A Neural Galerkin Solver for Accurate Surface Reconstruction

JIAHUI HUANG, HAO-XIANG CHEN, and SHI-MIN HU*,
BNRist, Department of Computer Science and Technology, Tsinghua University, China

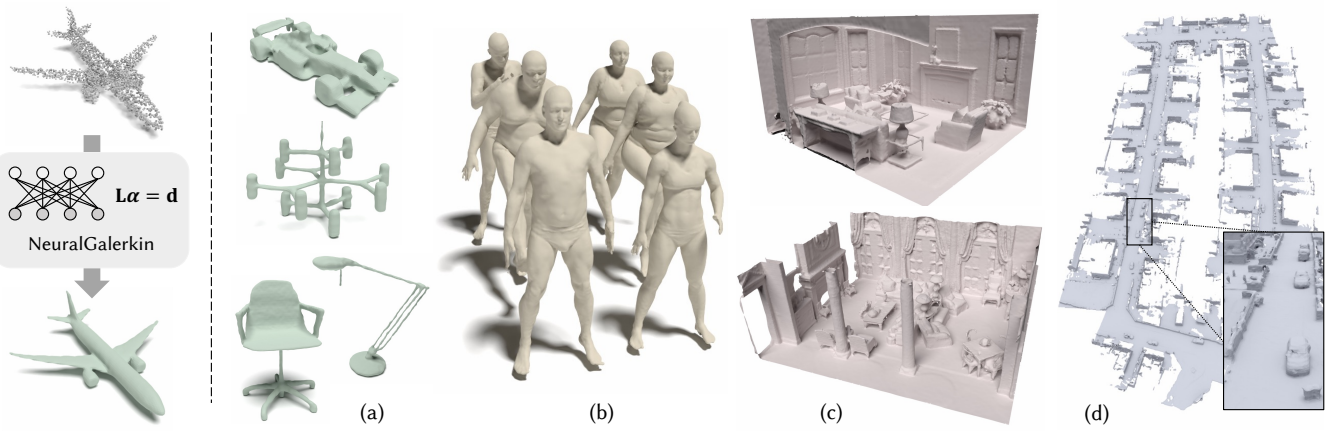


Fig. 1. Our method, termed as NeuralGalerkin, reconstructs surface meshes from point cloud data, using a combination of an adaptive sparse convolutional neural network and a fully differentiable solver to obtain an optimal implicit surface that fits the input (with optional normals). It is both accurate and scalable, being able to reconstruct (a) man-made objects (3K input points), (b) highly-detailed human bodies (10K input points), (c) indoor scene scans (100K input points), and (d) outdoor LiDAR captures (>1M input points), only within a few seconds.

To reconstruct meshes from the widely-available 3D point cloud data, implicit shape representation is among the primary choices as an intermediate form due to its superior representation power and robustness in topological optimizations. Although different parameterizations of the implicit fields have been explored to model the underlying geometry, there is no explicit mechanism to ensure the fitting tightness of the surface to the input. We present in response, NeuralGalerkin, a neural Galerkin-method-based solver designed for reconstructing highly-accurate surfaces from the input point clouds. NeuralGalerkin internally discretizes the target implicit field as a linear combination of a set of spatially-varying basis functions inferred by an adaptive sparse convolution neural network. It then solves differentiably for a variational problem that incorporates both positional and normal constraints from the data in closed form within a single forward pass, highly respecting the raw input points. The reconstructed surface extracted from the implicit interpolants is hence very accurate and incorporates useful inductive biases benefiting from the training data. Extensive evaluations on various datasets demonstrate our method's promising reconstruction performance and scalability.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **3D reconstruction**;

Additional Key Words and Phrases: surface reconstruction, point cloud, neural networks

ACM Reference Format:
Jiahui Huang, Hao-Xiang Chen, and Shi-Min Hu. 2022. A Neural Galerkin Solver for Accurate Surface Reconstruction. *ACM Trans. Graph.* 41, 6, Article 229 (December 2022), 16 pages. https://doi.org/10.1145/3550454.3555457

*Shi-Min Hu is the corresponding author.

Authors' address: Jiahui Huang, huang-jh18@mails.tsinghua.edu.cn; Hao-Xiang Chen, chx20@mails.tsinghua.edu.cn; Shi-Min Hu, shimin@tsinghua.edu.cn, BNRist, Department of Computer Science and Technology, Tsinghua University, China.

0730-0301/2022/12-ART229
https://doi.org/10.1145/3550454.3555457

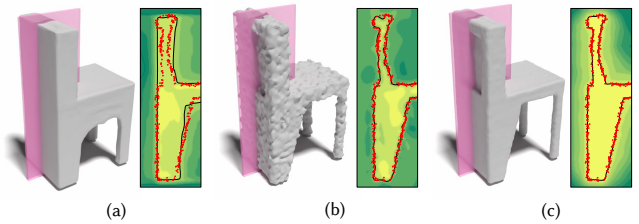
1 INTRODUCTION

Reconstructing 3D shapes from point cloud data is a long-standing problem in Computer Graphics. With its recent wide application in robotics, AR/VR, and assets digitalization, the problem of reconstructing highly-accurate geometry in a fast and robust way has attracted unprecedented attention from researchers. However, various requirements in these applications have posed numerous challenges to existing algorithms, including data sparsity, noisy measurements and surface occlusions, etc., making the task of reconstruction still an open problem.

At the core of an effective reconstruction algorithm is the underlying 3D representation. In comparison to staggered voxel grids, parametric surfaces, or solid geometries, implicit shape representation has been demonstrated to be very flexible due to its friendliness to topology changes and strong representation power, and has been used in many modern reconstruction algorithms. In such a representation, the surface $\mathcal{S}$ is defined by the level set of a function f, called an implicit field. Specifically,

$$\mathcal{S} := \{\boldsymbol{p} \in \mathbb{R}^3 \mid f(\boldsymbol{p}) = \delta\}, \quad f : \mathbb{R}^3 \mapsto \mathbb{R}, \tag{1}$$

where δ is usually a predefined constant level-set value. Discrete surface meshes can be easily extracted later using contouring algorithms for further processing.



(a) (b) (c)

Fig. 2. Comparisons of different implicit field fitting strategies. Contour maps of the field within the pink slices are shown on the right insets, where red points are the input points, and the black trace shows the extracted level set. (a) Methods (*e.g.*, [Mescheder et al. 2019]) with strong global priors can only roughly fit the general structure. (b) Traditional geometry-based approaches mistakenly interpolate the noisy data, unaware of the underlying semantics. (c) Our approach reaches an elegant balance between the two and provides the best and most intuitive fit.

To obtain the optimal function, traditional classic methods (*e.g.*, [Calakli and Taubin 2011; Carr et al. 2001]) aim to interpolate the input data faithfully and incorporate hand-crafted regularizations to encourage surface smoothness while preserving sharp features. However, designing such regularizations itself is challenging and inevitably leads to undesirable artifacts under the presence of large noise, because no high-level understanding of the input is exploited. Empowered by deep networks, modern reconstruction methods are being actively developed: On the one hand, the internal 'structure prior' [Williams et al. 2019] of the network function is utilized, and a smooth fitting can be achieved by simple overfitting [Gropp et al. 2020], but a long optimization time is needed due to the complexity of the function. On the other, the networks are instructed to learn geometric priors from a large corpus of training data [Choy et al. 2016; Mescheder et al. 2019], so that one can achieve direct feed-forward inference over the input. However, there is no mechanism ensuring that the learned surface actually 'respects' and fits the input [Tatarchenko et al. 2019] and how accurate the reconstruction can achieve, being biased by the learned priors.

The above dilemma, visualized in Fig. 2, motivates our approach. We try to answer the question of how to obtain accurate data fitting with sufficient tightness while leveraging data-driven inductive biases effectively. Most similar to our work is [Williams et al. 2021a], but their use of a dense kernel prevents their method from scaling to large-scale inputs. Hence our method also aims toward good scalability that could tackle millions of points.

Specifically, we present NeuralGalerkin, a neural Galerkin-method-based solver for accurate and scalable 3D reconstruction from point clouds. We parameterize the implicit field f as a linear combination of a spatially-varying set of bases composed of elementary functions, following the Galerkin method [Galerkin 1915] (as finite elements). The proposed method is internally composed of two tightly-coupled components: the surface fitting *solver* (§ 3) and the adaptive sparse convolutional *network* (§ 4). Given the input, the *network* applies sparse convolutions and adaptive structural prediction to generate a voxelized multi-scale scaffold of the geometry, where the inferred normal data and the basis functions are stored. The *solver* part then tries to solve in the closed form a surface fitting variational problem that takes both the inferred data and the raw input points into account, producing the coefficients of the basis functions that finally compose the target implicit field. The solver is fully differentiable and our whole pipeline can be trained end-to-end with simple supervision. Different from the meta-learning [Finn et al. 2017] where nested optimization is also used, our solver only requires solving a single sparse linear system, hence being much more efficient. A similar technique that uses the solver is concurrently explored in the mesh processing community [Aigerman et al. 2022], demonstrating the wide applicability of such methods. We demonstrate the efficacy of our algorithm through extensive evaluations on datasets captured at various scales and show superior performance on different reconstruction tasks, as visualized in Fig. 1.

In summary, we contribute:

(1) a fully differentiable closed-form surface fitting solver that solves for an optimal implicit field respecting input points,
(2) an adaptive sparse network producing the multi-scale learned geometric priors that are injected into the solver, and
(3) thorough experiments conducted on all scales of datasets showing state-of-the-art performance on reconstruction accuracy.

2 RELATED WORKS

This section reviews both classic and the most recent learning-based surface reconstruction methods. As the more general 'Shape-from-X' is a large field in Graphics, we only review point-cloud-based approaches, which is our main focus. Readers are referred to more systematic surveys as [Jin et al. 2020], [Huang et al. 2022] and [Li et al. 2022] on surface reconstruction for a more detailed review.

2.1 Non-learned surface reconstruction

One of the earliest and most renowned methods from [Curless and Levoy 1996] uses an analytic volumetric running mean fusion scheme to convert scans into 3D models. To gain more robustness under noisy or corrupted data, [Carr et al. 2001] introduces the radial basis function interpolant and solves the reconstruction problem under low-rank smoothness regularizations. MPU [Ohtake et al. 2003] takes a more explicit approach that models the shape by a union of multiple parametric surfaces fitted using local geometries. SSD [Calakli and Taubin 2011] approximates the signed distance field by solving a variational problem with linearly parameterized families of functions. Our method is inspired by the approach of Poisson reconstruction [Kazhdan et al. 2006] and its screened version [Kazhdan and Hoppe 2013], where PDEs are solved to recover from Hermite measurements. Such methods are empirically demonstrated to be well-performing and robust under various conditions and have been applied in many modern systems, *e.g.*, [Vizzo et al. 2021].

2.2 Learning-based shape modeling

Enabled by the power of deep networks and the unprecedented growth of large-scale 3D datasets, various learning-based modeling approaches are designed to effectively utilize the data prior for better reconstruction quality. Different 3D representations are exploited for a stronger representation power. Early works like [Choy

et al. 2016] use a dense voxel grid to represent occupancy information of the shapes. Adaptive voxel grids [Wang et al. 2017] or local variations inside each voxel [Liao et al. 2018] are later devloped for higher resolution outputs. [Hanocka et al. 2020] shows that deformation over template meshes can lead to good reconstruction quality by constraining the (self-)priors naturally on the manifold of surfaces. The most recent implicit representation uses a single feed-forward network to parameterize the geometry [Mescheder et al. 2019; Park et al. 2019] and devises novel constraints [Atzmon and Lipman 2020a; Lipman 2021] to regularize training. These representations are further extended to hybrid ones where local details are preserved, with the help of explicit voxel grids [Peng et al. 2020; Tang et al. 2021b], point cloud [Erler et al. 2020], or octrees [Tang et al. 2021a]. Dynamic reconstruction methods [Song et al. 2021] are also made available thanks to them. Although our representation is also implicit-function-based, it is instead parameterized by a set of simple basis functions that admit efficient optimizations.

2.3 Learning-based scene-scale reconstruction

While shape-level modeling leverages the global priors defined over the entire geometry, it is found to be more effective to model local priors for larger scenes. [Chen et al. 2021; Genova et al. 2020; Jiang et al. 2020] compose the scene with multiple independent implicit networks with limited supports. The local representation is further extended in [Huang et al. 2021] to achieve real-time tracking and mapping efficiently. To enable information exchange between neighboring voxels, [Chen et al. 2022] uses sparse convolution to enforce global consistency. [Ummenhofer and Koltun 2021] designs a novel adaptive convolution scheme that allows for multi-scale implicit field modeling. The octree network from [Wang et al. 2017] is further extended by [Wang et al. 2022] to a dual form that enhances feature propagation within the large-scale network. Approaches like [Azinović et al. 2021] exploit differentiable rendering to optimize the reconstructed geometry. The hashtable encoding strategies used in [Müller et al. 2022] could complete the optimizations within a few minutes. Notably, there is a growing interest in unsigned distance field (UDF) modeling [Chibane et al. 2020b; Ye et al. 2022] because scene-scale geometries are usually open surfaces. However, a standard mesh extraction method from UDF is still under-explored [Guillard et al. 2021].

