Reach for the Arcs: Reconstructing Surfaces from SDFs via Tangent Points

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

Yingying Ren
EPFL
Switzerland
yingying.ren@epfl.ch

Christopher Batty
University of Waterloo
Canada
christopher.batty@uwaterloo.ca

Oded Stein
University of Southern California
United States of America
ostein@usc.edu

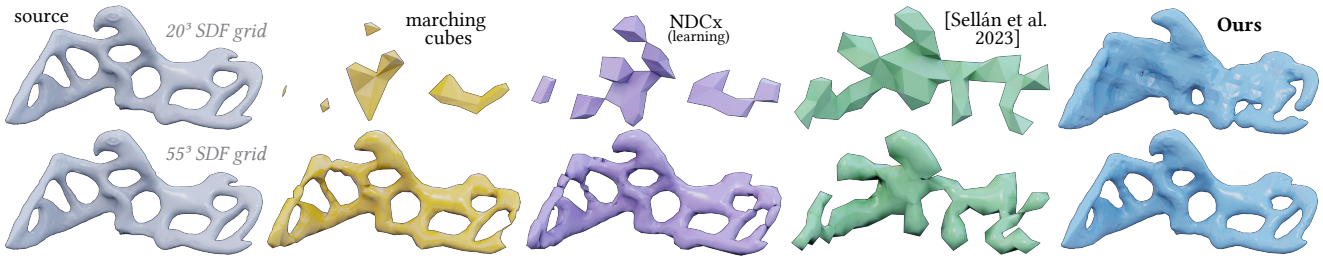


Figure 1: Our *Reach for the Arcs* algorithm consistently produces a more faithful mesh reconstruction from discrete SDF data at low and moderate resolutions compared to alternatives, as demonstrated on this nonzero genus Nightingale shape.

ABSTRACT

We introduce an algorithm to reconstruct a mesh from discrete samples of a shape's Signed Distance Function (SDF). A simple geometric reinterpretation of the SDF lets us formulate the problem through a point cloud, from which a surface can be extracted with existing techniques. We extract all possible information from the SDF data, outperforming commonly used algorithms and imposing no topological or geometric restrictions.

CCS CONCEPTS

• **Computing methodologies** → **Mesh models**; **Point-based models**; **Mesh geometry models**.

KEYWORDS

signed distance function, reconstruction, point cloud

ACM Reference Format:
Silvia Sellán, Yingying Ren, Christopher Batty, and Oded Stein. 2024. *Reach for the Arcs:* Reconstructing Surfaces from SDFs via Tangent Points. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24), July 27-August 1, 2024, Denver, CO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3641519.3657419

SIGGRAPH Conference Papers '24, July 27-August 1, 2024, Denver, CO, USA

ACM ISBN 979-8-4007-0525-0/24/07.
https://doi.org/10.1145/3641519.3657419

1 INTRODUCTION

Signed Distance Functions (SDFs) represent a shape by measuring the distance to its surface, using the function's sign to differentiate the shape's inside (negative) from its outside (positive). The attractive properties of SDFs have made them popular in applications from vision to graphics to applied mathematics [Gibson 1998; Osher and Fedkiw 2005; Sethian et al. 1999] to, recently, machine learning [Marschner et al. 2023; Park et al. 2019]. This popularity has made converting a discrete set of samples of an SDF into the explicit surface representations (e.g., triangle meshes) required by most downstream tasks a fundamental and critical research question.

Surprisingly, available methods for this task are lacking in reconstruction quality, robustness, or both. As shown recently by Sellán et al. [2023], both traditional and modern local, grid-based algorithms like Marching Cubes (MC) [Lorensen and Cline 1998] and Neural Dual Contouring (NDC) [Chen et al. 2022] heavily underutilize the global SDF information contained in the samples, leading to suboptimal results at low and medium sampling resolutions. On the other hand, while the recent *Reach for the Spheres* algorithm [Sellán et al. 2023] succeeds at exploiting all the information contained in the SDF, it manages to do so only for the simplest shapes of zero genus for which it does not encounter a flow singularity and fail.

We introduce *Reach for the Arcs*, an algorithm to recover meshes from SDFs that exploits all the information about the SDF samples without any topological restriction. Unlike Sellán et al. [2023], who pose the problem as that of finding a surface tangent to a set of spheres, we instead show that said tangency can be imposed on a set of smaller *arcs* or spherical regions (Figure 3). This lets us introduce a completely different algorithm, which explores the space of all

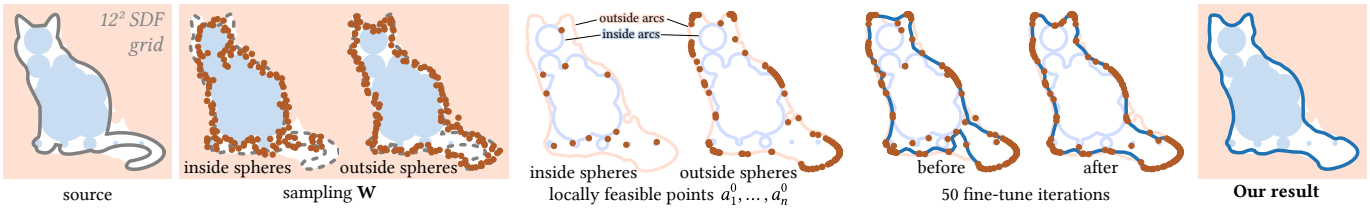


Figure 2: Given SDF data as input (left), we begin by sampling the empty volum free of SDF spheres (center left). We project the sample points onto each feasibility arc to obtain an initial tangency point cloud (center). We then fine-tune this point cloud (center right) and obtain our final output with a point-cloud-to-surface reconstruction algorithm (right).

possible tangency points in each arc, treats each choice of these as a point cloud, and extracts a mesh via the traditional Poisson Surface Reconstruction (PSR) [Kazhdan et al. 2006].

By combining SDFs and point clouds, we circumvent the main limitations of *Reach for the Spheres*, avoiding the appearance of flow singularities and allowing for reconstructions of any topology. However, like Sellán et al. [2023], our algorithm is successful at exploiting all the existing global SDF information, thus significantly outperforming local grid-based methods like MC and even neural approaches that are trained on large datasets like NDC (see Figure 1), especially for low and mid resolution inputs.

In this work, we show the robustness of our method across a large and diverse set of geometries and resolutions and use experimentation to justify our algorithmic and parametric choices. Through qualitative and large-scale quantitative comparisons, we conclude that our algorithm produces significantly more accurate reconstructions from discrete, low and mid resolution SDF data than existing methods, while either matching them or outperforming them at higher resolutions. This makes our algorithm the ideal choice for applications in which exact SDF data must be converted into an explicit mesh representation. Even further, we end by showing that our work can be easily extended to inexact signed distance data like noisy, clamped or bounded SDFs. While our algorithm's runtime is slower than alternatives like MC and NDC, we also show how it can be implemented in barely super-linear complexity.

2 RELATED WORK

2.1 Signed Distance Fields

SDFs have long seen use as a shape representation tool across many scientific fields, from computational physics [Osher and Fedkiw 2003; Sethian et al. 1999] to robotics [Liu et al. 2022] and manufacturing [Brunton and Rmaileh 2021]. Within computer graphics, their topological flexibility have made them the geometric representation of choice in many areas including image segmentation [Vese and Chan 2002], shape modeling [Museth et al. 2002], collision detection [Fisher and Lin 2001; Fuhrmann et al. 2003], liquid surface evolution [Foster and Fedkiw 2001] and rendering [Hart 1996]; including, more recently, *inverse* rendering [Bangaru et al. 2022; Jiang et al. 2020; Liu et al. 2020; Vicini et al. 2022].

Lately, the fact that SDFs represent geometric objects through mathematical functions that have regular properties and can be parametrized through neural networks [Park et al. 2019] has also led to their use in 3D Deep Learning applications [Marschner et al. 2023; Sharp and Jacobson 2022; Takikawa et al. 2021].

2.2 Isosurface Extraction

Constructing an explicit (usually triangle mesh) surface from the zero isosurface of an implicit surface representation is a classical problem, variously referred to as isosurfacing, polygonization, or simply implicit surface reconstruction. The survey by De Araújo et al. [2015] classifies such methods into three categories: spatial decomposition (e.g., marching cubes [Lorensen and Cline 1998] and dual contouring [Ju et al. 2002]), surface tracking (also known as advancing front) methods that build a mesh by crawling the zero isosurface, and shrinkwrapping/inflation methods [Hanocka et al. 2020; Stander and Hart 1997; Van Overveld and Wyvill 2004]. Spatial decomposition methods are the most common, presumably for their simplicity, efficiency, and robustness. In general, traditional isosurfacing methods do not explicitly place strong requirements on the implicit function being processed, but (smoothed) indicator functions or (exact or approximate) SDFs are common choices.

Isosurfacing methods that directly exploit the properties of signed distance fields have received less focus until relatively recently. A pair of learning-based approaches [Chen et al. 2022; Chen and Zhang 2021] improved marching cubes and dual contouring by training on large datasets of true distance fields and using this information to enhance the placement of vertices. These approaches use larger local windows of SDF data, and in doing so implicitly exploit the behavior of distance fields. Most directly relevant to our work is the method of Sellán et al. [2023], which used the tangent-spheres interpretation of SDFs to generate much higher-quality reconstructions. Its chief drawback is its reliance on evolving an explicit triangle mesh to gradually minimize an energy that measures how well the SDF data is satisfied. Since the method's remeshing

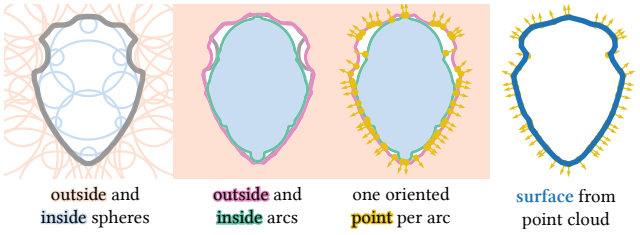


Figure 3: Sellán et al. [2023] interpret SDF data as a set of spheres (left) that the surface must be tangent to. Tangency can instead be required on a smaller set of exposed *arcs* (center). We collect the tangent points into a point cloud from which we reconstruct our final surface (right).

