NEURALSLICE: Neural 3D Triangle Mesh Reconstruction via Slicing 4D Tetrahedral Meshes

Chenbo Jiang [1 2] **Jie Yang** [1] **Shwai He** [3] **Yu-Kun Lai** [4] **Lin Gao** [1 5]

Abstract

Learning-based high-fidelity reconstruction of 3D shapes with varying topology is a fundamental problem in computer vision and computer graphics. Recent advances in learning 3D shapes using explicit and implicit representations have achieved impressive results in 3D modeling. However, the template-based explicit representation is limited by fixed topology, and the implicit representation, although flexible with arbitrary topology, requires a large number of sampled points to regress the surface, which is computationally expensive. In this work, we propose a novel 3D shape representation named NEURALSLICE, which represents a 3D shape as the intersection of a 4D tetrahedral mesh and a 4D hyperplane. A novel network is designed to incorporate the proposed representation flexibly, which learns a deformable 4D template and a parameter for slicing 4D hyperplane to reconstruct the 3D object. To learn the local deformation of the 4D template, we further propose a spatial-aware network to locate the 4D points within the 3D feature volume of input shape via positional encoding, which leverages the local geometrical feature to guide the 4D deformation. By addressing the 3D problem in a higher 4D space, our method supports flexible topology changes while being highly efficient. Our method is guaranteed to produce manifold meshes. NEURALSLICE outperforms the state-of-the-art explicit-based approaches in terms of reconstruction quality. Compared with implicit approaches, by avoiding point sampling, our method is 10 times faster than the implicit approaches, and better preserves thin structures. NEURALSLICE has the capability of representing various shapes and topologies using a single 4D tetrahedral mesh. The corresponding code can be found on GitHub at `https://github.com/IGLICT/NEURALSLICE`.

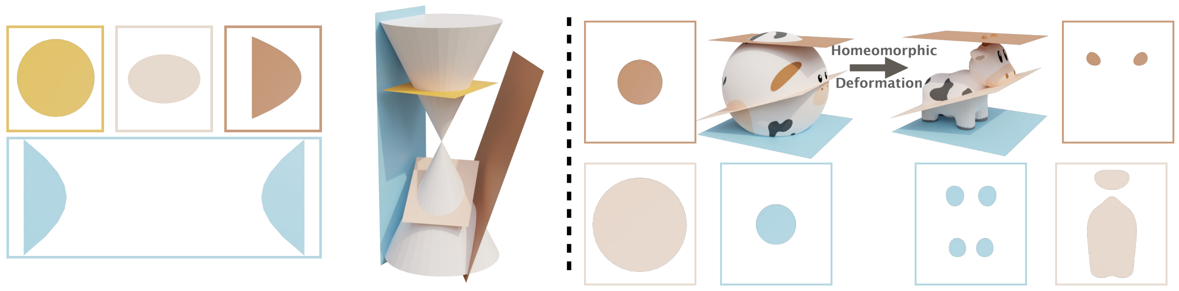
[1]Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy of Sciences [2]Nanjing University of Science and Technology Zijin College [3]University of Maryland, College Park [4]Cardiff University [5]University of Chinese Academy of Sciences. Correspondence to: Lin Gao <gaolin@ict.ac.cn>.

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

1. Introduction

Learning-based high-fidelity reconstruction of 3D shapes is a fundamental task for many applications in computer vision and graphics, *e.g.*, 3D scene modeling, virtual/augmented reality, architecture, gaming, film, etc. To cope with rich 3D shapes in the real world, 3D shape reconstruction methods should ideally be able to represent and recover detailed surface geometry and diverse topologies. The topology describes the intrinsic properties of shapes (Berger & Gostiaux, 2012; Lee, 2010). However, it is non-trivial to efficiently reconstruct meshes with correct, flexible topology, while recovering fine geometry details.

