean operations of SDFs are difficult to perform correctly. Common rephrasing in terms of basic comparative operations (e.g., the union of two shapes as the minimum of their SDFs) creates functions which only possess the correct zero level set. Away from the surface values are no longer necessarily true distances (see Figure 2). This makes the result ill-suit for downstream tasks, leading to potential decrease in efficiency (e.g., sphere tracing, see Figure 4) and accuracy (e.g., in collision avoidance [Li et al. 2020], surface reconstruction [Sellán et al. 2023] or medial axis extraction [Rebain et al. 2021]). Performing even simple distance-based tasks, like surface offsetting, on the results of boolean operations performed in this way can be catastrophic (Figure 3).

In this work, we propose characterizing true SDFs in terms of a closest point property on an eikonal implicit function. We add this property as a simple regularizer for neural implicit functions that ensures true signed distance results for boolean operations, including for infinite CSG operations such as swept volumes. We show that this addition allows one to efficiently carry out downstream tasks that rely on a shape being represented by a true SDF, like sphere tracing and morphological operations. Further, we showcase the advantages of our neural approach by generalizing over a set of parametric objects and swept volume trajectories (see Figure 1).

2 RELATED WORK

Our work concerns neural implicit representations, which encode 3D objects through the iso-contour of a function. Several works have explored different types of implicit function [Chibane et al. 2020; Mescheder et al. 2019; Venkatesh et al. 2021] and their applications in e.g., real-time rendering [Takikawa et al. 2021]. We refer the reader to [Xie et al. 2021] for a full survey of this field and instead focus our discussion on editing operations for neural

![Figure 2: An implicit field built from CSG operations on SDFs (left) is often not a true distance function, but rather a “Pseudo-SDF” (middle). It satisfies the eikonal property almost everywhere, but is not a true SDF (right).](image)

Signed Distance Functions (SDFs) [Park et al. 2019]—specifically, Constructive Solid Geometry (CSG) operations [Ricci 1973]. We focus on performing CSG operations on neural SDFs, in contrast to other works that focus on using neural networks to generate CSG objects [Kania et al. 2020; Ren et al. 2021; Sharma et al. 2018; Yu et al. 2023, 2022].

2.1 Neural Fields Editing

The increased recent attention on neural implicit representations has motivated the exploration of geometric editing operators defined on them. Perhaps the most direct way to edit a shape encoded by a neural implicit is by directly modifying network parameters [Park et al. 2019] or weights [Berzins et al. 2023; Davies et al. 2021]. However, direct weight editing often leads to undesired global changes to the output [Liu et al. 2022]. In response, a class of methods explore how to indirectly adjust network parameters. Their core idea, inspired by the level set method [Osher and Sethian 1988], is to formulate a target editing operation as a loss function, oftentimes a PDE. Minimizing the loss then serves as an indirect way of adjusting network weights to achieve the target edit, such as smoothing/sharpening [Mehta et al. 2022; Yang et al. 2021], physically-based deformation [Chen et al. 2022; Cuomo et al. 2022], and sculpting [Tzathas et al. 2023]. However, PDE-based approaches are quite slow as they require training the network at each time step to minimize the PDE objective. An alternative is to fix the network parameters and deform it with a vector field parameterized by another neural network (e.g., [Niemeyer et al. 2019]) to avoid re-training. Once trained, the neural vector field can directly deform any given neural implicit. By controlling the properties of the vector field, one can also achieve different editing behaviors [ Zhang et al. 2022]. For a subclass of neural implicit architectures that “exposes” part of the neural features to a spatial domain, such as vertices of an octree [Takikawa et al. 2021], one can edit neural implicit by directly moving special neural features to a different location [Abou-Chakra et al. 2022; Gao et al. 2020] and compositing with different features [Liu et al. 2020].