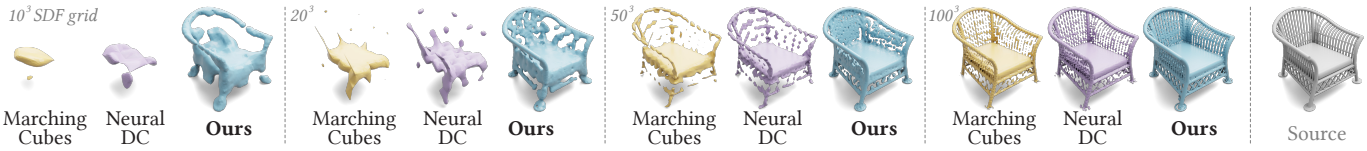


Figure 4: By extracting all information from the SDF data, we outperform Marching Cubes [Lorensen and Cline 1998] and Neural DC [Chen et al. 2022] in reconstruction quality for data at low and medium resolutions, matching them at higher ones.

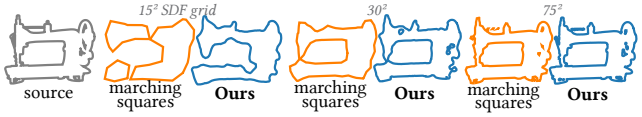


Figure 5: In 2D, our algorithm extracts significantly more geometric detail than Marching Squares on the same data.

implementation did not support topology changes, it fails whenever the mesh self-intersects, which occurs if the underlying surface is complex or has different topology from the initial triangle mesh (see Figure 6). While such failures could potentially be ameliorated using collision-aware topological remeshing (e.g., [Brochu and Bridson 2009; Wojtan et al. 2009]), such schemes add further overhead and the step sizes used for the geometric flow must still be limited to achieve a good result, leading to long optimization times.

Recent work on differentiable isosurfacing [Liao et al. 2018; Shen et al. 2021, 2023] has straddled the line between implicit (scalar field) representations, which are convenient for gradient-based optimization and topological evolution, and explicit (mesh) representations, which can often yield higher fidelity surfaces with sharp details.

2.3 Point Cloud Surface Reconstruction

Often captured from real-world data, point clouds are discrete, incomplete representations of an underlying geometry. Thus, algorithms that *reconstruct* complete surfaces from point clouds can be divided according to which *prior* they use to resolve this fundamentally underdetermined problem (see, e.g., the surveys by Berger et al. [2017] and Huang et al. [2022]). The appropriate prior is application-dependent, and can range from smoothness [Alexa et al. 2003; Amenta et al. 2001; Carr et al. 2001; Guennebaud and Gross 2007; Levin 2004; Ohtake et al. 2005] to topology [Dey and Goswami 2003], to self-similarity [Pauly et al. 2008; Williams et al. 2019] and similarity to simple primitives [Schnabel et al. 2009], to those present in a large training dataset [Groueix et al. 2018; Remil et al. 2017] or manually specified ones [Sharf et al. 2007].

The problem of point cloud reconstruction is not too dissimilar from the task considered in this paper, in which the incomplete surface information comes instead from a discrete set of SDF samples. By explicitly elucidating this duality, we reformulate SDF reconstruction as a modified point cloud reconstruction problem. As such, our work could theoretically employ any of the above listed methods and priors; however, in practice, we opt for the smoothness prior imposed by Poisson Surface Reconstruction (PSR) [Kazhdan et al. 2006] and its follow-ups [Hou et al. 2022; Kazhdan and Hoppe 2013; Sellán and Jacobson 2022, 2023].

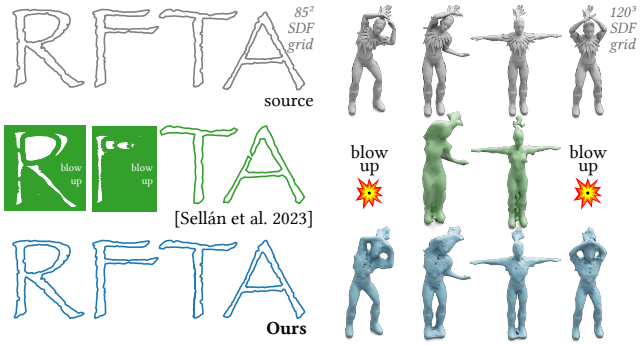


Figure 6: Unlike our method, *Reach for the Spheres* [Sellán et al. 2023] is unstable to changes in topology, and will often fail if given nonzero genus inputs.

3 METHOD

The input to our algorithm is a discrete set of SDF values $s_1, \ldots, s_n$ corresponding to positions $p_1, \ldots, p_n \in \mathbb{R}^d$. Our output will be a surface $\Omega^\star$ reconstructed from the data, discretized as a mesh.

We begin by building on recent work by Sellán et al. [2023], who propose reinterpreting each SDF sample (s_i, p_i) as defining a sphere $\mathcal{S}_i$ of radius $|s_i|$ centered at p_i (see Figure 7, left). Then, they state the reconstruction problem as finding a *feasible* surface, defined as one which is tangent to every sphere without intersecting any. Unfortunately, exploring the space of these feasible surfaces is a challenging task: even when avoiding singularities, the geometric flow proposed by Sellán et al. [2023] manages only to sample one of these surfaces, and only in the case in which the surface topology is known beforehand. Instead, we now introduce a more controllable way of navigating the space of feasible surfaces without any topological restriction, based on two simple observations.

First, we note that, in general, the spheres $\mathcal{S}_1, \ldots, \mathcal{S}_n$ intersect one another (see Figure 7, left). Since a feasible surface must be tangent to each $\mathcal{S}_i$ without intersecting any of the other $\mathcal{S}_{j\neq i}$, the tangency point must occur in the region of the sphere $\mathcal{S}_i$ that is free of intersection from all other spheres. In other words, a feasible surface must be tangent to every *feasibility arc*

$$\mathcal{A}_i = \mathcal{S}_i \setminus \bigcup_{j\neq i} \mathcal{S}_j \,, \tag{1}$$

Depending on how the spheres intersect, a single sphere's feasibility arc may in fact consist of multiple disjoint components. We further abuse terminology in 3D for brevity; there, the intersection-free portion of a sphere's surface ("feasibility arc") is a curvilinear polygon, with edges that are typically not geodesics.

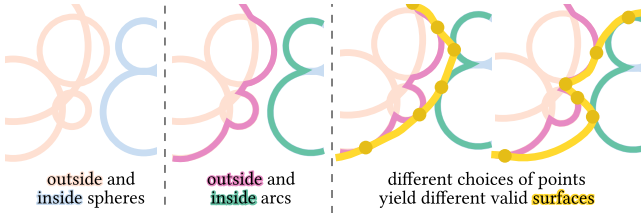


Figure 7: Left: SDF sample points interpreted as spheres. Center: Valid surface tangent points can only lie on a sphere's feasibility arc, not covered by other spheres. Right: Possible valid surfaces (yellow) are determined by the selection of a tangent point (darker yellow) per feasibility arc.

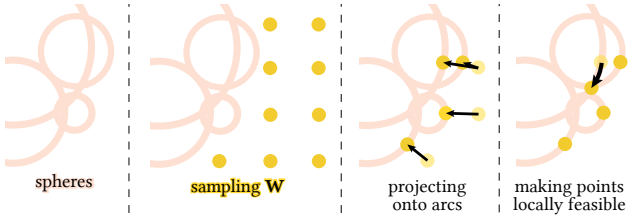


Figure 8: We produce an initial feasible tangency set a in three steps: We sample the empty space between the spheres W (center left), project the closest of these samples onto each sphere (center right) and locally displace those projections onto each sphere's feasibility arc (right).

Second, we note that we can partition the space of feasible surfaces based on the points at which the surface is tangent to each arc; i.e., through a *tangency set* $\mathbf{a} = \{a_1, \dots, a_n\}$ with $a_i \in \mathcal{A}_i$. Even further, we know that a_i will be the closest surface point to the center of the sphere p_i, thus each a_i will have an associated (unit) normal vector

$$n_i = \text{sign}(s_i)\frac{p_i - a_i}{\|p_i - a_i\|}. \tag{2}$$

Thus, one can explore the space of feasible surfaces by sampling a set of per-arc points $\mathbf{a} = \{a_1, \dots, a_n\}$ and computing their associated normal vectors $n_1, \dots, n_n$. This data forms an oriented point cloud that one can pass as input to any point-cloud-to-surface reconstruction algorithm to obtain a surface $\Omega(\mathbf{a})$ (see Figure 7, right). The choice of reconstruction algorithm corresponds to the choice of representative element in the partition class; in practice, it encodes any prior we may use to discriminate between all surfaces with the same tangency set. We choose screened Poisson Surface Reconstruction (sPSR) [Kazhdan and Hoppe 2013], which has been shown to promote smoothness [Sellán and Jacobson 2022, 2023].

Based on this exploration strategy, we propose a two-step algorithm for reconstructing a surface from any SDF data (see Figure 2; Algorithm 1 in Supplemental). First, we find an initial valid set with one point from each feasibility arc $a_1^0, \dots, a_n^0$. Then, we iteratively fine-tune these points based on a surface smoothness prior to obtain an oriented point cloud $(a_1^\star, n_1^\star), \dots, (a_n^\star, n_n^\star)$, from which we recover our final reconstructed surface $\Omega^\star$ using sPSR.

4 IMPLEMENTATION

Both algorithmic steps described above present implementation challenges related to the computational intractability of finding the exact feasibility arc $\mathcal{A}_i$ for each of a large number of spheres n. This difficulty complicates both sampling the initial tangency set $\{a_1^0, \dots, a_n^0\}$ as well as ensuring that any fine-tuning strategy only displaces each a_i *within* its arc $\mathcal{A}_i$.

