Enhancing Surface Neural Implicits with Curvature-Guided Sampling and Uncertainty-Augmented Representations

Lu Sang, Abhishek Saroha, and Maolin Gao Daniel Cremers

[1] Technical University of Munich
[2] Munich Center for Machine Learning
lu.sang, abhishek.saroha, maolin.gao, cremers@tum.de

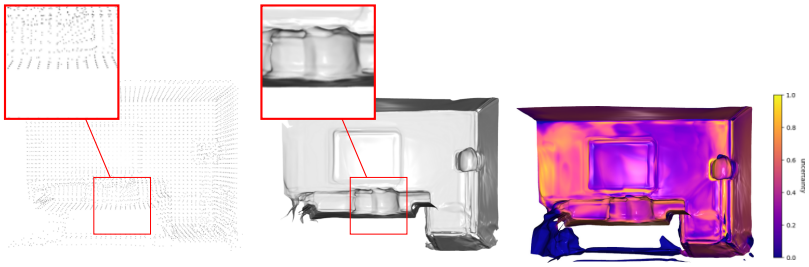


Fig. 1: Our Reconstruction of a room (middle) with sparse input contains only $6k$ points (left) and the estimated uncertainty (right).

Abstract. Neural implicit representations have become a popular choice for modeling surfaces due to their adaptability in resolution and support for complex topology. While previous works have achieved impressive reconstruction quality by training on ground truth point clouds or meshes, they often do not discuss the data acquisition and ignore the effect of input quality and sampling methods during reconstruction. In this paper, we introduce a method that directly digests depth images for the task of high-fidelity 3D reconstruction. To this end, a simple sampling strategy is proposed to generate highly effective training data, by incorporating differentiable geometric features computed directly based on the input depth images with only marginal computational cost. Due to its simplicity, our sampling strategy can be easily incorporated into diverse popular methods, allowing their training process to be more stable and efficient. Despite its simplicity, our method outperforms a range of both classical and learning-based baselines and demonstrates state-of-the-art results in both synthetic and real-world datasets.

Keywords: neural implicits · surface reconstruction · SDF

1 Introduction

Reconstructing neural implicit representations of surfaces is crucial for a range of downstream applications [1, 8]. While numerous approaches utilize point clouds

or meshes to model surfaces with Multi-Layer Perceptrons (MLPs), they often assume already available high-quality, dense input. How to acquire these inputs and how to deal with sparse and noisy training data has been often ignored. For example, some methods require (oriented) surface normals to produce satisfactory results [17]. In real-world scenarios, the requirement on high-quality point cloud and its normals is often hard to fulfill, which leads to failure cases of recent reconstruction methods [11,17]. Furthermore, the uneven distribution of training data and the influence of sampling strategies on training efficiency and output quality prompt a significant question: How can we tailor the sampling strategy to the characteristics of available input data to improve training?

In this paper, we introduce an innovative approach that directly leverages raw depth images from sensors to train uncertainty-aware neural implicit functions for high-fidelity 3D reconstruction tasks, ranging from single objects to large scenes. Our uncertainty-aware neural implicits can represent open surfaces, hence extending the representation capacity of SDF, which traditionally requires a clear notion of surface inside and outside. Our method builds an intermediate representation of a coarse voxel grid with various geometric properties, such as gradients and curvatures, based on input depth images, which remarkably enables continuous sampling inside the (discrete) grid. Consequently, our proposed method can handle sparse, low-quality input and reconstruct large open surfaces with high fidelity as shown in Fig. 1. In summary, our contributions are:

- We introduce a novel method, which can deal with raw input depth images to reconstruct surfaces ranging from single objects to large scenes under the same framework.
- We propose a method that computes mean curvature directly from input depth images and devises a simple yet effective sampling strategy to generate training data for faster and more accurate neural implicit fitting.
- Our sampling strategy is lightweight in terms of computational time and can be easily integrated into existing neural implicit reconstruction methods.
- Our uncertainty-aware implicit neural representation enables, for the first time, SDF-based open surface reconstruction.

2 Related Work

2.1 Surface Representation

Surface representation can be classified into two categories based on the stored surface properties: explicit surface representation (*e.g.*, polygon meshes or point clouds) and implicit surface representation (*e.g.*, signed distance fields). Explicit methods struggle with complex topologies, resolution adjustments, local modifications, and potential high memory consumption when storing high-resolution surfaces. In contrast, implicit representations represent the surfaces by storing indirect information about the surface, which overcomes the shortages of explicit methods. However, **classical** implicit methods still suffer from fixed resolution and high memory consumption issues. For example, the memory consumption

is $\mathcal{O}(n^3)$ for a voxel grid of size n^3. Luckily, **neural** implicit representations can encode the surface implicit information such as occupancies [10, 16, 28], signed/unsigned distance functions [9, 11, 17, 29, 31, 32] into neural networks. One can query any 3D points in space to obtain the corresponding attributes. This approach allows for recovering highly detailed surfaces at a lower memory cost than traditional surface representation methods such as classical signed distance field (SDF). Moreover, it is a continuous representation suitable for further mathematical analysis of the represented surface. A distance field stores the shortest (signed/unsigned) distance of a given point to a surface. The gradients of an SDF provide a normal vector of the surface, but using SDF requires well-defined "inside" and "outside" notions, which restricts representing open surfaces. An unsigned distance field (UDF) [34] is devoid of this specified concern. However, the lack of a sign introduces ambiguity in surface reconstruction, such as the flipped surface shown in Fig. 6. Consequently, the development of a novel approach is crucial to integrate the benefits of both signed and unsigned distance fields.

2.2 Surface Reconstruction

Similar to surface representations, surface reconstructions techniques can be categorized into traditional approaches, including **explicit** methods like Poisson surface reconstruction [22] and SSD [5]. However, these methods heavily depend on the quality of the input data and often struggle with complex geometries or sparse and noisy inputs. Furthermore, adjusting the output resolution with traditional methods requires repeating the entire reconstruction process, which is a significant drawback. On the other hand, **implicit** surface reconstruction approaches, such as Marching Cubes [25], rely on either classical discrete voxel grids or neural networks to derive surface features [10, 17, 29, 32, 33, 35]. Recent advances in implicit reconstruction methods, combined with neural surface representation, facilitate changing the output resolution without the need to retrain the network or update the voxel grid. These methods are also more adept at handling complex geometries. Typically, the input for these methods is 3D data, such as point clouds or meshes [21, 32]. Some methods also need point clouds with normals (oriented point clouds) to ensure satisfactory results [17, 35]. Some need ground truth meshes to provide supervision [31, 33]. Additionally, certain approaches incrementally use an initialized mesh for reconstruction instead of directly learning the implicit representation [19, 33].

2.3 Training Data Sampling

Sparse Training Data Learning-based methods sample data from input to train the neural implicit network. Sampling training data is easy when meshes are given [31], since it allows for infinite sampling. When only limited samples are available, such as point clouds [17, 35], the commonly used random sampling does not consider the data distribution and ignores the local shape geometry characteristic, however complex geometry area may need more learning attention. This gap may lead to bad reconstruction results [10, 11].

Biased Sampling Another problem of random sampling is that point clouds, especially those acquired directly from the real world, are not uniformly located on the surface [31, 40]. Moreover, points extracted from the iso-surface will likely be gathered near high-curvature areas [40]. Plus, complex surface areas need more points to represent their features. Random sampling does not consider these effects. To avoid this issue, Yang *et al*. [40] samples on and near the iso-surface with some tolerance. Novello *et al*. [31] proposes to sample according to principal curvatures of the surface points such that the sampled points are evenly distributed according to the curvatures. They divide points into different curvature categories according to the absolute sum of two principal curvatures. However, *ground-truth meshes are required* in their computation pipeline.

Input Data Acquisition The most commonly used input data to train neural implicits are point clouds or meshes. However, it is hard to get noise-free point clouds in real applications, let alone oriented point clouds. The most handy way to acquire point clouds in the real world is to use **RGB-D** sensor. Most previous papers do not discuss the input data acquisition step. Other depth-based methods focus on camera trajectory estimation [30] or accurate depth fusion [39].

In our work, we utilize depth as input but the fundamental differences between our work with other depth-based methods are twofold: firstly, we use depth images for geometrical feature computation that helps us to generate effective training samples. Secondly, our goal is to obtain the continuous neural implicit representation of the underlying surface captured by depth images.

To tackle the above challenges, we propose a method that computes an intermediate coarse voxel grid from depth images, effectively addressing the issues related to input points with normals. The voxel grid allows to locate query points in space and obtain the corresponding attributes such as SDF value by an efficient Taylor approximation, thereby resolving the sparse input problem. Moreover, we embed an uncertainty value into implicit surface representation to indicate the reliability of the SDF value. The uncertainty value helps to eliminate redundant areas and enables the reconstruction of open surfaces.

3 Method

Our goal is to train a network $f(\mathbf{x}, \theta) : \mathbb{R}^3 \rightarrow \mathbb{R} \times [0, 1]$, which predicts the SDF value together with its uncertainty, such that the reconstructed surface $\mathcal{S}$ lies on the level-set $\{\mathbf{x}|f(\mathbf{x}) = 0\}$. To achieve this goal, only a set of depth images $\{D_k\}$ of the object (or scene) of interest is required as input, which resembles a highly relevant application scenario in practice.

A full pipeline of our method is illustrated in Fig. 2. In summary, we first initialize a coarse voxel grid based on input depth images and further augment the voxel representation by integrating its corresponding curvature and uncertainty (Sec. 3.1). After the coarse voxel grid has been prepared, we sample points from the grid to generate data. We use curvature-guided sampling for on-surface samples to overcome the unevenly distributed points problem and use the off-surface sampling method to create samples in arbitrary positions, to solve the sparse training data

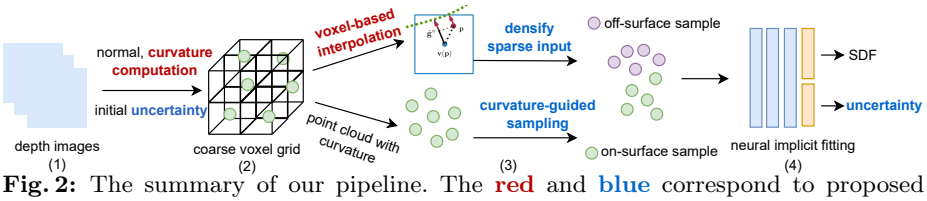


Fig. 2: The summary of our pipeline. The **red** and **blue** correspond to proposed theoretical and architectural improvement. After a customized coarse voxel initialization with uncertainty w^v, curvature H, and normal $\hat{\mathbf{g}}^v$, we use curvature-guided sample on the extracted point cloud and using voxel-based sampling to generate more training points in the space.

problem (Sec. 3.2). Finally, our network predicts uncertainty together with SDF values. We show that we can therefore extract non-watertight surfaces for open surfaces (Sec. 3.3). We also discuss the possibilities of incorporating our proposed techniques into existing methods and show a concrete example in the case of NeuralPull [26] (Sec. 3.4), showcasing the flexibility of our proposed techniques.