Shape reconstruction methods are inherently related to the underlying geometry representations. Existing methods can be broadly categorized as implicit and explicit approaches. However, neither explicit representations (*e.g.*, voxels (Wu et al., 2015; 2016; Choy et al., 2016), point clouds (Fan et al., 2017; Qi et al., 2017; Prokudin et al., 2019), meshes (Groueix et al., 2018; Yang et al., 2018; Bednarik et al., 2020; Deng et al., 2020b; Low & Lee, 2022)) nor implicit representations (*e.g.*, signed distance functions (SDF) (Park et al., 2019; Chen & Zhang, 2019; Mescheder et al., 2019), unsigned distance functions (UDF) (Guillard et al., 2022; Chibane et al., 2020)) can satisfy the above two important properties simultaneously in an efficient fashion. For explicit representations, voxel-based methods have struggled between efficient computation/storage and high-resolution surfaces, and point clouds are free of structure, which are ambiguous for complex shape and structures. On the contrary, a 3D mesh is capable of representing a 3D surface efficiently and compactly with a collection of *vertices*, *edges* and *faces*, but the predefined edges and faces limit its representation capability of flexible topology change since it is homeomorphic to the explicit template. On the other



(a) Conic Section (b) Sliced deformable template by different planes in 3D space

Figure 1. Our work is based on slicing a 4D tetrahedral mesh with a 4D hyperplane, which can be difficult to imagine. To help with understanding, we illustrate a lower-dimensional situation, i.e., slicing a 3D surface with a 3D plane, which bears similar principles. (a) slicing a cone surface to obtain different conic section curves (hyperbola, circle, ellipse, and parabola). Although the 3D surface is fixed, the 2D intersection curves can have different topologies. (b) a more complex example that deforms a sphere homomorphically to a cow. Although the 3D surface does not change topology, the 2D sliced curves have changed topology during deformation. We use different colors to indicate slices with different planes.

hand, while implicit representations can successfully handle diverse topology, they need a large number of sampling points to regress the diverse topology and extract mesh-based explicit representation by the Marching Cubes algorithm (Lorensen & Cline, 1987). The computation is expensive and it may not be possible to represent some thin structures (*e.g.*, slots of chairs, spokes) in an appropriate resolution. Recent advances have focused on finding a solution for efficient and flexible 3D representation. The mesh-based approaches pursue the goal to generalize to shapes with complex and thin structures via changing the explicit topology, *e.g.*, patch-based (Yang et al., 2018; Groueix et al., 2018; Low & Lee, 2022), part-based (Yang et al., 2022; Gao et al., 2019), etc. The implicit approaches are mainly dedicated to resolving the computationally expensive issues, *e.g.*, (Gao et al., 2020; Shen et al., 2021). However, their fundamental limitations still remain.

To tackle these challenges, we propose a novel 3D shape reconstruction method named NEURALSLICE, which represents and reconstructs each 3D shape as the intersection of a 4D tetrahedral mesh $\mathcal{T}$ and a 4D hyperplane Π. The interesting idea is inspired by level set theory (Lee, 2013). As 4D space is hard to imagine, we illustrate the core idea in Figure 1 using two 3D toy examples. The conic section example (a) shows that intersecting the same 3D shape with different planes can result in curves of different topologies. In (b), the homeomorphic deformation from a sphere to a cow demonstrates that topology-preserving deformation in 3D can lead to change of topology for intersection curves. In our approach, by deforming a 4D tetrahedral mesh and intersecting it with a 4D hyperplane, our method is able to reconstruct meshes with flexible topology, while avoiding expensive sampling needed for implicit approaches.

Furthermore, a novel architecture is designed to incorporate the proposed representation, which learns a deformable 4D tetrahedral mesh (3-manifold $\mathcal{T}\subset\mathbb{R}^4$) and a 4D hyperplane that slices $\mathcal{T}$ to reconstruct the 3D shape. To learn the deformation of $\mathcal{T}$, we propose a spatial-aware Embedding Net that establishes a mapping from the 4D tetrahedral mesh to 3D space. The Embedding Net performs a positional projection from each vertex on the 4D tetrahedral mesh to 3D space via integrating the global feature of the input shape, which determines the local geometrical feature from the 3D feature volume of the input shape via trilinear interpolation. Under the guidance of the local geometric feature, we introduce a Deformation Net with residual blocks to predict the deformed 4D tetrahedral mesh. In addition, to preserve the regular connection during deformation, we adopt Lipschitz Normalization and Laplacian regularization on the learnable weights and explicit meshes respectively. Finally, our whole architecture also predicts a learnable parameter to determine the hyperplane to slice the deformed 4D tetrahedral mesh to reconstruct explicit meshes.

