Surface Snapping Optimization Layer for Single Image Object Shape Reconstruction

Yuan-Ting Hu [1] **Alexander G. Schwing** [1] **Raymond A. Yeh** [2]

Abstract

Reconstructing the 3D shape of objects observed in a single image is a challenging task. Recent approaches rely on visual cues extracted from a given image learned from a deep net. In this work, we leverage recent advances in monocular scene understanding to incorporate an additional geometric cue of surface normals. For this, we proposed a novel optimization layer that encourages the face normals of the reconstructed shape to be aligned with estimated surface normals. We develop a computationally efficient conjugate-gradient-based method that avoids the computation of high-dimensional sparse matrices. We show this framework to achieve compelling shape reconstruction results on the challenging Pix3D and ShapeNet datasets.

1. Introduction

Humans can reason about the 3D geometry of objects even from a monocular image. This capability enables us to efficiently and effortlessly interact with our environment. Developing a system that has this capability is hence an important problem for applications across many fields, *e.g.*, robotics, the entertainment industry, and augmented and virtual reality.

Classical works (Horn, 1975; Nayar et al., 1991; Wolff et al., 1993; Pentland, 1987) on shape reconstruction rely on visual cues such as texture, shading, or camera focus. More recent methods aim to learn the visual cues from data via deep nets. These works study different shape representations and propose corresponding deep net architectures, *e.g.*, meshes (Wang et al., 2018), occupancy grids (Wu et al., 2015), octrees (Tatarchenko et al., 2017), implicit fields (Chen & Zhang, 2019; Mescheder et al., 2019; Park et al., 2019) or point clouds (Groueix et al., 2018). The main visual cue of these works is a deep feature extracted from an image via a convolutional network. Notably, these works often do not leverage explicit 2.5D sketches, such as depth maps or surface normals. Use of these sketches has been advocated by Marr (1982), and has previously been found to be useful by Bansal et al. (2016) and Wu et al. (2017).

In this work, we study *how* shape reconstruction methods can efficiently benefit from an explicit geometric cue of surface normals. We choose surface normals as its estimation has made tremendous progress in recent years (Ladický et al., 2014; Eigen & Fergus, 2015; Wang et al., 2015; 2016; Liao et al., 2019; Zhang et al., 2019; Hickson et al., 2019; Qi et al., 2020; Wang et al., 2020; Do et al., 2020; Zamir et al., 2020; Bae et al., 2021; Yu et al., 2022).

To incorporate surface normal estimates, we propose a novel optimization layer: an optimization problem viewed as a differentiable function that maps from its input to its solution (Amos & Kolter, 2017; Gould et al., 2022). The proposed optimization layer minimizes a cost function that explicitly encourages the normals of the reconstructed shape to match the surface normals predicted from the image.

We minimize each optimization layer objective to optimality during a forward pass through the deep net, which repositions the vertices of the reconstructed shape. Consequently, the reconstructed shape vertices "snap" to the observed geometry, hence the name **surface snapping**. The effect of surface snapping is illustrated in Fig. 1, where we observe a more accurately recovered shape.

To compute the exact solution of the optimization layer, we develop an efficient conjugate-gradient-based method that considers the sparsity structure of the problem to improve both the memory and computational efficiency. Using standard solvers without considering the sparsity structure is intractable with present-day hardware. The layer is interpretable and exposes a trainable parameter that controls the strength of the snapping. Finally, the developed surface snapping layer can be inserted into any deep net operating on meshes. This layer can be trained via gradient-based methods along with the standard layers.

[1]Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign [2]Department of Computer Science, Purdue University. Correspondence to: Yuan-Ting Hu <ythu2@illinois.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

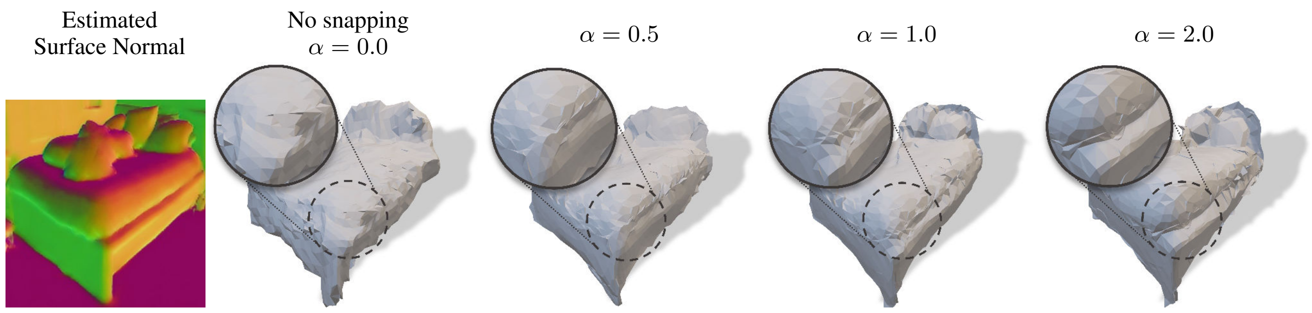

Figure 1. Effect of surface snapping. Larger values of α result in stronger snapping.

In experiments, we evaluate the proposed surface snapping technique on the challenging Pix3D (Sun et al., 2018) and ShapeNet (Chang et al., 2015) datasets. On both datasets, we observe that surface snapping, when incorporated into recent baselines, improves the reconstruction; especially in terms of the normal consistency metric. Qualitatively, we observe that the reconstructed geometry is less noisy, *i.e.*, the surface is smoother. This demonstrates the effectiveness of the proposed surface snapping layer at adjusting face normals for monocular shape reconstruction.

Our contributions:

- We propose a novel optimization layer for object shape reconstruction and a specialized solver to efficiently compute its solution.
- We demonstrate that the proposed layer leverages surface normal estimation to improve object shape reconstruction.

2. Related Work

Optimization as a layer. An optimization problem can be considered as a single “layer” in a deep net architecture because it can be viewed as a function mapping from its input to its exact solution. For such a layer, the derivative is computed through implicit differentiation. Amos & Kolter (2017) study deep net architectures that integrate optimization problems in the form of a quadratic program (OptNet). This formulation is further extended to disciplined convex programs by Agrawal et al. (2019). Recently, applications of optimization layers have emerged in reinforcement learning (Amos et al., 2018), logical reasoning (Wang et al., 2019), and image classification, segmentation and various computer vision tasks (Bai et al., 2019; 2020; Huang et al., 2021; Bai et al., 2022; Yeh et al., 2022; Pokle et al., 2022).

Our surface snapping formulation is also a quadratic program. However, the generic solver implementations, *e.g.*, OptNet, are not scalable to our problem size due to the large number of vertices involved in a shape. We hence develop a solver which benefits from the structure of the task.

Monocular 3D shape reconstruction. Shape reconstruction from a single image has received a considerable amount of attention (Dai et al., 2017; Izadinia et al., 2017; Wu et al., 2017; Zou et al., 2017; Wang et al., 2018; Tulsiani et al., 2018; Groueix et al., 2018; Kundu et al., 2018; Mahmud et al., 2020; Mescheder et al., 2019; Mo et al., 2019; Deng et al., 2020; Chen et al., 2020; Peng et al., 2020; Nash et al., 2020; Park et al., 2019; Paschalidou et al., 2020; Wu et al., 2020; Duggal & Pathak, 2022). These methods’ shape representations, which are often the focus of the work, generally differ. Among the most popular shape representations are meshes (Wang et al., 2018; Gao et al., 2022), occupancy grids (Wu et al., 2015), octrees (Tatarchenko et al., 2017), implicit fields (Chen & Zhang, 2019; Mescheder et al., 2019; Park et al., 2019) or point clouds (Groueix et al., 2018). Many of these methods assume that the image only depicts a single object, *i.e.*, only a single shape needs to be reconstructed per image. Consequently, these methods usually focus on datasets like ShapeNet (Chang et al., 2015), where a single object is illustrated per image.