3.1 Coarse Voxel Grid

We utilize a coarse voxel grid $\{\mathbf{v}_i\} \subset \mathbb{R}^3, i \in \mathcal{V}$ following the standard TSDF fusion method [30]. For each voxel $\mathbf{v}_i \in \mathbb{R}^3$, the SDF value $\psi_i^v \in \mathbb{R}$, the corresponding curvature $H_i^v \in \mathbb{R}$ and uncertainty $w_i^v \in [0, 1]$ are initialized. (the details of uncertainty computation are described in the supplementary.) Inspired by [36], we also integrate the gradient of SDF $\mathbf{g}_i^v \in \mathbb{R}^3$ for each voxel using the computed normal of each pixel in the depth images, as the SDF gradient is the normal direction of the underlying surface. Different from [36], we do not use a hash map to store voxels but a regular grid, which enables off-surface sampling (we will elaborate in Sec. 3.2). Moreover, we only require a coarse voxel grid (*e.g.*, of size 64^3 in our experiments) at this step, while being able to preserve details in the reconstruction, thanks to our efficient sampling strategy (*cf*. Sec. 3.2). With stored gradients in voxel elements, the corresponding point cloud of the object contained in the voxel grid can simply be extracted in one step (without any mesh extraction) by

$$\mathbf{x}_i = \mathbf{v}_i - \hat{\mathbf{g}}_i^v \psi_i^v, \quad \text{where} \quad \hat{\mathbf{g}}_i^v = \frac{\mathbf{g}_i^v}{\|\mathbf{g}_i^v\|}. \tag{1}$$

Direct Curvature From Depth Depth images can provide more geometric information other than normals [14,23]. The mean curvature and other differential geometry features, such as the Gaussian curvature, are local geometrical properties of the surface and reveal the local topological characteristics. The mean curvature H is the average of the principal curvatures, while the Gaussian curvature K is their product. We propose a method that computes curvature information directly based on (discrete) depth images *without (continuous) ground-truth meshes*. A depth image D can be viewed as a Monge patch of a surface, i.e. $z = D(m, n), (m, n) \in \Omega \subset \mathbb{R}^2$ with pixel coordinate (m, n) lies in the image

domain Ω [6,37]. Hence the Monge patch $\mathcal{M} : \Omega \rightarrow \mathbb{R}^3$ is defined as $\mathcal{M}(m,n) = (m, n, D(m,n))$. The two types of curvatures can be computed by

$$K(m,n) = \frac{D_{mm}D_{nn} - D_{mn}^2}{(1 + D_m + D_n)^2}\,, \tag{2}$$

$$H(m,n) = \frac{(1 + D_m^2)D_{nn} - 2D_m D_n D_{mn} + (1 + D_n^2)D_{mm}}{2(1 + D_m^2 + D_n^2)^{3/2}}\,, \tag{3}$$

where $D_m = \frac{\partial}{\partial m}D(m,n)$ is the partial derivative of depth w.r.t. x-axis. Similarly $D_n = \frac{\partial}{\partial n}D(m,n)$, $D_{mn} = \frac{\partial^2}{\partial m \partial n}D(m,n)$ and other second order derivatives. The computation is done on the fly per depth image. To our knowledge, we are the first to compute mean curvatures directly from depth images and integrate them into voxel grids for efficient sampling afterward (*cf*. Sec. 3.2). After associating a curvature to each pixel of depth images, we unproject 2D pixels into 3D space to determine its corresponding voxel element. Note that we can fuse curvatures which are computed under image coordinates, into world (voxel) coordinates due to the mean curvature $H(m,n)$ and Gaussian curvature $K(m,n)$ are invariant to changes of the parameterization on the smooth surface represented by $\mathcal{M}(m,n)$ [6]. A detailed explanation is provided in the supplementary. Fig. 3a illustrates that our estimated curvature indeed captures the local geometric properties of the surface.

Voxel attributes initialization For each voxel, we use a weighted averaging to iteratively update its gradient $\mathbf{g}^v$, curvature H^v, SDF ψ^v, and uncertainty w^v information. This process helps to reduce the error caused by noisy and sparse depth images (*cf*. supplementary for details). The time required for normal and curvature computation of a 480×640 depth image, plus updating voxel attributes according to this incoming depth, is $\sim$**50ms**. If camera poses are not available from the given depth images, we run camera pose estimation steps together with the voxelization step (see [36] and our supplementary). The whole voxelization step plus camera pose estimation (when ground truth camera poses are absent) takes $\sim$**120ms** per depth image.

3.2 Efficient Sampling Strategy

To fit a neural implicit function, pairs of 3D points and the corresponding SDF values are often needed as the supervision [10,31,32]. Many previous works only sample points in a point cloud [17,35] or need ground truth mesh to compute the signed distance when sampling in the space [31]. All these methods produce unsatisfactory results when input data are sparse and ground truth mesh is not available. In this section, we introduce an interpolation strategy that deals with the sparse input and deploys the estimated gradient and curvature stored in the coarse voxel grid to generate both on- and off-surface samples, which is crucial for effective neural fitting, as shown in our extensive experiments.

On-Surface Sampling To sample points on the surface, one naive approach is to uniformly (random) sample on surface points utilized using Eq. (1). However,

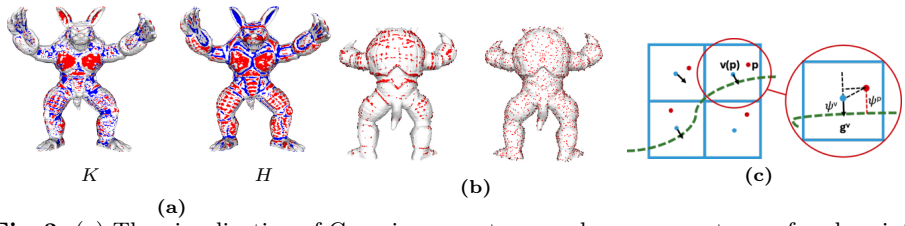


Fig. 3: (a) The visualization of Gaussian curvatures and mean curvatures of each point. The red color indicates a high curvature area, and the blue color indicates a low curvature area. A positive mean curvature ($H > 0$) signifies a convex surface, and a negative mean curvature ($H < 0$) indicates a concave surface. Positive Gaussian curvature ($K > 0$) indicates that the surface is locally like a dome, and negative Gaussian curvature ($K < 0$) indicates that the surface is locally saddle-shaped. (b) Points gathering on high curvature effect (left) and our sampling results after considering points mean curvature (right). (c) Illustration of Eq. (4). For each query point that falls in voxel $\mathbf{v}$, the SDF value is the SDF value of the voxel ψ^v plus the projected distance of the voxel center to the query point to the gradient direction.

the surface points extracted by this step are often concentrated at the high curvature area (see Fig. 3b left) [40] and this uneven distribution will be inherited by the uniform sampling during training. To avoid this uneven sampling problem, we divide sampled points into low, median, and high curvature regions based on the mean curvature, similarly as in [31]. Note that the surface point extracted by Eq. (1) inherit the integrated mean curvature H. Moreover, the Gaussian curvature can be employed as well and we prove that these two metrics are equivalent (see Fig. 3a and supplementary). Every epoch, m points are sampled in each of three curvature regions defined by the thresholds $\underline{\mathrm{H}}$ and $\bar{H}$, which are chosen to be 0.3 and 0.7 percentile of the entire curvature range.

Off-Surface Sampling To sample point $\mathbf{p}$ random in space and directly use it for neural implicit fitting, its SDF value ψ^p is required. This is easy to achieve when the input is mesh since one can easily compute the point-to-mesh distance with a sign. However, when only a discrete structure is available, sampling off the surface is not straightforward. Here we introduce an off-surface sampling strategy. With the coarse voxel grid $\{\mathbf{v}_i\}$ estimated in Sec. 3.1, we can randomly sample points $\mathbf{p} \in \Gamma$, where $\Gamma \subset \mathbb{R}^3$ is the space defined by the voxel grid. The voxel element corresponding to the sampled point $\mathbf{p}$ can be localized by a simple operation $\mathbf{v}(\mathbf{p}) = [\mathbf{p}/v_s]$, where $[\cdot]$ is the rounding operator, $\mathbf{v}(\mathbf{p})$ and v_s are the coordinate of the center and the size of the voxel cube respectively. Instead of inheriting the SDF value of the corresponding voxel ψ^v, we approximate it by the first-order Taylor expansion. Due to the stored gradients $\hat{\mathbf{g}}^v$ in the voxel grid, a randomly sampled point its SDF ψ^p this can be computed simply by (see Fig. 3c for an illustration):

$$\psi^p = \psi^v + \langle \hat{\mathbf{g}}^v, \mathbf{p} - \mathbf{v}(\mathbf{p}) \rangle \,, \tag{4}$$

The uncertainty of $\mathbf{p}$ is interpolated using

$$w^p = \frac{v_s - \psi^p}{v_s} w^v \,, \tag{5}$$

where w^v is the voxel uncertainty estimated in Sec. 3.1. Thanks to the stored gradient, our method does not need any trilinear interpolation or additional nearest neighbor search. It also enables *continuous* sampling in a *discrete* voxel grid, and at the same time retains as much information as possible. Eq. (5) sets the maximal possible uncertainty w^v to points on the surface (i.e. when $\psi^p = 0$) and reduces uncertainty when points are moving away from the surface since the accuracy of Taylor approximation reduces for distant samples away from the surface. Meanwhile, the point $\mathbf{p}$ inherits the normal of the voxel $\hat{\mathbf{g}}^p = \hat{\mathbf{g}}^v$.