2.4 Neural implicit representations

Parametrizing the shape using neural networks has been extensively studied in recent literature. On the one hand, the high complexity of the neural functions renders it necessary to apply appropriate regularizations to make the shape smooth [Gropp et al. 2020], while on the other hand, biases towards either feature frequencies [Sitzmann et al. 2020; Tancik et al. 2020] or multi-scales [Martel et al. 2021; Takikawa et al. 2021] are injected to improve geometric details in certain regions. Another line of work deals with implicit fields in a more principled manner: [Williams et al. 2021b] derives a solution to fit an infinitely-wide network to interpolate point data using the kernel method. The method is further extended to a learnable version in [Williams et al. 2021a]. Our method also aims at fitting an implicit field. Instead of using neural networks directly for the shape representation, we use a spatially-varying set of basis functions and solve the surface fitting problem in closed form. We show empirically that the representation power is similar, yet one could easily obtain an optimal fit without time-consuming optimizations.

3 SURFACE FITTING SOLVER

As an overview, our task of surface reconstruction takes a set of points $\mathcal{P} := \{\boldsymbol{p} \in \mathbb{R}^3\}$ as input, and aims at building a continuous implicit function f whose δ-level-set extracted as in Eq (1) well fits the geometry of $\mathcal{P}$. Mesh can be later built using the standard marching cubes [Lorensen and Cline 1987] algorithm. Our proposed method is composed of two tightly-coupled components, working in synergy: a surface fitting *solver* and an adaptive sparse convolutional neural *network*. As illustrated in Fig. 3, we first feed the point cloud into a learnable network that predicts useful priors about the input data. These predictions are then injected into the surface fitting solver, obtaining the best implicit field possible that respects both the point measurements and the learned data priors. The pipeline is fully differentiable that enables end-to-end optimizations.

In this section, we first introduce the differentiable formulation of the surface fitting solver that acts as the core component in our full pipeline. Specifically, we present the target energy minimization problem we want to solve in § 3.1. To tackle the problem in closed-form, we detail how we discretize the variables and the solution in § 3.2, and how the discretized linear system is efficiently solved in § 3.3. Details about the learnable parameters and their training strategies will be summarized in § 4.

3.1 Energy Formulation

Inspired by SPSR [Kazhdan and Hoppe 2013], we choose to solve a variational problem $f^\star := \arg\min_f E(f)$ for the best implicit function that represents the surface, where:

$$E(f) := \iiint_\Omega \left\| \nabla f - \vec{\mathbf{N}} \right\|_2^2 \mathrm{d}V + \lambda \sum_{\boldsymbol{p}\in\mathcal{P}} \left(f(\boldsymbol{p}) - \xi(\boldsymbol{p})\right)^2 . \qquad (2)$$

Here, the first term integrates the norm of the difference between f's derivative ∇f and a normal field $\vec{\mathbf{N}}$, defined in the 3D domain of Ω. The second term is a discrete data fitting term (*c.f.* a screening term) that encourages f to fit some predefined scalar value $\xi(\boldsymbol{p})$ at all input positions in $\mathcal{P}$. The two terms are balanced with a constant coefficient $\lambda \in \mathbb{R}^+$.

In Eq (2), the first term tries to recover the implicit field through its derivative. As we encourage $\vec{\mathbf{N}}$ to produce non-zero values only at the shape boundary, the recovered field is approximately a smoothed indicator function. Integrated within the entire domain, the term provides an overall orientational estimate of the general shape, strongly regularized by the normal field. In contrast, the second term explicitly considers input point measurements, strictly pushing the surface towards the points if $\xi(\boldsymbol{p}) \equiv \delta$. In regions with complicated geometric details, these point constraints could effectively enforce adequate fitting tightness, so that surface details can be made clearer. Our formulation could elegantly balance learned priors and input data fitting. While the former provides informative guidance to regularize the surface in the sense of orientation, the

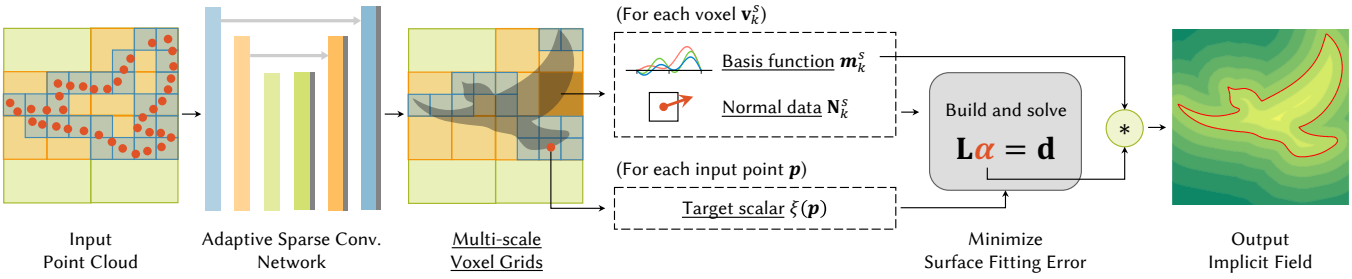


Fig. 3. Pipeline of NeuralGalerkin. The input point cloud is digested by our adaptive sparse convolutional neural network (§ 4), predicting a multi-scale voxel grid structure as well as the per-voxel and per-point priors (underlined). These learned priors are injected into our surface fitting solver that minimizes the surface error in closed-form (§ 3). Our pipeline is fully differentiable and is optimized end-to-end. The demo bird shape is from [Vecteezy 2022].

latter introduces surface details and possible generalizability. The use of $\xi(\boldsymbol{p})$ is mainly for denoising by allowing points to be located outside the surface.

3.2 Implicit Field Discretization

Suppose we are given a set of multi-scale voxel grids enclosing the target surface $\mathcal{S}$, denoted by $\mathcal{V} := \{\mathcal{V}^1, \mathcal{V}^2, ..., \mathcal{V}^S\}$ where S is the number of scales. The voxel grid $\mathcal{V}^s$ at scale s contains a sparse set of voxels $\{\mathbf{v}_k^s \mid k = 1, 2, ...\}$, each with a side length of $2^{s-1}b$, where b is the base voxel size. We constrain that the space covered by $\mathcal{V}^s$ is strictly contained within $\mathcal{V}^{s+1}$. The multi-scale structure resembles the adaptive octree used in [Meagher 1982; Wang et al. 2018], except that the shallow layers up to the root do not exist [Museth 2013].

Given such a scaffold, we adaptively set the integration domain $\Omega := \bigcup_s \bigcup_{\mathbf{v}_k^s \in \mathcal{V}^s} \mathrm{Rgn}(\mathbf{v}_k^s)$, where $\mathrm{Rgn}(\mathbf{v})$ denotes the enlarged region spanned by voxel $\mathbf{v}$ with a factor of 3^3. For the normal field we model it to be piece-wise constant within each voxel:

$$\vec{\mathbf{N}}(\boldsymbol{p}) := \sum_s \mathbf{N}_k^s, \quad \boldsymbol{p} \in \mathbf{v}_k^s, \quad \text{where } \mathbf{N}_k^s \in \mathbb{R}^3. \tag{3}$$

To allow for an efficient solution to the problem in Eq (2), as well as maintain a high flexibility for the surface representation, we define the target implicit function as follows:

$$f(\boldsymbol{p}) := \sum_s \sum_k \alpha_k^s \mathcal{B}_k^s(\boldsymbol{p}), \text{ with } \mathcal{B}_k^s(\boldsymbol{p}) := B_k^s\left(\frac{\boldsymbol{p} - \mathrm{Cntr}(\mathbf{v}_k^s)}{2^{s-1}b}\right), \tag{4}$$

where subscript k denotes the sum over all voxels in $\mathcal{V}^s$ and $\mathrm{Cntr}(\mathbf{v})$ is the centroid coordinate of voxel $\mathbf{v}$. $B_k^s : \mathbb{R}^3 \mapsto \mathbb{R}$ is the *basis function* that is fixed during energy minimization. It is spatially varying and, when weighted by the coefficient $\alpha_k^s \in \mathbb{R}$, describes the entire surface.

Following the Euler-Lagrange equation (detailed derivations in appendix) and the above discretization scheme, we could obtain the *optimal* surface fitting using the *best* possible linear coefficients $\boldsymbol{\alpha} := [..., \alpha_k^s, ...]^\top$ for the implicit field $f^\star$ by solving the equation in *closed-form*:

$$\begin{aligned} &\mathbf{L}\boldsymbol{\alpha} = \mathbf{d}, \\ \text{with} \quad &\mathbf{L}_{(\bar{k},\bar{l})} = \iiint_\Omega \nabla\mathcal{B}_{\bar{k}}^\top \nabla\mathcal{B}_{\bar{l}} \,\mathrm{d}V + \lambda \sum_{\boldsymbol{p}\in\mathcal{P}} \mathcal{B}_{\bar{k}}(\boldsymbol{p})\mathcal{B}_{\bar{l}}(\boldsymbol{p}), \\ &\mathbf{d}_{(\bar{k})} = \iiint_\Omega \nabla\mathcal{B}_{\bar{k}}^\top \vec{\mathbf{N}} \,\mathrm{d}V + \lambda \sum_{\boldsymbol{p}\in\mathcal{P}} \mathcal{B}_{\bar{k}}(\boldsymbol{p})\xi(\boldsymbol{p}), \end{aligned} \tag{5}$$

where the subscripts $\bar{k}, \bar{l}$ denote the row and column index of the matrix. All voxels across *all scales* are jointly considered here, with $\bar{k}, \bar{l}$ indexes into the product of the subscripts k and s (*i.e.*, $\mathcal{B}_k^s \equiv \mathcal{B}_{\bar{k}}$) used earlier.

The choice of the basis functions $B(\boldsymbol{p})$ (or equivalently $\mathcal{B}(\boldsymbol{p})$) is vital for the fitting performance, as they act as local geometries within the finite elements that model the shape. Different from SPSR where a fixed Bézier tensor basis is used, to inject more learned priors into the solution, we use a *parameterized family of basis functions*, whose parameters are predicted by our upstream network. This grants the solver more flexibility to reflect the inductive biases from the learned module.

Selecting a correct basis parameterization is challenging: On the one hand, it should be diverse enough to represent a broad range of functions, while on the other hand, due to the integration in Eq (5) it should not be too complicated to hinder runtime performance. As a fair trade-off we employ:

$$\begin{aligned} B_k^s(\boldsymbol{p}) := B(\boldsymbol{p}; \boldsymbol{m}_k^s) &:= \begin{cases} \prod_{a\in\{x,y,z\}} b^{\mathrm{a}}(a; \boldsymbol{m}_k^s), & \boldsymbol{p} \in \Omega_B \\ 0, & \boldsymbol{p} \notin \Omega_B \end{cases}, \\ b^{\mathrm{x}}(x; \boldsymbol{m}_k^s) &:= \sum_u m_u^{\mathrm{x}} q_u^{\mathrm{x}}(x), \quad m_u^{\mathrm{x}} \in \mathbb{R}, \quad q_u^{\mathrm{x}} : \mathbb{R} \mapsto \mathbb{R}, \end{aligned} \tag{6}$$

with $\boldsymbol{p} = [x, y, z]$ and $\Omega_B := [-1.5, 1.5]^3$ being the support of the basis. We define $b^{\mathrm{y}}(y), b^{\mathrm{z}}(z)$ similarly with $b^{\mathrm{x}}(x)$, and in the following discussions we omit the axes superscript for clarity.

In the above definition, q_u are predefined elementary functions that are blended by m_u which varies across voxels. The vectorized $\boldsymbol{m}_k^s := [..., m_u, ...]^\top$ hence parameterizes the continuous function B_k^s. We choose the set of elementary functions to be the power functions $q_u(x) \in \{1, x, x^2, x^3, ...\}$ that make $b(x)$ a polynomial. The piece-wise definition of $B(\boldsymbol{p})$ further leads us to the constraint of $b(-1.5) = b(1.5) = 0$ and $\frac{\partial b}{\partial x}(-1.5) = \frac{\partial b}{\partial x}(1.5) = 0$ for a C^1-continuous implicit

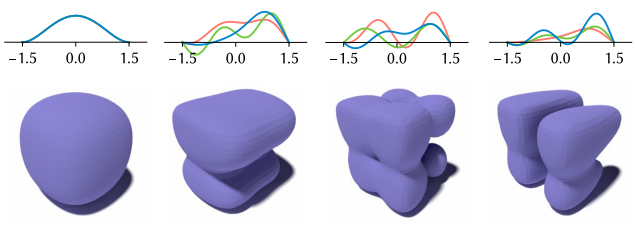
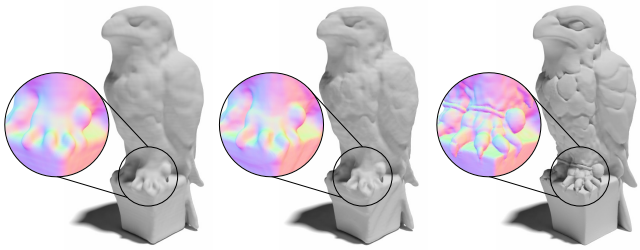


Fig. 4. Basis functions $B(\boldsymbol{p})$ visualized. The first column shows the default one as initialization. The above row shows the overlayed graph of $b^x(x)$, $b^y(y)$ and $b^z(z)$, while the bottom row shows in 3D the mean level set within the region of one basis.