Our method NEURALSLICE reconstructs plausible and accurate 3D meshes with flexible topology efficiently via a geometrically meaningful operation – a learned hyperplane slices a deformable 4D tetrahedron mesh. Experiments demonstrate that our method seeks a new solution to describing the diverse topology in an explicit fashion and outperforms existing explicit approaches with high accuracy and robustness for 3D shape reconstruction. Our method is superior to implicit approaches with 10 times faster speed for shape reconstruction and improved robustness to thin structures. To summarize, our major contributions are three-fold:

- We propose NEURALSLICE, a novel approach to representing 3D shapes with flexible topology in the explicit fashion, which represents a 3D shape as the intersection of a 4D deformable tetrahedral mesh and a learnable 4D hyperplane;

- A novel network architecture that incorporates NEURALSLICE to learn the deformable 4D tetrahedral mesh and 4D hyperplane for 3D shape reconstruction, which ensures the high fidelity of the obtained meshes with regular triangulation by introducing Lipschitz normalization and Laplacian regularization;

- Experiments demonstrate that our 3D reconstruction method accurately reconstructs 3D shapes of diverse topology, outperforming existing explicit methods in accuracy, and much faster than implicit methods. Furthermore, NEURALSLICE can represent different 3D shapes and topologies in one 4D tetrahedral mesh.

2. Related Work

As 3D shape reconstruction is closely related to the underlying 3D shape representation, we review methods on learning-based 3D representations (Xiao et al., 2020). Generally, learning-based 3D representations can be classified as explicit representation (e.g., voxels, point clouds and meshes) and implicit representation (e.g., SDF, UDF).

Explicit 3D Representations. Explicit representations mainly include voxels, point clouds and meshes. Voxel-based representation is one of the earliest learning-based representations for 3D reconstruction (Wu et al., 2015; 2016; Choy et al., 2016; Gkioxari et al., 2019). However, the capability of such methods in representing accurate 3D shapes is limited by high memory and computational costs, especially when the resolution of the voxel grid becomes higher. In contrast, point clouds (Fan et al., 2017; Qi et al., 2017; Prokudin et al., 2019) are a very efficient representation for 3D deep learning, but they cannot represent topological relations.

Meshes are a widely used representation for 3D deep learning. Some learning-based methods directly deform a mesh with fixed topology for shape reconstruction and/or generation. Pixel2Mesh (Wang et al., 2020; Wen et al., 2019) and NFM (Gupta & Chandraker, 2020) fit different objects by deforming a sphere mesh via graph neural network or neural ODE (Ricky et al., 2018). Instead of deforming a sphere, other works such as 3DN (Wang et al., 2019) and ShapeFlow (Jiang et al., 2020) deform source templates to target shapes. Nevertheless, the deformation process does not change the mesh connectivity, and such methods cannot reconstruct meshes with flexible topology.

To address the fixed-topology problem, patch-based methods (Groueix et al., 2018; Yang et al., 2018) learn to deform a set of mesh patches (known as an atlas) to represent target shapes. Some methods try to regularize patch distortion and overlap (Bednarik et al., 2020; Deng et al., 2020b) and some patch-based reconstruction methods achieve good results (Badki et al., 2020; Williams et al., 2019; Morreale et al., 2021). Although patch-based representation allows flexible topological changes, there is no connection between patches, and it is difficult to fully constrain patches to avoid visual cracks when produced shapes are rendered. Another approach to addressing fixed topology is to use part-based representations where each part is a mesh of fixed topology. GRASS (Li et al., 2017) and StructureNet (Mo et al., 2019) use recursive neural networks (RvNN), which can represent complicated topology. However, each part is represented as an oriented bounding box (Li et al., 2017) or a point cloud (Mo et al., 2019), which has limited accuracy. SDM-Net (Gao et al., 2019) and DSG-Net (Yang et al., 2022) instead use deformed meshes to represent parts, which can achieve high-quality representation. However, part-based methods require part-level semantic segmentation for training.

Some methods try to explicitly modify the topology through pruning. Low & Lee (2022) and Pan et al. (2019) modify topology by removing faces. TearingNet (Pang et al., 2021) introduces a "tearing" operation to split meshes. However, their modification leads to non-watertight and low-quality reconstruction, and their method still has limited capability for changing topology. None of these explicit methods produce watertight meshes with flexible topology. Our method is an explicit method that can reconstruct watertight meshes with arbitrary topology.