3.3 Uncertainty-Aware Neural Implicit Function

Our Loss To recover the neural implicit function $f : \mathbb{R}^3 \to (\psi, w) \subset \mathbb{R} \times [0, 1]$, such that the surface lies on the level-set $\{\mathbf{x}|f(\mathbf{x}) \in 0 \times (\tau, 1]\}$, where $f^{-1}(\{(\psi, w) \in \mathbb{R} \times [0, 1] \,|\, \psi = 0\})$ and w is the uncertainty of the predicted signed distance value ψ. τ is the uncertainty threshold. Given the points sampled in the voxel grid from the previous subsection and their associated SDF value and uncertainty (Eq. (4) and Eq. (5)), we have all the ingredients to train our neural implicit function, which predicts both the SDF value of the query points and its corresponding uncertainty. Finally, our total loss incorporating the geometric and the normal constraints reads

$$l_{\mathcal{X}}(\theta) = \frac{1}{|\Gamma^+|} \int_{\Gamma^+} |\psi - \psi^p| d\Gamma \,, \tag{6}$$

$$l_{\mathcal{W}}(\theta) = \frac{1}{|\Gamma|} \int_{\Gamma} |w - w^p| d\Gamma \,, \tag{7}$$

$$l_{\mathcal{N}}(\theta) = \frac{1}{|\Gamma^+|} \int_{\Gamma^+} (1 - \langle \frac{\nabla_\psi f(\mathbf{p}, \theta)}{\|\nabla_\psi f(\mathbf{p}, \theta)\|}, \hat{\mathbf{g}}^p \rangle) d\Gamma \,, \tag{8}$$

$$l_{\mathcal{E}}(\theta) = \frac{1}{|\Gamma|} \int_{\Gamma} |\, \|\nabla_\psi f(\mathbf{p}, \theta)\|^2 - 1| d\Gamma \,. \tag{9}$$

where Γ^+ indicates the area with the sampled uncertainty $w^p > 0$ and θ is the network parameter.

$$l(\theta, \theta_r) = l_{\mathcal{X}}(\theta) + \tau_n l_{\mathcal{N}}(\theta) + \tau_w l_{\mathcal{W}}(\theta_r) + \tau_e l_{\mathcal{E}}(\theta) \,. \tag{10}$$

The terms $l_{\mathcal{X}}$ and $l_{\mathcal{W}}$ penalize the SDF value and uncertainty differences between the network prediction and the pseudo ground truth interpolated by Eq. (4) and Eq. (5). The normal term $l_{\mathcal{N}}$ term evaluates the cosine similarity of surface normal and implicit function gradient. The Eikonal term $l_{\mathcal{E}}$ regularizes the underlying function to represent a valid signed distance field.

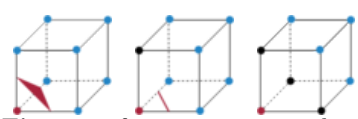


Fig. 4: surface extraction with uncertainty. Black vertices represent zero uncertainty points. Red and blue vertex mean points with negative and positive SDF values, respectively.

Open Surface Reconstruction While signed distance fields are flexible in representing objects with different topologies, they require the underlying object

to be water-tight. On the other hand, uncertainty can serve as an open area indicator. When extracting surface using the Marching cubes algorithm [25], we skip the area where the uncertainty is under a threshold w_c (*e.g.* w_c=0). As shown in Fig. 4, a single zero uncertainty vertex leads to a line (instead of a triangle), while two of this kind lead to a point [3,15]. Therefore, we can naturally reconstruct open surfaces, thanks to our uncertainty estimation.

3.4 Incorporating with Other Methods

Our curvature-guided sampling can be seamlessly incorporated into popular implicit surface reconstruction methods such as [17, 35]. These methods take point clouds as input. The points with curvature can be extracted using Eq. (1) is just $\psi^p = 0$ situation. Additionally, our interpolating method simplifies the method that needs nearest neighbor search, such as NeuralPull [26]. NeuralPull proposes to train a network to learn to pull a query point to the closest iso-surface. During training, the method samples random points $\mathbf{p}$ and uses nearest neighbor search to find the closest surface point $\mathbf{x}$. Then, a network f is trained to minimize the loss.

$$l_{\mathcal{X}}(\theta) = \frac{1}{|\Gamma|} \int_{\Gamma} \left\| \mathbf{x} - \mathbf{p} + \frac{\nabla f(\mathbf{p})}{\|\nabla f(\mathbf{p})\|} f(\mathbf{p}) \right\| d\Gamma \,, \tag{11}$$

with $\tau_n = 0$, $\tau_e = 0$. Our interpolating method (Sec. 3.2) eliminates the nearest neighbor search required in the original NeuralPull. As for a random point $\mathbf{p}$, after finding the corresponding voxel $\mathbf{v}(\mathbf{p})$, the closest surface point $\mathbf{x}$ can be easily located using (1). In Sec. 4, we show that it effectively reduces the noise during training and leads to better reconstruction quality. The modified NeuralPull outperformed the original implementation, especially under sparse inputs.

4 Evaluation

To demonstrate that our proposed method improves the robustness and accuracy of the neural implicit fitting, we validate our method extensively on both **synthetic** and **real-world** datasets, including **objects** and **scene** scenarios. For synthetic datasets with ground-truth mesh, we rendered noise-free depth images and camera poses to compute the coarse voxel grid $\{\mathbf{v}_i\}$. To test different quality inputs, we compare two different voxel resolutions, 64^3 (**coarse**) and 256^3 (**dense**). The coarse voxel grid needs 25 MB while the dense one takes 1.5 GB of memory. The real-world datasets contain RGB-D sequences with noisy, sparse depth images and noise camera poses. Using this setting, we show that even with the intermediate coarse voxel grid, our method can still reconstruct faithfully. We use an 8-layer multi-layer perceptron (MLP) with ReLU activation. Each layer has 256 nodes, and the last layer has 2 output heads for the SDF value and its uncertainty respectively. Note that due to the space constraint, we focus on showing the comparison methods. Ablation study of our loss function together with more comparison details please refer to our supplementary materials.

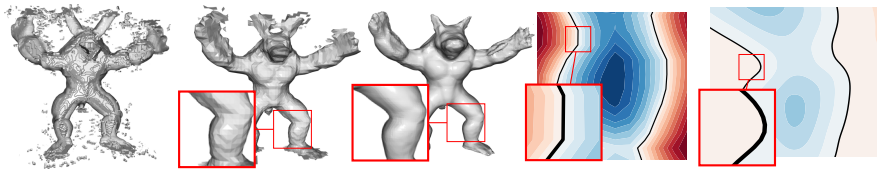


Fig. 5: Visualization of reconstructed meshes (first three columns) of Armadillo (more visualizations *cf*. supplementary). The right side is the visualization of the zoomed-in part in left meshes, the initial discrete SDF stored in 64^3 resolution cube, and continuous SDF represented by a neural network.

Method	**CD**↓($\times 10^2$)				**HD**↓			
	Bunny	Armadillo	Dragon	Buddha	Bunny	Armadillo	Dragon	Buddha
RoutedFusion	0.267	0.272	0.578	0.356	0.116	0.066	0.163	0.086
TSDF	0.073	0.080	0.220	0.287	**0.014**	0.008	0.038	0.018
gradient-SDF	0.111	0.210	0.165	**0.118**	0.028	0.027	**0.028**	0.024
Ours	**0.067**	**0.037**	**0.157**	0.125	0.016	**0.006**	0.040	**0.017**

Table 1: Quantitative comparison with depth-based methods. The best accuracy is bold-faced for each dataset.

Evaluation Metrics Quantitative evaluation is performed on four synthetic datasets, where the 3D ground truth is available. We compute the Chamfer distance (CD) [2] and Hausdorff distance (HD) [20] of the reconstruction w.r.t. the ground-truth.

4.1 Comparison with Depth-Based Methods

We first compare with both classical and learning-based depth-based methods. TSDF fusion [30] is one of the standard works in depth fusion. Gradient-SDF [36] and RoutedFusion [39] are two recent representative classical and learning-based methods respectively. For TSDF fusion, we employ a voxel grid of 128^3. Despite our initial input having only *half the resolution* of the TSDF, we achieve superior outcomes. Gradient-SDF employs a hash map to store individual voxel data, which constructs voxels solely around the projected depth points. This method doesn't specify a fixed voxel resolution; instead, it directly controls the voxel size to control the resolution. Hash map plus gradient stored in voxels helps to reduce the memory and computational time, but the discretely stored voxels make the method unreliable for sparse and noisy depth. RoutedFusion [39] leverages two networks—Routing and Fusion—utilizing both *ground truth camera poses* and *ground truth or previously derived meshes from TSDF* to facilitate their guided fusion. Hence we input noise-free depth images and ground truth camera poses for the experiments of RoutedFusion. Despite its high requirements for the inputs, RoutedFusion creates dense and high-resolution meshes however with a lot of artifacts around it. One reason could be that since the method trains one network for fusing different objects from the same dataset (similar shapes, like ShapeNet [7]), however, in our synthetic datasets the object sizes have very large varies. The method fails to generalize. The quantitative results can be found in Tab. 1 and the visualization results are shown in Fig. 5 (more analysis and

Metric	Dataset		Method							
			SSD	Poisson	IF-NET	SIREN	IGR	IGR (curv)	Ours	Ours (curv)
CD↓ ($\times 10^2$)	Bunny	sparse	0.204	0.278	0.303	8.745	0.481	0.447	0.068	**0.067**
		dense	0.224	0.242	0.315	7.582	0.501	0.491	0.073	**0.068**
	Armadillo	sparse	0.038	0.031	**0.025**	2.396	0.060	0.034	0.039	0.037
		dense	**0.024**	0.025	0.026	2.331	0.035	0.034	0.031	0.030
	Dragon	sparse	0.210	0.206	0.158	2.710	0.209	0.198	0.179	**0.157**
		dense	0.151	0.150	0.181	2.784	0.162	0.154	0.130	**0.126**
	Happy Buddha	sparse	0.180	0.240	0.258	2.717	0.307	0.259	0.135	**0.125**
		dense	0.210	**0.202**	0.258	2.720	0.248	0.240	0.214	0.267
HD↓	Bunny	sparse	0.057	0.074	0.112	0.817	0.149	0.142	0.026	**0.016**
		dense	0.071	0.065	0.107	0.816	0.153	0.155	0.020	**0.012**
	Armadillo	sparse	0.014	0.006	0.007	0.357	0.018	0.008	0.015	**0.006**
		dense	0.005	0.006	**0.004**	0.301	0.014	0.013	0.007	0.007
	Dragon	sparse	0.043	0.050	0.037	0.340	0.037	0.039	**0.030**	0.040
		dense	0.040	0.039	0.036	0.317	0.056	0.045	0.030	**0.025**
	Happy Buddha	sparse	0.048	0.061	0.059	0.338	0.092	0.081	0.023	**0.017**
		dense	0.075	0.057	0.058	0.332	0.059	0.057	0.054	**0.044**

Table 2: Quantitative evaluation of comparison methods: We compare our off-surface sampling method (second last column) and plus on-surface curvature-guided sampling method (last column) with other comparison methods. Additionally, we show our curvature-guided on-surface sampling method incorporated with IGR, compared with the original IGR, see column IGR(curv) and IGR.

results on other datasets please refer to supplementary). We also demonstrate the difference between TSDF (discrete SDF) and Ours (continuous SDF) by visualizing the resulting signed distance fields

4.2 Efficient Sampling Comparison

Off-Surface Point Sampling To evaluate the impact of off-surface sampling on the training of neural implicits and subsequent surface reconstruction—particularly in scenarios of sparse input—we conducted comparisons with traditional methods such as the smooth signed distance (SSD) method [5] and Poisson surface reconstruction [22], both of which generate polygon meshes from point cloud inputs. Additionally, for comparison within the learning-based neural implicit methods, we selected IGR [17], IF-NET [10], NDF [11], and SIREN [35]. Unlike NDF, which predicts the unsigned distance value (UDF) and produces a denser point cloud, the other methods estimate the signed distance function (SDF) values, subsequently utilizing MarchingCubes to derive meshes at specified resolutions. For the Poisson, SSD, IGR, NDF, and SIREN methods, inputs include both sparse and dense point clouds, derived from voxels using equation Eq. (1) to maintain consistent surface point density with normals. Meanwhile, IF-NET employs meshes generated from initialized voxels through MarchingCubes as its input. Fig. 6 shows the visual comparison and the Tab. 2 shows the quantitative comparison. All methods produce satisfactory results when the input points are dense. Our method still gives satisfactory results when the input is sparse.