Unfortunately, most editing operators mentioned above do not preserve the structural properties of the underlying implicit field, such as the distance property of neural SDFs. This results in the deformed neural “SDF” not being a distance function, preventing subsequent geometry processing operations (see Figure 3). One solution is to incorporate additional regularization to further control
the editing operator. Many mesh-based regularizers have been proposed to encourage surface smoothness [Hertz et al. 2020; Kato et al. 2018; Liu et al. 2019; Wang et al. 2018] or isometry [Rakotosaona and Ovsjanikov 2020], but they are designed specifically for the surface of an object, instead of the entire implicit field. The most relevant work is the Eikonal regularization by Gropp et al. [2020] which encourages the gradient of the implicit field to have unit norm. This is an essential property all (signed) distance fields satisfy. However, in section 4, we show that the eikonal property alone is insufficient because many gradient-norm-one implicit functions are not SDFs. Our proposed regularization complements existing regularizations by encouraging the implicit field to be a true signed distance function. Our method is based on the closest point property of SDFs (see section 4), which also has proven useful in previous works [Gomes and Faugeras 2003; Ma et al. 2020]. In section 6, we demonstrate how our method can be combined with many neural editing operators to deform the implicit function while preserving the (signed) distance properties.

2.2 Constructive Solid Geometry

Constructive Solid Geometry is a modeling tool that describes a shape via boolean operations of other simpler shapes called primitives. As a general framework, CSG trees have been a part of Computer Aided Design since its infancy [Laidlaw et al. 1986; Requicha and Voelcker 1977], with more recent works proposing strategies to infer them from a given finalized shape using classic techniques [Du et al. 2018] or machine learning [Sharma et al. 2018].

Our work focuses on the atomic boolean operations that make up a CSG tree, whose efficient and robust computation has received much attention by the Computer Graphics research community. This proves a particularly difficult task for the case of explicit polyg-

![Figure 4](https://via.placeholder.com/150)

**Figure 4:** Using sphere tracing to render a Pseudo-SDF (middle) constructed by subtracting two SDFs (left) requires more iterations than rendering an exact SDF (right) due to incorrect exterior distance values.

onal meshes [Bieri and Nef 1988; Fabri and Pion 2009].

In contrast, computing implicit booleans is simple: for example, the union of two shapes is encoded by the minimum of the two implicit functions representing them. Wyvill et al. [1999] [Barthe et al. 2004; Schmidt et al. 2007; Wyvill and Van Overveld 1997] exploit this fact to build full implicit modeling systems that combine booleans with other operations like blending, smoothing and deforming.

SDFs present many advantages over general implicit functions; among them is the possibility to render SDF surfaces efficiently via sphere tracing [Hart 1996]. This forms the basis for popular interactive modeling tools like Shadertoy [Quellez and Jeremias 2017] or Adobe Substance 3D modeler. While these typically employ SDF primitive shapes, the quality of the final representation degrades with each subsequent boolean operation (see, e.g., [Takikawa et al. 2022]). In some cases, this low quality may translate only into a less efficient rendering (Figure 4); in other applications, it could be catastrophic (see Figure 3).

A particular case of CSG operations are swept volumes, which may be phrased as a continuous union of shapes along a trajectory. Given the challenge of directly computing swept volumes of explicit meshes [Abrams and Allen 2000; Zhang et al. 2009], implicit functions like SDFs are used even in cases where both input and output are meshes [Schmidt and Wyvill 2005; Sellan et al. 2021]. Since these works rely heavily on the implicit boolean definition in terms of comparative functions, they will not produce a true SDF output. In this work, we present what we believe to be the first fully implicit sweep volume algorithm that outputs a true SDF.