Bézier Tensor Basis
12k voxels of size 0.04^3

Bézier Tensor Basis
50k voxels of size 0.02^3

Our Basis
12k voxels of size 0.04^3

Fig. 5. Improved representation power using our proposed basis functions. Under the same normal field $\vec{\mathbf{N}}$ (left), simply improving the voxel resolutions is less effective in recovering surface details (middle). Our proposed bases introduce more flexibility and successfully overfit the statue (right).

field (implying that the effect of the bases should smoothly fade towards the boundaries), which we impose by setting the coefficients $\boldsymbol{m}_k^s$ to lie within a linear null space of these constraints.

Although the chosen family of bases contains no advanced functions, the ability to represent highly complicated shapes is maintained (*c.f.* Taylor series). Furthermore, to incorporate high-frequency biases, we introduce sine waves to the set of elementary functions, *i.e.*, $q_u(x) \in \{\sin(\frac{\pi n_p}{3}x - \frac{\pi n_p}{2} + \pi),\ n_p \in [1, N_p]\}$. Such a Fourier-style family is widely adopted both in traditional geometric modeling [Zhang 1996] and in modern deep learning [Tancik et al. 2020]. Although this downgrades the C^1-continuity to C^0, empirically it helps training. Visualizations of randomly sampled basis functions are shown in Fig. 4.

Performance-wise, customized bases only have little computational overhead, even with the integral of inner products. This is because the coefficients $\boldsymbol{m}_k^s$ can be factored out under our formulation, and the remaining part of the integral related only to the relative positions of the voxels can be precomputed. Detailed derivations are in the appendix.

3.3 Differentiable Coarse-to-fine Solver

The multi-scale nature of the linear system in Eq (5) enables us to a coarse-to-fine linear solver. It works by dividing the linear system $\mathbf{L}\boldsymbol{\alpha} = \mathbf{d}$ into blocks relating to different scales of the voxels, *i.e.*,

$$\begin{bmatrix} \mathbf{L}^{11} & \mathbf{L}^{12} & \dots & \mathbf{L}^{1S} \\ \mathbf{L}^{21} & \mathbf{L}^{22} & \dots & \mathbf{L}^{2S} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{L}^{S1} & \mathbf{L}^{S2} & \dots & \mathbf{L}^{SS} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}^1 \\ \boldsymbol{\alpha}^2 \\ \vdots \\ \boldsymbol{\alpha}^S \end{bmatrix} = \begin{bmatrix} \mathbf{d}^1 \\ \mathbf{d}^2 \\ \vdots \\ \mathbf{d}^S \end{bmatrix}, \tag{7}$$

and obtaining the solution $\boldsymbol{\alpha}$ in a coarse-to-fine manner. Mathematically equivalent to an upstroke-only V-cycle multigrid solver, the coarse-to-fine approximation is first proposed in [Kazhdan et al. 2006] and demonstrated to be fast and effective for solving Poisson systems.

Our fully differentiable pipeline requires us to back-propagate the gradient through the solver. We hence derive a backpropagation scheme using the rule of implicit function theorem, so that given the derivative of the final loss (§ 4.3) w.r.t. the solution $\boldsymbol{\alpha}$, we obtain its derivative w.r.t. both $\mathbf{L}$ and $\mathbf{d}$. The detailed forward and backward computation of the solver is listed in Alg. 1, with $\otimes$ denoting the cross product.

Algorithm 1: Differentiable coarse-to-fine solver for $\mathbf{L}\boldsymbol{\alpha} = \mathbf{d}$.

Forward Pass

Input : $\{\mathbf{L}^{ss'}\}$ and $\{\mathbf{d}^s\}$.
Output: Solution $\{\boldsymbol{\alpha}^s\}$.
for $s \leftarrow 1$ **to** S **do**
  for $s' \leftarrow 1$ **to** $s-1$ **do**
    $\mathbf{d}^s \leftarrow \mathbf{d}^s - \mathbf{L}^{ss'}\boldsymbol{\alpha}^{s'}$.
  Solve for $\mathbf{L}^{ss}\boldsymbol{\alpha}^s = \mathbf{d}^s$.

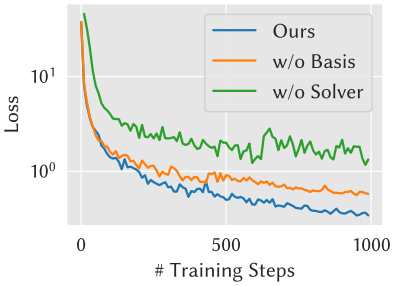
Backward Pass

Input : $\{\nabla\boldsymbol{\alpha}^s\}$.
Output: $\{\nabla\mathbf{L}^{ss'}\}$ and $\{\nabla\mathbf{d}^s\}$.
for $s \leftarrow 1$ **to** S **do**
  Solve for $(\mathbf{L}^{ss})^\top \nabla\mathbf{d}^s = \nabla\boldsymbol{\alpha}^s$.
  $\nabla\mathbf{L}^{ss} \leftarrow -\nabla\mathbf{d}^s \otimes \boldsymbol{\alpha}^s$.
  for $s' \leftarrow 1$ **to** $s-1$ **do**
    $\nabla\mathbf{L}^{ss'} \leftarrow -\nabla\mathbf{d}^s \otimes \boldsymbol{\alpha}^{s'}$.

As the chosen basis functions are compactly-supported (*i.e.*, defined within the domain of 3^3 voxels), all the linear systems involved in the algorithm for both forward and backward passes are sparse and symmetric. This offers a good opportunity for accelerating the solvers. In practice, we first apply a sparse conjugate gradient method and check whether the residual diverges (using a threshold of 10^{-2} relative residual), in which case we fall back to the sparse Cholesky factorization approach. The latter leads to a more stable convergence.

3.4 Discussion

To empirically justify the efficacy of the proposed reconstructor, we include two baselines under the setting of overfitting a given surface with gradient descent, setting both $\{\mathbf{N}_k^s\}$ and $\{\boldsymbol{m}_k^s\}$ as learnable parameters: (1) 'w/o' Basis, where bases of all voxels are fixed as $b^x = b^y = b^z = b_{\text{init}}$, and $b_{\text{init}}(x) := 0.146x^4 - 0.657x^2 + 0.739$, which is a function that resembles the Bézier basis as visualized in the first



column of Fig. 4. (2) 'w/o Solver', where the coefficients $\{\boldsymbol{\alpha}_k^s\}$ are treated as a free parameter to optimize instead of being enforced as $\mathbf{L}^{-1}\mathbf{d}$. With the voxel grid structure fixed, the training curves in the inset (in logarithm coordinates) show a clear advantage of the proposed method, where both bases and normal fields are being optimized, with the coefficients being solved through the linear solver.

Furthermore, we show in Fig. 5 that the introduction of spatially-varying basis functions could largely enhance the representation power of the implicit field. Under the same $\vec{\mathbf{N}}$, compared to naïvely increasing the resolution of the voxel grids, the learnable bases endow the recovered geometry with more details and more appealing looks.

4 NETWORK AND TRAINING

With our core surface fitting solver defined, in this section, we detail how it is used in conjunction with the neural network and learn useful priors from the data. We first define the learning task by summarizing the attributes that the network needs to predict in § 4.1. Then we present in § 4.2 the detailed neural network architecture. The whole pipeline is eventually trained end-to-end with the strategies and loss functions proposed in § 4.3.

4.1 The Learning Task

There are four key attributes the network needs to predict for the surface fitting solver: (1) The multi-scale voxel grid structure $\{\mathcal{V}^s\}$ to form Ω. Such a structure is defined by predicting the 3-category status of the voxels, as detailed in § 4.2. For each of the existing voxels $\mathbf{v}_k^s$, we need (2) its normal data $\mathbf{N}_k^s$ and (3) the blending coefficients for its basis function $\boldsymbol{m}_k^s$. Finally for each point $\boldsymbol{p}$ in $\mathcal{P}$, we learn (4) the target scalar $\xi(\boldsymbol{p})$ in Eq (2) for denoising. The above attributes, as underlined in Fig. 3, will be used to compute the linear system in Eq (5) and assemble the desired implicit field f via Eq (4).

Note that the predictions are conditioned on the network weights which are the only learnable parameters in our framework. While the network models in a local manner the geometric prior of both surface orientations (2) and local volumetric shapes (1,3,4), the closed-form solver in § 3 helps generate a globally coherent and faithful reconstruction.

4.2 Network Architecture

Our network is an adaptive sparse convolutional neural network, as illustrated in Fig. 6. It leverages the sparsity of geometric data to gain scalability and is 'adaptive' in that the output sparse structure is multi-scale to recover details at different levels. Generally speaking, the network follows the Convolutional U-Net encoder-decoder structure with skip connections, which gradually summarizes low-level features of the input points into rich high-level semantics where information is exchanged. The sliding-window nature and translation-equivariant property of the convolutional operators make it applicable to data of arbitrary scales.

4.2.1 Encoder. The encoder branch is adopted from [Peng et al. 2020], but tailored to a sparsified version, with the same voxel size as the one used in § 3 and S layers. Specifically, the input points $\mathcal{P}$ are first distributed into voxels, where a PointNet [Qi et al. 2017] is used to pool all points within each voxel into features. We then perform convolution and pooling operations layer-by-layer over the feature grid to extract deep features, similar to a traditional U-Net. As the feature grid is sparsely allocated in space, we use the sub-manifold [Graham et al. 2018] versions of the operators above for efficiency.

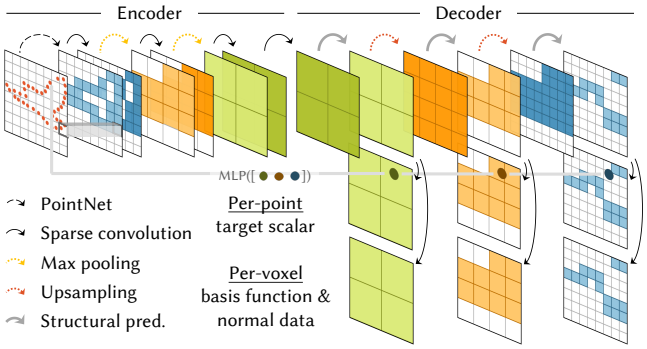


Fig. 6. Illustration of a 3-layer sparse convolutional neural network. Taking the raw point cloud as input, the network infers both per-point and per-voxel information. Here the skip links are omitted for clarity.

4.2.2 Decoder. The decoder branch is borrowed from the adaptive convolutional network of [Wang et al. 2018]. It generates the voxel grids $\mathcal{V}$ that defines the domain of the solver, and the number of layers is the same as the number of scales S. Given the upsampled voxel grid from the coarser layer, a structural prediction branch is used to classify each voxel into three categories: voxels that should be 'deleted', 'subdivided' or 'kept-as-is'. The rules follow that all voxels that are not 'deleted' will form the voxel grid $\mathcal{V}^s$ of the current layer/scale s, while only voxels that are 'subdivided' would be kept for the finer layer $s-1$, in consideration for being included in $\mathcal{V}^{s-1}$. Such a structure prediction naturally satisfies the recursive containing requirement from the surface fitting solver. To prevent empty layers, we additionally impose that voxels at scale $s > M$ will not be 'kept as-is'. Here, M could be dubbed as the maximum adaptive depths, as voxels with $s > M$ could never become leaf nodes.

4.2.3 Point prediction. For the per-point prediction $\xi(\boldsymbol{p})$, we leverage the trilinear-interpolated feature concatenated from all scales of voxels that contain $\boldsymbol{p}$, a way similar to [Chibane et al. 2020a]. We additionally note that only the relative coordinates to the voxel are used in both the encoding and decoding stage of the points, making predictions translation-invariant.

4.3 Training

The full training process of our pipeline is divided into two phases. In the first phase, we only train the normal and structural prediction branch, supervised by $\mathcal{L}_{\text{p1}} := \omega_{\text{vn}}\mathcal{L}_{\text{vn}} + \omega_{\text{struct}}\mathcal{L}_{\text{struct}}$, where

$$\mathcal{L}_{\text{vn}} := \sum_{s,k} \left|\mathbf{N}_k^s - (\mathbf{N}_k^s)_{\text{gt}}\right|, \quad \mathcal{L}_{\text{struct}} := \sum_{s,k} \text{CE}\left(c_k^s, (c_k^s)_{\text{gt}}\right), \tag{8}$$

with a fully supervised L1 loss on voxel normal and a cross-entropy (denoted by CE) between the predicted 3-category voxel state c_k^s

(see § 4.2 decoder) and the ground-truth one $(c_k^s)_{\text{gt}}$. Here, $(c_k^s)_{\text{gt}}$ is computed by inspecting whether the corresponding voxel $\mathbf{v}_k^s$ or its children exist in the ground-truth voxel grids. Due to the different geometric properties of the datasets we use, detailed specifications of such supervision are postponed to the respective experiment sections.