On-Surface Point Sampling We show that curvature-guided sampling helps in two aspects. First, it stabilizes the learning procedure, leading to a faster convergence of the minimum solution. We show this visually by rendering reconstructed meshes during early training epochs to see the learning efficiency of the

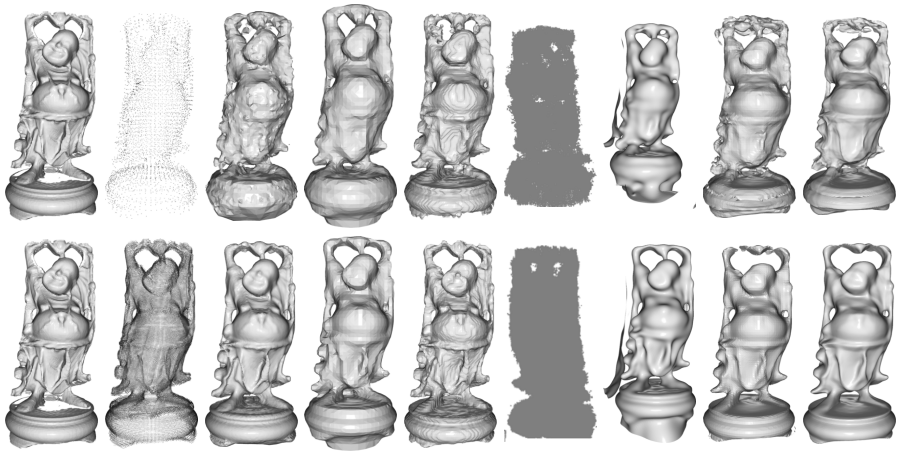


Fig. 6: Comparison results with IGR [17], SIREN [35] and IF-NET [10] with two different density input on synthetic datasets. Sparse input *happy_ buddha* [13] has $\sim 5k$ points, and dense one has $\sim 96k$ points.

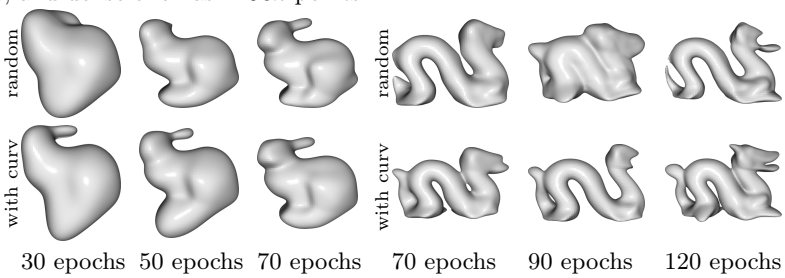


Fig. 7: We extract surface during training to compare the curvature-guided sampling and random sampling.

different sampling methods (see Fig. 7). We then compute these CD and HD to show that curvature-guided sampling has a smoother error curve. In Fig. 9, the solid lines and dash lines are CD errors of curvature-guided sampling and random sampling, respectively. We normalized the CD errors by dividing the maximum error within each dataset to draw all lines in one figure. The curvature-guided sampling lines have a smoother trend and reach a lower error faster. Second, we show that curvature-guided sampling also increases the accuracy of reconstructed meshes. We test our method (with enabled off-surface sampling) with and without curvature-guided sampling (Tab. 2 Ours(curv) and Ours) for a comparison. Both comparison pairs show that considering curvature information during training improves the results.

4.3 Improving Other Methods with Our Sampling Strategy

On-Surface Point Sampling We now show that our on-surface curvature-guided sampling can be incorporated seamlessly with previous neural fitting methods and improve accuracy. We change the sampling method in IGR [17] to

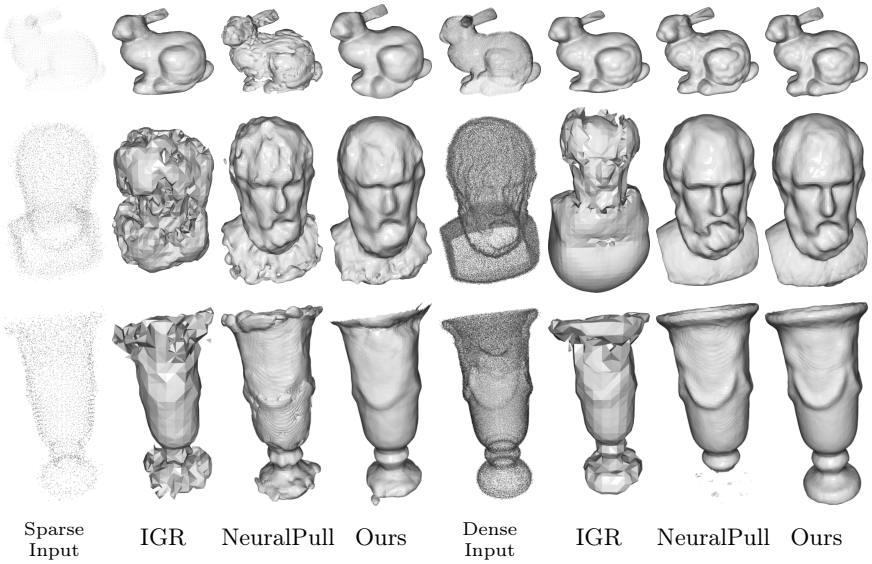


Fig. 8: Comparison results with NeuralPull [26] and IGR [17]. Sparse input on synthetic dataset *bunny* has $\sim 5k$ points and dense one with $\sim 100k$ points. Two real-world datasets *sokrates* have $\sim 9k$ points and $150k$ points in sparse and dense situations, *vase* has $\sim 4k$ and $\sim 81k$ respectively.

curvature-guided sampling during training and compare it to original random sampling (Tab. 2 IGR(curv) and IGR). Note that the point cloud inherits the curvature information from our initialized voxel grids since the points are extracted from the voxel grid.

Off-Surface Point Sampling Additionally, we show that our off-surface sampling method can be easily incorporated into previous works such as NeuralPull [26]. NeuralPull outperforms the other methods when normal information is not available. It uses Eq. Eq. (11) as the geometric loss to learn the pulling vector. However, it has to sample training points using nearest neighbor search and it fails easily when the points are too sparse or noisy. Our off-surface sampling strategy bypasses the nearest neighbor search and thus is more robust compared to the original NeuralPull. In Fig. 8, we show the result of our sampling method incorporated with NeuralPull loss compared to the original method together with the baseline method IGR. Our method and NeuralPull do not use normals in their loss terms, while IGR still uses point normals. The first two rows are synthetic datasets, and the last two rows are noisy real-world datasets *vase* and *sokrates* [41]. The noise originates from both the depth images and camera poses. IGR [17] works well for synthetic datasets, whereas NeuralPull fails in sparse inputs. Our method compares favorably across all scenarios. Note

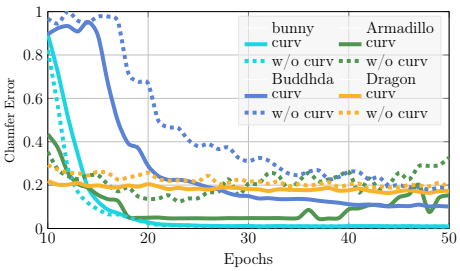


Fig. 9: The CD error vs. training epochs.

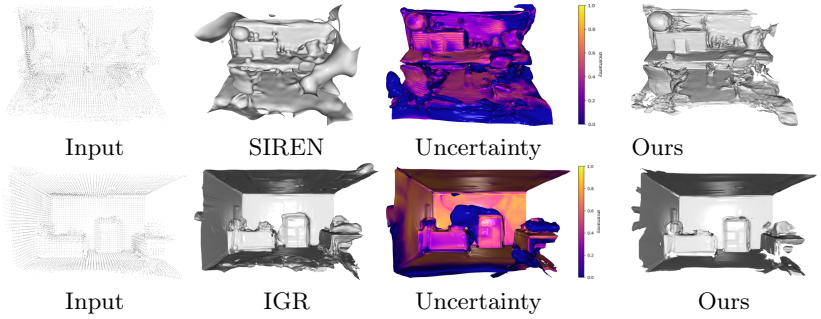


Fig. 10: Scene reconstruction results with sparse input on real-world dataset *TUM_rgbd* (first rows, sparse points $\sim 14k$), with noisy camera poses. Synthetic dataset *icl_nium* (last rows, sparse points $\sim$ $14k$) with ground truth camera poses. The complete comparison is included in supplementary material.

that the key improvement is bypassing the nearest neighbor search step, thus for the works with similar step [24,27], can easily employ our strategy.

4.4 Uncertainty Prediction

In this section, we show the uncertainty prediction result in Fig. 10, which illustrates that the uncertainty helps to eliminate redundant areas. We focus on showing the results on the scene dataset to show that with the help of uncertainty, we can also represent an open surface. The camera poses are estimated during the voxelization step. Many previous works [11,35] have also trained neural networks to represent scene-level surfaces. However, a method such as [35] produces extra artifacts outside the surface. Although the authors propose one term in the loss function to penalize off-surface points for creating SDF values close to 0, it can not eliminate all artifacts, especially when the input is sparse and noisy. This problem can be solved by considering uncertainty during surface extraction as described in Sec. 3.3. Due to space constraints, we show a subset of comparison results. For more results, please refer to the supplementary material.

5 Conclusion

Summary In this work, we have presented novel curvature-guided sampling methods with uncertainty-augmented surface implicits representation. Our method is easily transferred to existing methods that can efficiently deal with low-quality inputs. Our approach operates on depth images, which can be directly acquired from hardware. We propose a method that computes surface geometric properties: normals and curvatures on depth images instead of using ground truth mesh, which is more suitable for real-world applications. The by-product uncertainty gives a reliability indication for the predicated signed distance value and can help with non-closed surface representations.

Limitations and Future Work Our approach does not specialize in surface completion. It will fail to recover any missing areas in depth. Next, since the presented method can be easily integrated with other neural reconstruction methods

and shape completion techniques, we plan to incorporate shape completion and neural rendering to deal with missing areas.