3 PSEUDO-SDFS

We consider the signed distance function \( f_{\theta} : \mathbb{R}^n \rightarrow \mathbb{R} \) parameterized by a neural network with weights \( \theta \). A neural SDF \( f_{\theta} \) has the same properties as a regular SDF. A surface \( \Sigma \) bounding a solid region may be implicitly encoded by the zero level set of an SDF: the set of all points \( x \) such that \( f_{\theta}(x) = 0 \). The function outputs a positive distance from a query point \( x \) to its closest point on the surface \( \Sigma \) if the point \( x \) is outside of the shape, and the negative distance for interior points:

\[
\begin{align*}
    f_{\theta}(x) &= \begin{cases} 
        d(x, \Sigma) & \text{if } x \text{ is outside of } \Sigma, \\
        -d(x, \Sigma) & \text{otherwise,}
    \end{cases} \\
    d(x, \Sigma) &= \inf_{\sigma \in \Sigma} ||x - \sigma||,
\end{align*}
\]

where \( d(x, \Sigma) \) denotes the (positive) distance from \( x \) to \( \Sigma \). This implies that a true SDF must satisfy the eikonal property

\[
||\nabla f_{\theta}(x)|| = 1,
\]

with the editing operator. Many mesh-based regularizers have been proposed to encourage surface smoothness [Hertz et al. 2020; Kato et al. 2018; Liu et al. 2019; Wang et al. 2018] or isometry [Rakotosaona and Ovsjanikov 2020], but they are designed specifically for the surface of an object, instead of the entire implicit field. The most relevant work is the Eikonal regularization by Gropp et al. [2020] which encourages the gradient of the implicit field to have unit norm. This is an essential property all (signed) distance fields satisfy. However, in section 4, we show that the eikonal property alone is insufficient because many gradient-norm-one implicit functions are not SDFs. Our proposed regularization complements existing regularizations by encouraging the implicit field to be a true signed distance function. Our method is based on the closest point property of SDFs (see section 4), which also has proven useful in previous works [Gomes and Faugeras 2003; Ma et al. 2020]. In section 6, we demonstrate how our method can be combined with many neural editing operators to deform the implicit function while preserving the (signed) distance properties.

![Figure 5](https://via.placeholder.com/150)

**Figure 5:** For a given boundary there is a uniquely defined exact SDF that is eikonal and satisfies the distance property, but there are many ways in which the conditions can be violated to different degrees. Pseudo-SDFs are functions that satisfy the eikonal property almost everywhere, but not the distance property (Equation 1). Conservative SDFs are not eikonal nor do they satisfy the distance property, but the function values are bounded by the actual distances. Implicit functions are generic level set functions that do not have any guarantees.
We will first focus on discussing the family of SDF-like functions which states that the norm of the gradient is unit everywhere. How-
conservative distance property is enforced, certain algorithms used as the additional structure imposed by the distance property allows
SDF is preferable to general implicit functions in many applications,
eroded/dilated SDF. The circle does not intersect the zero level set, as it should in an exact SDF.

which states that the norm of the gradient is unit everywhere. However, the eikonal property is merely a necessary, but not sufficient, condition to guarantee that the underlying implicit function is a true SDF. As a result, solely encouraging an implicit function to be eikonal (e.g., [Gropp et al. 2020]) is insufficient to obtain a true SDF. We will first focus on discussing the family of SDF-like functions that are not SDFs, summarized in Figure 5, and then present our solution in section 4.

3.1 A Categorization of Implicit Functions

Exact SDF. When a function obeys the distance property in Equation 1, the function exactly encodes the distance and is thus a signed distance field. The unambiguity of this definition means that there is always exactly one function that satisfies this definition for any surface Σ, which we refer to as the exact SDF for emphasis. An exact SDF is preferable to general implicit functions in many applications, as the additional structure imposed by the distance property allows for more efficient and correct behavior in many applications, such as in Figure 3 and 4.

Conservative SDF. However, obtaining an exact SDF in practice is often difficult due to, for instance, discretization error. Thus one often enforces the conservative distance property; namely,
\[ |f_0(x)| \leq |d(x, Σ)|, \quad Σ = f^{-1}_0(0). \] (3)

Intuitively, this constraints the output distance of \( f_0 \) to have a smaller magnitude than the true distance \( d \). A function that obeys this property is called a conservative SDF (see Figure 5). Because the conservative distance property is enforced, certain algorithms used on exact SDFs remain applicable, albeit with looser guarantees: for example, ray marching will still converge to a correct result as the steps are guaranteed to not overshoot the boundary.