3D shape reconstruction methods (Izadinia et al., 2017; Tulsiani et al., 2018; Kundu et al., 2018; Gkioxari et al., 2019; Nie et al., 2020; Zhang et al., 2021) which reconstruct multiple objects in a given image usually either leverage a detection network (Izadinia et al., 2017; Kundu et al., 2018; Gkioxari et al., 2019) like Faster/Mask-RCNN (Ren et al., 2015; He et al., 2017) or assume instance-level bounding boxes are readily available (Tulsiani et al., 2018). Using given or detected bounding boxes, the 3D shape of each instance is inferred separately for each bounding box. For this, appearance information in the form of a spatial high-dimensional feature vector is obtained for each bounding box from the object detector or a feature pyramid network.

Different from these works, we study the use of surface normals as an additional geometric cue. While there are prior deep learning approaches for 3D reconstruction that utilize surface normals, they are either not directly applicable to meshes (Qi et al., 2020) or simply introduce a normal loss for end-to-end training (Wu et al., 2017; Wang et al., 2018). In contrast, the proposed surface snapping layer operates di-

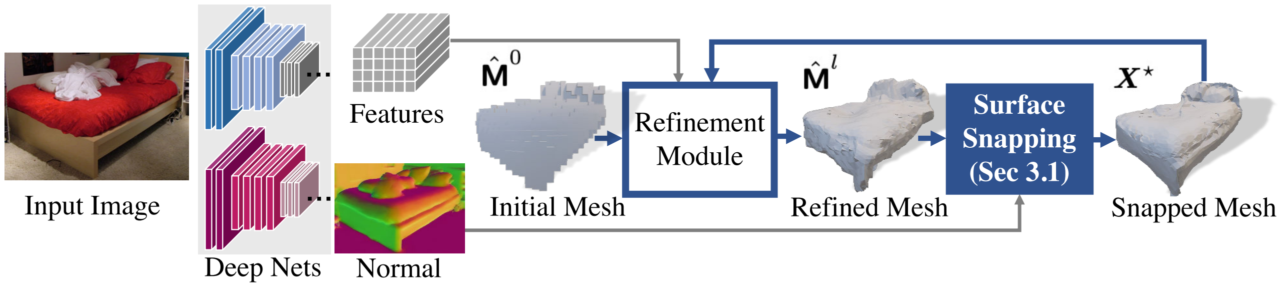

Figure 2. Object shape reconstruction with surface snapping and an explicit model. With surface snapping, the snapped mesh exhibits sharper edges and a much smoother surface. *E.g.*, the crisp edges near the bed frame or the outline of the pillows.

rectly on a mesh representation, finds the exact solution, and incorporates the surfaces normal directly into the inference procedure (forward-pass).

Monocular normal estimation. Fouhey et al. (2013) introduce a data-driven approach that leverages 3D geometric primitives and high-level constraints. Ladický et al. (2014) additionally incorporate pixel-based and segment-based cues. The first deep net for this task was proposed by Wang et al. (2015), and multi-scale network architectures have been studied by Eigen & Fergus (2015). These were later refined to incorporate skip connections (Bansal et al., 2016) as well as a dense conditional random field to refine the deep net output (Wang et al., 2016). Increasingly accurate results have been reported and methods have also been scaled to efficiently run on edge devices (Hickson et al., 2019).

To incorporate surface normals into shape reconstruction we formulate a novel optimization layer that optimizes a cost function to ensure that the reconstructed surface "snaps" to the estimated normals.

3. Approach

Our goal is to leverage surface normal estimates for the task of 3D shape reconstruction. For this, we propose surface snapping, an optimization layer that explicitly aligns the predicted shape to be consistent with the predicted surface normal. In §3.1, we introduce the surface snapping layer by defining its forward and backward operation. We incorporate surface snapping layers into shape reconstruction methods, including both explicit and implicit models (§3.2). We discuss how to train these models in §3.3.

3.1. Surface snapping Optimization Layer

An object's shape can be characterized by a triangular mesh $\mathbf{M} = (\boldsymbol{V}, \boldsymbol{F})$, which is specified by a tuple $\boldsymbol{V} \in \mathbb{R}^{N_V \times 3}$ of N_V 3-dimensional vertices, and a tuple $\boldsymbol{F} \in \mathbb{Z}_+^{N_F \times 3}$ of N_F faces. Here, each face is represented by three vertex indices.

Given a predicted shape $\hat{\mathbf{M}}^l = (\hat{\boldsymbol{V}}^l, \hat{\boldsymbol{F}})$ at the l^{th} layer and a face normal estimate $\hat{\boldsymbol{N}}$ obtained from the image, the proposed surface snapping aims to refine the vertices into a shape $\hat{\mathbf{M}}^{l+1} = (\hat{\boldsymbol{V}}^{l+1}, \hat{\boldsymbol{F}})$ that is more consistent with the surface normal $\hat{\boldsymbol{N}}$. *I.e.*, we view surface snapping as a layer

$$\hat{\mathbf{M}}^{l+1} = \texttt{SurfaceSnapping}(\hat{\mathbf{M}}^l, \hat{\boldsymbol{N}}), \tag{1}$$

which takes as input a shape $\hat{\mathbf{M}}^l$ and a normal map $\hat{\boldsymbol{N}}$, and yields another shape $\hat{\mathbf{M}}^{l+1}$ as its output.

Forward operation definition. To update the vertices $\hat{\boldsymbol{V}}^l$, we formulate `SurfaceSnapping` as an optimization layer. The goal of this optimization problem is to align ("snap") the predicted vertices to face normal estimates $\hat{\boldsymbol{N}}$. Specifically, `SurfaceSnapping` minimizes the weighted sum of two cost functions, a vertex cost $\mathcal{C}_{\text{V}}$ and a normal cost $\mathcal{C}_{\text{N}}$, *i.e.*,

$$\hat{\boldsymbol{V}}^{l+1} \triangleq \boldsymbol{X}^\star = \arg\min_{\boldsymbol{X}} \left(\mathcal{C}_{\text{V}}(\boldsymbol{X}, \hat{\boldsymbol{V}}^l) + \alpha \mathcal{C}_{\text{N}}(\boldsymbol{X}, \hat{\boldsymbol{N}}) \right), \tag{2}$$

where $\alpha \in \mathbb{R}_+$ is a positive trainable parameter and $\boldsymbol{X} \in \mathbb{R}^{N_V \times 3}$ denotes the optimization variable in Eq. (2).

The vertex cost $\mathcal{C}_{\text{V}}$ encourages similarity between optimization variable $\boldsymbol{X}$ and the current vertices $\hat{\boldsymbol{V}}^l$ by penalizing distances via an ℓ_2-norm:

$$\mathcal{C}_{\text{V}}(\boldsymbol{X}, \hat{\boldsymbol{V}}^l) = \|\boldsymbol{X} - \hat{\boldsymbol{V}}^l\|_2^2. \tag{3}$$

The normal cost $\mathcal{C}_{\text{N}}$ controls the consistency of the mesh prediction and the estimated normals via

$$\mathcal{C}_{\text{N}}(\boldsymbol{X}, \hat{\boldsymbol{N}}) = \sum_{i=1}^{N_F} \sum_{j,k \in \hat{\boldsymbol{F}}_i} \langle \hat{\boldsymbol{N}}_i, \boldsymbol{X}_j - \boldsymbol{X}_k \rangle^2, \tag{4}$$

where N_F denotes the number of faces. This cost accumulates the inner products between adjacent edges $\boldsymbol{X}_j - \boldsymbol{X}_k$ of face i (defined by its vertex indices $\hat{\boldsymbol{F}}_i$) and the face normal $\hat{\boldsymbol{N}}_i$. Intuitively, the estimated face normal and the edges should be orthogonal to each other. Hence we penalize the squared norm of the inner product. As face normals are only estimated for visible faces, we set the normal of non-visible faces to a zero vector.

Note, we use the term *cost function* $\mathcal{C}$ to refer to the optimization problem that *defines the forward operation* of the surface snapping layer. A cost function $\mathcal{C}$ is *not* the loss function for training the model. We will discuss training losses in §3.3.