References

1. Atzmon, M., Novotny, D., Vedaldi, A., Lipman, Y.: Augmenting implicit neural shape representations with explicit deformation fields. arXiv preprint arXiv:2108.08931 (2021) 1
2. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer-Verlag TELOS, Berlin, Heidelberg, 2 edn. (2000) 10
3. Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., Levy, B.: Polygon Mesh Processing. CRC Press (2010) 9
4. Bylow, E., Sturm, J., Kerl, C., Kahl, F., Cremers, D.: Real-time camera tracking and 3d reconstruction using signed distance functions. In: Robotics: Science and Systems (2013) 22, 23, 24
5. Calakli, F., Taubin, G.: Ssd: Smooth signed distance surface reconstruction. Comput. Graph. Forum **30**(7), 1993–2002 (2011) 3, 11, 18, 27, 29
6. do Carmo, M.P.: Differential geometry of curves and surfaces. Prentice Hall (1976) 6
7. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: Shapenet: An information-rich 3d model repository (2015) 10, 25
8. Chen, H., Zheng, C., Wampler, K.: Local deformation for interactive shape editing (2023) 1
9. Chen, Z., Zhang, H.: Learning implicit fields for generative shape modeling. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5932–5941 (2019) 3
10. Chibane, J., Alldieck, T., Pons-Moll, G.: Implicit functions in feature space for 3d shape reconstruction and completion. CoRR **abs/2003.01456** (2020) 3, 6, 11, 12, 18, 22, 25, 27, 29
11. Chibane, J., Mir, A., Pons-Moll, G.: Neural unsigned distance fields for implicit function learning. In: Advances in Neural Information Processing Systems (NeurIPS) (December 2020) 2, 3, 11, 14, 18, 25, 27, 29
12. Choi, S., Zhou, Q.Y., Miller, S., Koltun, V.: A large dataset of object scans. arXiv:1602.02481 (2016) 18, 21, 22, 28, 29
13. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. p. 303–312. SIGGRAPH '96, Association for Computing Machinery, New York, NY, USA (1996) 12, 18, 27, 29
14. Di Martino, J.M., Fernández, A., Ferrari, J.A.: 3d curvature analysis with a novel one-shot technique. In: 2014 IEEE International Conference on Image Processing (ICIP). pp. 3818–3822 (2014) 5
15. Farin, G.: Curves and Surfaces for CAGD: A Practical Guide. Computer graphics and geometric modeling, Elsevier Science (2002) 9
16. Genova, K., Cole, F., Sud, A., Sarna, A., Funkhouser, T.A.: Deep structured implicit functions. ArXiv **abs/1912.06126** (2019) 3
17. Gropp, A., Yariv, L., Haim, N., Atzmon, M., Lipman, Y.: Implicit geometric regularization for learning shapes. In: Proceedings of Machine Learning and Systems 2020, pp. 3569–3579 (2020) 2, 3, 6, 9, 11, 12, 13, 18, 24, 25, 26, 27, 28, 29

18. Handa, A., Whelan, T., McDonald, J., Davison, A.: A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In: IEEE Intl. Conf. on Robotics and Automation, ICRA. Hong Kong, China (May 2014) 18, 29
19. Hanocka, R., Metzer, G., Giryes, R., Cohen-Or, D.: Point2mesh: a self-prior for deformable meshes. ACM Transactions on Graphics **39**(4) (Aug 2020) 3
20. Hausdorff, F.: Set Theory. American Mathematical Soc. (2005) 10
21. Huang, J., Gojcic, Z., Atzmon, M., Litany, O., Fidler, S., Williams, F.: Neural kernel surface reconstruction (2023) 3
22. Kazhdan, M.M., Bolitho, M., Hoppe, H.: Poisson surface reconstruction. In: Sheffer, A., Polthier, K. (eds.) Proceedings of the Fourth Eurographics Symposium on Geometry Processing. SGP '06, vol. 256, pp. 61–70. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2006) 3, 11, 18, 27, 29
23. Kurita, T.: Computation of surface curvature from range images using geometrically intrinsic weights (09 1999) 5
24. Li, T., Wen, X., Liu, Y.S., Su, H., Han, Z.: Learning deep implicit functions for 3d shapes with dynamic code clouds (2022) 14
25. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. ACM siggraph computer graphics **21**(4), 163–169 (1987) 3, 9
26. Ma, B., Han, Z., Liu, Y., Zwicker, M.: Neural-pull: Learning signed distance functions from point clouds by learning to pull space onto surfaces. CoRR **abs/2011.13495** (2020) 5, 9, 13, 18, 25, 28, 29
27. Ma, B., Liu, Y.S., Han, Z.: Reconstructing surfaces for sparse point clouds with on-surface priors (2022) 14
28. Mescheder, L.M., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3d reconstruction in function space. CoRR **abs/1812.03828** (2018) 3
29. Michalkiewicz, M., Pontes, J.K., Jack, D., Baktashmotlagh, M., Eriksson, A.P.: Deep level sets: Implicit surface representations for 3d shape inference. CoRR **abs/1901.06802** (2019) 3
30. Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohi, P., Shotton, J., Hodges, S., Fitzgibbon, A.: Kinectfusion: Real-time dense surface mapping and tracking. In: 2011 10th IEEE international symposium on mixed and augmented reality. pp. 127–136. Ieee (2011) 4, 5, 10, 22
31. Novello, T., Schardong, G., Schirmer, L., da Silva, V., Lopes, H., Velho, L.: Exploring differential geometry in neural implicits (2022) 3, 4, 6, 7
32. Park, J.J., Florence, P., Straub, J., Newcombe, R.A., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. CoRR **abs/1901.05103** (2019) 3, 6
33. Peng, S., Jiang, C.M., Liao, Y., Niemeyer, M., Pollefeys, M., Geiger, A.: Shape as points: A differentiable poisson solver (2021) 3
34. Richa, J.P., Deschaud, J.E., Goulette, F., Dalmasso, N.: Unsigned distance field as an accurate 3d scene representation for neural scene completion (2022) 3
35. Sitzmann, V., Martel, J.N., Bergman, A.W., Lindell, D.B., Wetzstein, G.: Implicit neural representations with periodic activation functions. In: Proc. NeurIPS (2020) 3, 6, 9, 11, 12, 14, 18, 26, 27, 28, 29
36. Sommer, C., Sang, L., Schubert, D., Cremers, D.: Gradient-SDF: A semi-implicit surface representation for 3d reconstruction. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2022) 5, 6, 10, 18, 20, 21, 23, 24, 29

37. Spivak, M.: A Comprehensive Introduction to Differential Geometry. No. Bd. 1 in A Comprehensive Introduction to Differential Geometry, Publish or Perish, Incorporated (1999) 6
38. Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D.: A benchmark for the evaluation of rgb-d slam systems. In: Proc. of the International Conference on Intelligent Robot Systems (IROS) (Oct 2012) 18, 21, 29
39. Weder, S., Schönberger, J.L., Pollefeys, M., Oswald, M.R.: Routedfusion: Learning real-time depth map fusion. CoRR **abs/2001.04388** (2020) 4, 10, 22, 23, 24, 29
40. Yang, G., Belongie, S., Hariharan, B., Koltun, V.: Geometry processing with neural fields. In: Thirty-Fifth Conference on Neural Information Processing Systems (2021) 4, 7
41. Zollhöfer, M., Dai, A., Innmann, M., Wu, C., Stamminger, M., Theobalt, C., Nießner, M.: Shading-based refinement on volumetric signed distance functions. ACM Transactions on Graphics (TOG) **34**(4), 1–14 (2015) 13, 18, 29

A Appendix

A2 Code, Datasets and baseline methods

Our code and evaluation scripts will be publicly available upon acceptance. We will also provide the detailed information about the code of baseline methods.

	name	type	year	link	license
[12]	Redwood	dataset	2016	http://www.redwood-data.org/3dscan/	Public Domain
[13]	The Stanford 3D	dataset	1994	http://graphics.stanford.edu/data/3Dscanrep/	Public Domain
[41]	multi-view dataset	dataset	2015	http://graphics.stanford.edu/projects/vsfs/	CC BY-NC-SA 4.0
[18]	ICL-NUIM	dataset	2014	https://www.doc.ic.ac.uk/~ahanda/VaFRIC/iclnuim.html	CC BY 3.0
[38]	TUM-rgbd	dataset	2012	https://cvg.cit.tum.de/data/datasets/rgbd-dataset	CC BY 4.0
[36]	gradient-SDF	code	2022	https://github.com/c-sommer/gradient-sdf	BSD-3
[17]	IGR	code	2020	https://github.com/amosgropp/IGR	-
[35]	SIREN	code	2019	https://github.com/vsitzmann/siren	MIT license
[10]	IF-NET	code	2020	https://virtualhumans.mpi-inf.mpg.de/ifnets/	-
[26]	Neural-Pull	code	2021	https://github.com/bearprin/neuralpull-pytorch	-
[11]	NDF	code	2020	https://virtualhumans.mpi-inf.mpg.de/ndf/	-
[22]	Poisson	code	2006	http://www.open3d.org/	-
[5]	SSD	code	2011	http://mesh.brown.edu/ssd/software.html	-

Table A.3: Used datasets and code in our submission, together with reference, link, and license. We did our real-world experiments on two datasets, multi-view dataset [41] (for which ground truth poses exist), and Redwood [12] (without ground truth poses). Two synthetic dataset, the Stanford 3D [13], which is an object dataset, and ICL-NUIM dataset [18], which is a scene dataset. For the comparison methods, we use the code listed in the table.

This Supplement contains information on code, datasets, and more comparison results. The citation numbers are the same as in the main paper. **Our code and evaluation scripts will be publicly available upon acceptance.**

A3 Mathematical detail

A3.1 Math Notations

We summarize important math notation we used in the paper and appendix in Table Tab. A.4.

Symbol	Description	Symbol	Description
$\mathbf{x} \in \mathbb{R}^3$	3D points	$\mathcal{P} \subset \mathbb{R}^3$	point cloud set
$\mathcal{V} \subset \mathbb{N}^+$	points index set	$\mathcal{S} \subset \mathbb{R}^3$	continuous surface
$f(\mathbf{x}, \theta)$	neural implicit function	$\theta \in \mathbb{R}^{n \times m}$	learnable parameter
$w^p \in [0, 1]$	points uncertainty	$w^v \in [0, 1]$	voxel uncertainty
$\psi^v \in \mathbb{R}$	voxel SDF value	$\psi^p \in \mathbb{R}$	point SDF value
$\hat{\mathbf{g}}^v \in \mathbb{R}^3$	normalized distance gradient	$\mathbf{g}^v \in \mathbb{R}^3$	voxel distance gradient
∇	differential operator	$H^p \in \mathbb{R}$	point mean curvature
$\Gamma \subset \mathbb{R}^3$	sample domain	$\gamma \in \mathbb{R}$	SDF threshold
$K \in \mathbb{R}$	Gaussian curvature	$H \in \mathbb{R}$	mean curvature
k_1, k_2	principal curvature	$D \subset \mathbb{R}^2$	depth image
$\Omega \subset \mathbb{R}^2$	image domain	$\mathcal{M} \subset \mathbb{R}^3$	Monge path
$d_{\mathcal{S}}(\cdot)$	signed distance to surface $\mathcal{S}$	$\Gamma^+ \subset \mathbb{R}^3$	sample domain with positive uncertainty
$\bar{H} \in \mathbb{R}$	higher threshold	$\underline{\mathrm{H}} \in \mathbb{R}$	lower threshold of curvature
$\mathbf{n}_i \in \mathbb{R}^3$	known points normal	$\boldsymbol{Q} \in \mathbb{R}^{3 \times 3}$	camera intrinsic matrix
$\boldsymbol{R} \in SO(3)$	camera rotation matrix	$\mathbf{t} \in \mathbb{R}^3$	camera translation vector

Table A.4: Summary of our notation in the main paper and the supplementary material.