The ground-truth normal defined on the voxel is computed by distributing the point normal into its nearest voxels using trilinear weights, with proper normalizations:

$$(\mathbf{N}_k^s)_{\text{gt}} := 8^{-s+1} \sum_{\boldsymbol{p} \in \mathcal{P}_{\text{gt}}} \frac{1}{Z_{\boldsymbol{p}}} \cdot w(\boldsymbol{p}, \mathbf{v}_k^s) \cdot \boldsymbol{n}_{\boldsymbol{p}}, \tag{9}$$

with $\boldsymbol{n}_{\boldsymbol{p}}$ being the ground-truth normal of $\boldsymbol{p}$. The factor $Z_{\boldsymbol{p}}$ is the point density estimated as in [Kazhdan et al. 2006], which normalizes the scales to make $\mathbf{N}_k^s$ invariant to point density or voxel sizes. $w(\boldsymbol{p}, \mathbf{v})$ is the normalized trilinear weight of $\boldsymbol{p}$ w.r.t. the 8 nearby voxels to allow for sub-node precision. This term evaluates to a non-zero value only if the point belongs to scale s.

In the second phase of training, we set the supervision as $\mathcal{L}_{\text{p2}} := \omega_{\text{struct}}\mathcal{L}_{\text{struct}} + \omega_{\text{surf}}\mathcal{L}_{\text{surf}} + \omega_{\text{norm}}\mathcal{L}_{\text{norm}}$, and

$$\mathcal{L}_{\text{surf}} := \sum_{\boldsymbol{p} \in \mathcal{P}_{\text{gt}}} \left(f(\boldsymbol{p}) - \delta\right)^2, \;\; \mathcal{L}_{\text{norm}} := \sum_{\boldsymbol{p} \in \mathcal{P}_{\text{gt}}} \left(1 - \boldsymbol{n}_{\boldsymbol{p}}^{\top} \frac{\nabla_{\boldsymbol{p}} f}{\|\nabla_{\boldsymbol{p}} f\|}\right). \tag{10}$$

Such two terms force both the predicted implicit function and its first-order derivative $\nabla_{\boldsymbol{p}} f \in \mathbb{R}^3$ to fit the surface. Thanks to the well-defined subspace of the variational solution, we do not need to impose further constraints over positions away from the surface like, *e.g.*, [Sitzmann et al. 2020]. To compute the derivative $\nabla_{\boldsymbol{p}} f$, one only has to evaluate the derivative of the elementary basis functions. This is in contrast to neural surfaces [Gropp et al. 2020] where a back-propagation through the network is needed.

In total, we need three supervisions for training: (1) per-voxel structural state $(c_k^s)_{\text{gt}}$, (2) per-voxel normal $(\mathbf{N}_k^s)_{\text{gt}}$, and (3) sampled surface points/normals from $\mathcal{P}_{\text{gt}}$. We manually initialize the basis prediction branch to produce b_{init} for all axes within each voxel for a more stable training. Empirically, we have found that training the network from scratch without the 2-phase-strategy or the basis initialization also leads to converged results without an apparent decrease in performance. However, the proposed approach achieves the fastest convergence and takes less time to train as the first training phase does not back-propagate through the solver.

5 EXPERIMENTS

In this section, we first introduce the detailed parameters and settings used (§ 5.1). Our method is evaluated on multiple different datasets with different scales, *i.e.*, object-level shapes (§ 5.2), human body scans (§ 5.3), room-level scenes (§ 5.4), and qualitative results on driving scenes from LiDAR scans (§ 5.5). The design choices are then thoroughly validated in § 5.6, and timing and memory analysis is in § 5.7.

5.1 Implementation Details

Our implementation is based on a customized version of [Tang et al. 2022] (that internally uses a Cuckoo hashtable [Pagh and Rodler 2004]) for fast neighborhood queries and cuBLAS/cuSOLVER for solving sparse linear problems. The adaptive sparse convolutional network is instructed to share its neighborhood map with the solver. For both of our training phases, we use an AMSGrad optimizer [Reddi et al. 2019] with a standard learning rate of 10^{-3} and a batch size of 2. We empirically set $\lambda = 64.0$, $N_p = 3$, $\omega_{\text{vn}} = 25.0$, $\omega_{\text{struct}} = 1.0$, $\omega_{\text{surf}} = 20.0$ and $\omega_{\text{norm}} = 200.0$, with $S = 4$ layers of voxel grids and varying M for different datasets. The level set δ is fixed to be 0.1 as an initial approximation of the integration of Eq (9) along the surface normal. Some of these choices will be validated in our ablation study. For all the datasets we train on, 100K points are used for supervision. Our experiments are conducted on an NVIDIA GeForce RTX 3090 graphics card with 24GiB of video memory and an Intel i9-10900K CPU.

5.2 Single Shape Reconstruction

For object-level evaluations we leverage ShapeNet [Chang et al. 2015], a large-scale repository containing over 50k unique 3D models for deep learning. We follow the common practice in the literature and use the training-validation-test split from [Choy et al. 2016], containing 13 categories of objects. In total, 8751 unseen objects from all these categories are used for benchmarking. Same as [Mescheder et al. 2019], virtually-fused watertight meshes are created as ground-truth instead of the original (possibly) non-manifold raw meshes. All meshes are uniformly scaled to fit a unit cube. We explore three settings in our experiment, *i.e.*, 'No noise, 1K points', 'Small noise, 3K points' and 'Large noise, 3K points', where small noise and large noise refer to Gaussian noises with standard deviations of 0.005 and 0.025 respectively added onto the input points. For the metrics, we use F-score, chamfer distances, and normal consistency: F-score (%) ranges from 0 to 100, and balances surface precision and recall (completeness), both with a distance threshold of 0.01. Chamfer distance (scaled by 10^3, using L_1-norm) measures the fitting tightness between the predicted surfaces and the ground truth. Normal consistency (%, abbreviated as 'Normal C.') reflects how the normals of the two surfaces agree.

As the geometric variations in the ShapeNet objects are approximately uniformly-distributed, with few areas containing less detailed surfaces such as planes, we find it unnecessary to use an adaptive voxel grid structure in this case. Hence we set the maximum adaptive depth M to 1, *i.e.*, all voxels at scales $s > 1$ cannot be 'kept-as-is'. In our structural supervision $(c_k^s)_{\text{gt}}$, we build the ground-truth voxel grid such that every point on the ground-truth surface is covered by a voxel for all scales. A voxel size of $b = 0.02$ is found enough to model the shapes accurately.

Regarding the baselines we choose the screened Poisson surface reconstruction (SPSR) in [Kazhdan and Hoppe 2013], neural splines (NS) in [Williams et al. 2021b], the most recent local implicit grid approach DI-Fusion from [Huang et al. 2021], convolutional occupancy networks (ConvONet) from [Peng et al. 2020], shape-as-points (SAP) from [Peng et al. 2021] and IMLSNet [Liu et al. 2021]. All learnable baselines are re-trained from scratch except for [Huang et al. 2021] where strict local modeling is ensured. For a fair comparison we train two versions of our method, *with* and *without (w/o)* point normal. For the version with normal, the point normal is concatenated as extra input channels to the network's encoder, while the normal

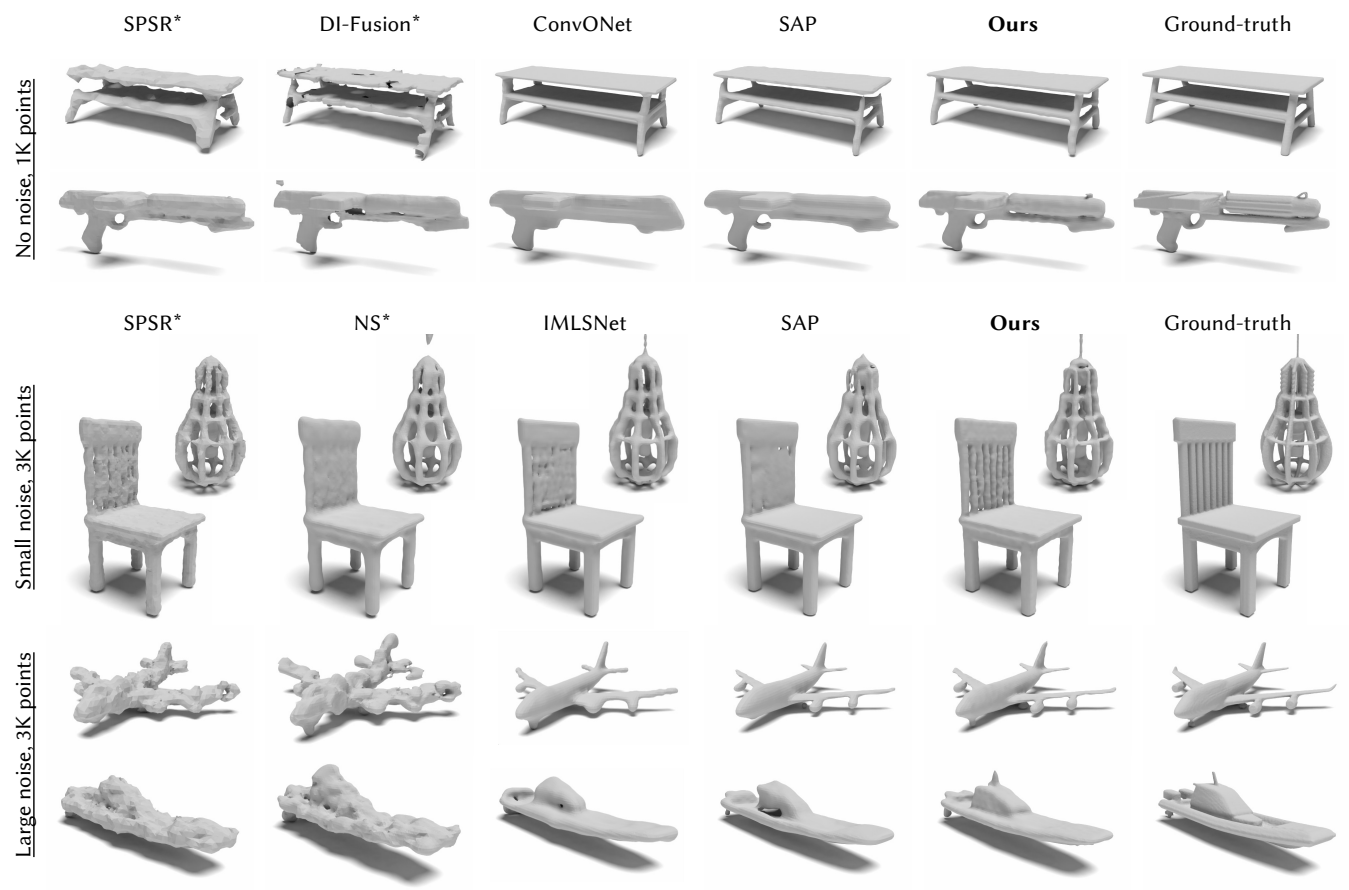


Fig. 7. Results on ShapeNet [Chang et al. 2015] dataset under the 3 settings. Methods marked with '*' use point normal while others do not. Note that our method can faithfully recover fine geometric details (such as the wheels of the plane and the engine of the boat) and thin structures.

Table 1. Quantitatve comparisons on ShapeNet [Chang et al. 2015]. The three methods above the middle bar require normal input, while the rest (including ours) do not. The 'mean' and 'std.' (standard deviation) statistics are reported across the 13 categories. ↑/↓: The higher/lower the better.

		No noise, 1K points						Small noise, 3K points						Large noise, 3K points					
		F-Score ↑		Chamfer ↓		Normal C. ↑		F-Score ↑		Chamfer ↓		Normal C. ↑		F-Score ↑		Chamfer ↓		Normal C. ↑	
		mean	std.	mean	std.	mean	std.	mean	std.	mean	std.	mean	std.	mean	std.	mean	std.	mean	std.
with Normal	SPSR [Kazhdan and Hoppe 2013]	84.3	8.50	6.26	2.13	89.4	3.45	95.8	2.84	3.84	0.95	92.1	2.27	55.5	12.7	10.7	1.99	81.9	4.54
	NS [Williams et al. 2021b]	90.6	3.13	4.74	0.95	91.9	2.51	95.8	2.22	3.78	0.78	93.5	2.25	49.9	8.80	12.0	1.81	82.8	4.90
	DI-Fusion [Huang et al. 2021]	88.9	6.13	4.97	1.34	88.4	3.66	91.8	5.07	4.43	1.28	86.7	4.19	39.5	**2.40**	15.1	0.78	61.8	3.29
	Ours	**98.9**	**1.01**	**2.47**	**0.65**	**96.3**	**1.85**	**99.5**	**0.55**	**2.51**	**0.53**	**96.4**	**1.75**	**95.8**	2.42	**3.93**	**0.71**	**94.4**	**2.43**
w/o Normal	ConvONet [Peng et al. 2020]	89.3	5.17	6.07	1.64	92.4	3.04	94.2	4.09	4.35	1.40	93.8	2.63	82.3	7.30	7.31	2.00	91.1	3.36
	SAP [Peng et al. 2021]	96.2	2.47	3.44	0.86	93.8	2.61	97.5	1.90	3.30	0.77	94.5	2.42	89.5	4.40	5.34	1.16	91.7	3.22
	IMLSNet [Liu et al. 2021]	96.8	2.40	3.15	**0.75**	93.9	2.70	98.2	1.55	3.08	**0.58**	94.4	2.52	82.0	5.09	6.58	0.95	89.5	3.77
	Ours	**97.4**	**1.92**	**2.91**	0.84	**95.0**	**2.28**	**98.6**	**1.12**	**2.89**	0.60	**95.2**	**2.17**	**90.2**	**4.03**	**5.06**	**0.89**	**92.0**	**2.99**

data for the surface fitting solver is still predicted by the network. The comparisons are shown in Tab. 1, and visualized in Fig. 7.