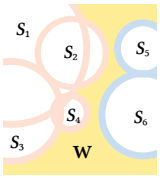
4.1 Finding the initial tangency set $\{a_1^0, \dots, a_n^0\}$

We begin by building an Axis-Aligned Bounding-Box tree data structure containing all input spheres, making queries of the type "is point x contained in any sphere?", "what is the signed distance from x to the set of all spheres?" and "what are the k closest spheres to x?" logarithmic in complexity. These atomic operations will form the basic building blocks of our algorithm.

Sampling the feasibility arcs is far from trivial. While perhaps possible in 2D, analytically computing $\mathcal{A}_i$ for each SDF data position p_i and value s_i is intractable in 3D. Trying to compute even one a_i via rejection sampling can be incredibly inefficient even with logarithmic-complexity rejection/acceptance queries, since $\mathcal{A}_i$ is in practice often significantly smaller than its containing sphere $\mathcal{S}_i$.

Fortunately, the set of feasibility arcs $\mathcal{A}_i$ form the boundary of the *feasibility volume*, the region of space exterior to every sphere

$$\mathbf{W} = \mathbf{B} \setminus \bigcup_{i=1}^{n} \text{int}(\mathcal{S}_i),$$



where $\mathbf{B}$ is a box containing all p_i and $\text{int}(\mathcal{S})$ is the ball whose boundary is $\mathcal{S}$. Thus, if we procure a dense sampling $w_1, \dots, w_m$ of $\mathbf{W}$ and find (with a K-D tree) the closest sample to each sphere

$$w^i = \text{argmin}_j d(w_j, \mathcal{S}_i), \tag{3}$$

it is likely that w^i is *very close* to a feasible arc point a_i (see Figure 8).

Densely sampling $\mathbf{W}$ is its own challenge. Strategies like naive rejection sampling become highly inefficient as the number of spheres increases and the size of $\mathbf{W}$ relative to $\mathbf{B}$ becomes smaller. One also encounters a similar behavior when using a Metropolis-Hastings strategy based on the minimum sphere distance. One may be able to sample $\mathbf{W}$ through complex importance sampling strategies using purpose-made data structures; instead, we propose a simpler alternative: rasterizing the spheres on a d-dimensional image grid to obtain a set of *empty* cells, from which we can easily sample.[1] This process not only scales linearly with the number of spheres, regardless of the relative size of $\mathbf{W}$, but also can be easily computed on a GPU for massive performance gains (see Figure 10).

Notably, making use of a rasterization image grid for this internal step of our algorithm in no way assumes any structure (grid or otherwise) present in the input SDF positions p_i; thus, it does not harm our algorithm's generality. Furthermore, while the rasterization grid resolution r is an additional parameter in our method, we experimentally find our reconstructions to be relatively stable to it (see Figure 11), and opt for the heuristic $r = 64\lceil \sqrt[d]{n}/16 \rceil$.

If we sampled $\mathbf{W}$ perfectly and with infinite density, then simply selecting w^i as the closest point to the i-th sphere would yield a point on its feasibility arc $\mathcal{A}_i$. However, for a finite set of samples

[1] Even simpler, it is actually enough to sample only a narrow band of two pixels around the spheres, as other pixels will not contribute to (3).

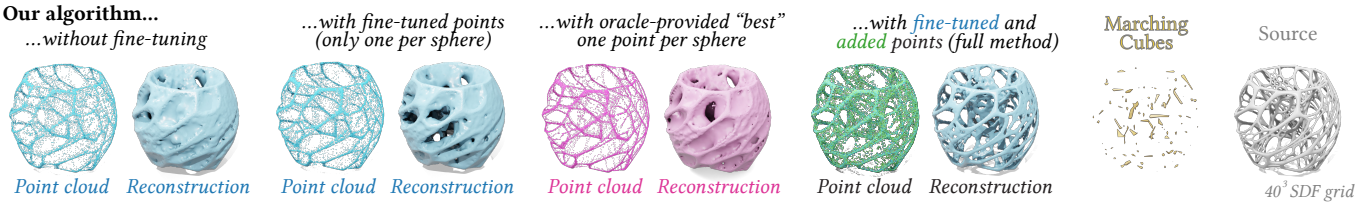


Figure 9: Preventing intersections by adding tangency points is critical, and makes our full method (center right) outperform even a fictional, oracle-given "best" tangency set (center) in which all the points are positioned exactly on the groundtruth.

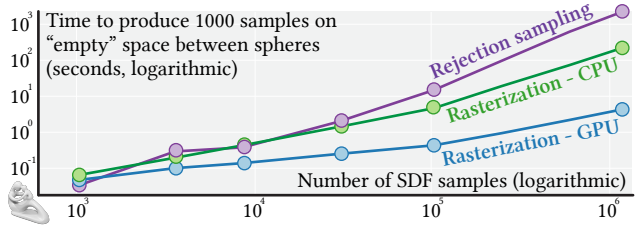


Figure 10: We find rasterization to be the most efficient strategy for producing samples on the empty space W, especially given the possibility of a GPU implementation (blue line).

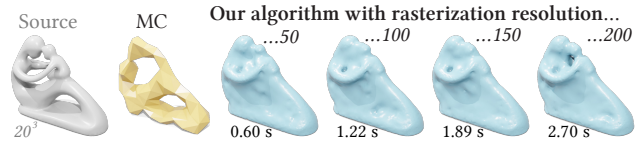
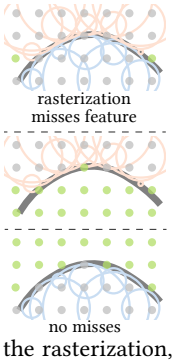


Figure 11: Our algorithm is relatively independent of the resolution of the rasterization grid for initial point sampling.

of $\mathbf{W}$, the closest point w^i will likely not lie exactly *on* the sphere $\mathcal{S}_i$, and projecting it onto $\mathcal{S}_i$ may produce a point $\text{Proj}_{\mathcal{S}_i}(w^i)$ that is *just outside* the feasibility arc $\mathcal{A}_i$ (see Figure 8).

Should $\text{Proj}_{\mathcal{S}_i}(w_i)$ be outside the feasibility arc for $\mathcal{S}_i$, we make it feasible using a local iterative procedure. For up to 20 steps (or until feasibility is achieved, whichever happens first), we collect the first $k_{\text{search}} = \lceil 2\sqrt[d]{n} \rceil$ other spheres the point is contained in, and replace w^i with the closest point tangent to $\mathcal{S}_i$ outside all of these spheres. We choose this closest point from among the intersection of $\mathcal{S}_i$ with one ($d = 2$, $d = 3$ but tuples of spheres do not intersect) or two ($d = 3$ using the algorithm of Fang [1986]) other sphere(s). We follow this process for every sphere $\mathcal{S}_i$ in a parallelized loop (see Pseudocode 2 in Supplemental).



Sign separation. The feasibility volume $\mathbf{W}$ separates the region containing the positive-sign spheres (with $s_i > 0$) and the negative sign ones (with $s_i < 0$). By definition of the SDF, these two regions must never intersect each other; however, they can be arbitrarily close to one another, sometimes causing our rasterization-based strategy to miss entire cells and produce closest points w^i that are far from the spheres.

We resolve this issue by instead considering these two sets of spheres separately, repeating the rasterization, projection and local displacement approach first to obtain initial tangency points for all the positive-sign spheres, and next for all the negative-sign spheres. Because these two sets cannot intersect one another, the information from a negative sphere will never be relevant to finding the tangency arc for a positive sphere, and vice versa, so no information loss is incurred through this separation. By combining the tangency points for both sets, we end this algorithmic step with an initial set for most spheres $\{a_1^0, \ldots, a_n^0\}$.

4.2 Fine-tuning the tangency set

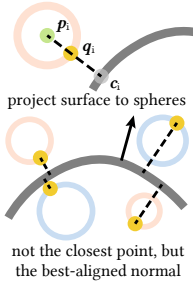
Since any discrete SDF data admits infinitely many possible tangency sets defining infinitely many possible feasible surfaces, the process from Section 4.1 only produces our initial tangency set $\mathbf{a}^0 = \{a_1^0, \ldots, a_n^0\}$. However, in general, this initial surface is undesirable: it often contains high-frequency noise, and may not even be feasible under our tangency definition.

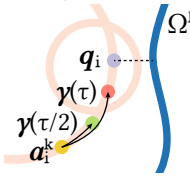
We exploit the smoothness prior imposed by sPSR in a local-global iterative optimization that reconstructs a surface from a tangency set and then moves each a_i such that it agrees with the reconstructed surface as much as possible, occasionally adding points to the cloud to enforce the surface's feasibility with respect to the SDF. Specifically, at the k-th fine-tuning iteration, we begin by using sPSR to construct the triangle mesh $\Omega^k = \Omega(\mathbf{a}^k)$, for which we build an AABB data structure to make closest-point and distance queries possible in logarithmic complexity.

Then, for each i-th sphere $\mathcal{S}_i$, we find its signed distance t_i to the surface Ω^k and its closest point c_i on Ω^k. We project c_i onto $\mathcal{S}_i$ to obtain the ideal new tangency point

$$q_i = p_i + \text{sign}(s_i t_i)|s_i| \frac{c_i - p_i}{\|c_i - p_i\|},$$

where $\text{sign}(s_i t_i)$ is intended to ensure that spheres with $s_i < 0$ are inside and spheres with $s_i > 0$ are outside the surface.