Pseudo-SDF. A Pseudo-SDF is a piecewise differentiable implicit function that satisfies the eikonal property wherever the gradient is well defined, but does not satisfy the distance property. If a Pseudo-SDF is continuous, it must be conservative since the eikonal property bounds the maximum rate at which the function values can change. The fact that any Pseudo-SDF will satisfy the eikonal property implies that the widely-used eikonal regularization [Gropp et al. 2020] will necessarily convert a function into an exact SDF, motivating the development of our method.

These SDF-like functions, especially Pseudo-SDFs, arise naturally when performing geometric editing on exact SDFs. Even basic distance-based offsetting will result in Pseudo-SDFs in regions with non-zero curvature (see Figure 6). A more representative class is CSG operations, where Pseudo-SDFs are ubiquitous. CSG operations, including union \( \cup \), intersection \( \cap \), and difference \( \setminus \), on implicit functions \( f_1, f_2 \) are often rephrased with \( \max \) and \( \min \) operations
\begin{align*}
f_1 \cup f_2 &= \min(f_1, f_2) \\
f_1 \cap f_2 &= \max(f_1, f_2) \\
f_1 - f_2 &= f_1 \cap -f_2 = \max(f_1, -f_2),
\end{align*} (4)

but performing CSG operations with these formulae does not produce an exact SDF (see Figure 7). This phenomenon also generalizes to swept volumes, which can be perceived as performing an infinite amount of CSG operations (see Figure 17).

In the context of neural “SDFs,” the function obtained from a network may also be far from an exact SDF. This often happens in applications, such as in shape reconstruction and interpolation, where the ground truth supervision is not presented (see Figure 9). A popular (insufficient) remedy is to add the eikonal regularization
This energy measures how far from the zero level set a point is pointing towards the direction of maximal energy for a set of points \( x \in \mathcal{X} \) as

\[
E_{CP} = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} f_0(x - f_0(x)\nabla_x f_0(x))^2.
\]

This energy measures how far from the zero level set a point is mapped by the closest point function by computing the norm of the SDF at the mapped point. In Figure 10, we visualize the computation of the energy and show how our \( E_{CP} \) identifies the difference between Pseudo-SDFs and exact SDFs.

Figure 9: A parametric neural implicit \( f_0(x, w) \) is trained to fit a moon for \( f_0(x, 0) \) and a star for \( f_0(x, 1) \). For \( 0 < w < 1 \), the vanilla neural net produces non-SDFs results (top). Our method is able to encourage the interpolation to be exact SDFs (bottom).

Figure 10: The closest point loss detects the regions where a Pseudo-SDF is not a distance function. Schematically, the loss is computed for a set of points (left) by applying the closest point formula Equation 6 (middle), yielding a loss based on the value of the mapped points in \( f_0 \) (right), shown here with higher opacity denoting higher values of the closest point loss.

4.1 Regularizations for Exact SDFs

Our closest point energy in Equation 7 is the missing piece needed to encourage a neural implicit function to be an exact Distance Function (DF), either an SDF or unsigned distance function. In practice, it is always necessary for the problem formulation to include some specification of the sign of different regions, to break the symmetry inherent from the fact that the orientation of a surface is not determined by its zero level set alone. In our problems, the editing energies break this symmetry, giving us exact signed distance fields. If an eikonal function \( f_0 \) has zero level set \( \Sigma \) and at every point \( x \in \mathbb{R}^n \) the point \( \hat{x} \) computed from Equation 6 is in \( \Sigma \), then \( f_0 \) is an exact DF. This can be proven by contradiction: if \( f \) is not an DF, then at some \( x \) it does not measure the distance to its closest point on \( \Sigma \); in other words, \( \hat{x} \) is not the closest point to \( x \) on \( \Sigma \). If that is the case, then there exists some other point \( y \in \Sigma \) such that \( |f(x)| = |\hat{x} - x| > |x - y| \). But by eikonality, \( |f(x) - f(y)| < |x - y| \), meaning that \( |f(x) - f(y)| < |f(x)| \) and thus necessarily \( |f(y)| > 0 \), which is impossible since \( y \in \Sigma \).