Without learnable modules SPSR and NS fail to recover geometric details with points as few as 1K, and suffer from the noise in the input, producing bulging geometries. DI-Fusion shows an even worse performance with large noise because the local implicit grid used is unaware of the full shape structure. Our reconstruction accuracy surpasses ConvONet and IMLSNet (except for some cross-category variations), where the implicit fields / IMLS points are directly inferred with no fitness guarantees. Notably, our voxel grid resolution is close to ConvONet, where neural-network-parameterized local implicit fields are used. Yet, we could reach a better performance

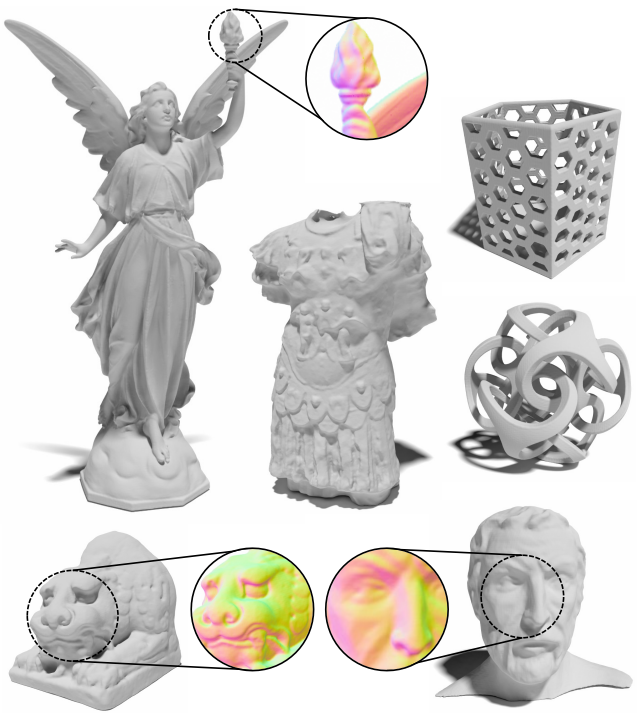


Fig. 8. Domain generalization to Thingi10k [Zhou and Jacobson 2016] dataset using our model with normal. Insets show the normal maps.

with a higher f-score of ∼8%. This echoes that our elementary-function-based bases indeed possess strong representation power. SAP solves an un-screened version of the Poisson equation and can only use a discrete dense grid, while our representation is implicit and sparse, leading to our more detailed and faithful modeling of the shapes.

We additionally offer qualitative results of reconstructions on selected shapes from Thingi10k [Zhou and Jacobson 2016] in Fig. 8 to show the strong generalizability of our model. We use the version *with* input point normal because we find such information effectively helps disambiguate surface orientations for novel shapes. Trained only on ShapeNet, we could recover highly-detailed geometries such as the clothes, wrinkles, and sharp edges of unseen categories and scales. We attribute the generalizability to a synergy of the network and our surface fitting solver, which naturally incorporates informative point constraints with orientational predictions.

5.3 Human Body Reconstruction

Compared to man-made objects, human bodies' geometries exhibit finer details that are more challenging to recover. To demonstrate our applicability, we utilize D-FAUST [Bogo et al. 2017] dataset. It contains raw 4D recordings of 10 subjects performing different actions. We choose eight subjects, subsample each sequence every four frames, and randomly split the frames into training, validation, and test set. For a better description, the above test set is denoted as the 'orig.' (original) subset. The remaining two subjects (50025, 50027) are used in a separate held-out test set that contains novel geometries to evaluate our generalizability, which we call the 'novel' subset. All baseline methods take 10K uniformly sampled points as input. The metrics are the same as the ones in § 5.2, except that we change the threshold of the F-score to 0.002 (which we denote as 'F-Score+') for clearer comparisons.

Table 2. Comparisons on D-FAUST [Bogo et al. 2017] dataset. The two columns ('orig.' and 'novel') are results of different test subsets.

		F-Score+ ↑		Chamfer ↓		Normal C. ↑	
		orig.	novel	orig.	novel	orig.	novel
with Normal	IGR [Gropp et al. 2020]	51.5	54.1	24.7	22.7	88.3	87.8
	SALD [Atzmon and Lipman 2020b]	58.6	62.0	2.73	2.63	97.3	97.0
	ConvONet [Peng et al. 2020]	80.8	83.2	2.71	2.09	96.6	96.3
	SPSR [Kazhdan and Hoppe 2013]	93.4	93.6	1.36	1.58	98.2	97.5
	Ours	**95.3**	**97.0**	**1.12**	**1.05**	**98.7**	**98.6**
w/o Normal	ConvONet [Peng et al. 2020]	71.7	73.6	2.78	2.21	96.2	95.9
	SAP [Peng et al. 2021]	86.0	88.8	1.55	1.46	97.4	97.2
	Ours	**95.2**	**96.8**	**1.13**	**1.06**	**98.6**	**98.5**

Different from the dense one used in ShapeNet, we employ different structural supervision $(c_k^s)_{\text{gt}}$. This is because the human body is inherently multi-scale, as more geometric variations are focused on, *e.g.*, faces or fingers. Specifically, we adopt the subdivision-based policy proposed in [Tang et al. 2021a] to build the ground-truth structure, where the geometric variation within each voxel is evaluated as:

$$Q_k^s := \sum_{\text{axis}\in\{x,y,z\}} \text{std.}\left(\left\{(\boldsymbol{n}_{\boldsymbol{p}})_{\text{axis}}\right\}_{\boldsymbol{p}\in\mathbf{v}_k^s}\right), \tag{11}$$

where the standard deviations of the point normals within the voxel are summed over all axes. Starting from the coarsest layer $\mathcal{V}^S$, each voxel $\mathbf{v}_k^s$ is determined to be 'subdivided' if $Q_k^s > 0.02$, otherwise it should be either 'deleted' or 'kept-as-is' based on whether there is ground-truth geometry within that voxel. We empirically set the maximum adaptive depths M to 2, and the voxel size b to a value of 5×10^{-3}.

Similar to ShapeNet, we provide two versions (with & without normal) of our model. For the version with normal, we additionally compare to single-MLP-parametrized IGR [Gropp et al. 2020] and SALD [Atzmon and Lipman 2020b]. Results are compared in Tab. 2. We find SPSR to be a rather strong baseline that surpasses the learnable ConvONet. While our method employs a similar energy formulation as SPSR, inferred priors from the networks are effectively merged into the solver, and we are hence able to recover more details. Our method outperforms the baselines by a large margin and shows good generalizability to the novel subjects. As visualized in Fig. 9 quantitatively, although IGR could generate more details on the faces and skins than SAP, it spawns a large number of spurious planes away from the input and takes ∼2 minutes to optimize the latent vector for a single shape. On the other hand, SALD generates smooth shapes with a single forward pass, but it suffers from a large systematic misalignment that does not respect input poses, resulting in a large quantitative error. Our method, in contrast, can reconstruct the most details faithfully and accurately while taking only ∼1 second to compute (without the meshing operation).

Fig. 9. Results on D-FAUST [Bogo et al. 2017] dataset. Our method is compared to IGR [Gropp et al. 2020], SALD [Atzmon and Lipman 2020b] (using the version with normal) and SAP [Peng et al. 2021] (using the version without normal), all digesting 10K points. We could recover sharp facial and hand details, without spurious surfaces or costly test-time optimization (~2min for IGR). The upper row shows subjects from the 'orig.' subset, while the lower row is from the 'novel' subset (containing novel unseen subjects).

5.4 Room-level Scene Reconstruction

The scalability of our method is further tested using Matterport3D [Chang et al. 2017] dataset consisting of real-world scans of indoor rooms using a panoramic depth camera. The dataset contains 90 buildings in total. We follow the official train/test split and further divide each building into individual rooms, with each room being reconstructed and evaluated separately. The test set contains 399 rooms. The same strategy in § 5.3 is used to generate the structures used for supervision, producing voxels at a larger scale in planar regions such as walls and grounds. We set $M = 2$ and $b = 0.012$.

Critically, the indoor scans are all single-sided surfaces, yet due to the nature of our solver to produce watertight geometries, excessive floaters will be presented at the domain boundary. Thanks to the truncated domain Ω used, the floaters are small in area and never appear undesirably at places far from the surface. In our experiment, we employ a k-NN approach to trim the vertices on the generated mesh that are too far away from the input points. To enable a fair comparison, we report both our untrimmed and trimmed versions, using the additional metrics that incorporate the uni-direction performance irrespective of excessive geometries (*i.e.*, uni-directional chamfer and normal consistency from the ground-truth to the predictions, and the surface recall rate).

For quantitative comparisons, we follow the standard practices and subsample the input point cloud to 10K. We compare SPSR, DI-Fusion (without re-training), ConvONet and SAP, also on two versions with and without input normals. For SAP we try to increase the resolution of the FFT solver from 256 to 512 but find the higher resolution is hard to converge, resulting in noisy particles around the surface. SAP also suffers from memory issues due to its cubic algorithm complexity.

Table 3. Results on Matterport [Chang et al. 2017] dataset. '⇆' indicates the original bi-directional metrics, while '←' is the uni-directional version (for f-score this corresponds to surface recall) that disregards excessive floaters.

		F-Score ↑		Chamfer ↓		Normal C. ↑	
		⇆	←	⇆	←	⇆	←
with Normal	SPSR [Kazhdan and Hoppe 2013]	87.0	96.9	10.4	3.65	92.3	93.6
	DI-Fusion [Huang et al. 2021]	90.1	93.3	5.13	4.34	90.1	90.9
	ConvONet [Peng et al. 2020]	92.8	98.7	4.80	3.98	94.9	94.8
	Ours	92.7	**99.7**	4.10	2.60	95.6	**97.1**
	Ours (trimmed)	**97.8**	**99.7**	**2.87**	**2.48**	**96.8**	**97.1**
w/o Normal	ConvONet [Peng et al. 2020]	88.9	91.8	6.21	5.79	92.3	92.0
	SAP [Peng et al. 2021]	91.2	94.5	4.17	4.12	91.4	91.4
	Ours	93.5	**99.3**	3.93	2.72	94.1	95.6
	Ours (trimmed)	**97.3**	**99.3**	**3.04**	**2.63**	**95.3**	**95.7**

The results are compared in Tab. 3 and Fig. 10. Our method performs favorably over all the baselines even without trimming, introducing nice and smooth complete room-level reconstructions with abundant geometric details. The fitting accuracy is more distinct with the uni-directional metrics, where we could achieve a nearly 29% and 34% reduction in chamfer distances compared to the closest counterpart. Moreover, our method enjoys an intuitive property that the reconstruction accuracy improves with more points as input, without fine-tuning the network. We note that such a property is not guaranteed in many learning-based approaches, such as ConvONet. This is illustrated in Fig. 11, the details on the furniture/plants can be faithfully recovered with 50K input, while under the same setting, ConvONet exhibits holes and noise.

We additionally compare to two overfitting-based methods, *i.e.* SIREN [Sitzmann et al. 2020] and ACORN [Martel et al. 2021] in Fig. 12. As both of the methods require first-order optimizations,

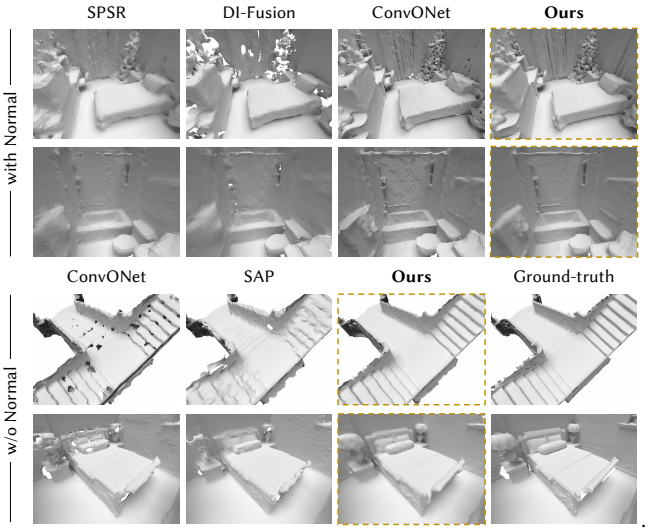


Fig. 10. Comparisons on Matterport [Chang et al. 2017] dataset with 10K input points. Yellow dashed boxes indicate our method.

they are inherently slow in time. For SIREN, a one-minute run is not enough for proper convergence, leaving blurred surfaces and floating chunks in the scene. Fully converged SIREN that takes an hour of optimization could reach a comparable quality to our method that runs only for a few seconds, while the ACORN model falsely introduces noisy surfaces possibly due to its weakness in regularizing geometries. As the optimality of our method is obtained through the closed-form Euler-Lagrange formulation that requires only a linear solve, we could reconstruct the scenes both fast and accurately.