If a_i^k is not null (because Section 4.1 or a previous iteration found a feasible point), we now move a_i^k towards q_i. Consider the shortest arc-length parametrized geodesic γ on the sphere $\mathcal{S}_i$ from a_i^k to q_i. If $\gamma(\tau)$ (we choose $\tau = 10^{-2}$) is not inside any other sphere $\mathcal{S}_{j \neq i}$, then $\gamma(\tau) \in \mathcal{A}_i$ and we set $a_i^{k+1} = \gamma(\tau)$ (if q_i is closer to a_i^k than $\gamma(\tau)$, we set $a_i^{k+1} = q_i$). If, on the other hand, $\gamma(\tau)$ is covered by another sphere, we adjust $\tau \leftarrow \tau/2$ and check again, halving iteratively $\gamma(\tau) \in \mathcal{A}_i$, and then set $a_i^{k+1} = \gamma(\tau)$ (we know

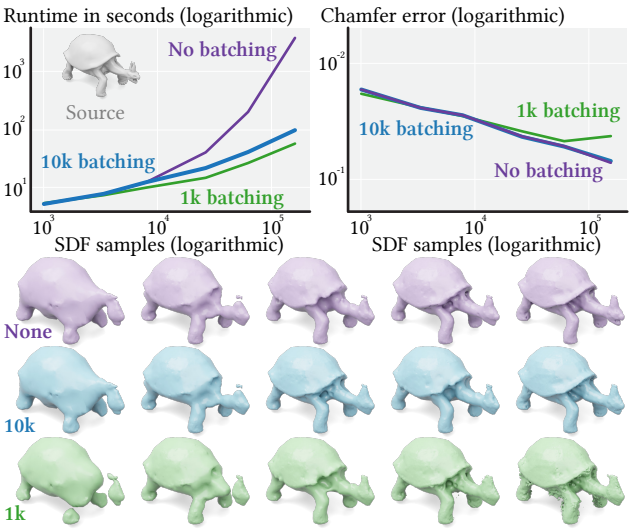


Figure 12: We opt for a batch size of $10,000$, which significantly diminishes our algorithm's runtime without a noticeable penalty to accuracy.

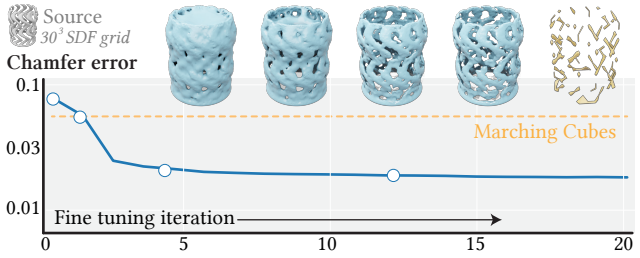


Figure 13: Our fine-tuning strategy rapidly decreases reconstruction error, with eventual diminishing returns.

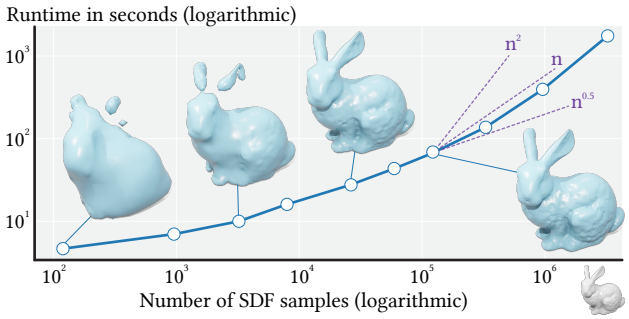
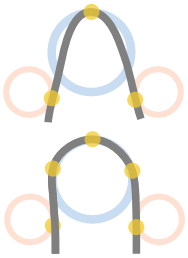


Figure 14: While we did not optimize our algorithm's runtime beyond asymptotics, its experimental complexity is, as expected, super-linear: $O(n \log n)$.

this point exists, as $\gamma(0) = a_i^k \in \mathcal{A}_i$). This could yield a point a_i^{k+1} that is not on an uninterrupted arc from a_i^k to a_i^{k+1}, but this is not a problem, since any $a_i^{k+1} \in \mathcal{A}_i$ is valid. Finally, if a_i^k is null, meaning that we are yet to find a single feasible point on the arc $\mathcal{A}_i$, we give the i-th sphere a "second chance" by testing whether q_i happens to be on $\mathcal{A}_i$ (i.e., outside every $\mathcal{S}_{j\neq i}$), in which case we make $a_i^{k+1} = q_i$. Otherwise, we maintain $a_i^{k+1} = \text{null}$.

We repeat this process in a parallelized loop over every sphere $\mathcal{S}_i$ and for $k_{\max}$ total fine-tuning iterations, leading to a final tangency set $\mathbf{a}^\star$, from which we obtain our final reconstruction $\Omega^\star = \Omega(\mathbf{a}^\star)$.

Preventing intersections. The method described thus far could still, even in the limit of fine-tuning where Ω^k passes through every point a_k^i, produce a surface that intersects a sphere (see inset). Fortunately, these cases can be identified during each fine-tuning iteration by checking whether $|t_i| < |s_i|$, and prescribing more than one tangency point for such spheres (up to $\kappa_{\max}$ per sphere): Instead of just a single tangency point a_i^k, we now have a set $\mathbf{a}_i^k = \{a_{i,1}^k, \ldots, a_{i,\kappa}^k\}$. We construct this set as follows (see Figure 9 and Algorithm 3).



- If the intersection is determined to be severe ($|t_i| < \frac{4}{5}|s_i|$), not only do we add a new point a_i^{k+1} as already described, but we also keep all the existing points in $\mathbf{a}_i^k$ and move them towards their projected closest points on Ω. This results in the addition of *one* new point, i.e., $\mathbf{a}_i^{k+1}$ has one more element than $\mathbf{a}_i^k$
- If there is no severe intersection, we add the new point a_i^{k+1} as described, and *discard* the point in $\mathbf{a}_i^k$ farthest away from Ω^k (all other points get moved as in the severe case). In this scenario, $\mathbf{a}_i^{k+1}$ has the same number of elements as $\mathbf{a}_i^k$

Batching. Our algorithm's memory and runtime complexity are dominated by storing the AABB tree containing all spheres $\mathcal{S}_i$ and answering queries related to it; mainly, the "what are the k closest spheres to x?" operation critical for projecting w^i and q_i onto the feasibility arc $\mathcal{A}_i$ in Sections 4.1 and 4.2. This is not dissimilar from the challenge encountered by Sellán et al. [2023], which they propose tackling by *batching* the spheres and using only a subset of them for each iteration of their flow. We propose a similar strategy: before finding the first tangency set $\mathbf{a}^0$ and before each fine-tuning iteration, we randomly select a *batch* of b spheres. We use this subset in all feasibility queries, while still looping over all spheres elsewhere in the algorithm (e.g., we generate a tangency point a_i for all spheres, not just the batched ones). In practice, we find that a batch size of $b = 10^4$ massively improves our algorithm's scalability without a significant penalty to accuracy (see Figure 12).

Implementation details. We implemented our algorithm in a combination of C++ and Python, using LIBIGL [Jacobson et al. 2018] and GPYTOOLBOX [Sellán and Stein 2023] for common geometry processing subroutines in each respective language. The GPU sphere rasterization described in Section 4.1 was implemented in C++ using WebGPU. Our comparisons to the SDF reconstruction algorithms by Chen et al. [2022] and Sellán et al. [2023] use their respective official Python implementations, which we thank the authors for releasing publicly. Our 3D results are rendered in Blender using the rendering toolbox of Liu [2023]. All our reported runtimes were measured on a 20-Core M1 Ultra Mac Studio with 128GB RAM.

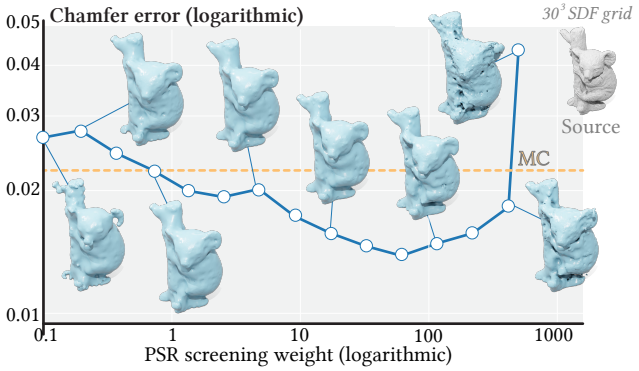


Figure 15: The screening weight used in our call to sPSR balances smoothness with adherence to the point cloud.

5 RESULTS

5.1 Experiments

Executing our algorithm requires specifying many parameters. Fortunately, for the majority of them, we find our algorithm to be surprisingly stable; for the rest, we make use of experiments to decide on reasonable default values.

For example, as shown in Figure 13, we find that most reconstruction error is corrected in the first few iterations of tangency set fine-tuning, so we set $k_{\max} = 10$. Similarly, as evidenced by Figure 11, the resolution of our rasterization grid r does not have much of an effect as it is large enough, motivating our heuristic from Section 4.1. On the other hand, we note that the batch size b can provoke failures if set too low (see Figure 12); inspired by Sellán et al. [2023], we also choose $b = 10,000$ as a default.

Finally, sPSR, the surface reconstruction algorithm that we use at each fine-tuning iteration step and as a final step before outputting the surface $\Omega^\star$, also has its own parameters. Most notably, the *screening weight* balancing smoothness and adherence to the point cloud is particularly critical for our application, since our fine-tuning strongly relies on the surface not perfectly adhering to the point cloud. We explore the effects of this parameter in Figure 15, and choose to fix it at 10 for all others.

Both of our main algorithmic steps constitute a loop over every one of the n SDF samples. Within each iteration of these loops, we find that the computational bottleneck is the sphere AABB tree traversal necessary to project w^i onto $\mathcal{A}_i$ in Section 4.1, and to check whether $\gamma(\tau)$ is on $\mathcal{A}_i$ in Section 4.2. For a set of randomly generated spheres, this traversal should average logarithmic complexity on the number of spheres n. Unfortunately, we find that the specific structure of our problem, in which vast numbers of spheres heavily intersect each other and our query points are very close to these intersections, causes these traversals to be closer to linear in complexity, making our full algorithm (without batching) quadratic $O(n^2)$. Fortunately, as evidenced in Figure 12, our proposed batching strategy makes these traversals have sublinear cost, resulting in our complete algorithm having subquadratic complexity $O(n \log n)$. We show that this is the case at medium and high resolutions in Figure 14; at low resolutions, the sub-linear cost of our GPU rasterization dominates instead.