A3.2 Voxelization Details

Given an incoming depth $D(m, n), (m, n) \in \Omega$ with $z = D(m, n) \in \mathbb{R}$ and the estimated pose $\boldsymbol{R}, \mathbf{t}$, the 3D points in world coordinates are

$$\mathbf{x} = \boldsymbol{R}\boldsymbol{Q}^{-1} \begin{bmatrix} m \\ n \\ 1 \end{bmatrix} z \,, \tag{12}$$

$$\boldsymbol{Q} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \,, \tag{13}$$

where Q is the camera intrinsic matrix. Here we describe the standard voxel integration procedure. For each voxel $\mathbf{v}_i$, we project voxel center to the current depth image using $\mathbf{p} = (p_x, p_y, p_z) = \boldsymbol{R}^\top(\boldsymbol{Q}\mathbf{v}_i - \mathbf{t})$ and $(m, n, 1) = \mathbf{p}/p_z$ to find the corresponding pixel coordinates (m, n) on depth, then compare the $z - axis$ value, which stands for the distance of voxel $\mathbf{v}_i$ to the camera center. Then we

compute the difference of q_z with depth value on the corresponding pixel, then the project is to the normal direction to compute the point-to-plan distance (see Eq. (14)). We use the convention that inside the surface is positive and the outside surface is negative, such that the gradient of SDF has the same sign of surface normal. Then, the uncertainty of each voxel is computed using the distance $d_{\mathcal{S}}(\mathbf{v}_i)$. For visible voxel from current depth, which means $d_{\mathcal{S}}(\mathbf{v}_i)$ is negative, the uncertainty is set to 1, which means this update of distance is valid. To allow some noise, we set the depth integration threshold T (we use $T = 5$) as a truncate threshold. If $d_{\mathcal{S}}(\mathbf{v}_i)$ is positive and smaller Tv_s, then the uncertainty drops linearly to 0 (see Eq. (16)). Written in more simplified mathematical expression is, the SDF value of points $\mathbf{x}_j$ and its normal $\mathbf{n}_j$ and curvature H_j computed from the depth image D, then the voxel grid $\{\mathbf{v}_i\}$ SDF and uncertainty is computed by

$$d_{\mathcal{S}}(\mathbf{v}_i) = (\mathbf{x}_{j^*} - \mathbf{v}_i)^\top \hat{\mathbf{g}}_i \tag{14}$$

$$\nabla d_{\mathcal{S}}(\mathbf{v}_i) = \mathbf{g}_i = \boldsymbol{R}\mathbf{n}_{j^*} \tag{15}$$

$$w^v(\mathbf{v}_i) = \begin{cases} 1, & d_{\mathcal{S}}(\mathbf{v}_i) \leq 0 \\ 1 - \frac{d_{\mathcal{S}}(\mathbf{v}_i)}{v_s T}, & 0 < d_{\mathcal{S}}(\mathbf{v}_i) \leq v_s T \\ 0, & \text{else} \end{cases} \tag{16}$$

$$j^* = \arg\min_j \|\mathbf{x}_j - \mathbf{v}_i\| \tag{17}$$

where v_s is the voxel size, $T \in \mathbb{N}^+$ is the truncate voxel number, in this paper, voxel size is set to $0.8cm$ for 64^3 grid and $0.2cm$ for 256^3 grid with $T = 5$. Iterating over all depth image, the SDF ψ_i^v, uncertainty w_i^v, gradient $\mathbf{g}_i$ and curvature H_i^v is updated by

$$\psi_i^v \longleftarrow \frac{w_i^v \psi_i^v + w^v(\mathbf{v}_i) d_{\mathcal{S}}(\mathbf{v}_i)}{w_i^v + w^v(\mathbf{v}_i)} \tag{18}$$

$$\mathbf{g}_i \longleftarrow \frac{w_i^v \mathbf{g}_i^v + w^v(\mathbf{v}_i) \boldsymbol{R}\mathbf{g}_i^v}{w_i^v + w^v(\mathbf{v}_i)} \tag{19}$$

$$H_i^v \longleftarrow \frac{w_i^v K_i^v + w^v(\mathbf{v}_i) K_{j^*}}{w_i^v + w^v(\mathbf{v}_i)} \tag{20}$$

$$w_i^v \longleftarrow w_i^v + w^v(\mathbf{v}_i) \tag{21}$$

The attributes in voxels are integrated using weighted averages to be more robust to the noise, especially considering that the normal and curvature are computed using neighborhood information of one pixel on the depth map.

A3.3 Camera pose estimation

We can estimate the camera pose when we initialize the voxel grid using the same method as Gradient-SDF [36]. We use the first depth map with $\boldsymbol{R}_0 = \boldsymbol{I}$ and $\mathbf{t}_0 = \mathbf{0}$ to initial voxel grid as described before in Appendix A3.2, then the initial surface $\mathcal{S}$ in contained in voxels. Then from the second depth, each depth,

e.g. depth k is an incoming point cloud $\mathcal{P}^k = \{\mathbf{p}_j^k\}_j$ (outside index means iterate over j) after we project them using Eq. (12). The problem then is convert to find $\boldsymbol{R}_k$, $\mathbf{t}_k$, such that

$$(\boldsymbol{R}_k, \mathbf{t}_k) = \underset{\substack{\boldsymbol{R}\in SO(3),\\ \mathbf{t}\in\mathbb{R}^3}}{\arg\min} \sum_j w_j d_{\mathcal{S}}(\boldsymbol{R}_k \mathbf{p}_j^k + \mathbf{t}_k)\,, \tag{22}$$

where $SO(3)$ is 3-dimensional Lie group and $d_{\mathcal{S}}(\mathbf{p})$ denotes the signed distance from the point $\mathbf{p}$ to surface $\mathcal{S}$

$$|d_{\mathcal{S}}(\mathbf{p})| = \min_{\mathbf{p_s}\in\mathcal{S}} \|\mathbf{p} - \mathbf{p}_s\|\ . \tag{23}$$

Eq. (22) is solved by Gaussian-Newton and since the gradient $\nabla d_{\mathcal{S}}(\mathbf{p}) = \hat{\mathbf{g}}^v$ is pre-computed and stored, it also accelerates the optimization steps, as described in [36]. The camera pose is not given for Redwood [12] (sofa, kiosk, washmachine sequences) and TUM_RGBD [38] (household) datasets, the poses are estimated as described in this section.

A3.4 Proof of the Curvature Integration

In the paper section Sec. 3.1, we mention that the transformation between two depth coordinates has non-zero Jacobian (non-zero determinant); hence, integrating curvatures from depth images makes sense. Here is the formulation and proof.

The determinant of the Jacobian of the parameter transformation of the parameterization in the two depth images is non-zero; the mean curvature $H(x, y)$ and Gaussian curvature $K(x, y)$ are invariant.

Proof. Given two depth images D_1 and D_2 taken at two different positions. Suppose the transformation from position 1 to position 2 is a rigid body motion $\boldsymbol{T} = [\boldsymbol{R}, \mathbf{t}]$, where $\boldsymbol{R} \in SO(3)$ is a rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ is a translation vector. Let pixel $p = (m, n)$ in D_1, $0 \neq z = D_1(m, n)$, and $\boldsymbol{Q} \in \mathbb{R}^{3\times 3}$ be the camera intrinsic matrix, then the transformation of pixel p to $\bar{p} = (\bar{x}, \bar{y})$ in D_2 is

$$\bar{z}\begin{bmatrix}\bar{x}\\ \bar{y}\\ 1\end{bmatrix} = \boldsymbol{Q}\mathbf{x}\,, \tag{24}$$

where $\mathbf{x}$ is computed using Eq. (12) and $\boldsymbol{Q}$ is same as Eq. (13). is invertible, f_x, f_y is the camera focal length and c_x, c_y is the principal points. The pixel $3D$ coordinates in under two camera view is $\boldsymbol{Q}^{-1}(x, y, 1)^\top z$ and $\boldsymbol{Q}^{-1}(\bar{x}, \bar{y}, 1)^\top \bar{z}$. Let $\{r_{ij}\}_{ij}, i, j \in \{1, 2, 3\}$ be the element in $\boldsymbol{R}$ and $\mathbf{t} = (t_1, t_2, t_3)^\top$, compute the right side, we have $\bar{z} = r_{31}x + r_{32}y + r_{33}z + t_3$ is the depth value after a rigid body motion and $\bar{z} \neq 0$ since it does not fall to image plane of D_2 as we assume the point is visible in both camera position.

$$\det(\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^{-1}) = \det(\boldsymbol{Q})\det(\boldsymbol{R})\det(\boldsymbol{Q})^{-1} = 1\,, \tag{25}$$

it is because $\det(\boldsymbol{Q}) \neq 0$, $\det(\boldsymbol{R}) = 1$ and $\det(\boldsymbol{Q}^{-1}) = \det(\boldsymbol{Q})^{-1}$. Thus, the transformation Jacobian of the 3D points is non-zero. For the Jacobian of the transformation from (x, y) to $(\bar{x}, \bar{y})$, we only need to consider the upper-left 2×2 submatrix of $\boldsymbol{QRQ}^{-1}$. Since the upper-left 2×2 matrix of $\boldsymbol{R}$ represents the rotation matrix in the xy plane, thus it is non-zero. The upper-left 2×2 matrix of $\boldsymbol{Q}$ is diagonal also non-zero. Hence, we get the determinant of Jacobian from (x, y) to $(\bar{x}, \bar{y})$ is also non-zero.