5.5 Large-scale Driving Scene

To further demonstrate our scalability and generalizability to large scales, we apply our method to KITTI dataset [Geiger et al. 2012], using the pre-trained weights from Matterport. We take our input from the LiDAR scans in the odometry subset. They are transformed into world coordinates using the provided camera trajectories. Ground-truth semantic annotations from [Behley et al. 2019] are used to prune dynamic objects and distracting vegetation from the LiDAR points. Due to the differences in point distributions, instead of using the predicted voxel grid structure, we opt to replace it with a hand-crafted one based on point density, similar to [Kazhdan et al. 2006]. To accelerate processing, the full scene is divided into 51.2m^3 blocks, and points within each block are down-sampled using 5cm^3 voxels. Our method is executed sequentially on each block, and the final meshes are blended.

As shown in Fig. 13, our method successfully generalizes to large-scale scenes, achieving visual-appealing reconstructions with sufficient details. This is further confirmed by back-projecting color images onto our geometry and observing the well-aligned textures. Thanks to the sparsified domain Ω, no broad-scale trimming is needed. Moreover, our multi-scale voxel grids naturally enable a multi-scale mesh extraction mechanism [Shekhar et al. 1996], as illustrated in Fig. 14, to reduce the number of triangles.

Table 4. Comparison of different noise handling strategies. 'mean' and 'std.' (standard deviation) are measured at instance level.

	L_2-Chamfer ↓		F-Score ↑		Normal C. ↑	
	mean	std.	mean	std.	mean	std.
No handling	1.84	1.37	98.2	2.88	95.2	3.42
$E_{\text{alter}}(f)$	1.99	1.53	98.0	2.85	95.2	3.29
$E(f)$ **(Ours)**	**1.69**	**1.21**	**98.4**	**2.55**	**95.6**	**3.25**

5.6 Ablation Studies

Our ablation studies are conducted over ShapeNet and Matterport datasets. For ShapeNet, we use the 'Small noise, 3K points' setting and choose its chair subset for training, validation, and testing in the following experiments. Under the same parameter setting, our model trained only on the subset could perform similarly to the model trained on the full dataset (F-score 98.4 vs. 98.6). Apart from the F-score and normal consistency metrics, we change the chamfer distance from L_1-norm to L_2-norm (denoted by 'L_2-Chamfer' below) to show clearer differences due to its outlier-sensitivity. The L_2 distance is scaled by 10^5 for better display.

5.6.1 Energy formulation. The balancing factor λ in Eq (5) controls how close the fitted surface should snap to input point observations. In the extreme case where $\lambda = 0$, the formulation falls back to a simple Poisson solution as in [Kazhdan et al. 2006], focusing only on $\vec{\mathbf{N}}$ which is predicted by the network in our case. As shown in Fig. 15, all metrics improve with a growing λ, especially L_2-Chamfer distance. Such an observation confirms the merit of respecting the input points. The worsened performance with a small λ reveals that the network alone could not generate a surface with high precision. Note that the predicted normal data is equally important by offering informative guidance of the surface orientation, and our solver will not converge without the first term in Eq (5) (*i.e.*, when $\lambda \to \infty$).

5.6.2 Point noise handling. A common way of handling input noise is to predict the weight of each point, as done in previous works such as [Ben-Shabat and Gould 2020; Williams et al. 2021a]. Being an intuitive alternative, we also implement such a scheme by minimizing the following energy function in replacement of $E(f)$ in Eq (2):

$$E_{\text{alter}}(f) := \iiint_{\Omega} \left\| \nabla f - \vec{\mathbf{N}} \right\|_2^2 \mathrm{d}V + \lambda \sum_{\boldsymbol{p} \in \mathcal{P}} \xi(\boldsymbol{p}) \left(f(\boldsymbol{p}) - \delta \right)^2, \quad (12)$$

where the per-point target scalar prediction $\xi(\boldsymbol{p})$ acts instead as a varying weight of the second data fitting term. The strategies are compared in Tab. 4, with an additional 'No handling' baseline setting both λ and ξ as constant. By explicitly treating the predicted scalars as constraints within the implicit field, our strategy enables a more detailed depiction of the point noise, using the information from all the points. Predicting per-point weight is less effective in our case, producing ill-posed fitting behaviors when all points are falsely down-weighed, hindering the learning process.

5.6.3 Adaptive structure and number of scales. We show in Tab. 5 the effect of the number of scales S (that equals the number of layers in our network) and the maximum adaptive depth M, using the Matterport dataset adopting the same metrics as in § 5.4. With deeper

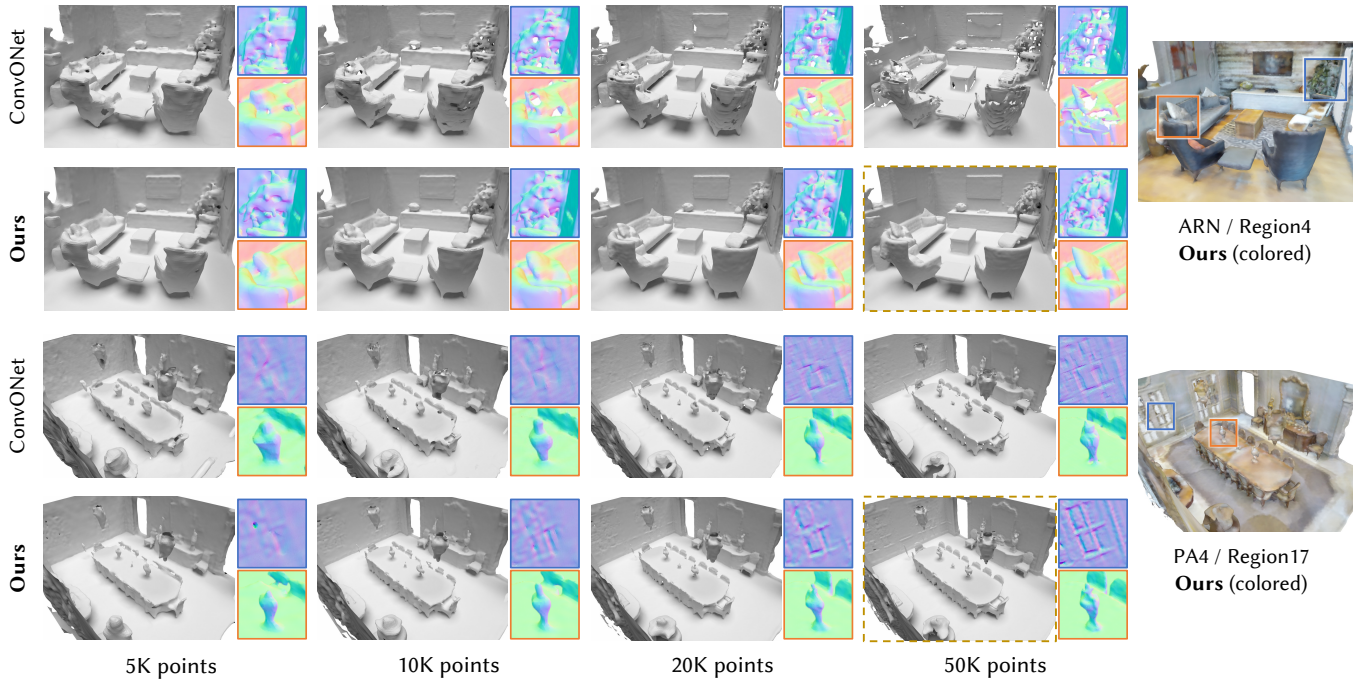


Fig. 11. Reconstruction results on Matterport [Chang et al. 2017] with increasing number of points. Both ConvONet and our method are trained only with 10K points, while only our method shows a gradually improving geometry thanks to the surface fitting formulation. The blue and orange crops are taken from the regions annotated in the rightmost figure, where our colored geometry is generated by setting the vertex color as the average of nearby input points.

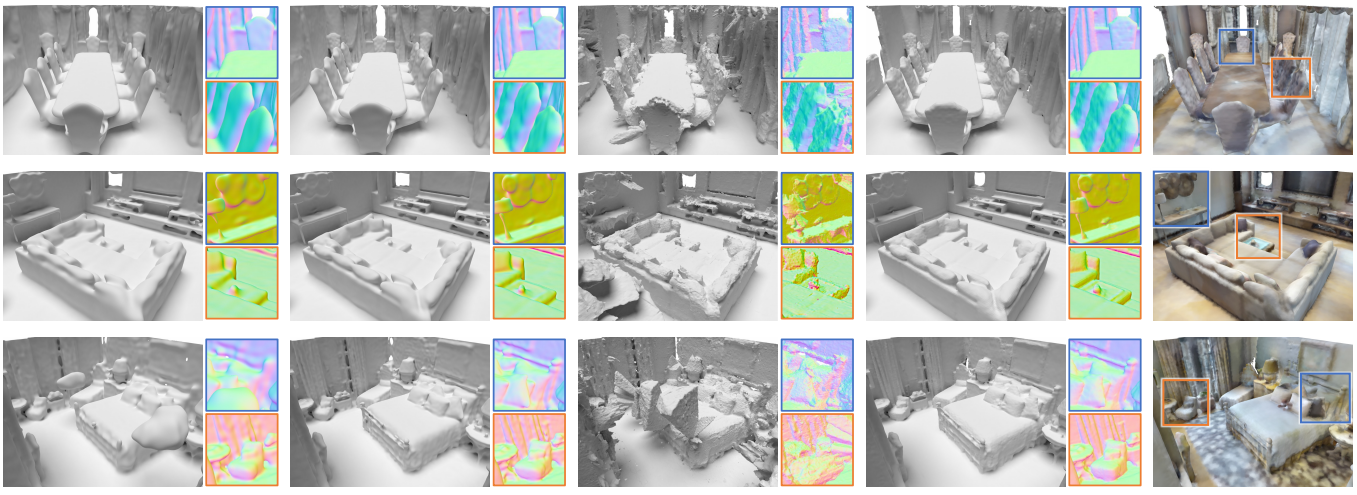


Fig. 12. Surface quality and run time comparison with SIREN [Sitzmann et al. 2020] and ACORN [Martel et al. 2021], using 50K oriented points as input. We reach comparable reconstruction quality with these overfitting-based methods, while being significantly faster.

networks and more scales, the surface could be more accurately fitted. The bi-directional metric decreases drastically with growing M due to the increased surface area of un-trimmed floaters. Regarding the fitting precision reflected by the single-directional metrics, we find similar or even better performance with a larger M. It also enables fair speed boosts by reducing the number of voxels involved in the solver.

5.6.4 Learnable basis functions. We observe similar behaviors in the learning-based setting as in overfitting (Fig. 5), that using our proposed spatially-varying bases composed of elementary functions

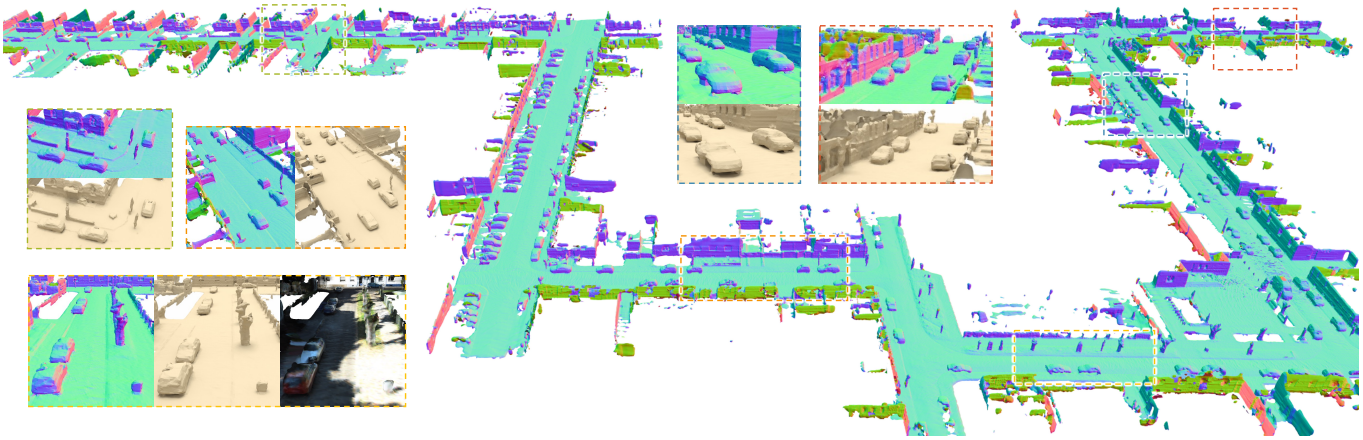


Fig. 13. Results on KITTI [Geiger et al. 2012] odometry dataset (Sequence 00, frame 3000 to 4000). Normal maps along with the rendered mesh are used for visualization to highlight geometric details. Zoomed-in insets correspond to the locations marked on the full map, with the same dashed boundary color.

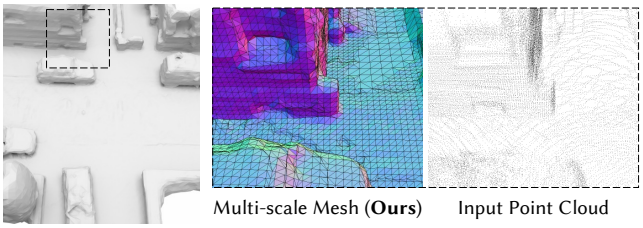


Fig. 14. Our method enables extracting multi-scale meshes thanks to the adaptive voxel grid structure built. Note the coarsened triangulations on the top-right corner where input points are sparse.