The above observation proves that the eikonal property (Equation 2) and the closest point property (Equation 6) combined are sufficient to characterize exact SDFs. This forms the foundation of our proposed regularization for neural implicit. When training a neural network \( f_0 \) to find the optimal parameters \( \theta \), we propose augmenting the original loss function \( L \) with the closest point loss \( E_{CP} \) and with the eikonal regularization \( E_{eik} \) [Gropp et al. 2020] as:

\[
\theta^* = \arg\min_{\theta} L(\theta) + \lambda_{CP} E_{CP}(\theta, X) + \lambda_{eik} E_{eik}(\theta, X).
\]

Our regularizations are sufficient conditions to encourage fitting an exact SDF. In the inset, we visualize a neural swept volume with \( E_{CP} \) but without \( E_{eik} \) (see Figure 19 for its setup). This ablation demonstrates the importance of having both regularization terms: without \( E_{eik} \), training can get stuck in local minima like this one, where all the values are essentially zero. In section 5, we will discuss in detail how we augment our regularizations for each neural field editing operation.
5 LOSS FUNCTIONS FOR NEURAL SDF EDITING

To encourage CSG style editing operations to output exact neural SDFs, we train a network \( f_0 \) using a combination of editing loss functions, which specify the editing operations and are described in subsection 5.1 for CSG operations and subsection 5.2 for swept volumes, and the regularization terms described in Equation 8.

5.1 Constructive Solid Geometry

For this operation, we aim to learn a network representing the SDF of a sequence of CSG operations \( \text{CSG}(x) = ((s_0 \oplus s_1) \oplus (s_2 \ldots)) (x) \) where each \( s_i \) is a CSG operation (union, intersection, and subtraction) and \( s_i \) are the input SDFs. The \( s_i \) can be represented in any queryable form—such as a primitive SDF defined through math operations, an SDF on a voxel grid, or even a neural SDF. For the given sequence of CSG operations, we can compute the Pseudo-SDF of the result using min and max functions as described in Equation 4. We denote the output of this series of min and max operations \( \text{CSG}_\alpha \). Because this sequence of operations results in a Pseudo-SDF, the conservative distance property holds. That is,

\[
|\text{CSG}(s_1), \{\oplus\}| \geq |\text{CSG}_\alpha|.
\]

We therefore want to constrain our network \( f_0 \) to obey the inequality

\[
|f_0(x)| \geq |\text{CSG}_\alpha| \quad \forall x \in \mathbb{R}^n.
\]

This condition can be equivalently written using the Heaviside step function as

\[
H(-\text{sgn}(\text{CSG}_\alpha(x)))(f_0(x) - \text{CSG}_\alpha(x)) = 0 \quad \forall x \in \mathbb{R}^n.
\]

While this function correctly formulates CSG operations with a loss function, it is not practical as the step function has no meaningful gradients. To overcome this, we relax the step function to a narrow sigmoid parameterized by a width parameter \( \alpha \), leading to the final loss function which, computed at a set of points \( X \), is

\[
E_{\text{CSG}}(f_0, X; \alpha) = \frac{1}{|X|} \sum_{x \in X} \sigma(-\alpha \text{sgn}(\text{CSG}_\alpha(x)))(f_0(x) - \text{CSG}_\alpha(x))
\]

where \( \sigma \) denotes the sigmoid function. This loss \( E_{\text{CSG}} \) ensures the correct zero level set is learned. When combined with our regularization losses \( E_{\text{CP}}, E_{\text{Eik}} \) (see Equation 8), this will encourage the CSG result to be an exact SDF.