A3.5 Main Curvature Vs. Gaussian Curvature

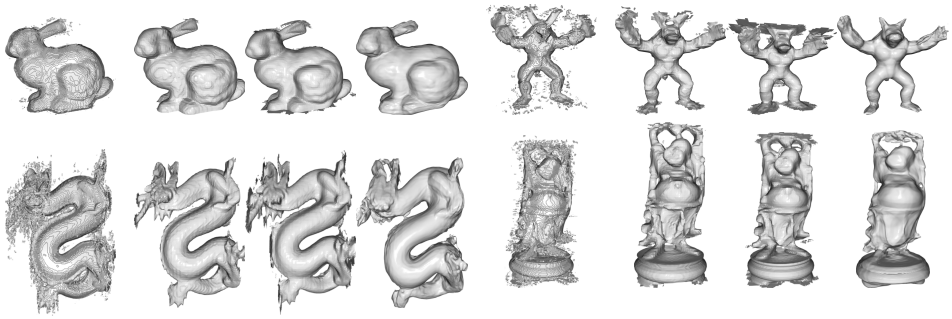
In this paper, we focus only on the main curvature and have not provided an ablation study to compare the influence of two curvatures. The reasons are the following. First, from the curvature visualization (Fig. 3a and Fig. 3b), the distribution of low, median, and high main curvature and Gaussian curvature are more or less similar. Since we only need the curvature as a guide information to sample points, by controlling the low, median, and high curvature threshold and proportion of each category, using main curvature and Gaussian curvature should not make much difference. Second, from a mathematical perspective, main curvature is the addition of two principal curvatures, *i.e.* $H = \frac{k_1+k_2}{2}$ and Gaussian curvature is $K = k_1 k_2$. We can have the following bound $K \leq H^2$. Thus, the difference between using the main curvature and the Gaussian curvature is negligible.

A4 Depth-based Method Comparison

We show more comparison results with the depth-based methods. All time evaluations are done on a computer with Intel Xeon(R) CPU @ 3.60GHz and 12 GeForce GTX TITAN X GPU. Fig. A.11 shows the visualization results of synthetic datasets and Fig. A.12 shows the results on real-world datasets.

RoutedFusion [39] provides networks trained for depth fusion across datasets. Ground truth or pre-generated mesh and camera poses are needed during training. For synthetic datasets, ground truth meshes and camera poses are given. For real-world datasets (*e.g.* [12]), pre-estimated camera poses using our pipeline and pre-generated mesh using TSDF [30] are given. The training time for two networks: Rout and Fusion needs around 70 hours for 360 (640×480) images from 4 synthetic datasets for each network. We tested on their pre-trained model and our trained model, pre-trained model offers better (less noisy) results. Thus we show the fusion results from their pre-trained model. During inference, ground truth poses and meshes are still needed for a guided fusion. The inference time for fusion is around 1.5 seconds per frame with CUDA. As shown in Fig. A.11 and Fig. A.12, RoutedFusion [39] tends to still perseve contour of voxels the same as IF-NET [10] (see Fig. 6 in main paper and Fig. A.16 in supplementary). Plus it creates noise around the meshes.

TSDF [30] we re-implemented TSDF tracker according to [4]. The method builds a dense voxel grid for tracking and depth fusion. The code is implemented purely



RoutedFusion TSDF Grad-SDF Ours RoutedFusion TSDF Grad-SDF Ours

Fig. A.11: The visualization of the reconstructed meshes on synthetic datasets. Compared with depth based methods: RoutedFusion [39], TSDF [4], Gradient-SDF (Grad-SDF) [36] and our method. TSDF uses 128^3 resolution voxel grid with $v_s = 0.02$ (m) voxel size, Grad-SDF uses $v_s = 0.02$ (m), ours is built on coarse voxel grid with 64^3 resolution voxel with $v_s = 0.04$ (m).

on CPU, and each voxel store a signed distance and a weight. We use a 128^3 resolution grid with $2cm$ voxel size, which need 16 Mb memory. For each depth, all voxels are project to image plane to compare with the depth value. Thus, the running time dependents on voxel resolution and depth image size. Fusion time is around $23ms$ per frame (640×480). TSDF has a limited reconstruction range, which equals resolution times the voxel size. Because of the truncation and missing data in depth images, it might create holes on the surface.

Gradient-SDF [36] uses a hash-map to store sparse voxels. For each voxel it stores a signed distance, a weight and a gradient of the signed distance. Different from traditional volumetric grid methods, instead of updating every pre-defined voxel, it updates according to each pixel in the incoming depth image, *i.e.*, for each incoming depth image, points are unprojected to 3D coordinates, it then computes which voxels need to be updated according to these points. Thus, the running time per frame depends on depth image size solely. However, it also means that for each depth, the number of updated voxels is at most equal to the pixel number of the depth image. The memory consumption of Gradient-SDF is dynamical, depends on each datasets and how many voxels are created during the fusion, (*e.g.*, for bunny the memory consumption is 4.4 Mb and for Armadillo is around 1.5 Mb.) The fusion time is around $30ms$ per frame (640×480). One possible

Our voxelization builds a dense voxel grid with resolution 64^4, stores a signed distance, a gradient of signed distance, an uncertainty, and curvature of each voxel. The memory consumption is 6 Mb. The fusion time is around $31ms$ per frame.

A5 Loss Ablation Study

Training details We use an 8-layer multi-layer perceptron (MLP) with ReLU activations. Each layer has 256 nodes, and the last layer has 2 output nodes

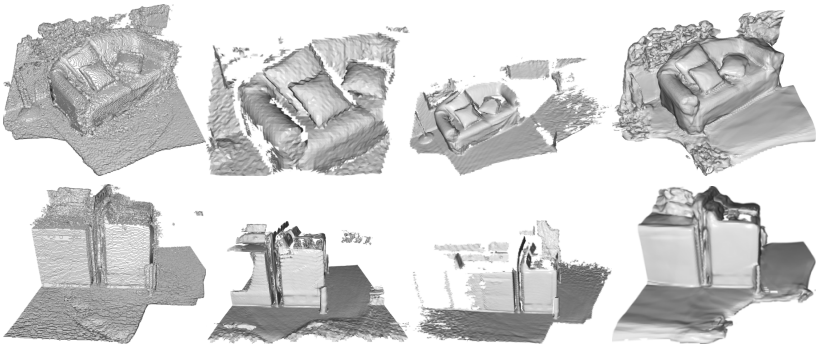


Fig. A.12: The Visualization of reconstructed meshes on real world datasets. For Gradient-SDF (Grad-SDF) [36], TSDF [4] and our voxelization, the camera poses are estimated during depth-fusion, RoutedFusion [39] takes estimated camera poses from our methods. The voxel setting is same as synthetic datasets.

for the SDF and uncertainty. We set the learning rate to 10^{-4} with decay. The batch size is $10k$, and we train for $10k$ epochs for each dataset. Our PyTorch implementation takes approximately 15 minutes to train on a GeForce GTX TITAN X GPU with CUDA for each dataset. The setting is the same for all the experiments for the proposed method.

Our loss function is composed of four distinct terms, and in this section, we detail how each contributes to the overall results. Table Tab. A.5 presents a quantitative assessment of the reconstructed meshes, indicating the impact of each loss term. The Eikonal term Eq. (9), ensures that our neural network functions as a signed distance field. The geometric loss term Eq. (6) is key in clarifying the orientation of surfaces. The uncertainty term Eq. (7) helps eliminate undesired areas. Furthermore, we investigate the effects of off-surface sampling in scenarios lacking normal information. Although our process incorporates both normal and curvature information—deriving the normal from depth—we still explore the role of normal information in processing sparse and dense inputs. This is to demonstrate that point cloud density and normal information jointly contribute to resolving surface orientation challenges, as shown in Tab. A.6, where we compare the error metrics of our method against IGR [17] with and without the geometric loss term. Interestingly, for certain datasets, omitting the geometric loss Eq. (6) or Eikonal loss Eq. (9) leads to reduced error rates. This suggests that at certain points, the model struggles to simultaneously satisfy both constraints, according to our analysis.

Even one goal of the proposed method is to solve the normal acquisition problem, especially on real-world datasets. In the ablation study, we still test the influence of off-surface sampling on the reconstruction under the no-normal information case. Fig. A.13 shows the visualization of two methods, IGR [17] and ours, with $\tau_n = 0$ in Eq. (10). Our method is robust and stable for sparse input even without normal input. IGR can reconstruct satisfactory meshes for dense

Metric	Dataset	**Method** (Sparse)					**Method** (Dense)				
		w/o $l_{\mathcal{E}}$	w/o $l_{\mathcal{W}}$	w/o $l_{\mathcal{N}}$	w/o $l_{\mathcal{N},\mathcal{W}}$	Ours	w/o $l_{\mathcal{E}}$	w/o $l_{\mathcal{W}}$	w/o $l_{\mathcal{N}}$	w/o $l_{\mathcal{N},\mathcal{W}}$	Ours
CD ($\times 10^2$)	Bunny	0.152	0.176	0.171	0.265	**0.067**	0.153	0.139	0.138	0.395	**0.068**
	Armadillo	0.089	0.052	**0.037**	1.180	**0.037**	0.104	0.041	0.052	1.987	**0.030**
	Dragon	0.387	0.182	0.226	0.251	**0.157**	0.138	0.158	0.184	0.196	**0.126**
	Buddha	0.378	0.289	0.467	0.184	**0.125**	0.248	0.301	0.253	0.295	0.267
HD	Bunny	0.058	0.044	0.039	0.074	**0.016**	**0.010**	0.139	0.012	0.039	0.043
	Armadillo	0.009	0.016	**0.006**	0.104	**0.006**	0.019	0.013	0.028	0.251	**0.007**
	Dragon	0.044	0.038	0.048	0.047	**0.030**	0.024	0.045	0.065	0.041	**0.025**
	Buddha	0.118	0.057	0.146	0.045	**0.017**	0.052	0.078	0.053	0.142	**0.044**

Table A.5: Ablation study for the individual loss term that we presented on Eq. (10). The error numbers indicate with full losses, the reconstructed meshes are best in most situation quantitatively.

Metric	Dataset	**Method** (Sparse)				**Method** (Dense)			
		w/o $l_{\mathcal{N}}$		full losses		w/o $l_{\mathcal{N}}$		full losses	
		IGR	Ours	IGR	Ours	IGR	Ours	IGR	Ours
CD ($\times 10^2$)	Bunny	0.489	0.171	0.481	0.067	0.169	0.139	0.501	0.068
	Armadillo	0.078	0.037	0.060	0.037	0.060	0.052	0.035	0.030
	Dragon	0.225	0.226	0.209	0.157	0.160	0.184	0.162	0.126
	Buddha	0.405	0.467	0.307	0.125	0.257	0.253	0.248	0.267
HD	Bunny	0.109	0.039	0.149	0.016	0.050	0.012	0.153	0.043
	Armadillo	0.032	0.006	0.018	0.006	0.015	0.028	0.014	0.007
	Dragon	0.050	0.048	0.037	0.030	0.046	0.065	0.056	0.025
	Buddha	0.087	0.146	0.092	0.017	0.097	0.053	0.059	0.044

Table A.6: Quantitative comparison with IGR [17] to show the influence of sprase and dense input under no normal information situation.

situations without normal but it fails when the input is sparse, especially when the surface is planar. Dense input can, therefore, resolve the ambiguity of the surface orientation to some extent.

A5.1 More Visualization Results

In this section, we show more visualization results and include the full uncertainty visualization for Fig. 10 in the main paper.