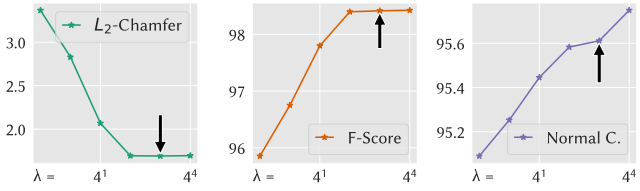


Fig. 15. Effect of the data fitting term in Eq (5). We show the fitting performance using different weights λ (all networks are re-trained from scratch). Black arrows indicate our chosen $\lambda = 64$.

Table 5. Effect of different number of scales S and maximum adaptive depth M. '⇆' is the bi-directional metric and '←' only considers single direction.

		F-Score ↑		Chamfer ↓		Normal C. ↑	
S	M	⇆	←	⇆	←	⇆	←
4	1	95.7	96.8	3.52	3.24	92.4	92.6
4	2	93.5	**99.3**	3.93	**2.72**	**94.1**	**95.6**
4	3	82.3	99.0	8.47	3.07	92.5	95.4
3	1	**96.3**	97.8	**3.35**	3.00	93.0	93.4
3	2	90.4	98.5	4.63	2.97	92.5	94.2
2	1	93.8	96.4	3.89	3.40	88.5	89.5

Table 6. Effect of different basis functions. 'Order' means the order of the polynomials. '+ Sine' indicates whether sine waves are added to the function.

Order	+ Sine	L_2-Chamfer ↓	F-Score ↑	Normal C. ↑
Bézier		1.75	97.8	94.2
0	✓	1.70	97.7	93.8
5		1.74	97.6	94.2
6		1.70	97.9	94.8
6	✓	1.69	98.4	**95.6**
7		1.67	98.2	95.0
7	✓	**1.66**	**98.5**	95.1

improves the representation power of the implicit field and leads to better surface fitting accuracy. Listed in Tab. 6, in comparison to the fixed Bézier tensor basis, the learnable ones all reach a better surface fitting accuracy. As the bases are essentially local geometric priors, introducing higher-frequency sine waves could effectively help enlarge the solution space of the implicit function and encourage more variations in the generated shapes. Interestingly, the performance does not grow monotonically as the basis functions become more complex, which could be partially attributed to the instability in the training process caused by an increased number of parameters. Regarding the computation overhead brought by the additional bases, we observe no apparent decrease in runtime efficiency: Compared to the Bézier basis, using the most complicated 7th-order polynomial with sine waves only adds < 1% additional test time. We additionally tested the DSIF basis from [Genova et al. 2019], but due to the instability of their integral formulation, this led to training divergence.

5.6.5 Loss functions. As shown in [Peng et al. 2021], the supervision over the entire implicit field could lead to better convergence. We hence design a similar loss that writes:

$$\mathcal{L}_{\text{field}} := \iint_{\Omega} \left| f - \hat{f}_{\text{gt}} \right| \mathrm{d}V, \quad \hat{f}_{\text{gt}}(\boldsymbol{p}) := \text{Sigmoid}\left(d(\boldsymbol{p}, \mathcal{S}_{\text{gt}})\right), \quad (13)$$

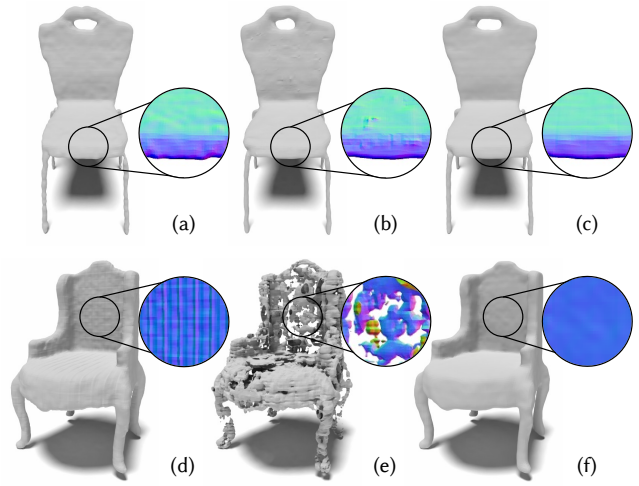


Fig. 16. Qualitative comparisons of different noise handling strategies (a-c) and different loss functions (d-f). Please refer to the text for more explanations.

Table 7. Comparison of different loss functions. 'mean' and 'std.' (standard deviation) are measured at instance level. Ours is at the last row.

$\mathcal{L}_{\text{field}}$	$\mathcal{L}_{\text{surf}}$	$\mathcal{L}_{\text{norm}}$	L_2-Chamfer ↓		F-Score ↑		Normal C. ↑	
			mean	std.	mean	std.	mean	std.
✓			4.14	14.7†	97.4	3.54	93.8	3.69
✓		✓	2.32	3.06	97.9	3.14	95.5	3.30
	✓		9.51	2.60	73.0	4.55	63.3	5.05
	✓	✓	**1.69**	**1.21**	**98.4**	**2.55**	**95.6**	**3.25**

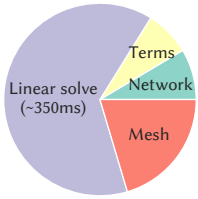
†: Due to some severe failure cases.

where $d(\boldsymbol{p}, \mathcal{S})$ measures the signed distance from the point $\boldsymbol{p}$ to the surface $\mathcal{S}$ (inner values are positive). In the above loss, the pseudo-ground-truth field $\hat{f_{\text{gt}}}$ is approximated as a smoothed indicator field, and directly supervises the predicted implicit function f. However, as shown in Tab. 7, such an approximation leads to inferior results. Fig. 16 (d) further highlights the gridded artifact introduced. We now point out that the optimal $f^\star$ produced by the solver is not a simple indicator or occupancy function of the geometry. Rather, the solution lies in the null space of our variational energy where the geometric bias is more intricately defined, especially with our customized varying bases where the solution space is intractable to define.

The surface normal loss $\mathcal{L}_{\text{norm}}$, is found to be vital in the learning process. It helps guide correct surface orientations and prevents the optimization from falling to a trivial solution where $f \equiv \delta$. The recovered surface is completely corrupted without such a loss, as shown in Tab. 7 and Fig. 16 (e).

5.7 Timing and Memory

The most time-consuming operations in our pipeline are divided into four parts, whose detailed breakdown is shown in the inset figure. Here, 'Network' is the feed-forward time of our adaptive sparse convolutional network, 'Terms' refer to the computation of the integrals in Eq (5), 'Linear solve' is the time of our linear solver, and 'Mesh' includes the time to densely evaluate function values over a uniform voxel grid and to perform marching cubes. On average, the test time for a single sample/chunk is ~0.6s/1.0s/1.0s/6.5s for ShapeNet, D-FAUST, Matterport, and KITTI, respectively, using the default setting. Among all the components, the main bottleneck is the linear solver (taking up ~60% time), which we wish to further optimize in the future.



The runtime memory fluctuates around 6-7GiB during inference. Note that we have not made a heavy code-level optimization, and the statistics may include the overhead of the deep learning library. Possible speed-ups and memory reduction can be achieved using meta-operators [Hu et al. 2020] or low-level languages.

6 CONCLUSION

Despite the promising performance as demonstrated, it suffers from the following main limitations: (1) Time and memory cost of the solver is still high. Although we use sparse structures throughout our pipeline, the overheads of hashtable manipulations and the linear solver contribute a non-negligible constant to the algorithm complexity. (2) The reconstruction is not transformation equivariant. This is not only due to the convolutional operations we use in our network, but also due to the chosen family of basis functions. For ease of computation, we perform axis-aligned factorizations, which inevitably introduce anisotropic bias. (3) The heavy reliance on point cloud data makes it challenging to apply our method to high-level semantic tasks such as shape completion. The solution given by the solver prefers empty regions where no input point exists even if the network predicts correct scaffolds. This could be partially solved by allowing the network to 'hallucinate' novel points, which we leave as future works.

In this paper, we propose NeuralGalerkin for accurate surface reconstruction from point cloud data. By effectively combining priors generated by the network and the raw point measurements using a closed-form surface fitting solver, we can strike a nice balance between the fidelity of the point fitting and the respect to learned inductive biases. Such a formulation also enables higher-definition outputs and a good generalization to unseen data, as demonstrated in the various datasets we have tested upon. In the future, we hope to explore more possible families of basis functions and to develop more efficient ways to optimize the fitting energy.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. This work was supported by the National Key R&D Program of China (No. 2021ZD0112902), Research Grant of Beijing Higher Institution Engineering Research Center and Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology.

REFERENCES

Noam Aigerman, Kunal Gupta, Vladimir G. Kim, Siddhartha Chaudhuri, Jun Saito, and Thibault Groueix. 2022. Neural Jacobian Fields: Learning Intrinsic Mappings of Arbitrary Meshes. *ACM Trans. Graph.* 41, 4, Article 109 (2022), 17 pages.

Matan Atzmon and Yaron Lipman. 2020a. Sal: Sign agnostic learning of shapes from raw data. In *Proc. IEEE Conf. CVPR*. 2565–2574.

Matan Atzmon and Yaron Lipman. 2020b. SALD: Sign Agnostic Learning with Derivatives. In *Int. Conf. Learning Representations*.

Dejan Azinović, Ricardo Martin-Brualla, Dan B Goldman, Matthias Nießner, and Justus Thies. 2021. Neural RGB-D surface reconstruction. *arXiv preprint arXiv:2104.04532* (2021).

Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. 2019. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proc. IEEE Int. Conf. Computer Vision*. 9297–9307.

Yizhak Ben-Shabat and Stephen Gould. 2020. Deepfit: 3d surface fitting via neural network weighted least squares. In *European Conf. Computer Vision*. Springer, 20–34.

Federica Bogo, Javier Romero, Gerard Pons-Moll, and Michael J Black. 2017. Dynamic FAUST: Registering human bodies in motion. In *Proc. IEEE Conf. CVPR*. 6233–6242.

Fatih Calakli and Gabriel Taubin. 2011. SSD: Smooth Signed Distance Surface Reconstruction. *Comput. Graph. Forum* 30, 7 (2011), 1993–2002.

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*. ACM, 67–76.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. 2017. Matterport3D: Learning from RGB-D Data in Indoor Environments. *Int. Conf. 3D Vision* (2017), 667–676.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. 2015. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012* (2015).

Haoxiang Chen, Jiahui Huang, Tai-Jiang Mu, and Shi-Min Hu. 2022. CIRCLE: Convolutional Implicit Reconstruction and Completion for Large-scale Indoor Scene. In *European Conf. Computer Vision*. Springer, 1000–1010.

Zhang Chen, Yinda Zhang, Kyle Genova, Sean Fanello, Sofien Bouaziz, Christian Häne, Ruofei Du, Cem Keskin, Thomas Funkhouser, and Danhang Tang. 2021. Multiresolution deep implicit functions for 3d shape representation. In *Proc. IEEE Int. Conf. Computer Vision*. 13087–13096.

Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. 2020a. Implicit functions in feature space for 3d shape reconstruction and completion. In *Proc. IEEE Conf. CVPR*. 6970–6981.

Julian Chibane, Gerard Pons-Moll, et al. 2020b. Neural unsigned distance fields for implicit function learning. *NeurIPS* 33 (2020), 21638–21652.

Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 2016. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conf. Computer Vision*. Springer, 628–644.

Brian Curless and Marc Levoy. 1996. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 303–312.

Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J Mitra, and Michael Wimmer. 2020. Points2surf learning implicit surfaces from point clouds. In *European Conf. Computer Vision*. Springer, 108–124.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Int. Conf. Machine Learning*. PMLR, 1126–1135.

BG Galerkin. 1915. On electrical circuits for the approximate solution of the Laplace equation. *Vestnik Inzh* 19 (1915), 897–908.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proc. IEEE Conf. CVPR*. IEEE, 3354–3361.

Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. 2020. Local deep implicit functions for 3d shape. In *Proc. IEEE Conf. CVPR*. 4857–4866.

Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas A Funkhouser. 2019. Deep Structured Implicit Functions. *arXiv preprint arXiv:1912.06126* (2019).

Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 2018. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proc. IEEE Conf. CVPR*. 9224–9232.

Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. 2020. Implicit Geometric Regularization for Learning Shapes. In *Int. Conf. Machine Learning*. PMLR, 3789–3799.

Benoit Guillard, Federico Stella, and Pascal Fua. 2021. MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks. *arXiv preprint arXiv:2111.14549* (2021).

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. 2020. Point2Mesh: a self-prior for deformable meshes. *ACM Trans. Graph.* 39, 4 (2020), 126–1.

Shi-Min Hu, Dun Liang, Guo-Ye Yang, Guo-Wei Yang, and Wen-Yang Zhou. 2020. Jittor: a novel deep learning framework with meta-operators and unified graph execution. *Science China Information Sciences* 63, 12 (2020), 1–21.