5.2 Swept Volumes

The swept volume problem involves computing the region of space covered by an object as it moves along a path. We consider a general formulation of this problem: the geometry of a shape moving in time is input as an SDF spacetime function \( s(x, t) \); to solve the swept volume problem we seek to compute the infinite union of the SDFs \( s(x, t_i) \) for all \( t_i \in [0, 1] \). This problem is harder than the CSG problem as there is no feasible way to explicitly construct an approximate SDF representing the swept volume for the general case, so an overall loss derived simply from bounding the approximate SDF as obtained in the previous subsection is not feasible. Instead, we rely on the following observation: if there exists a spacetime point \( (x, t) \) satisfying \( s(x, t) < 0 \), then the value at \( x \) in the exact swept volume, \( SV(x) \), must satisfy

\[
SV(x) \leq s(x, t).
\]

This bound allows us to write a loss analogous to \( E_{\text{CSG}} \) for the swept volume operation:

\[
E_{\text{SV}}(f_0, X; \alpha) = \frac{1}{|X|} \sum_{x \in X} |s(x, t) \leq 0| \|\alpha (f_0(x) - s(x, t))\|.
\]

where the term \( s(x, t) \leq 0 \) is a scalar mask. This loss encodes the structure we know about the interior of the SDF of the swept volume. However, this loss alone cannot solve the swept volume problem as one can construct many different exact SDFs that satisfy \( E_{\text{SV}}(f_0, X; \alpha) = 0 \) for all points. Thus, we add additional loss term which captures the idea that every point not required to be negative by \( E_{\text{SV}} \) should be positive:

\[
E_{\text{SV}}(f_0, X; \alpha) = \frac{1}{|X|} \sum_{x \in X} \sigma(-f_0(x)).
\]

The final editing loss for swept volumes is constructed from the sum of these two losses (see subsection 6.2).

5.3 Implementation Details

We represent our implicit function with a multi-layer perceptron with ReLU non-linearities. For all tests, we use 7 hidden layers with a hidden size of 128, initialized with the method by He et al. [2015]. The network is trained using Adam [Kingma and Ba 2015], with step size \( 10^{-4} \). At each epoch of training, the loss functions are evaluated on a set of points, which are generated during training. To reduce the time spent generating sample points, we reuse the same point set for \( N \approx 20 \) epochs.

5.3.1 Sampling. As our loss is defined uniformly across a set of sample points \( X \) (see Equation 7), the way we sample points \( X \) during training creates a bias towards different regions. Because the zero level set of \( f_0 \)—the surface represented by the SDF—has much more visual importance than the rest of the function, we choose to sample points in a way that weights the values of losses near the zero level set more. The inset shows the importance of this in achieving a visually sharp zero level set: in the union computed with no importance sampling, Spot’s horns—a
sharp feature—do not show up in the zero level set. To implement this sampling strategy, we generate two point sets, \( \mathcal{X} = \mathcal{X}_{\text{imp}} + \mathcal{X}_{\text{amb}} \). The points \( x \in \mathcal{X}_{\text{imp}} \) are importance sampled based on their nearness to the zero level sets of the shape SDFs defining the operation. We implement this with rejection sampling: points \( x \) are sampled using stratified sampling in the entire domain (\( \mathbb{R}^3 \) for CSG and \( \mathbb{R}^{m+1} \) for swept volume) and accepted when they satisfy the criteria

\[
\text{rand}(0, 1) \geq G(s(x), \sigma),
\]

where \( G(\mu, \sigma) \) is a Gaussian with mean \( \mu \) and standard deviation \( \sigma \), and \( \text{rand}(0, 1) \) is a random number from the uniform distribution between 0 and 1. This procedure produces points near the zero level sets of the original functions, a super set of the zero level set of the edited shape.