Implicit 3D Representations. Recently, implicit surface representations are becoming increasingly popular for 3D deep learning (Park et al., 2019; Chen & Zhang, 2019; Mescheder et al., 2019). Implicit signed distance field (SDF) is an excellent 3D representation, which can represent and generate smooth, continuous, and arbitrary topology models. Liu et al. (2021) and Peng et al. (2020) employ local features to improve reconstruction performance. Other works (Atzmon & Lipman, 2020a;b; Boulch et al., 2021; Ma et al., 2021) proposed methods to learn an SDF from unoriented point clouds. But SDF cannot represent open surfaces. To address this, Unsigned distance functions (UDF) (Guillard et al., 2022; Chibane et al., 2020) are used to cope with open surfaces. Some works accelerate implicit methods using the hierarchical octree structure (Wang et al., 2022; Tang et al., 2021), but the speedup is limited to 2-3 times. However, such implicit methods cannot directly generate meshes. To extract a mesh from implicit fields, the marching cubes (Lorensen & Cline, 1987) algorithm is needed for iso-surface extraction. It also requires dense sampling of points around the target surface to extract an accurate surface mesh, so its generation efficiency is low.

BSP-Net (Chen et al., 2020) and CvxNet (Deng et al., 2020a)

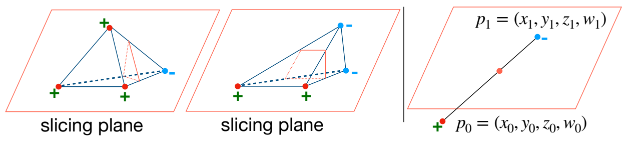


Figure 2. Left: Two possible cross sections when slicing a tetrahedron with a plane: a triangle or a quadrilateral; Right: Computing an intersection point. Note although this illustration is in 3D, the same principle also works in 4D.

approximate an implicit field by many convex parts. But these approximations are not suitable for reconstructing shapes with fine details and thin structures. DMTet and DefTet (Shen et al., 2021; Gao et al., 2020) learn implicit fields via deformable 3D tetrahedral meshes. Although deforming 3D tetrahedral meshes may appear similar to our work, we have different motivations to solve this problem with 4D deformation. Moreover, NEURALSLICE is more general than DMTet: we can encode multiple 3D meshes into a single 4D tetrahedral mesh by slicing with different hyper-planes which DMTet cannot handle. HyperNeRF (Park et al., 2021) improves topological flexibility of traditional Neural Radiance Fields (NeRF) (Mildenhall et al., 2020), by "slicing" a low-dimensional object from a high-dimensional space, which is conceptually similar to our idea. However, they use an MLP (Multi-Layer Perceptron) to output the "sliced" dimension, rather than explicitly constructing and slicing a high dimensional geometry itself.

3. Methodology

Our method aims to reconstruct a 3D mesh for a given (potentially noisy) point cloud. We formulate this as a problem of deforming a 4D tetrahedral mesh as a template, which is then sliced by a 4D hyperplane to produce a 3D mesh, with flexible topological change. In the following, we first present the preliminary for 4D tetrahedral meshes (Sec. 3.1) and how to obtain 3D meshes from them through slicing (Sec. 3.2). After that, we present our neural network to learn to deform a 4D template and slice it to produce a 3D mesh given an input point cloud (Sec. 3.3). In the end, we describe our refinement step to produce finer meshes (Sec. 3.4).

3.1. Preliminary: 4D Tetrahedral Meshes

Previous works such as Atlas-O (Groueix et al., 2018), Pix2mesh (Wang et al., 2020), NFM (Gupta & Chandraker, 2020), SDM-Net (Gao et al., 2019) all use neural networks to predict the deformation of a 3D template with fixed topology, and the deformation process for each vertex can be written as: $F_\Theta : \mathbb{R}^3 \to \mathbb{R}^3$, where Θ represents the network parameters. As a result, the final obtained shape is homeomorphic to the template mesh. As illustrated in Figure 1, which shows a 3D toy example, slicing 3D shapes with homeomorphic deformation allows generating 2D curves with changing topology. We therefore uplift the deformable mesh to 4D, leading to a 4D tetrahedral mesh.

4D Tetrahedron Mesh. A manifold triangle mesh is a discrete 2-manifold embedded in 3D Euclidean space and the basic unit of a 3D mesh is a 2-simplex (i.e., triangle). Our 4D tetrahedral mesh is a discrete 3-manifold embedded in 4D Euclidean space. We use 3-simplexes (i.e., tetrahedra) as the basic unit of 4D tetrahedron mesh.