We illustrate the behavior of this layer in Fig. 1. When α increases, we observe the edge of the bed more clearly as a larger α more strongly encourages snapping the face normals of the predicted shape to the estimated normals. In summary, the program in Eq. (2) considers vertex and normal cost functions which balance between the normal predictions and the current vertex location estimates. We will next discuss the layers' forward pass and backward pass computation.

Forward operation computation. A forward pass requires solving the weighted objective in Eq. (2). Observe that it can be formulated as a linear least-squares problem of the form $\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$, as

$$\min_{\boldsymbol{x}} \left(\|\boldsymbol{x} - \hat{\boldsymbol{v}}^l\|_2^2 + \alpha \|\mathbf{N} \odot \mathbf{D}\boldsymbol{x}\|_2^2 \right) \tag{5}$$

$$= \min_{\boldsymbol{x}} \left\| \underbrace{\begin{bmatrix} \boldsymbol{I} \\ \sqrt{\alpha}\mathbf{N} \odot \mathbf{D} \end{bmatrix}}_{\boldsymbol{A}} \boldsymbol{x} - \underbrace{\begin{bmatrix} \hat{\boldsymbol{v}}^l \\ \sqrt{\alpha}\boldsymbol{0} \end{bmatrix}}_{\boldsymbol{b}} \right\|_2^2 . \tag{6}$$

Here, $\boldsymbol{x} = \texttt{vec}(\boldsymbol{X})$ and $\hat{\boldsymbol{v}}^l = \texttt{vec}(\hat{\boldsymbol{V}}^l) \in \mathbb{R}^{3 \cdot N_V}$ refer to the flattened vertex coordinates in column-major order, *i.e.*, $\texttt{vec}(\hat{\boldsymbol{V}}^l) = [\hat{\boldsymbol{V}}^l_{\texttt{col(1)}}; \hat{\boldsymbol{V}}^l_{\texttt{col(2)}}; \hat{\boldsymbol{V}}^l_{\texttt{col(3)}}]$. Moreover, $(\mathbf{N} \odot \mathbf{D})\boldsymbol{x}$ computes the inner product between a face normal and the edges of each face in the mesh. For this we construct the edges by using matrix $\mathbf{D} \in \mathbb{R}^{9N_F \times 3N_V}$ which is defined as

$$\mathbf{D} = \begin{bmatrix} \boldsymbol{D} & 0 & 0 \\ 0 & \boldsymbol{D} & 0 \\ 0 & 0 & \boldsymbol{D} \end{bmatrix}, \ \boldsymbol{D}_{i,j} = 1 \text{ and } \boldsymbol{D}_{i,k} = -1 \tag{7}$$

$\forall j < k \in \hat{\boldsymbol{F}}_{\lfloor i/3 \rfloor}$, and 0 otherwise. *I.e.*, each row of $\boldsymbol{D}$ forms an edge $\boldsymbol{X}_j - \boldsymbol{X}_k$ between vertex j and k. Next, the face normals are stored in matrix $\mathbf{N} \in \mathbb{R}^{9N_F \times 9N_F}$ as follows:

$$\mathbf{N} = \begin{bmatrix} \mathbf{N}^1 & 0 & 0 \\ 0 & \mathbf{N}^2 & 0 \\ 0 & 0 & \mathbf{N}^3 \end{bmatrix}, \ \mathbf{N}^h_{i,i} = \begin{cases} \hat{\boldsymbol{N}}_{i,h}, & i \in \{1, \dots, N_F\} \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where $\hat{\boldsymbol{N}}_{i,h}$ is the normal of face i for dimension h.

In theory, a forward pass through the surface snapping layer can be solved analytically via $(\boldsymbol{A}^\intercal\boldsymbol{A})^{-1}\boldsymbol{A}^\intercal\boldsymbol{b}$ as formulated in Eq. (6). However, this is inefficient and memory intensive due to matrix inversion of a large matrix $(\boldsymbol{A}^\intercal\boldsymbol{A})^{-1}$. Hence, we develop a conjugate gradient method leveraging the system's sparsity structure, which we discuss next.

Efficient conjugate gradient solver. The least-squares solution is equivalent to solving a linear system of the form $\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{t}$, where

$$\boldsymbol{Q} = \boldsymbol{A}^\intercal\boldsymbol{A} \quad \text{and} \quad \boldsymbol{t} = \boldsymbol{A}^\intercal\boldsymbol{b}. \tag{9}$$

As $\boldsymbol{Q}$ is positive-definite, the conjugate gradient method is applicable. We refer to a conjugate gradient solver via $\texttt{CGSolve}(\boldsymbol{Q}, \boldsymbol{t})$.

Importantly, a standard conjugate gradient solver does not work out of the box, due to memory and computation constraints. Instead, we develop a custom conjugate gradient solver with GPU support which leverages the structure of $\boldsymbol{A}$. We exploit the sparsity patterns in the system of equations, observed in Eq. (7) and Eq. (8), to avoid redundancies. Please see the Appendix for more details. Here, we spotlight some aspects which led to significant speedups:
① *Sparse element-wise multiplication:* When computing $\mathbf{N} \odot \mathbf{D}$ in Eq. (6), use of dense matrices is inefficient. Instead, we use sparse element-wise multiplications on $\boldsymbol{D}$ and $\mathbf{N}^1$, $\mathbf{N}^2$, $\mathbf{N}^3$ to avoid constructing the full matrices $\mathbf{D}$ and $\mathbf{N}$. This removes many unnecessary multiplications with zeros.

② *Advanced indexing to create edges:* When computing $\mathbf{D}\boldsymbol{x}$ in Eq. (6), we avoid this large matrix multiplication, by using advanced indexing to select the vertices and taking their differences with element-wise vector operations.

Backward operation. To enable end-to-end training, we need to back-propagate through the solution $\boldsymbol{x}^\star$ obtained by solving the program given in Eq. (6). Concretely, for a given loss function $\mathcal{L}$ which depends on the solution $\boldsymbol{x}^\star$, we need to compute

$$\frac{\partial \mathcal{L}(\boldsymbol{x}^\star)}{\partial \hat{\boldsymbol{v}}^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^\star} \cdot \frac{\partial \boldsymbol{x}^\star}{\partial \boldsymbol{t}} \cdot \frac{\partial \boldsymbol{t}}{\partial \hat{\boldsymbol{v}}^l}. \tag{10}$$

We can view the solution $\boldsymbol{x}^\star$ as a functional mapping of $\hat{\boldsymbol{v}}^l$, *i.e.*, $\boldsymbol{x}^\star(\hat{\boldsymbol{v}}^l)$, in which case the gradient can be traced through the optimization procedure. To see this, recall, $\boldsymbol{t}$ is a linear combination of $\hat{\boldsymbol{v}}^l$, as defined in Eq. (5) and Eq. (9). Automatic differentiation can handle $\frac{\partial \boldsymbol{t}}{\partial \hat{\boldsymbol{v}}^l}$. Hence, we only need to compute, $\frac{\partial \mathcal{L}(\boldsymbol{x}^\star)}{\partial \boldsymbol{t}}$, the gradient of a loss function w.r.t. $\boldsymbol{t}$.

Recall, $\boldsymbol{x}^\star$ is the solution of the linear system $\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{t}$, *i.e.*, $\boldsymbol{x}^\star = \boldsymbol{Q}^{-1}\boldsymbol{t}$. From the chain rule, we obtain

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^\star} = \boldsymbol{Q}^\intercal \frac{\partial \mathcal{L}}{\partial \boldsymbol{t}}. \tag{11}$$

This is again a linear system where we can use the developed $\texttt{CGSolver}$ which exploits sparsity. Hence, we compute the

Table 1. Quantitative comparison explicit methods on Pix3D dataset split S_1 using the Setup ❶ by Gkioxari et al. (2019). Green indicates that adding the surface snapping layer improves the performance.