The second set of points, \( \mathcal{X}_{\text{amb}} \) is simply constructed through stratified sampling on the entire domain. It is necessary to ensure the function is an SDF everywhere in the trained region. This sampling procedure has two parameters, the \( \sigma \) of the Gaussian used for importance sampling and the fraction of points sampled by importance sampling.

### 6.3.2 Parametric Neural SDFs

A particular advantage of neural SDFs is their ability to represent families of shapes in their latent space. This is especially applicable to our application, as it allows for the computation of entire families of CSG operations and swept volumes by training a single network—such as a network parametric over the path of the swept volume or over parameters of shapes involved in CSG operations. Such a network requires a single up-front training cost, after which specific instances of the problem can be solved through a simple network evaluation. To implement this, we increase the dimension of the network: in \( n \) dimensional space, a network representing an SDF with \( m \) parameters will be a function \( f_0 : \mathbb{R}^{n+m} \rightarrow \mathbb{R} \). With this change, we can apply the same loss functions as before. Note that the gradient \( \nabla_x f_0 \) term in our losses is the spatial gradient, excluding the additional parametric dimensions.

### 6 RESULTS

We evaluate how our method can be used to train neural implicit.s of exact SDFs under some geometric editing operations, specifically CSG operations and swept volumes. Our model is implemented in PyTorch [Paszke et al. 2019] and trained on a NVIDIA GeForce RTX 3090 GPU.

#### 6.1 Constructive Solid Geometry Operations

We use our regularizations from Equation 8 to compute CSG operations without obtaining a Pseudo-SDF, as the naïve method does. We define the loss function as

\[
E = \lambda_1 E_{\text{CSG}} + \lambda_2 E_{\text{CP}} + \lambda_3 E_{\text{eik}}.
\]

(16)

We use the same hyper parameters for each of the tests: the weightings of the losses are chosen to be \( (\lambda_1, \lambda_2, \lambda_3) = (15, 1, 1) \) and the sharpness of the sigmoid in \( E_{\text{CSG}} \) is \( \alpha = 300 \).

In Figure 12, we show how one can use our method to learn SDFs of CSG operations on 3D objects. Unlike the Pseudo-SDFs created by computing CSG operations with min and max, we can correctly compute dilations and erosions of our result (see Figure 18). In addition to computing SDFs of single CSG operations, we can apply our method to CSG operations over parametric shapes to obtain a parametric neural SDF (see subsubsection 5.3.2) that outputs exact SDFs for a family of shapes. In Figure 16, we train a neural network to represent the entire family of pin shapes. Once trained (\( \approx 10 \) hours on a single NVIDIA GeForce RTX 3090 GPU), one can easily obtain the exact SDF of any instance within the family via a single network evaluation.

In the inset, we perform an ablation study to investigate the influence of the closest point loss. Removing \( E_{\text{CP}} \) from Equation 16 without changing the initialization causes the network to get stuck in a local minimum which is eikonal but has the incorrect zero level set (not pictured). When initialized with the Pseudo-SDF, optimizing with \( E_{\text{eik}} \) only remains at this minimum. This shows the importance of incorporating both \( E_{\text{CP}} \) and \( E_{\text{eik}} \) as regularizers to encourage convergence to the exact SDF.

#### 6.2 Swept Volumes

For swept volumes problems, the loss function is

\[
E = \lambda_1 E_{\text{SV},-} + \lambda_2 E_{\text{SV},+} + \lambda_3 E_{\text{CP}} + \lambda_4 E_{\text{eik}}.
\]

(17)

As before, we use the same set of hyper-parameters for all problems described: \( (\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (10, 0.5, 0.1, 0.1) \), and for the sharpness of the sigmoid functions we choose \( \alpha_1 = \alpha_2 = 300 \).