In detail, our 4D tetrahedral mesh is made up of a set of tetrahedra including vertices V and tetrahedra T, denoted as $\mathcal{T} = \{V, T\}$. Every vertex in the vertex set $v_i \in V$ has 4 dimensions $v_i = (x_i, y_i, z_i, w_i)$, and every tetrahedron in the tetrahedron set $t_i \in T$ has 4 points $t_i = (v_i^1, v_i^2, v_i^3, v_i^4)$. To construct the 4D tetrahedral mesh template, as it is a 3-manifold embedded in $\mathbb{R}^4$, we first build a 3D tetrahedral mesh $\mathcal{T}^3$ by splitting a 3D grid into tetrahedra. We then map each vertex $v \in \mathbb{R}^3$ to $\mathbb{R}^4$ using a mapping $f : \mathbb{R}^3 \to \mathbb{R}^4$. This is conceptually similar to mapping from the UV texture space to a 3D mesh for texturing. Details are provided in Appendix B.1.

3.2. Slicing 4D Tetrahedral Meshes to 3D Meshes

Visualizing 4D objects is a fascinating and challenging problem with a long history (Abbott, 1884; Chu et al., 2009; Hilbert & Cohn-Vossen, 1932; Emmer & Banchoff, 1990; Liu & Zhang, 2022). These visualization techniques inspired many methods on 4D objects, such as manipulation (Yan et al., 2012; Hui & Hanson, 2006), dynamic simulation (Bosch, 2020) and iso-surface extraction (Bhaniramka et al., 2000). None of the existing work addresses shape reconstruction. In this work, we use the *Slice* operation that cuts through a 4D tetrahedral mesh with a 4D hyperplane to obtain a 3D mesh.

We now present how to generate a 3D mesh $\mathcal{M} = \{\mathcal{E}, \mathcal{V}\}$ with a given 4D tetrahedral mesh $\mathcal{T} = \{P, T\}$ and the slice dimension $w = \alpha$. $\mathcal{E}$ means edges of a 3D mesh, $\mathcal{V}$ means vertices of a 3D mesh. This is the intersection between our 4D tetrahedral mesh and a hyper-plane $\tilde{P}$: $\tilde{a}x + \tilde{b}y + \tilde{c}z + \tilde{d}w + \tilde{e} = \tilde{f}(x, y, z, w) = 0$ in the implicit form. Then, $\mathcal{M} = Slice(\mathcal{T}, \tilde{P})$. Our method is similar to (Chu et al., 2009).

A tetrahedron in the 4D tetrahedral mesh has 4 faces, each of which is a subset of a hyperplane. Let $P_0 : a_0x + b_0y + c_0z + d_0w + e_0 = 0$ be such a hyperplane. This hyperplane may intersect with the slicing hyperplane $\tilde{P}$. For simplicity, we assume that none of the vertices of the tetrahedral mesh is on the hyperplane $\tilde{P}$, i.e., $\tilde{f}(v) \neq 0, \forall v \in V$. We will discuss how degenerative cases are handled later. There