Jiahui Huang, Shi-Sheng Huang, Haoxuan Song, and Shi-Min Hu. 2021. DI-Fusion: Online Implicit 3D Reconstruction with Deep Priors. In *Proc. IEEE Conf. CVPR*. 8932–8941.

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).

Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. 2020. Local implicit grid representations for 3d scenes. In *Proc. IEEE Conf. CVPR*. 6001–6010.

Yiwei Jin, Diqiong Jiang, and Ming Cai. 2020. 3d reconstruction using deep learning: a survey. *Communications in Information and Systems* 20, 4 (2020), 389–413.

Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Trans. Graph.* 32, 3 (2013), 1–13.

Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing, Cagliari, Sardinia, Italy, June 26-28, 2006 (ACM International Conference Proceeding Series, Vol. 256)*, Alla Sheffer and Konrad Polthier (Eds.). Eurographics Association, 61–70.

Jianwei Li, Wei Gao, Yihong Wu, Yangdong Liu, and Yanfei Shen. 2022. High-quality indoor scene 3D reconstruction with RGB-D cameras: A brief review. *Computational Visual Media* (2022), 1–25.

Yiyi Liao, Simon Donne, and Andreas Geiger. 2018. Deep marching cubes: Learning explicit surface representations. In *Proc. IEEE Conf. CVPR*. 2916–2925.

Yaron Lipman. 2021. Phase Transitions, Distance Functions, and Implicit Neural Representations. In *Int. Conf. Machine Learning*. PMLR, 6702–6712.

Shi-Lin Liu, Hao-Xiang Guo, Hao Pan, Peng-Shuai Wang, Xin Tong, and Yang Liu. 2021. Deep implicit moving least-squares functions for 3d reconstruction. In *Proc. IEEE Conf. CVPR*. 1788–1797.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM Trans. Graph.* 21, 4 (1987), 163–169.

Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. 2021. Acorn: adaptive coordinate networks for neural scene representation. *ACM Trans. Graph.* 40, 4 (2021), 1–13.

Donald Meagher. 1982. Geometric modeling using octree encoding. *Computer graphics and image processing* 19, 2 (1982), 129–147.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy networks: Learning 3d reconstruction in function space. In *Proc. IEEE Conf. CVPR*. 4460–4470.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages.

Ken Museth. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph.* 32, 3 (2013), 1–22.

Yutaka Ohtake, Alexander G. Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. 2003. Multi-level partition of unity implicits. *ACM Trans. Graph.* 22, 3 (2003), 463–470.

Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proc. IEEE Conf. CVPR*. 165–174.

Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. 2021. Shape as points: A differentiable poisson solver. *NeurIPS* 34 (2021), 13032–13044.

Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020. Convolutional occupancy networks. In *European Conf. Computer Vision*. Springer, 523–540.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proc. IEEE Conf. CVPR*. 652–660.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. 2019. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237* (2019).

Raj Shekhar, Elias Fayyad, Roni Yagel, and J Fredrick Cornhill. 1996. Octree-based decimation of marching cubes surfaces. In *Proceedings of Seventh Annual IEEE Visualization'96*. IEEE, 335–342.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. *NeurIPS* 33 (2020), 7462–7473.

Haoxuan Song, Jiahui Huang, Yan-Pei Cao, and Tai-Jiang Mu. 2021. HDR-Net-Fusion: Real-time 3D dynamic scene reconstruction with a hierarchical deep reinforcement network. *Computational Visual Media* 7, 4 (2021), 419–435.

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 *Proc. IEEE Conf. CVPR*. 11358–11367.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. 2020. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS* 33 (2020), 7537–7547.

Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. 2022. TorchSparse: Efficient Point Cloud Inference Engine. In *Conference on Machine Learning and Systems (MLSys)*.

Jiapeng Tang, Jiabao Lei, Dan Xu, Feiying Ma, Kui Jia, and Lei Zhang. 2021b. SA-ConvONet: Sign-Agnostic Optimization of Convolutional Occupancy Networks. In *Proc. IEEE Int. Conf. Computer Vision*. 6504–6513.

Jia-Heng Tang, Weikai Chen, Bo Wang, Songrun Liu, Bo Yang, Lin Gao, et al. 2021a. OctField: Hierarchical Implicit Functions for 3D Modeling. *NeurIPS* 34 (2021).

Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. 2019. What do single-view 3d reconstruction networks learn?. In *Proc. IEEE Conf. CVPR*. 3405–3414.

Benjamin Ummenhofer and Vladlen Koltun. 2021. Adaptive Surface Reconstruction with Multiscale Convolutional Kernels. In *Proc. IEEE Int. Conf. Computer Vision*. 5651–5660.

Vecteezy. 2022. *Nature Vectors by Vecteezy*. Retrieved May 1, 2022 from https://www.vecteezy.com/free-vector/nature

Ignacio Vizzo, Xieyuanli Chen, Nived Chebrolu, Jens Behley, and Cyrill Stachniss. 2021. Poisson surface reconstruction for LiDAR odometry and mapping. In *IEEE Int. Conf. Robotics Automation*. IEEE, 5624–5630.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Trans. Graph.* 36, 4 (2017), 1–11.

Peng-Shuai Wang, Yang Liu, and Xin Tong. 2022. Dual Octree Graph Networks for Learning Adaptive Volumetric Shape Representations. *arXiv preprint arXiv:2205.02825* (2022).

Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. 2018. Adaptive O-CNN: A patch-based deep representation of 3D shapes. *ACM Trans. Graph.* 37, 6 (2018), 1–11.

Francis Williams, Zan Gojcic, Sameh Khamis, Denis Zorin, Joan Bruna, Sanja Fidler, and Or Litany. 2021a. Neural Fields as Learnable Kernels for 3D Reconstruction. *arXiv preprint arXiv:2111.13674* (2021).

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

Francis Williams, Matthew Trager, Joan Bruna, and Denis Zorin. 2021b. Neural splines: Fitting 3d surfaces with infinitely-wide neural networks. In *Proc. IEEE Conf. CVPR*. 9949–9958.

Jianglong Ye, Yuntao Chen, Naiyan Wang, and Xiaolong Wang. 2022. GIFS: Neural Implicit Function for General Shape Representation. *arXiv preprint arXiv:2204.07126* (2022).

Jiwen Zhang. 1996. C-curves: an extension of cubic curves. *Computer Aided Geometric Design* 13, 3 (1996), 199–217.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).

A OPTIMAL SOLUTION TO THE SURFACE FITTING VARIATIONAL PROBLEM

The energy in Eq (2) could be transformed as:

$$E(f) := \iiint_\Omega \left(\left\| \nabla f - \vec{\mathbf{N}} \right\|_2^2 + \lambda \sum_{\boldsymbol{p}\in\mathcal{P}} \left(f(\boldsymbol{p}) - \xi(\boldsymbol{p})\right)^2 \mathbb{1}(\boldsymbol{x}-\boldsymbol{p}) \right) \mathrm{d}\boldsymbol{x}, \tag{14}$$

where $\mathbb{1}(\cdot)$ is the Dirac delta function satisfying $\iiint_\Omega \mathbb{1}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 1$ and $\forall \boldsymbol{x} \neq 0, \mathbb{1}(\boldsymbol{x}) = 0$.

Taking the multi-dimensional extension of Euler-Lagrange equation, we could find the stationary point of E satisfies:

$$\Delta f - \nabla \cdot \vec{\mathbf{N}} + \lambda \sum_{\boldsymbol{p}\in\mathcal{P}} \left(f(\boldsymbol{p}) - \xi(\boldsymbol{p})\right) \mathbb{1}(\boldsymbol{x}-\boldsymbol{p}) = 0. \tag{15}$$

As a differentiable functional is always stationary at its local extrema, the quasi-convexity of the above problem would generate almost surely the optimal solution to the surface fitting energy, solved by our NGS.

Per Galerkin method, converting the strong form in Eq (15) to a weak form by introducing the test function ω leads to:

$$\forall \omega, \iiint_\Omega \omega \left(\Delta f - \nabla \cdot \vec{\mathbf{N}} \right) \mathrm{d}\boldsymbol{x} + \lambda \sum_{\boldsymbol{p}\in\mathcal{P}} \omega(\boldsymbol{p}) \left(f(\boldsymbol{p}) - \xi(\boldsymbol{p})\right). \tag{16}$$

The first integrand could be simplified using integration by parts $\omega\Delta f = \nabla\cdot(\omega\nabla f) - \nabla f\cdot\nabla\omega$, and stokes theorem that exploits the boundary conditions over Ω. We further replace the arbitrary test functions with the basis functions, along with $f(\boldsymbol{p}) = \sum_{\bar{k}} \alpha_{\bar{k}} \mathcal{B}_{\bar{k}}(\boldsymbol{p})$ using the product subscript, obtaining the following discretization:

$$\begin{aligned} \forall \bar{k}, \quad \sum_{\bar{l}} \alpha_{\bar{l}} \iiint_\Omega \nabla\mathcal{B}_{\bar{k}}^\top \nabla\mathcal{B}_{\bar{l}} \,\mathrm{d}V + \lambda \sum_{\bar{l}} \alpha_{\bar{l}} \sum_{\boldsymbol{p}\in\mathcal{P}} \mathcal{B}_{\bar{k}}(\boldsymbol{p})\mathcal{B}_{\bar{l}}(\boldsymbol{p}) \\ = \iiint_\Omega \nabla\mathcal{B}_{\bar{k}}^\top \vec{\mathbf{N}} \,\mathrm{d}V + \lambda \sum_{\boldsymbol{p}\in\mathcal{P}} \mathcal{B}_{\bar{k}}(\boldsymbol{p})\xi(\boldsymbol{p}), \end{aligned} \tag{17}$$

giving the exact same form $\mathbf{L}\boldsymbol{\alpha} = \mathbf{d}$ as in Eq (5).

B EFFICIENT COMPUTATION OF THE INTEGRAL

As $\mathcal{B}(\boldsymbol{p})$ only applies simple translation and uniform scaling to $B(\boldsymbol{p})$, it could be similarly decomposed axis-wise into three components composed of elementary functions. With the form $\mathcal{B}_{\bar{k}}(\boldsymbol{p}) = \sum_u m^{\mathrm{x}}_{u,\bar{k}} q^{\mathrm{x}}_{u,\bar{k}}(x) \cdot \sum_u m^{\mathrm{y}}_{u,\bar{k}} q^{\mathrm{y}}_{u,\bar{k}}(y) \cdot \sum_u m^{\mathrm{z}}_{u,\bar{k}} q^{\mathrm{z}}_{u,\bar{k}}(z)$ (we omit the zero branch in Eq (6) for clarity), the integral in $\mathbf{L}$ can be factored out as:

$$\iiint_\Omega \nabla\mathcal{B}_{\bar{k}}^\top \nabla\mathcal{B}_{\bar{l}} \,\mathrm{d}V = \dot{\theta_x}\theta_y\theta_z + \theta_x\dot{\theta_y}\theta_z + \theta_x\theta_y\dot{\theta_z}, \tag{18}$$

where

$$\begin{aligned} \theta_a &= \sum_u \sum_v m^a_{u,\bar{k}} m^a_{v,\bar{l}} \int q^a_{u,\bar{k}}(a) q^a_{v,\bar{l}}(a) \,\mathrm{d}a, \\ \dot{\theta_a} &= \sum_u \sum_v m^a_{u,\bar{k}} m^a_{v,\bar{l}} \int \frac{\partial q^a_{u,\bar{k}}}{\partial a} \cdot \frac{\partial q^a_{v,\bar{l}}}{\partial a} \,\mathrm{d}a, \\ & a \in \{x, y, z\}. \end{aligned} \tag{19}$$

Here, in both θ_a and $\dot{\theta_a}$ all integrals are axis-aligned and one-dimensional. As these integrals do not contain the variables $\{m_u\}$ predicted by the network, they could be pre-computed either by hand or via automatic symbolic packages (in our implementation as a pre-processing step), as q_u are all elementary functions.

Similarly, the terms in $\mathbf{d}$ can be factored out as:

$$\iiint_\Omega \nabla\mathcal{B}_{\bar{k}}^\top \vec{\mathbf{N}} \,\mathrm{d}V = \sum_{\mathbf{v}_{\bar{l}} \in \mathrm{Rgn}(\mathbf{v}_{\bar{k}})} \mathbf{N}_{\bar{l}}^\top \begin{bmatrix} \dot{\rho_x}\rho_y\rho_z \\ \rho_x\dot{\rho_y}\rho_z \\ \rho_x\rho_y\dot{\rho_z} \end{bmatrix}, \tag{20}$$

where

$$\begin{aligned} \rho_a &= \sum_u m^a_{u,\bar{k}} \int_{a\in\mathrm{Rgn}(\mathbf{v}_{\bar{k}})} q^a_{u,\bar{k}}(a) \,\mathrm{d}a, \\ \dot{\rho_a} &= \sum_u m^a_{u,\bar{k}} \int_{a\in\mathrm{Rgn}(\mathbf{v}_{\bar{k}})} \frac{\partial q^a_{u,\bar{k}}}{\partial a} \,\mathrm{d}a, \\ & a \in \{x, y, z\}. \end{aligned} \tag{21}$$

With the pre-computed terms, the evaluation of the 3-dimensional integral could be reduced to simple multiplications and additions that are readily parallelizable.