Method / %	*chair*	*sofa*	*table*	*bed*	*desk*	*bkcs*	*wrdrb*	*tool*	*misc*	AP$^{\text{box}}$	AP$^{\text{mask}}$	AP$^{\text{mesh}}$	Normal	Normal$^{\text{Vis}}$
Pixel2Mesh^{+} (Wang et al., 2018)	30.9	59.1	40.2	40.5	30.2	50.8	62.4	18.2	26.7	93.5	**88.4**	39.9	18.0	39.8
Sphere-Init	40.9	75.2	44.2	50.3	28.4	48.6	42.5	26.9	7.0	**94.1**	87.5	40.5	15.3	39.0
Mesh R-CNN (Gkioxari et al., 2019)	48.2	71.7	60.9	53.7	42.9	70.2	63.4	21.6	27.8	94.0	**88.4**	51.2	21.6	46.5
GCN Transformer	**49.9**	74.3	**67.3**	50.5	42.8	**75.4**	68.9	**37.4**	33.3	**94.1**	88.3	55.5	23.3	49.6
`Snap+` Pixel2Mesh^{+}	32.3	61.8	43.9	42.8	33.5	46.0	74.3	4.5	26.7	93.5	**88.4**	40.7	19.4	43.3
`Snap+` Sphere-Init	35.1	61.7	44.1	40.0	31.3	55.7	46.1	23.6	13.5	**94.1**	87.6	39.0	17.1	39.0
`Snap+` Mesh R-CNN	49.0	74.8	65.3	55.7	**46.1**	73.8	**70.9**	21.6	27.8	**94.1**	88.3	54.1	23.0	48.8
`Snap+` GCN Transformer	49.5	**76.5**	64.3	**56.0**	44.3	73.8	**70.9**	31.8	**33.4**	**94.1**	88.3	**55.6**	**23.8**	**50.4**

Table 2. Quantitative comparison of explicit methods on Pix3D dataset split S_2 using the **Setup ❶** by Gkioxari et al. (2019). Green indicates that adding the surface snapping layer improves the performance.

Method / %	*chair*	*sofa*	*table*	*bed*	*desk*	*bkcs*	*wrdrb*	*tool*	*misc*	AP$^{\text{box}}$	AP$^{\text{mask}}$	AP$^{\text{mesh}}$	Normal	Normal$^{\text{Vis}}$
Pixel2Mesh^{+} (Wang et al., 2018)	26.7	58.5	10.9	38.5	7.8	34.1	**3.4**	10.0	0.0	71.1	63.4	21.1	19.5	42.2
Sphere-Init	32.9	**75.3**	15.8	40.1	10.1	45.0	1.5	0.8	0.0	**72.6**	**64.5**	24.6	15.7	40.0
Mesh R-CNN (Gkioxari et al., 2019)	**42.7**	70.8	27.2	40.9	18.2	**51.1**	2.9	5.2	0.0	72.2	63.9	28.8	21.4	46.5
GCN Transformer	42.9	68.8	26.5	40.7	22.9	44.6	1.2	0.5	0.0	72.4	64.0	27.5	22.1	48.8
`Snap+` Pixel2Mesh^{+}	26.9	60.9	10.0	38.7	10.0	25.3	4.2	**10.1**	0.0	71.1	63.4	20.7	20.5	44.7
`Snap+` Sphere-Init	31.2	72.7	11.7	42.1	7.8	38.2	1.0	1.2	0.0	**72.6**	**64.5**	22.9	16.0	40.2
`Snap+` Mesh R-CNN	41.4	74.7	**28.1**	**42.6**	**20.1**	50.4	2.9	3.7	0.0	72.4	64.0	**29.9**	23.0	**49.7**
`Snap+` GCN Transformer	42.2	75.3	28.6	41.3	21.1	50.0	2.8	6.1	0.0	72.4	64.0	29.7	**23.3**	**49.7**

gradient efficiently via

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{t}} = \texttt{CGSolve}\left(\boldsymbol{Q}^{\intercal}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{\star}}\right). \tag{12}$$

We will next explain how to incorporate our surface snapping layer into deep nets for shape reconstruction.

3.2. Surface Snapping for Shape Reconstruction

To apply our surface snapping layer, we consider two main shape reconstruction paradigms. We also discuss how to obtain the surface normal estimation from an image.

Surface snapping with explicit models. Surface snapping can be applied to models which use an explicit mesh representation (Wang et al., 2018; Gkioxari et al., 2019). These models start with an initial estimate of the object mesh $\hat{\mathbf{M}}^0 = (\hat{\boldsymbol{V}}^0, \hat{\boldsymbol{F}})$, *e.g.*, a sphere (Wang et al., 2018) or a cubified mesh obtained from voxel prediction (Gkioxari et al., 2019). Next, these models iteratively update the initial mesh with L refinement modules to update the vertices (Gkioxari et al., 2019; Lin et al., 2021), *i.e.*, refinement modules directly operate on meshes.

As the surface snapping layer also takes meshes as input, it naturally fits the design of explicit models with refinement modules. Specifically, we insert our surface snapping layer after each refinement module. See Fig. 2 for a high-level illustration. We back-propagate through the surface snapping layer and train the weighting term α jointly with the other differentiable layers. See §3.3 for details.

Surface Snapping with implicit models. The surface snapping layer can also be incorporated into models which use an implicit representation. Given an image, implicit models reconstruct objects via an implicit function, *e.g.*, a function predicting the sign distance of a point to the surface (Park et al., 2019) or predicting whether a point lies inside or outside the mesh (Mescheder et al., 2019). Subsequently, a marching cube algorithm is used to extract object meshes. Hence, we incorporate the surface snapping layer after the marching cube algorithm to snap the mesh vertices to the estimated surface normals. As the classic marching cubes algorithm is not differentiable, we do not train the surface snapping layer jointly with the implicit function. This could potentially be addressed by levering differentiable marching cubes techniques (Liao et al., 2018).

Normal estimation model. For a fair comparison we train a transformer-based dense prediction network (Ranftl et al., 2021) to predict surface normals *on the same training data* that is used for shape reconstruction. Note, the surface normals are extracted from ground truth meshes. *I.e.*, we do not use any additional data beyond what is already provided in datasets for this task. Further note, the obtained surface normals only capture the visible portion of the object. To obtain the face normals $\hat{\boldsymbol{N}}$, we align the surface normal at each pixel to a face $\hat{\boldsymbol{F}}_i$. To do so, we rasterize the predicted mesh $\hat{\mathbf{M}}^{l-1}$ for each $l \in \{1, \ldots, L\}$ to obtain the region that each face $\hat{\boldsymbol{F}}_i$ projects to in the image plane, as well as

Table 3. Quantitative comparison on Pix3D dataset using the setup in (Zhang et al., 2021). We report per-category Chamfer distance and the averaged Chamfer distance (↓) scaled by 10^3. Green indicates better results with surface snapping.

Method	*bed*	*bkcs*	*chair*	*desk*	*sofa*	*table*	*tool*	*wrdrb*	*misc*	Avg
AtlasNet (Groueix et al., 2018)	9.03	6.91	8.37	8.59	6.24	19.46	6.95	4.78	40.05	12.26
TMN (Pan et al., 2019)	7.78	5.93	6.86	7.08	4.25	17.42	4.13	4.09	**23.68**	9.03
MGN (Nie et al., 2020)	5.99	6.56	5.32	**5.93**	**3.36**	14.19	3.12	**3.83**	26.93	8.36
IM3D (Zhang et al., 2021)	4.11	3.96	5.45	7.85	5.61	11.73	**2.39**	4.31	24.65	7.78
`Snap+` IM3D	**3.98**	**3.91**	**5.05**	7.90	5.32	**11.61**	2.47	4.06	24.99	**7.70**

information about its visibility. Finally, to compute the face normal $\hat{\boldsymbol{N}}_i$, we average the surface normals within each face region. Importantly, we set the normal of the non-visible faces to $\mathbf{0}$ to ensure that non-visible faces do not influence the output of the surface snapping layer.