Table 1: Average Chamfer error across one hundred shapes at different resolutions (see Table 2 for full results).

Resolution	Marching Cubes	Neural DC	RFTS	**Ours** (RFTA)
6^3	0.2644	0.1852	0.0865	**0.0672**
10^3	0.1706	0.1166	0.0559	**0.0472**
20^3	0.0688	0.0470	0.0514	**0.0295**
30^3	0.0322	0.0217	0.0497	**0.0203**
40^3	0.0234	0.0166	0.0521	**0.0156**
60^3	0.0129	**0.0087**	0.0423	0.0109
100^3	0.0054	**0.0040**	0.0332	0.0069

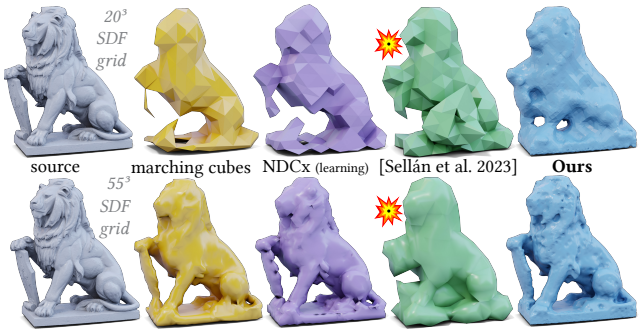


Figure 16: We outperform MC and NDC at low and medium resolutions, avoiding the catastrophic singularities of *RFTS*.

5.2 Comparisons

Qualitative comparisons. By defining and exploring the set of feasible reconstruction surfaces, our algorithm extracts the maximum amount of global information possible from the input SDF samples (Figure 20). This behavior is in marked contrast to commonly used grid-based algorithms like Marching Cubes [Lorensen and Cline 1998] and Dual Contouring [Ju et al. 2002], which utilize only the SDF information in each grid cell to decide on the local shape of the surface in it. Their neural counterparts [Chen et al. 2022; Chen and Zhang 2021] use larger input windows (7^3), but are still restricted to a limited set of stencils for a given grid cell. As evidenced in Figure 4 and Figure 5, these conventional algorithms often miss out on capturing detailed surface features, a limitation that cannot be circumvented by artificially upsampling the input data (see Figure 17) and which is not present in our approach, as further highlighted in Figure 1, Figure 9, Figure 13, and Figure 16. As seen in Figure 21, our method is better than such previous work at recovering thin features, but smooths out sharp features.

To produce these high-quality reconstructions, our algorithm builds on the theoretical observations by Sellán et al. [2023] in their work *Reach for the Spheres* (RFTS). Our method addresses the main limitation in their algorithm, by removing the requirement that the input SDF has genus zero and instead allowing for any reconstruction surface topology. We exemplify this in Figure 6, where both reconstructions are qualitatively similar for the zero genus shapes but completely different for the more topologically rich examples: ours produces valid results while theirs encounters a flow singularity and fails. Interestingly, our algorithm outperforms RFTS even for some zero genus shapes in which the latter also encounters singularities, as shown in Figure 18.

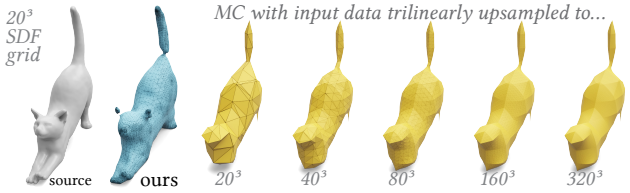


Figure 17: Our method's use of global SDF information outperforms MC, even if the input is naively upsampled via interpolation (example from Sellán et al. [2023, Fig. 12]).

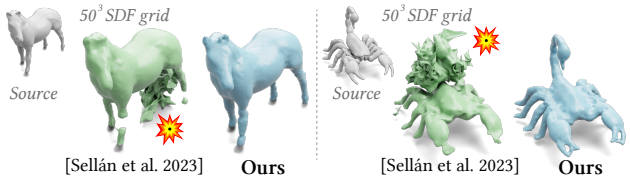


Figure 18: The flow by Sellán et al. [2023] hits a singularity and fails for these two simple shapes, unlike ours.

Quantitative comparisons. For a principled evaluation of our algorithm's performance, we randomly select 100 shapes from Thingi10k [Zhou and Jacobson 2016], preprocessed through TetWild [Hu et al. 2018] to ensure that they are closed. For each, we sample its SDF on a grid of varying resolution, and pass it as an input to our algorithm, RFTS, MC and NDC. Whenever RFTS encounters a singularity, we use the flow's last converged result.

We then compare the reconstruction errors of each method as given by the Hausdorff and Chamfer distances to the ground truth. The full results are summarized in Table 1, and visualized graphically in Figure 19. For simplicity, we also include the averages over all shapes for each method and resolution in Table 2. We find that our quantitative results match our qualitative observations: our algorithm outperforms Marching Cubes and Neural Dual Contouring at low and medium resolutions, while matching them at higher ones. Our algorithm also matches or outperforms RFTS for genus zero shapes, while vastly outperforming it for more topologically complex ones. Based on these results, we believe our proposed method can be considered the new state-of-the-art for recovering surfaces from any discrete SDF data at low and medium resolutions.

5.3 Generalizations & Applications

Our algorithm can be used in any application requiring reconstruction from SDFs. We prototype a collision-detection task in Figure 22, in which we generate random projectiles and check whether they impact the surface. Our algorithm's reconstruction is more accurate than MC and NDC, with significantly fewer false negatives.

On the other hand, many purported applications of SDFs use functions that are not strictly signed distance fields, and for which many of our algorithmic assumptions (i.e., the lack of intersection between inside and outside spheres) are violated. For example, they may utilize noisy SDFs, to which our algorithm is reasonably stable (see Figure 23) until the noise magnitude is large enough that the feasible volume becomes almost nonexistent. A more interesting example is the computation of swept volumes: despite having been shown to benefit from SDF-like representations [Sellán et al. 2021], as noticed by Marschner et al. [2023], they are often represented by *pseudo*-SDFs that do not necessarily satisfy the distance property. As shown in Figure 25, our algorithm can also accommodate these.

Finally, machine learning applications often *clamp* SDFs them to avoid spending network resolution in irrelevant regions. As long as the clamp value is known, these SDFs can also be incorporated into our algorithm (see Figure 24), by simply refraining from adding tangency points to any sphere whose data value matches the clamp.

6 LIMITATIONS & CONCLUSION

We have introduced *Reach for the Arcs*, an algorithm for reconstructing meshes from discrete samples of SDFs that outperforms existing schemes at low and medium resolutions, while at the same time overcoming the strongest limitation of its most relevant competitor.

Our algorithm works by codifying the SDF information through a *tangency point cloud*, from which a surface can be produced through standard point-cloud-to-surface reconstruction techniques. While we opt for screened Poisson Surface Reconstruction [Kazhdan and Hoppe 2013], the specific reconstruction method used can encode more complex priors about the input geometry (e.g., data-driven). While we experimented with this possibility only briefly in Figure 26, we believe it to be a promising avenue for future work.

We optimized our algorithm's computational complexity to be only super-linear; however, its wallclock runtime remains orders of magnitude worse than competing methods like Marching Cubes. Promising avenues to reduce this discrepancy in order to completely supplant previous methods in every use-case may include prioritizing certain spheres over others in the fine-tuning strategy, or introducing batching in the initially generated tangency points.

On a more fundamental level, our work advocates for exploiting the observed duality between SDFs and point clouds. By shining a light on this underexplored fact, we hope to inspire a new generation of work which combines the vast literatures that have resulted from studying each of these shape representations independently.

ACKNOWLEDGMENTS

This work received support from NSERC (Grant RGPIN-2021-02524), CFI (JELF Project 40132), an NSERC Vanier Scholarship, and SNSF (Grant FNS 514543 / CF 1156). We thank Abhishek Madan, Nicholas Sharp, Chenxi Liu, Victor Rong, Dylan Rowe, and Kinjal Parikh for technical help and proofreading; Michael Kazhdan for his help with the official sPSR implementation; Eitan Grinspun and David Levin for their insightful suggestions regarding rasterization; and Daniella Sarit Levy for providing the mesh and poses for Figure 6.

We acknowledge and thank the authors of the 3D models used throughout this paper. Figures in this work contain the nightingale [Emm 2021], chair [Kacie Hultgren 2012], lamp [Nervous System Studio 2012], fertility [Aim@Shape 2009], spiral cup [Mark Durbin 2011], bunny [The Stanford 3D Scanning Repository 1994], turtle pope [Ronan Murphy 2022], koala [Thunk3D scanner 2019], lion [Niederösterreich 3D 2020], spot [Crane 2013], pyramid [ChapinEagle02 2015], cat [billyd 2016], archer [Roy Mac 2020], metratron [Bathsheba Grossman 2012], scorpion [YahooJAPAN 2013], horse [Cyberware Inc. 2023], skull [Yahoo! JAPAN 2013], and triceratops [Hanocka et al. 2020] meshes.

REFERENCES

Aim@Shape. 2009. Fertility.

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. 2003. Computing and rendering point set surfaces. 9, 1 (2003), 3–15.

Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. 2001. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*. 249–266.

Sai Praveen Bangaru, Michaël Gharbi, Fujun Luan, Tzu-Mao Li, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. 2022. Differentiable rendering of neural SDFs through reparameterization. In *SIGGRAPH Asia 2022 Conference Papers*. Article 5, 9 pages. https://doi.org/10.1145/3550469.3555397

Bathsheba Grossman. 2012. metratron. https://www.thingiverse.com/thing:16673.

Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. 2017. A survey of surface reconstruction from point clouds. In *Computer graphics forum*, Vol. 36. Wiley Online Library, 301–329.

Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. 1999. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics* 5, 4 (1999), 349–359.

billyd. 2016. Cat Stretch. https://www.thingiverse.com/thing:1565405.

Tyson Brochu and Robert Bridson. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing* 31, 4 (2009), 2472–2493.

Alan Brunton and Lubna Abu Rmaileh. 2021. Displaced Signed Distance Fields for Additive Manufacturing. *ACM Trans. Graph.* 40, 4, Article 179 (2021), 13 pages. https://doi.org/10.1145/3450626.3459827

Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 67–76.

ChapinEagle02. 2015. The chichen Itza pyramid. https://www.thingiverse.com/thing:675795.

Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. 2022. Neural dual contouring. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–13.

Zhiqin Chen and Hao Zhang. 2021. Neural marching cubes. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–15.

Keenan Crane. 2013. Spot. https://www.cs.cmu.edu/~kmcrane/Projects/ModelRepository/.

Cyberware Inc. 2023. Horse. https://github.com/alecjacobson/common-3d-test-models/blob/master/data/horse.obj (year denotes online access).

Bruno Rodrigues De Araújo, Daniel S Lopes, Pauline Jepp, Joaquim A Jorge, and Brian Wyvill. 2015. A survey on implicit surface polygonization. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–39.

Tamal K Dey and Samrat Goswami. 2003. Tight cocone: a water-tight surface reconstructor. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*. 127–134.

Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. 1983. On the shape of a set of points in the plane. *IEEE Transactions on information theory* 29, 4 (1983), 551–559.

Emm. 2021. Stravinsky Fountain: the Nightingale. https://sketchfab.com/3d-models/stravinsky-fountain-the-nightingale-66589dd1f2b04c2f9d2e828be5241d77.

Bertrand T. Fang. 1986. Trilateration and extension to Global Positioning System navigation. *Journal of Guidance, Control, and Dynamics* 9, 6 (1986), 715–717.

Susan Fisher and Ming C Lin. 2001. Deformed distance fields for simulation of non-penetrating flexible bodies. In *Computer Animation and Simulation 2001: Proceedings of the Eurographics Workshop in Manchester, UK, September 2–3, 2001*. Springer, 99–111.

Nick Foster and Ronald Fedkiw. 2001. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 23–30.

Arnulph Fuhrmann, Gerrit Sobotka, and Clemens Groß. 2003. Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon*, Vol. 2003. 58–65.

Sarah FF Gibson. 1998. Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of the 1998 IEEE symposium on Volume visualization*. 23–30.

Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. 2018. A papier-mâché approach to learning 3d surface generation. 216–224.

Gaël Guennebaud and Markus Gross. 2007. Algebraic point set surfaces. In *ACM siggraph 2007 papers*. 23–es.

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. 2020. Point2mesh: A self-prior for deformable meshes. *arXiv preprint arXiv:2005.11084* (2020).

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

Fei Hou, Chiyu Wang, Wencheng Wang, Hong Qin, Chen Qian, and Ying He. 2022. Iterative Poisson Surface Reconstruction (IPSR) for Unoriented Points. *ACM Trans. Graph.* 41, 4, Article 128 (2022), 13 pages. https://doi.org/10.1145/3528223.3530096

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral meshing in the wild. *ACM Trans. Graph.* 37, 4 (2018), 60–1.

Zhangjin Huang, Yuxin Wen, Zihao Wang, Jinjuan Ren, and Kui Jia. 2022. Surface reconstruction from point clouds: A survey and a benchmark. *arXiv preprint arXiv:2205.02413* (2022).

Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. 2020. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1251–1261.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 339–346.

Kacie Hultgren. 2012. 1:24 Wicker Furniture Set. https://www.thingiverse.com/thing:32225.

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7. 0.

Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* 32, 3 (2013), 1–13.

David Levin. 2004. Mesh-independent surface interpolation. In *Geometric modeling for scientific visualization*. Springer, 37–49.

Yiyi Liao, Simon Donne, and Andreas Geiger. 2018. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2916–2925.

Hsueh-Ti Derek Liu. 2023. BlenderToolbox. https://github.com/HTDerekLiu/BlenderToolbox.

Puze Liu, Kuo Zhang, Davide Tateo, Snehal Jauhri, Jan Peters, and Georgia Chalvatzaki. 2022. Regularized Deep Signed Distance Fields for Reactive Motion Generation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 6673–6680. https://doi.org/10.1109/IROS47612.2022.9981456

Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. 2020. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019–2028.

William E Lorensen and Harvey E Cline. 1998. Marching cubes: A high resolution 3D surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*. 347–353.

Mark Durbin. 2011. Spiral Cup. https://www.thingiverse.com/thing:11687.

Zoë Marschner, Silvia Sellán, Hsueh-Ti Derek Liu, and Alec Jacobson. 2023. Constructive Solid Geometry on Neural Signed Distance Fields. In *SIGGRAPH Asia 2023 Conference Papers*. 1–12.

Ken Museth, David E Breen, Ross T Whitaker, and Alan H Barr. 2002. Level set surface editing operators. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 330–338.

Nervous System Studio. 2012. Cellular Lamp. https://www.thingiverse.com/thing:19104.

Niederösterreich 3D. 2020. Löwe. https://sketchfab.com/3d-models/lowe-4522a4cdc1c14190bf1a8811fa27da32.

Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. 2005. Multi-level partition of unity implicits. In *Acm Siggraph 2005 Courses*. 173–es.

Stanley Osher and Ronald Fedkiw. 2003. *Level set methods and dynamic implicit surfaces*. Springer New York. https://doi.org/10.1007/b98879

Stanley Osher and Ronald P Fedkiw. 2005. *Level set methods and dynamic implicit surfaces*. Vol. 1. Springer New York.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 165–174.

Mark Pauly, Niloy J Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J Guibas. 2008. Discovering structural regularity in 3D geometry. (2008), 1–11.

Oussama Remil, Qian Xie, Xingyu Xie, Kai Xu, and Jun Wang. 2017. Surface reconstruction with data-driven exemplar priors. 88 (2017), 31–41.

Ronan Murphy. 2022. Elden Ring - Turtle Pope. https://www.thingiverse.com/thing:5373873.

Roy Mac. 2020. RPG mini - Human Bow Captain Female. https://www.thingiverse.com/thing:4612035.

Ruwen Schnabel, Patrick Degener, and Reinhard Klein. 2009. Completion and reconstruction with primitive shapes, Vol. 28. Wiley Online Library, 503–512.

Silvia Sellán, Noam Aigerman, and Alec Jacobson. 2021. Swept volumes via spacetime numerical continuation. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–11.

Silvia Sellán, Christopher Batty, and Oded Stein. 2023. Reach For the Spheres: Tangency-Aware Surface Reconstruction of SDFs. In *SIGGRAPH Asia 2023 Conference Papers*. Article 73, 11 pages.

Silvia Sellán and Alec Jacobson. 2022. Stochastic Poisson Surface Reconstruction. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–12.

Silvia Sellán and Alec Jacobson. 2023. Neural Stochastic Poisson Surface Reconstruction. In *SIGGRAPH Asia 2023 Conference Papers*. 1–9.

Silvia Sellán and Oded Stein. 2023. gpytoolbox: A Python Geometry Processing Toolbox. https://gpytoolbox.org/.

James A Sethian et al. 1999. *Level set methods and fast marching methods*. Vol. 98. Cambridge Cambridge UP.

Andrei Sharf, Thomas Lewiner, Gil Shklarski, Sivan Toledo, and Daniel Cohen-Or. 2007. Interactive topology-aware surface reconstruction. 26, 3 (2007), 43–es.

Nicholas Sharp and Alec Jacobson. 2022. Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–16.

Nicholas Sharp and Maks Ovsjanikov. 2020. Pointtrinet: Learned triangulation of 3d point sets. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*. Springer, 762–778.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems* 34 (2021), 6087–6101.

Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. 2023. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–16.

Barton T. Stander and John C. Hart. 1997. Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. 279–286. https://doi.org/10.1145/258734.258868

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11358–11367.

The Stanford 3D Scanning Repository. 1994. Stanford Bunny. http://graphics.stanford.edu/data/3Dscanrep/.

Thunk3D scanner. 2019. koala bear. https://sketchfab.com/3d-models/koala-bear-221d8d6519944a65b473ea56fc032570.

Kees Van Overveld and Brian Wyvill. 2004. Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface. *The visual computer* 20 (2004), 362–379. https://doi.org/10.1007/s00371-002-0197-4

Luminita A Vese and Tony F Chan. 2002. A multiphase level set framework for image segmentation using the Mumford and Shah model. *International journal of computer vision* 50 (2002), 271–293.

Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable signed distance function rendering. *ACM Trans. Graph.* 41, 4, Article 125 (2022), 18 pages. https://doi.org/10.1145/3528223.3530139

Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. 2019. Deep geometric prior for surface reconstruction. 10130–10139.

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2009. Deforming meshes that split and merge. In *ACM SIGGRAPH 2009 papers*. 1–10.

Yahoo! JAPAN. 2013. Skull. https://www.thingiverse.com/yahoojapan/designs.

YahooJAPAN. 2013. Scorpion. https://www.thingiverse.com/thing:182363.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).

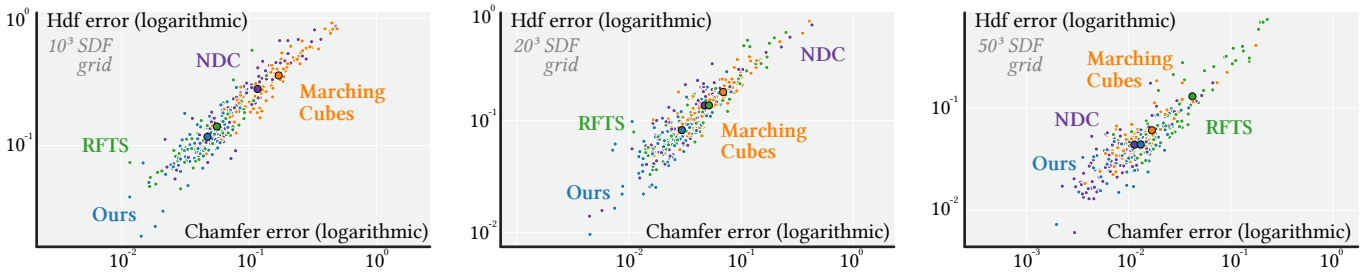


Figure 19: Each dot corresponds to one shape in our one-hundred-shape experiment. Our algorithm's error is consistently below our main competitors at lower resolutions (left, middle) and closely matches neural approaches at higher resolutions (right). Colored circles indicate averaged errors.