A6 Failing Cases Analysis

Chibane *et al.* show in [11] successful results in ShapeNet [7]. However, we did not get satisfactory results on both object and scene datasets. We suspect the method needs a lot of training data for one single shape, *e.g.*, ShapeNet has multiple point clouds for a single shape. We only have one (sparse) point cloud for a shape. We also see a similar issue reported in the authors' Github issues. Fig. A.18 shows different failing cases. The first line is the failing case on open surface reconstruction for sparse input ($\sim 6k$) *lr_kt0* datasets. IF-NET [10] can not handle open surfaces. NeuralPull [26] easily fails when there is a flat plane.

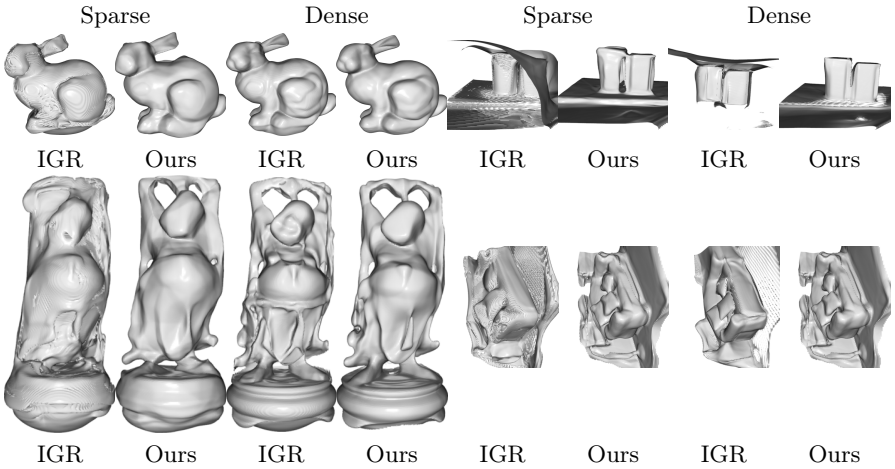


Fig. A.13: No normal information reconstruction results on IGR [17] and our method. Our method outperforms IGR on all sparse input situations.

Fig. A.14: The visualization of Gaussian curvatures and mean curvatures of *happy_buddha* and *dragon* dataset, computed using depth images as described in main paper Sec. 3.1.

One reason is NeuralPull learns to pull the sampled point to the closest surface point. If the sampled point is already on the surface, it learns to pull back and forth on the same plane. SIREN [35] tends to create artifices in non-surface areas when dealing with open surfaces as it use periodic activation functions after each layer, it creates random area when initialize the neural network. The second row is failing cases for complicated shape *Dragon* (sparse input, $\sim 6k$ points). We also fail to recover satisfactory results using our method (last column), and the modified NeuralPull method with our sampling strategy also fails to recover the correct shape of the Dragon (last second column). Our analysis is that large

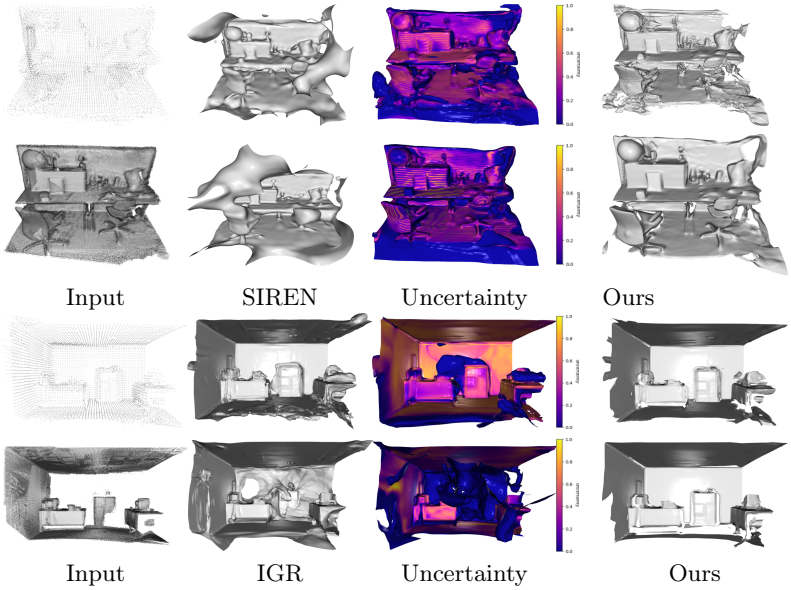


Fig. A.15: Scene reconstruction results with sparse input on real-world dataset *TUM_ rgbd* (first rows, sparse points $\sim 14k$, dense points $\sim 330k$) with noisy camera poses. Synthetic dataset *icl_ nium* (last rows, sparse points $\sim 14k$, and dense points $\sim 215k$) with ground truth camera poses.

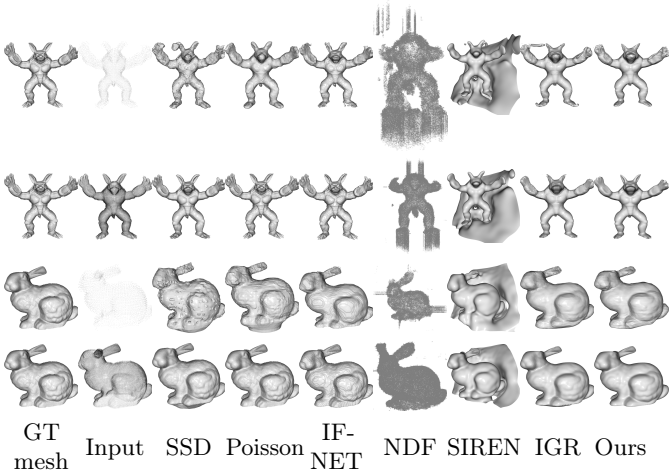


Fig. A.16: Comparison results with SSD [5], Poisson surface recontruction [22], NDF [11], IGR [17], SIREN [35] and IF-NET [10] with two different density input on synthetic datasets [13]. *Armadillo* has $\sim 8k$ in sparse input and $\sim 146k$ in dense input. *Bunny* has $\sim 5k$ sparse points and dense one with $\sim 100k$ points

details are gathered around the head part of the Dragon, and the interpolation fails to overcome too sparse input.

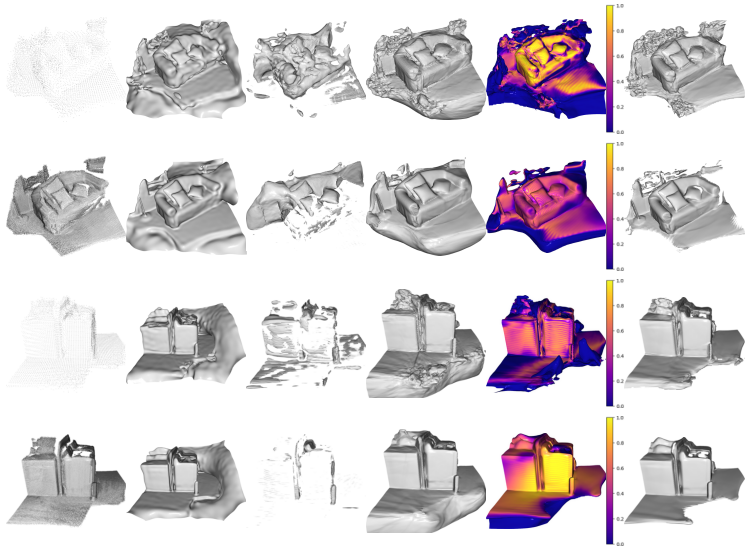


Fig. A.17: Two **Real world** Scene datasets [12] results comparison with SIREN [35], NeuralPull [26] and IGR [17]. NeuralPull fails in most of the cases, while SIREN creates a redundant area. *Sofa* has $\sim 6k$ points and $\sim 111k$ points in sparse and dense situation, respectively. *Washmachine* with $\sim 8k$ sparse points and $\sim 180k$ dense points.

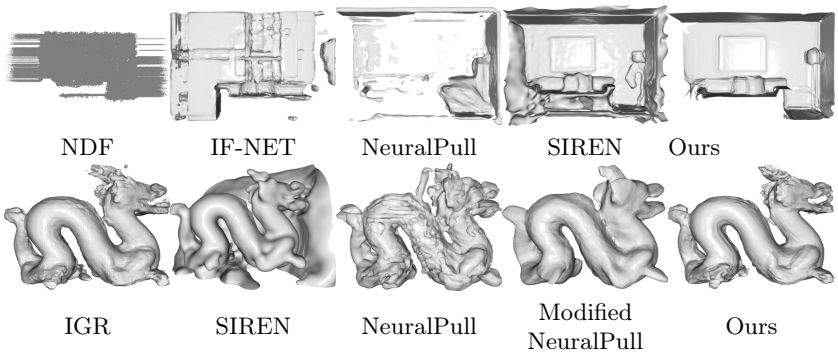


Fig. A.18: Failing cases in different methods.

A7 Code, Datasets, and Baseline Methods

The following table contains detailed information (link, license) of used code and datasets in the main paper.

name	type	year	link	license
[12]Redwood dataset		2016	http://www.redwood-data.org/3dscan/	Public Domain

[13]	The Stanford 3D	dataset	1994	http://graphics.stanford.edu/data/3Dscanrep/	Public Domain
[41]	multi-view dataset	dataset	2015	http://graphics.stanford.edu/projects/vsfs/	CC BY-NC-SA 4.0
[18]	ICL-NUIM	dataset	2014	https://www.doc.ic.ac.uk/~ahanda/VaFRIC/iclnuim.html	CC BY 3.0
[38]	TUM-rgbd	dataset	2012	https://cvg.cit.tum.de/data/datasets/rgbd-dataset	CC BY 4.0
[36]	gradient-SDF	code	2022	https://github.com/c-sommer/gradient-sdf	BSD-3
[17]	IGR	code	2020	https://github.com/amosgropp/IGR	-
[39]	Routed-Fusion	code	2020	https://github.com/weders/RoutedFusion	-
[35]	SIREN	code	2019	https://github.com/vsitzmann/siren	MIT license
[10]	IF-NET	code	2020	https://virtualhumans.mpi-inf.mpg.de/ifnets/	-
[26]	NeuralPull	code	2021	https://github.com/bearprin/neuralpull-pytorch	-
[11]	NDF	code	2020	https://virtualhumans.mpi-inf.mpg.de/ndf/	-
[22]	Poisson	code	2006	http://www.open3d.org/	-
[5]	SSD	code	2011	http://mesh.brown.edu/ssd/software.html	-

Table A.7: Used datasets and code in our submission, with reference, link, and license. We did our real-world experiments on two datasets, multi-view dataset [41] (for which ground truth poses exist), and Redwood [12] (without ground truth poses). There are two synthetic datasets: the Stanford 3D [13], an object dataset, and the ICL-NUIM dataset [18], a scene dataset. For the comparison methods, we use the code listed in the table.