3.3. Training Details

Following Gkioxari et al. (2019), we train both the model parameters of the refinement modules and the surface snapping strength α in Eq. (2) by minimizing the sum of a Chamfer loss $\mathcal{L}_{\text{cham}}$, the normal distance $\mathcal{L}_{\text{norm}}$, and an edge regularizer $\mathcal{L}_{\text{edge}}$, *i.e.*, $\mathcal{L} = \mathcal{L}_{\text{cham}} + \lambda_1 \mathcal{L}_{\text{norm}} + \lambda_2 \mathcal{L}_{\text{edge}}$.

Given a predicted mesh $\hat{\mathbf{M}}^L$ and a ground-truth mesh $\mathbf{M}$, we first sample faces from each of the meshes. The probability of sampling a face is proportional to the area of the face. We then uniformly sample points from the surface of the face using differentiable sampling (Smith et al., 2019). Let the two sets of sampled point clouds be $\hat{\mathcal{P}}$ and $\mathcal{P}$ where $\hat{\mathcal{P}}$ is sampled from the predicted mesh $\hat{\mathbf{M}}^L$ and $\mathcal{P}$ is sampled from the ground-truth mesh $\mathbf{M}$. Let $\Gamma(\mathcal{P}, \hat{\mathcal{P}}) = \{(p, \arg\min_{\hat{p} \in \hat{\mathcal{P}}} \|p - \hat{p}\|), \forall p \in \mathcal{P}\}$ be the set of pairs of every point in $\mathcal{P}$ and its nearest neighbor in $\hat{\mathcal{P}}$. The bi-directional Chamfer and normal losses between $\mathcal{P}$ and $\hat{\mathcal{P}}$ are defined as

$$\mathcal{L}_{\text{cham}}(\hat{\mathcal{P}}, \mathcal{P}) = |\hat{\mathcal{P}}|^{-1} \sum_{(\hat{p}, p) \in \Gamma(\hat{\mathcal{P}}, \mathcal{P})} \|\hat{p} - p\|^2 + |\mathcal{P}|^{-1} \sum_{(p, \hat{p}) \in \Gamma(\mathcal{P}, \hat{\mathcal{P}})} \|p - \hat{p}\|^2,$$

$$\mathcal{L}_{\text{norm}}(\hat{\mathcal{P}}, \mathcal{P}) = -|\hat{\mathcal{P}}|^{-1} \sum_{(\hat{p}, p) \in \Gamma(\hat{\mathcal{P}}, \mathcal{P})} |\boldsymbol{n}_p^\intercal \boldsymbol{n}_{\hat{p}}| - |\mathcal{P}|^{-1} \sum_{(p, \hat{p}) \in \Gamma(\mathcal{P}, \hat{\mathcal{P}})} |\boldsymbol{n}_{\hat{p}}^\intercal \boldsymbol{n}_p|,$$

where $|\mathcal{P}|$ is the cardinality of $\mathcal{P}$ and $\boldsymbol{n}_p$ denotes the unit normal vector of the face that point $p \in \mathbb{R}^3$ is sampled from. Note that the normal *loss* is not the normal *cost* for surface snapping. The normal loss uses ground-truth normal from the ground-truth mesh, which is unavailable at test time. Contrarily, the normal cost uses the estimated surface normal which can be used to solve the objective in Eq. (2) to optimality at test time. Finally, the edge regularizer of a mesh penalizing long edges is given by

$$\mathcal{L}_{\text{edge}}(\hat{\boldsymbol{V}}^L, \hat{\boldsymbol{F}}) = \frac{1}{N_F} \sum_{i=1}^{N_F} \sum_{j,k \in \hat{\boldsymbol{F}}_i} \left\| \hat{\boldsymbol{V}}_j^L - \hat{\boldsymbol{V}}_k^L \right\|^2. \quad (13)$$

4. Experiments

We quantitatively evaluate the proposed surface snapping layer on two widely used datasets, Pix3D (Sun et al., 2018) and ShapeNet (Chang et al., 2015). We show that the proposed surface snapping layer enhances existing explicit methods (Mesh R-CNN (Gkioxari et al., 2019) and GCN Transformer inspired by Lin et al. (2021)) and implicit methods (Im3D (Zhang et al., 2021)). We also show qualitative comparisons illustrating the improved meshes, *e.g.*, sharper edges and smooth surfaces, when using surface snapping layers. Finally, we conclude with an ablation study and discussion.

4.1. Pix3D Setup & Results

Pix3D (Sun et al., 2018) is a challenging dataset for single image reconstruction. It consists of 10,069 real-world photos of nine object categories with potentially cluttered backgrounds and different lighting conditions. Evaluation for the proposed surface snapping with the *explicit* methods follows Gkioxari et al. (2019), where object bounding boxes are unknown, which we refer to as **Setup ❶**. Evaluation for the proposed surface snapping with *implicit* methods follows Zhang et al. (2021), where ground truth object bounding boxes are given, which we refer to as **Setup ❷**.

Evaluation metrics. As explicit and implicit methods use different evaluation setups, we strictly follow the corresponding setup for a fair comparison. **Setup ❶**: Gkioxari et al. (2019) report average precision of the bounding box AP^{box} and average precision of the mask AP^{mask}, as well as average precision of the shape AP^{mesh}, which is defined as the average area under the precision-recall curve, per-category, with F1@0.3 and a threshold of 0.5. In addition, we also report the normal consistency metric which is one minus the normal distance described in §3.3. We report the normal consistency computed on the whole shape (Normal) and on the visible part only ($\text{Normal}^{\text{Vis}}$). **Setup ❷:** Zhang et al. (2021) report per-category Chamfer distance and the averaged Chamfer distance. The predicted mesh is aligned with the ground-truth mesh via ICP (Arun et al., 1987) and 10K points are sampled to compute the Chamfer distance.

Baselines. For Setup ❶, we compare our approach to five methods: Voxel-Only, Sphere-Init, Pixel2Mesh^+, Mesh R-CNN (Gkioxari et al., 2019) and GCN Transformer.

Mesh R-CNN extends Mask R-CNN for multi-object shape reconstruction. This is done by first predicting an intermediate voxel representation and refining it to a mesh output for each of the objects. Pixel2Mesh^+ augments Pixel2Mesh (Wang et al., 2018) by attaching an ROI head to