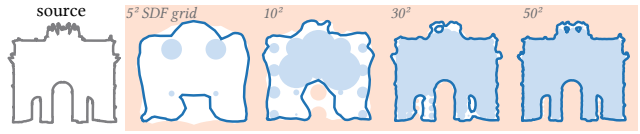


Figure 20: Our approach manages to use even very small amounts of global information to reconstruct a surface, thus producing good rough shapes even when almost no data is available at all. As the available data increases, our method reconstructs the ground truth more and more faithfully.

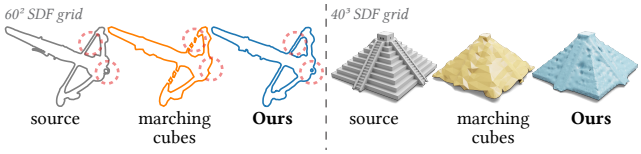


Figure 21: Our method reconstructs thin features better than marching cubes *(left)*, but smooths out sharp features much like marching cubes *(right)*.

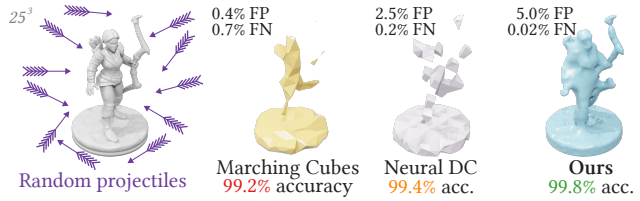


Figure 22: As an additional error metric, we study the accuracy of our method to detect collisions with randomly sampled projectiles. Our algorithm produces a more conservative reconstruction from the SDF data, producing a very low number of false negatives and higher overall accuracy.

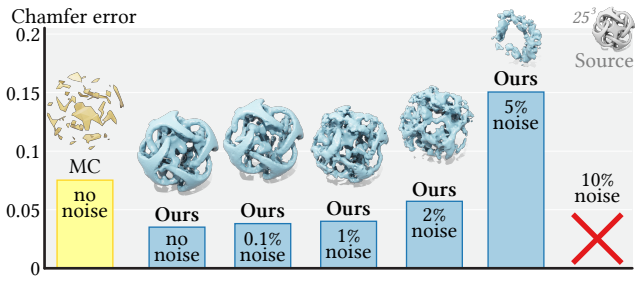


Figure 23: Our reconstruction is relatively robust to moderate noise in the input SDF, even outperforming MC (with noise-free input) for up to 2% noise amplitude. However, at large amplitudes, our algorithm cannot find feasible tangency sets and can fail (see rightmost bars).

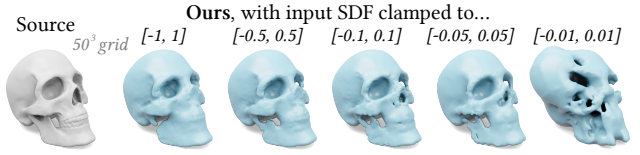


Figure 24: Our method can handle *clamped SDFs* with only minor modifications, where we trust the distance data only within certain bounds. As the clamps get tighter, our method degrades gracefully, until the very end, where not much information is left to aid with reconstruction.

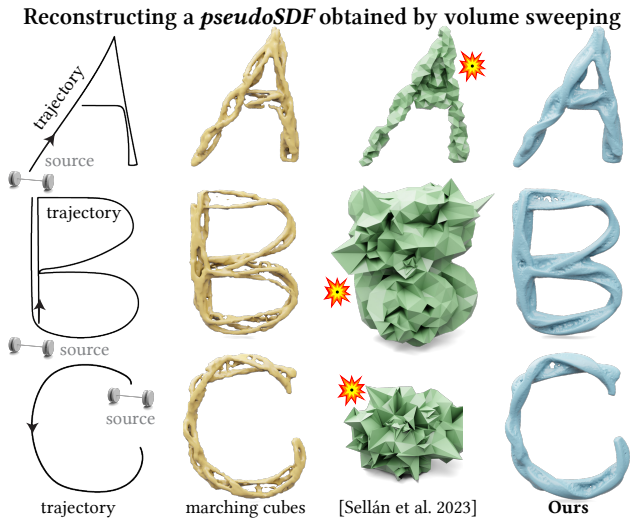


Figure 25: Our algorithm can be used to reconstruct surfaces from *pseudo*-SDFs like those obtained from sweeping closed surfaces and studied by Marschner et al. [2023]. We recover thin features not captured by MC (left) and avoid the singularities encountered in all three cases by RFTS.

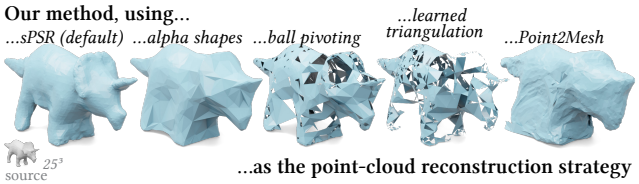


Figure 26: Our choice of point-cloud-to-surface reconstruction method is incidental, and can be made differently depending on the application and the prior knowledge about the reconstruction surface. By default, we choose sPSR [Kazhdan and Hoppe 2013], but also experiment with alpha shapes [Edelsbrunner et al. 1983], ball pivoting [Bernardini et al. 1999], learned triangulations [Sharp and Ovsjanikov 2020] and Point2Mesh [Hanocka et al. 2020].

SUPPLEMENTAL MATERIAL

Parameters

By default, we run our algorithm with $k_{\max} = 10$ fine-tuning iterations, $b = 10,000$ batch size, 10.0 sPSR screening weight, $n_{\max} = 3$ maximum points added per sphere and 20 local search iterations. Below, we specify the non-default parameters used for some of the figures in the paper:

Figure 1, Figure 5 and Figure 6. $k_{\max} = 50$.

Figure 4, Figure 20 and Figure 22. $k_{\max} = 50$, $n_{\max} = 50$.

The 2D results for Sellán et al. [2023] are limited to 100 iterations, since the official implementation does not terminate on 2D blow-ups and iterations get very expensive.

Figure 11 and Figure 24. $k_{\max} = 20$.

Figure 9. $k_{\max} = 20$, $n_{\max} = 30$.

Figure 13. $n_{\max} = 30$.

Figure 16. $k_{\max} = 100$, $n_{\max} = 16$.

Figure 26. $k_{\max} = 5$.

Pseudocode Implementation

This section features pseudocode for the major routines mentioned in Section 4 in Algorithms 1, 2, and 3.

Detailed Quantitive Evaluation Data

The full results of our large-scale experiment over nine different resolutions and one hundred different shapes are provided in the supplemental file *results.csv*. The average errors over all shapes are also presented in Table 2 for ease of reference.

Algorithm 1 Reach for the Arcs

function REACH_FOR_THE_ARCS($\{p_1, \dots, p_n\}$, $\{s_1, \dots, s_n\}$)
  separate sphere indices i into $\{\mathcal{S}_i^{(I)}\}_i$ (inside) and $\{\mathcal{S}_i^{(O)}\}_i$ (outside)
  for $\sigma = I, O$ **do**
    $\mathbf{w}^{(\sigma)} \leftarrow \text{rasterize}\left(\{\mathcal{S}_i^{(\sigma)}\}_i\right)$
    for $\mathcal{S}_i^{(\sigma)} \in \{\mathcal{S}_i^{(\sigma)}\}_i$ **do**
      $w^i \leftarrow \text{argmin}_{j \in \mathbf{w}^{(\sigma)}} d(w_j, \mathcal{S}_i^{(\sigma)})$
      $w^i \leftarrow \text{make\_feasible}\left(w^i, i, \text{p}(\mathcal{S}_i^{(\sigma)}), \text{s}(\mathcal{S}_i^{(\sigma)})\right)$
  $\mathbf{a}^0 \leftarrow \{w^1, \dots, w^n\}$
  for $k = 1, \dots, k_{\max}$ **do**
    $\mathbf{a}^k \leftarrow \text{fine\_tune}(\mathbf{a}^{k-1}, \{p_1, \dots, p_n\}, \{s_1, \dots, s_n\})$
  return point_cloud_to_mesh($\mathbf{a}^{k_{\max}}$)

Algorithm 2 Make feasible

function MAKE_FEASIBLE(w, i, $\{p_1, \dots, p_n\}$, $\{s_1, \dots, s_n\}$)
  for 20 times **do**
    if w not within any sphere **then**
      break
    $\mathbf{B} \leftarrow$ up to k_{search} spheres containing w
    $\mathbf{C} \leftarrow \{\}$
    if $d = 2$ or $|\mathbf{B}| = 2$ **then**
      for $(b, r) \in \mathbf{B}$ **do**
        $C \leftarrow C \cup$ {intersections of (p_i, s_i) with (b, r) not within any sphere}
    else if $d = 3$ **then**
      for $(b_1, r_1), (b_2, r_2) \in \mathbf{B}$ **do**
        $C \leftarrow C \cup$ {intersections of (p_i, s_i) with (b_1, r_1) and (b_2, r_2) not within any sphere}
    $w \leftarrow \text{argmin}_{v \in C} d(v, w)$
  return w