In Figure 19, we evaluate our method on computing a swept volume along a single cubic path. In Figure 13, we test our swept volume method on SDFs computed from brush profiles included in Photoshop. Because of the complex profile, these shapes lead to especially incorrect interior SDFs in the stamped result, while our network learns an exact SDF of the resulting sweep. We can also take advantage of neural implicit’s ability to learn swept volumes over a family of paths. In Figure 14, we take a data-driven approach, collecting a set of handwriting data from [Marti and Bunke 2002] as our training data, represented as cubic Bézier curves. This ensures our network spends its capacity on the distribution of handwriting, instead of infinite arbitrary paths. Our method enables a trained network to output an exact SDF of a “hello” hand-written by another writer who is not included in the training data. In addition to cubic Bézier curves, our method is applicable to swept volume of different kinds of path, such as those including rotations (see Figure 15). We can train networks parametric over properties of the path, allowing...
We propose regularizing neural implicits with the closest point without rotations. Our method learns an exact SDF in 3D volumes (right), as opposed to the incorrect SDF resulting from stamping (middle). Combined with geometric editing losses (see section 5), and eikonal losses to encourage the result to be an exact SDF (see Equation 8), our approach is able to correctly produce exact SDFs for CSG objects and swept volumes. Our current approach, however, requires hours of re-training to repair a Pseudo-SDF; exploring fast fine-tuning (e.g., low-rank updates [Hu et al. 2022]) of a pre-trained model could boost the efficiency of the repairing process.

An exciting future direction is to evaluate our regularization on neural SDF applications beyond computation of CSG operations, such as test-time reconstruction [Duggal et al. 2022] or generation [Chen and Zhang 2019].

ACKNOWLEDGMENTS

The second author is supported in part by an NSERC Vanier Doctoral scholarship. The MIT Geometric Data Processing group acknowledges the generous support of Army Research Office grants W911NF2010168 and W911NF2110293, of Air Force Office of Scientific Research award FA9550-19-1-031, of National Science Foundation grants IIS-1838071 and CTS-1955697, from the CSAIL Systems that learn program, from the MIT–IBM Watson AI Laboratory, from the Toyota–CSAIL Joint Research Center, from a gift from Adobe Systems, and from a Google Research Scholar award.

We thank the MIT.nano Immersion Lab and Steve Marschner for providing VR equipment and Annie, Heidi, and Steve Marschner for contributing VR drawings.

REFERENCES


Applications. IEEE, 205–213.
Figure 16: Given a family of pin shapes controlled by $(h_1, h_2, r_2, h_3)$ (left), we use our method to train a parametric neural implicit to represent the exact SDFs of all CSG operations in this family (middle). One the right, we compare the implicit function of a particular instance within this pin family computed naively with min and max operations (top right) and as a neural SDF with our regularization applied during training (bottom right).

Figure 17: A swept volume can be perceived as a union of an infinite amount of shape “stamps” along a path.

Figure 18: With Pseudo-SDFs computed from CSG operations based on min/max (first, third), one does not obtain the correct erosion and dilation behavior. Our method, which does not produce the same Pseudo-SDF artifacts, is able to obtain correct results.

Figure 19: Computing the swept volume of a star shape along a cubic curve (top first) via the naïve union of many stamps leads to a non-SDF function (top fourth). With our method we compute an SDF of the sweep (top third) that is much closer to the ground truth (top second), which is reflected quantitatively by the lower L2 error over the shown domain for our result. Having a correct sweep is important for later tasks, for instance taking the erosion of the stroke (bottom).
Figure 20: Our method is used to create the swept volume of a space shuttle flying along a fixed linear path. The rotation of the space shuttle around its vertical axis through the path is learned as a one dimensional latent space, sampled here at many points. Model courtesy of NASA.

Figure 21: We train a single network to learn the swept volume of a shape along any cubic Bézier path. We show networks for two different sweep shapes here, evaluated on cubic Bézier curves fit to data recorded from sketching in VR. We must train this model only once, and are then able to compute swept volumes simply through a network evaluation.