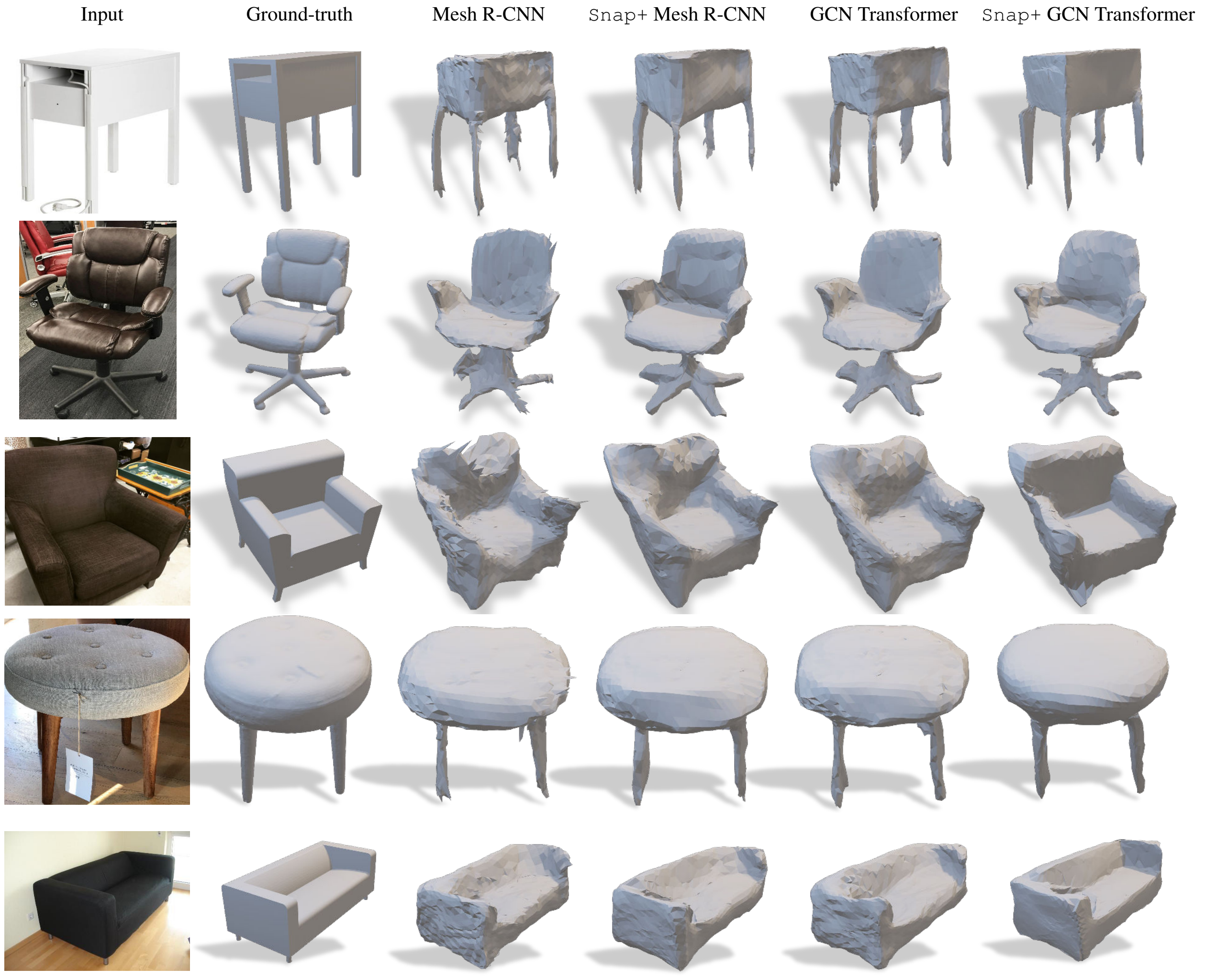

Figure 3. Qualitative comparisions to baselines Mesh R-CNN (Gkioxari et al., 2019) and GCN Transformer on Pix3D.

Mask R-CNN to support multi-object shape reconstruction. The Voxel-Only method refers to the intermediate voxel representation of Mesh R-CNN. The Sphere-Init refers to the baseline of initializing from a sphere mesh and performing vertex refinements, similar to Pixel2Mesh^{+} but without subdivision. Finally, GCN Transformer is a baseline inspired by Lin et al. (2021), where we apply the self-attention mechanism before graph convolutions in a refinement stage.

For Setup ❷, we compare to baselines AtlasNet (Groueix et al., 2018), TMN (Pan et al., 2019), MGN (Nie et al., 2020), and IM3D (Zhang et al., 2021).

Quantitative results (Setup ❶). We evaluate using the proposed surface snapping with the Mesh R-CNN framework and provide quantitative results for the Pix3D split S_1 in Tab. 1 and for the Pix3D split S_2 in Tab. 2. Note that split S_2 is more challenging than split S_1 as split S_2 guarantees that 3D models which appear in training do not appear in the test set, while split S_1 does not guarantee this.

On both splits, we find that incorporating surface snapping generally improves (shown in green) the quality of the reconstructed shape in terms of AP^{mesh} when compared to the corresponding baselines. As expected, the face normals of the mesh predicted with the proposed surface snapping are more consistent with the ground truth. We observe that surface snapping improves nearly all baselines on the normal consistency metric computed on both the whole shape (Normal) and the visible part only (Normal$^{\text{Vis}}$). These results indicate that surface snapping effectively utilizes the estimated normals and optimizes the vertices of the predicted mesh to snap to the observed geometry.

Quantitative results (Setup ❷). We evaluate using the proposed surface snapping following the setup in (Zhang et al., 2021) and compare it to the baselines in Tab. 3. We observe that incorporating surface snapping with existing methods further improves the reconstruction quality.

Qualitative results. We illustrate the effectiveness of the

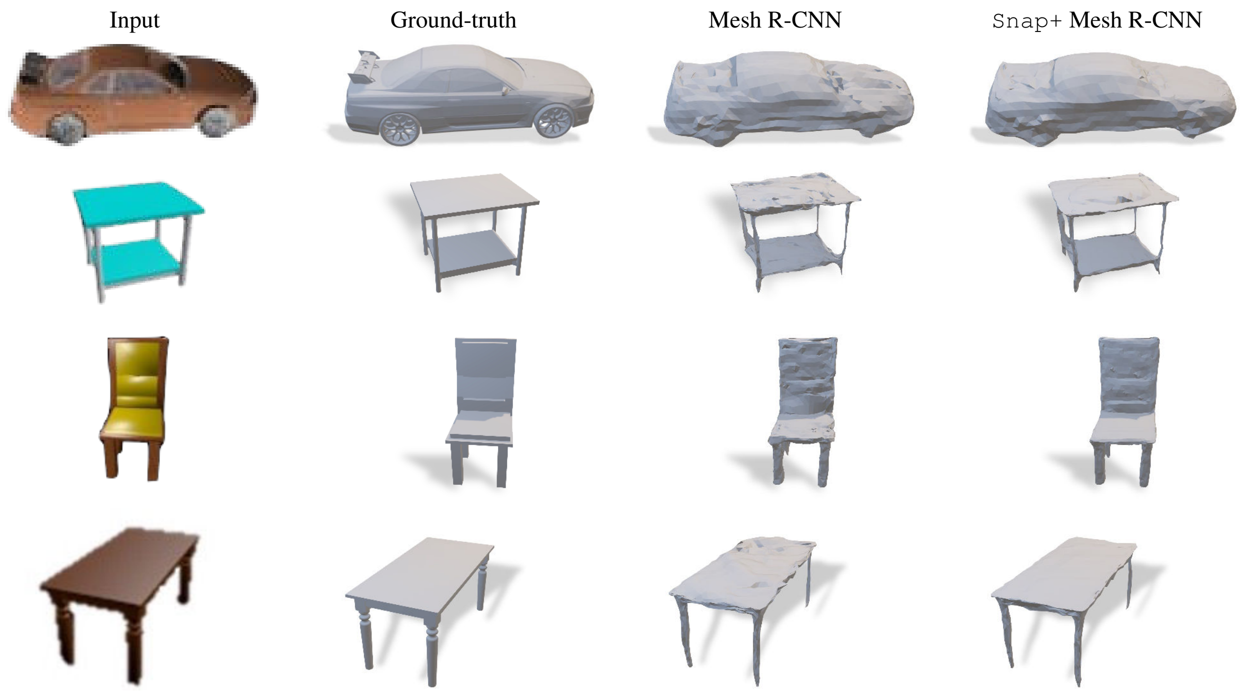

Figure 4. Qualitative results of our method compared to the baseline Mesh R-CNN (Gkioxari et al., 2019) on ShapeNet.

Table 4. Quantitative comparison on ShapeNet (**Best**). Green indicates better results with surface snapping.

Method	F1@0.1(↑)	F1@0.3(↑)	F1@0.5(↑)	Chamfer(↓)	Normal(↑)
Sphere-Init	38.3	86.5	**95.1**	**0.132**	0.711
Pixel2Mesh$^+$ (Wang et al., 2018)	38.3	86.6	**95.1**	**0.132**	0.707
Mesh R-CNN (Gkioxari et al., 2019)	**39.2**	**86.8**	**95.1**	0.133	0.725
`Snap+` Sphere-Init	37.6	85.6	94.5	0.133	0.716
`Snap+` Pixel2Mesh$^+$	37.0	85.3	94.6	0.134	0.717
`Snap+` Mesh R-CNN	38.0	86.0	94.7	0.134	**0.727**

Table 5. Quantitative comparison on ShapeNet (**Pretty**). Green indicates better results with surface snapping.