Algorithm 3 Fine tune

function FINE_TUNE($\mathbf{a}$, $\{p_1, \dots, p_n\}$, $\{s_1, \dots, s_n\}$)
  Ω = point_cloud_to_mesh($\mathbf{a}$)
  for i = 1, ..., n **do**
    $\mathbf{a}_i \leftarrow \{a \in \mathbf{a} \mid \text{idx}(a) = i\}$
    $c_i \leftarrow$ closest point on Ω to p_i
    if c_i on sphere (p_i, s_i) **then**
      break
    $q_i \leftarrow \text{proj\_onto}(c_i, p_i, s_i)$
      ▷ idx returns the sphere index of a tangency point
    if c_i outside sphere (p_i, s_i) **then**
      $\mathbf{a}_i \leftarrow \{\text{transport}(\text{argmin}_{a \in \mathbf{a}_i} d(a, q_i), q_i, p_i, s_i)\}$
    else if c_i inside sphere (p_i, s_i) **then**
      $a_{\text{new}} \leftarrow \text{transport}(\text{argmin}_{a \in \mathbf{a}_i} d(a, q_i), q_i, p_i, s_i)$
      for $a_i^{(1)}, \dots, a_i^{(\kappa)} \in \mathbf{a}_i$ **do**
        $c_i^{(j)} \leftarrow$ closest point on Ω to $a_i^{(j)}$
        $q_i^{(j)} \leftarrow \text{proj\_onto}(c_i^{(j)}, p_i, s_i)$
        $a_{\text{new}}^{(j)} \leftarrow \text{transport}(a_i^{(j)}, q_i, p_i, s_i)$
      if c_i only barely inside (p_i, s_i) or $\kappa = \kappa_{\max}$ **then**
        $\mathbf{a}_i \leftarrow \{a_{\text{new}}, a_{\text{new}}^{(1)}, \dots, a_{\text{new}}^{(\kappa-1)}\}$
      else
        $\mathbf{a}_i \leftarrow \{a_{\text{new}}, a_{\text{new}}^{(1)}, \dots, a_{\text{new}}^{(\kappa)}\}$
  $\mathbf{a} \leftarrow \mathbf{a}_1 \cup \dots \cup \mathbf{a}_n$
  for $a \in \mathbf{a}$ **do**
    $a \leftarrow \text{make\_feasible}(a, \text{idx}(a), \{p_1, \dots, p_n\}, \{s_1, \dots, s_n\})$
  return $\mathbf{a}$
function PROJ_ONTO(c, p, s)
  $t \leftarrow$ signed distance from c to p
  $q \leftarrow p + \text{sign}(ts)|s| \frac{c-p}{\|c-p\|}$
  return q
function TRANSPORT(x, y, p, s)
  $\tau \leftarrow 0.01$
  repeat
    $z \leftarrow x$ transported τ towards y along geodesic on (p, s)
    $\tau \leftarrow \tau/2$
  until z not within any sphere
  return z

Table 2: Reconstruction accuracy of our method compared to Marching Cubes [Lorensen and Cline 1998], Neural Dual Contouring [Chen et al. 2022] and *Reach for the Spheres* [Sellán et al. 2023] on one hundred randomly selected inputs from Thingi10K [Zhou and Jacobson 2016]. Accuracy is measured via Chamfer error (left), Hausdorff distance (middle) and the SDF energy introduced by Sellán et al. [2023] (right). Our algorithm, which requires no training data, outperforms every other method at low and medium resolutions, and is only narrowly bested by neural approaches at higher resolutions.

Grid	Chf MC	Chf NDC	Chf RFTS	Chf RFTA	Hdf MC	Hdf NDC	Hdf RFTS	Hdf RFTA	$\mathcal{E}_{SDF}$ MC	$\mathcal{E}_{SDF}$ NDC	$\mathcal{E}_{SDF}$ RFTS	$\mathcal{E}_{SDF}$ RFTA
Average results over all test shapes												
6^3	0.2644	0.1852	0.0865	**0.0672**	0.4837	0.3873	0.1956	**0.1519**	64.1815	25.2839	4.0003	**0.0611**
10^3	0.1706	0.1166	0.0559	**0.0472**	0.3546	0.2778	0.1410	**0.1169**	28.2937	13.1524	0.3003	**0.0271**
20^3	0.0688	0.0470	0.0514	**0.0295**	0.1799	0.1366	0.1368	**0.0822**	5.3163	2.9343	1.9020	**0.0125**
30^3	0.0322	0.0217	0.0497	**0.0203**	0.1038	0.0753	0.1395	**0.0614**	1.0028	0.1903	2.7906	**0.0057**
40^3	0.0234	0.0166	0.0521	**0.0156**	0.0811	0.0593	0.1414	**0.0491**	0.8186	0.2274	2.7921	**0.0031**
50^3	0.0169	**0.0114**	0.0421	0.0131	0.0611	**0.0440**	0.1311	0.0442	0.4964	0.0586	2.7342	**0.0025**
60^3	0.0129	**0.0087**	0.0423	0.0109	0.0502	**0.0370**	0.1247	0.0390	0.3190	0.0369	2.9077	**0.0020**
80^3	0.0074	**0.0062**	0.0351	0.0084	**0.0325**	0.0325	0.1056	0.0332	0.0598	0.0612	2.6553	**0.0015**
100^3	0.0054	**0.0040**	0.0332	0.0069	0.0294	**0.0203**	0.1209	0.0356	0.0507	0.0092	2.7148	**0.0013**
Average results over all test shapes with genus zero												
6^3	0.2733	0.1607	0.0582	**0.0570**	0.4963	0.3450	0.1319	**0.1285**	67.7507	18.6811	0.0737	**0.0399**
10^3	0.1525	0.0893	0.0439	**0.0412**	0.3272	0.2360	0.1160	**0.0956**	20.3723	6.9478	0.1881	**0.0186**
20^3	0.0530	0.0304	0.0364	**0.0254**	0.1461	0.0977	0.0996	**0.0662**	2.9292	0.5201	1.0359	**0.0098**
30^3	0.0265	**0.0182**	0.0307	0.0183	0.0942	0.0596	0.0950	**0.0539**	0.7241	0.1472	0.2928	**0.0076**
40^3	0.0227	0.0174	0.0361	**0.0138**	0.0834	0.0607	0.1003	**0.0407**	1.0747	0.2745	1.2115	**0.0034**
50^3	0.0170	**0.0103**	0.0188	0.0119	0.0606	**0.0372**	0.0572	0.0381	0.9437	0.0344	0.0465	**0.0029**
60^3	0.0139	**0.0071**	0.0170	0.0099	0.0501	**0.0293**	0.0529	0.0341	0.6008	0.0160	0.1912	**0.0024**
80^3	0.0063	**0.0048**	0.0158	0.0070	0.0293	**0.0201**	0.0537	0.0293	0.0499	0.0109	0.4676	**0.0016**
100^3	0.0036	**0.0024**	0.0142	0.0049	0.0233	**0.0148**	0.0602	0.0330	0.0194	0.0031	0.4471	**0.0011**
Average results over all test shapes with genus over zero												
6^3	0.2598	0.1980	0.1014	**0.0725**	0.4771	0.4096	0.2291	**0.1642**	62.3076	28.7504	6.0618	**0.0723**
10^3	0.1790	0.1293	0.0615	**0.0499**	0.3674	0.2972	0.1527	**0.1268**	31.9904	16.0479	0.3527	**0.0311**
20^3	0.0760	0.0546	0.0583	**0.0314**	0.1955	0.1544	0.1540	**0.0896**	6.4152	4.0456	2.3007	**0.0137**
30^3	0.0350	0.0233	0.0588	**0.0213**	0.1084	0.0828	0.1606	**0.0650**	1.1353	0.2108	3.9780	**0.0048**
40^3	0.0238	**0.0162**	0.0593	0.0164	0.0800	0.0587	0.1600	**0.0529**	0.7026	0.2061	3.5082	**0.0030**
50^3	0.0169	**0.0119**	0.0539	0.0138	0.0613	0.0475	0.1686	**0.0473**	0.2688	0.0709	4.1016	**0.0023**
60^3	0.0125	**0.0095**	0.0546	0.0114	0.0503	**0.0406**	0.1594	0.0413	0.1828	0.0470	4.2207	**0.0018**
80^3	0.0080	**0.0071**	0.0460	0.0092	**0.0342**	0.0396	0.1352	0.0353	0.0655	0.0898	3.8992	**0.0015**
100^3	0.0062	**0.0047**	0.0406	0.0077	0.0318	**0.0224**	0.1448	0.0367	0.0630	0.0116	3.6081	**0.0014**
Average results over all test shapes with genus over ten												
6^3	0.3101	0.2243	**0.0815**	0.0859	0.6170	0.4880	0.1677	**0.1664**	89.2837	32.3581	0.2148	**0.0665**
10^3	0.1998	0.1463	0.0681	**0.0587**	0.4105	0.3627	0.1500	**0.1327**	35.0728	12.8121	0.2259	**0.0261**
20^3	0.1049	0.0856	0.0687	**0.0347**	0.2667	0.2079	0.1637	**0.0912**	12.4083	12.0323	1.0960	**0.0105**
30^3	0.0415	**0.0268**	0.0836	0.0292	0.1170	0.0885	0.2486	**0.0806**	1.4963	0.1589	8.6610	**0.0072**
40^3	0.0318	0.0239	0.0704	**0.0222**	0.0922	0.0782	0.1839	**0.0638**	0.9870	0.4756	3.8019	**0.0040**
50^3	0.0217	**0.0160**	0.1058	0.0175	0.0666	**0.0500**	0.3309	0.0555	0.3969	0.0419	12.0395	**0.0031**
60^3	0.0164	**0.0139**	0.0794	0.0144	0.0582	0.0562	0.2172	**0.0498**	0.3163	0.1029	5.7565	**0.0026**
80^3	0.0115	**0.0108**	0.0522	0.0116	0.0467	0.0475	0.1391	**0.0426**	0.0968	0.0958	2.9109	**0.0023**
100^3	0.0095	**0.0072**	0.0702	0.0095	0.0509	**0.0311**	0.2272	0.0409	0.1829	0.0242	4.2791	**0.0020**