Method	F1@0.1(↑)	F1@0.3(↑)	F1@0.5(↑)	Chamfer(↓)	Normal(↑)
Sphere-Init	34.5	82.2	92.9	0.175	0.718
Pixel2Mesh$^+$ (Wang et al., 2018)	**34.9**	82.3	92.9	0.175	0.727
Mesh R-CNN (Gkioxari et al., 2019)	34.8	**82.4**	**93.1**	0.176	0.699
`Snap+` Sphere-Init	34.6	82.2	92.9	0.174	0.722
`Snap+` Pixel2Mesh$^+$	**34.9**	82.3	93.0	0.175	**0.731**
`Snap+` Mesh R-CNN	**34.9**	**82.4**	**93.1**	**0.171**	0.707

proposed method for mesh reconstruction on Pix3D images in Fig. 3. We train a dense prediction transformer (Ranftl et al., 2021) using the Pix3D data to predict surface normals. We observe that the results with surface snapping are generally less noisy and more visually appealing.

4.2. ShapeNet Setup & Results

ShapeNet (Chang et al., 2015) is a commonly used synthetic dataset for training and evaluating single image object reconstruction. We use the rendered images provided by Choy et al. (2016) and use the training and test splits provided by Wang et al. (2018). The rendered images have a black background. The training split consists of 840,189 images and the test split contains 210,051 images. All images are of size 137×137.

Evaluation Metrics. For evaluation, we report the Chamfer distance and normal consistency (one minus normal distance) as described in §3.3 as well as the F-score with various distance thresholds following Gkioxari et al. (2019).

Baselines. We use the same set of explicit baselines as we did for Pix3D data, except for GCN Transformer as it utilizes too much GPU memory for the given batch size.

Quantitative results. We provide the quantitative results for ShapeNet in Tab. 4 and Tab. 5. Following Gkioxari et al. (2019), we evaluate baselines using two settings: **Pretty** and **Best**. The two settings differ in the shape regularizer. Specifically, the **Pretty** setting enables the edge regularizer, while the **Best** does not use one. While the images are of low resolution, which makes it challenging to estimate accurate normals, we still find our method to improve the normal consistency metric in the **Best** setting and to improve the Chamfer distance in the **Pretty** setting.

Qualitative results. In Fig. 4, we visualize the reconstructed shapes on ShapeNet. Incorporating surface snapping leads to less noisy reconstruction that better matches the observed geometry.

4.3. Additional analysis

Ablation study on α. We study the effect of the weight α on the normal cost in the surface snapping objective given

Table 6. Ablation on α in Eq. (2) for `Snap+` Mesh R-CNN.

	Pix3D S_1			Pix3D S_2		
	AP$^{\text{mesh}}$	Normal	Normal$^{\text{Vis}}$	AP$^{\text{mesh}}$	Normal	Normal$^{\text{Vis}}$
$\alpha = 0.0$	53.4	21.5	45.4	29.1	21.4	46.5
$\alpha = 1.0$	53.4	22.2	47.1	28.7	20.8	44.7
$\alpha = 2.0$	52.7	21.4	44.8	28.5	21.6	44.6
α learned	**54.1**	**23.0**	**48.8**	**29.9**	**23.0**	**49.7**

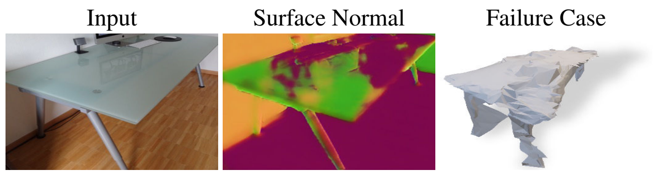

Figure 5. Failure mode of our approach. If the surface normal estimation is not accurate the proposed reconstruction degrades.

in Eq. (2) using Mesh R-CNN. We report quantitative results of models trained with different values of α in Tab. 6. We find the learning of α to improve the overall performance.

Surface snapping implementation comparison. We ablate different aspects of our implementation and report inference time and memory usage on meshes with 5,280 faces with a batch size of two. We compare the following settings (on an Nvidia Tesla V100 GPU): (a) Without considering any structure in the system of equations, the solver reports an out-of-memory error; (b) Leveraging the sparsity structure and solving with `torch.solve` takes 1.71s and consumes 4437MiB in GPU memory; (c) Leveraging the sparsity structure and using the developed `CGSolver` takes 0.46s and consumes 1203MiB in GPU memory.

Limitations. In Fig. 5, we show that surface snapping struggles when the estimated surface normal is not accurate. On the flip side, we expect the proposed technique to further improve if normal estimation accuracy increases.

5. Conclusion

We develop surface snapping, an optimization layer, for 3D shape reconstruction. This layer ensures that the reconstructed shape 'snaps' to predicted normal estimates by solving an optimization problem that encourages the edges in the predicted mesh to be orthogonal to the predicted normals. To run this layer efficiently, we develop a conjugate-gradient-based method that leverages the sparsity of the problem. During training, we back-propagate through this layer via implicit differentiation. On the challenging Pix3D and ShapeNet datasets, we show that incorporating surface snapping into existing methods leads to competitive shape reconstruction performance and yields results with improved normal consistency. More generally, we believe advances in scene understanding can aid recently developed monocular object reconstruction techniques. The developed surface snapping is a step in this direction that explicitly utilizes estimated surface normal geometry.

Acknowledgments: Work supported in part by NSF Grants 2008387, 2045586, 2106825, MRI 1725729, and NIFA award 2020-67021-32799. We thank NVIDIA for providing a GPU.

References

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. In *Proc. NeurIPS*, 2019. 2

Amos, B. and Kolter, J. Z. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. ICML*, 2017. 1, 2

Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable MPC for end-to-end planning and control. In *Proc. NeurIPS*, 2018. 2

Arun, K. S., Huang, T. S., and Blostein, S. D. Least-squares fitting of two 3-D point sets. 1987. 6

Bae, G., Budvytis, I., and Cipolla, R. Estimating and exploiting the aleatoric uncertainty in surface normal estimation. In *Proc. ICCV*, 2021. 1

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. In *Proc. NeurIPS*, 2019. 2

Bai, S., Koltun, V., and Kolter, J. Z. Multiscale deep equilibrium models. In *Proc. NeurIPS*, 2020. 2

Bai, S., Geng, Z., Savani, Y., and Kolter, J. Z. Deep equilibrium optical flow estimation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2

Bansal, A., Russell, B., and Gupta, A. Marr revisited: 2D-3D alignment via surface normal prediction. In *Proc. CVPR*, 2016. 1, 3

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., and Yu, F. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*, 2015. 2, 6, 8

Chen, Z. and Zhang, H. Learning implicit fields for generative shape modeling. In *Proc. CVPR*, 2019. 1, 2

Chen, Z., Tagliasacchi, A., and Zhang, H. BSP-Net: Generating compact meshes via binary space partitioning. In *Proc. CVPR*, 2020. 2

Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *Proc. ECCV*, 2016. 8

Dai, A., Ruizhongtai Qi, C., and Nießner, M. Shape completion using 3D-encoder-predictor cnns and shape synthesis. In *Proc. CVPR*, 2017. 2

Deng, B., Genova, K., Yazdani, S., Bouaziz, S., Hinton, G., and Tagliasacchi, A. CvxNet: Learnable convex decomposition. In *Proc. CVPR*, 2020. 2

Do, T., Vuong, K., Roumeliotis, S. I., and Park, H. S. Surface normal estimation of tilted images via spatial rectifier. In *Proc. ECCV*, 2020. 1

Duggal, S. and Pathak, D. Topologically-aware deformation fields for single-view 3d reconstruction. In *Proc. CVPR*, June 2022. 2

Eigen, D. and Fergus, R. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proc. ICCV*, 2015. 1, 3

Fouhey, D., Gupta, A., and Hebert, M. Data-driven 3D primitives for single image understanding. In *Proc. ICCV*, 2013. 3

Gao, W., Wang, A., Metzer, G., Yeh, R. A., and Hanocka, R. Tetgan: A convolutional neural network for tetrahedral mesh generation. 2022. 2

Gkioxari, G., Malik, J., and Johnson, J. Mesh R-CNN. In *Proc. ICCV*, 2019. 2, 5, 6, 7, 8

Gould, S., Hartley, R., and Campbell, D. Deep declarative networks. *IEEE TPAMI*, Aug 2022. 1

Groueix, T., Fisher, M., Kim, V. G., Russell, B. C., and Aubry, M. A papier-mâché approach to learning 3D surface generation. In *Proc. CVPR*, 2018. 1, 2, 6, 7

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask R-CNN. In *Proc. ICCV*, 2017. 2

Hickson, S., Raveendran, K., Fathi, A., Murphy, K., and Essa, I. Floors are Flat: Leveraging Semantics for Real-Time Surface Normal Prediction. In *ICCV Workshop*, 2019. 1, 3

Horn, B. K. Obtaining shape from shading information. *The psychology of computer vision*, 1975. 1

Huang, Z., Bai, S., and Kolter, J. Z. $(Implicit)^2$: Implicit layers for implicit representations. *Adv. Neural Inform. Process. Syst.*, 2021. 2

Izadinia, H., Shan, Q., and Seitz, S. M. IM2CAD. In *Proc. CVPR*, 2017. 2

Kundu, A., Li, Y., and Rehg, J. M. 3D-RCNN: Instance-level 3D object reconstruction via render-and-compare. In *Proc. CVPR*, 2018. 2

Ladický, L., Zeisl, B., and Pollefeys, M. Discriminatively trained dense surface normal estimation. In *Proc. ECCV*, 2014. 1, 3

Liao, S., Gavves, E., and Snoek, C. G. M. Spherical regression: Learning viewpoints, surface normals and 3D rotations on n-spheres. In *Proc. CVPR*, 2019. 1

Liao, Y., Donne, S., and Geiger, A. Deep marching cubes: Learning explicit surface representations. In *Proc. CVPR*, 2018. 5

Lin, K., Wang, L., and Liu, Z. Mesh graphormer. In *Proc. ICCV*, 2021. 5, 6, 7

Mahmud, J., Price, T., Bapat, A., and Frahm, J.-M. Boundary-aware 3D building reconstruction from a single overhead image. In *Proc. CVPR*, 2020. 2

Marr, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. MIT Press, 1982. 1

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. Occupancy networks: Learning 3D reconstruction in function space. In *Proc. CVPR*, 2019. 1, 2, 5

Mo, K., Zhu, S., Chang, A. X., Yi, L., Tripathi, S., Guibas, L. J., and Su, H. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *Proc. CVPR*, 2019. 2

Nash, C., Ganin, Y., Eslami, S. A., and Battaglia, P. PolyGen: An autoregressive generative model of 3D meshes. In *Proc. ICML*, 2020. 2

Nayar, S. K., Ikeuchi, K., and Kanade, T. Shape from interreflections. *IJCV*, 1991. 1

Nie, Y., Han, X., Guo, S., Zheng, Y., Chang, J., and Zhang, J. J. Total3DUnderstanding: Joint layout, object pose and mesh reconstruction for indoor scenes from a single image. In *Proc. CVPR*, 2020. 2, 6, 7

Pan, J., Han, X., Chen, W., Tang, J., and Jia, K. Deep mesh reconstruction from single rgb images via topology modification networks. In *Proc. ICCV*, 2019. 6, 7

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. DeepSDF: Learning continuous signed distance functions for shape representation. In *Proc. CVPR*, 2019. 1, 2, 5

Paschalidou, D., Gool, L. V., and Geiger, A. Learning unsupervised hierarchical part decomposition of 3D objects from a single RGB image. In *Proc. CVPR*, 2020. 2

Peng, S., Niemeyer, M., Mescheder, L., Pollefeys, M., and Geiger, A. Convolutional occupancy networks. In *Proc. ECCV*, 2020. 2

Pentland, A. P. A new sense for depth of field. *IEEE TPAMI*, 1987. 1

Pokle, A., Geng, Z., and Kolter, Z. Deep equilibrium approaches to diffusion models. In *Proc. NeurIPS*, 2022. 2

Qi, X., Liu, Z., Liao, R., Torr, P. H., Urtasun, R., and Jia, J. GeoNet++: Iterative geometric neural network with edge-aware refinement for joint depth and surface normal estimation. *IEEE TPAMI*, 2020. 1, 2

Ranftl, R., Bochkovskiy, A., and Koltun, V. Vision transformers for dense prediction. *Proc. ICCV*, 2021. 5, 8

Ren, S., He, K., Girshick, R., and Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proc. NeurIPS*, 2015. 2

Smith, E., Fujimoto, S., Romero, A., and Meger, D. Geometrics: Exploiting geometric structure for graph-encoded objects. In *Proc. ICML*, 2019. 6

Sun, X., Wu, J., Zhang, X., Zhang, Z., Zhang, C., Xue, T., Tenenbaum, J. B., and Freeman, W. T. Pix3D: Dataset and methods for single-image 3D shape modeling. In *Proc. CVPR*, 2018. 2, 6

Tatarchenko, M., Dosovitskiy, A., and Brox, T. Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs. In *Proc. ICCV*, 2017. 1, 2

Tulsiani, S., Gupta, S., Fouhey, D. F., Efros, A. A., and Malik, J. Factoring shape, pose, and layout from the 2D image of a 3D scene. In *Proc. CVPR*, 2018. 2

Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W., and Jiang, Y.-G. Pixel2Mesh: Generating 3D mesh models from single RGB images. In *Proc. ECCV*, 2018. 1, 2, 5, 6, 8

Wang, P., Shen, X., Russell, B., Cohen, S., Price, B., and Yuille, A. L. Surge: Surface regularized geometry estimation from a single image. In *Proc. NeurIPS*, 2016. 1, 3

Wang, P.-W., Donti, P., Wilder, B., and Kolter, Z. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proc. ICML*, 2019. 2

Wang, R., Geraghty, D., Matzen, K., Szeliski, R., and Frahm, J.-M. VPLNet: Deep single view normal estimation with vanishing points and lines. In *Proc. CVPR*, 2020. 1

Wang, X., Fouhey, D., and Gupta, A. Designing deep networks for surface normal estimation. In *Proc. CVPR*, 2015. 1, 3

Wolff, L. B., Shafer, S. A., and Healey, G. E. *Physics-Based Vision: Principles and Practice: Radiometry*. 1993. 1

Wu, J., Wang, Y., Xue, T., Sun, X., Freeman, B., and Tenenbaum, J. MarrNet: 3D shape reconstruction via 2.5D sketches. *Proc. NeurIPS*, 30, 2017. 1, 2

Wu, R., Zhuang, Y., Xu, K., Zhang, H., and Chen, B. PQ-NET: A generative part seq2seq network for 3D shapes. In *Proc. CVPR*, 2020. 2

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3D shapenets: A deep representation for volumetric shapes. In *Proc. CVPR*, 2015. 1, 2

Yeh, R. A., Hu, Y.-T., Ren, Z., and Schwing, A. G. Total variation optimization layers for computer vision. In *Proc. CVPR*, 2022. 2

Yu, Z., Peng, S., Niemeyer, M., Sattler, T., and Geiger, A. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *Proc. NeurIPS*, 2022. 1

Zamir, A. R., Sax, A., Cheerla, N., Suri, R., Cao, Z., Malik, J., and Guibas, L. J. Robust learning through cross-task consistency. In *Proc. CVPR*, 2020. 1

Zhang, C., Cui, Z., Zhang, Y., Zeng, B., Pollefeys, M., and Liu, S. Holistic 3D scene understanding from a single image with implicit representation. In *Proc. CVPR*, 2021. 2, 6, 7

Zhang, Z., Cui, Z., Xu, C., Yan, Y., Sebe, N., and Yang, J. Pattern-affinitive propagation across depth, surface normal and semantic segmentation. In *Proc. CVPR*, 2019. 1

Zou, C., Yumer, E., Yang, J., Ceylan, D., and Hoiem, D. 3D-PRNN: Generating shape primitives with recurrent neural networks. In *Proc. ICCV*, 2017. 2