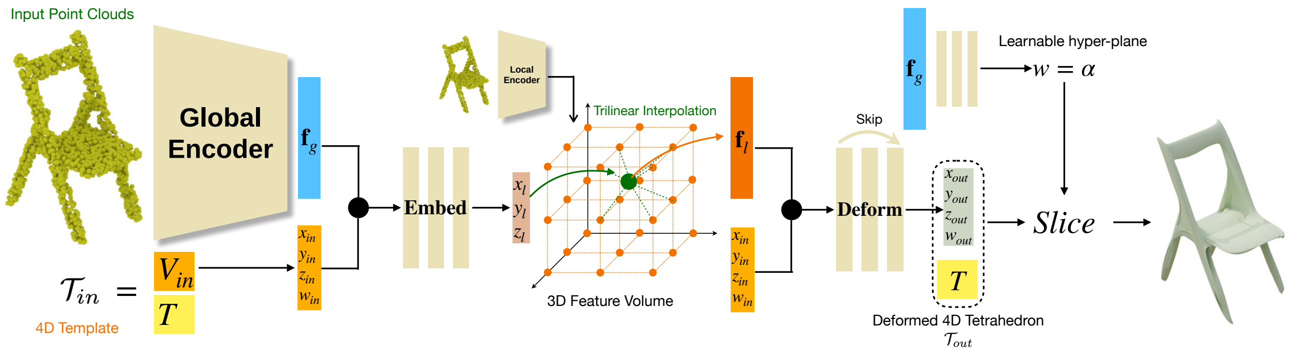


Figure 3. The pipeline of 3D mesh reconstruction via slicing a deformed 4D template tetrahedral mesh. To better predict suitable deformations, we employ both global and local features extracted from the input point cloud. To achieve this, based on PointNet, we build a Global Encoder that extracts the global feature $\mathbf{f}_g$ that characterizes the input point cloud as a whole. We also construct a Local Encoder to extract features at each grid point of a 3D Feature Volume. To extract local feature at each vertex of the 4D template mesh, we develop Embedding Net **Embed** to map 4D coordinates along with global feature $\mathbf{f}_g$ to 3D coordinates $[x_l, y_l, z_l]$ and use them for trilinear interpolation on the 3D Feature Volume to obtain the local feature $\mathbf{f}_l$. These are then used to deform 4D template $\mathcal{T}_{in} = (V_{in}, T)$ through Deformation Net **Deform**, to obtain deformed 4D tetrahedral mesh $\mathcal{T}_{out} = (V_{out}, T)$. The global feature $\mathbf{f}_g$ passes through a fully connected network to predict a hyperplane parameter α that determines the hyperplane $w = \alpha$ to slice our deformed 4D tetrahedral mesh, producing the output 3D mesh.

are only two possible situations where we need to compute points on $\tilde{P}$: Two vertices of the tetrahedron are on one side of $\tilde{P}$ and the other two on the other side, or one vertex of the tetrahedron is on one side of $\tilde{P}$ and the remaining three on the other side, as illustrated in Figure 2(left).

All intersection points between the tetrahedral mesh $\mathcal{T}$ and the hyperplane $\tilde{P}$ by intersecting an edge of $\mathcal{T}$ with $\tilde{P}$. For this to happen, the two end vertices of the edge, denoted as $p_0 = (x_0, y_0, z_0, w_0)$, $p_1 = (x_1, y_1, z_1, w_1)$ need to be on opposite sides of $\tilde{P}$, i.e., $\tilde{f}(p_0) > 0$, $\tilde{f}(p_1) < 0$ (or vice versa), as illustrated in Figure 2(right).

In practice, we only need to consider hyperplanes orthogonal to w-axis, i.e., $w = \alpha$. The intersection point $\hat{p} = (\hat{x}, \hat{y}, \hat{z}, \hat{w})$ can be calculated as

$$\begin{cases} \hat{x} = \dfrac{(\alpha - w_0)m}{q} + x_0, \\ \hat{y} = \dfrac{(\alpha - w_0)n}{q} + y_0, \\ \hat{z} = \dfrac{(\alpha - w_0)p}{q} + z_0, \\ \hat{w} = \alpha, \end{cases} \quad \begin{cases} m = x_0 - x_1, \\ n = y_0 - y_1, \\ p = z_0 - z_1, \\ q = w_0 - w_1, \end{cases} \tag{1}$$

After computing the intersection points, we join then up to form the triangle mesh $\mathcal{M}$. Details and pseudocode are in Appendix B.2 and B.3.

Constant-Rank Level Set Theorem (Theorem 5.12 in(Lee, 2013)) proves situations when a general level set is a submanifold of another manifold. As a specific case, for a 3-manifold M and a function $\Phi : \mathbb{R}^4 \to \mathbb{R}$ that projects 4D points to a scalar value. The theorem states that if the Jacobian matrix of Φ is of rank 1 for all points, then the level set of the scalar value forms a 2-manifold. The mapping may degenerate if the rank of the Jacobian matrix is 0 for certain points. In the discrete situation, we can guarantee no vertices of our 4D tetrahedral mesh are on the slicing hyperplane to avoid degeneration. This can be easily implemented by a very small offset to each vertex on the slicing hyperplane. In this case, we prove that the slice of a 4D tetrahedral mesh is guaranteed to be a manifold triangle mesh; see Appendix B.4 for the detailed proof.

3.3. NEURALSLICE Architecture

Given an input point cloud, our NEURALSLICE network needs to predict suitable deformation to the 4D template tetrahedral mesh $\mathcal{T}_{in} = (V_{in}, T)$ where V_{in} is the 4D template vertex positions, and T is a set of tetrahedra that determines the topology of the 4D template $\mathcal{T}_{in}$. The aim of our network is to predict a suitable deformed 4D tetrahedral mesh $\mathcal{T}_{out} = (V_{out}, T)$ with the same topology as the 4D template, but with vertex positions changed. We also need to predict a parameter α, which determines the slicing hyperplane $w = \alpha$. Finally, the reconstructed mesh is directly obtained $\mathcal{M} = Slice(\mathcal{T}_{out}, w = \alpha)$.

For better prediction, we extract both global and local geometric features from the input point cloud. Similar to (Groueix et al., 2018; Gupta & Chandraker, 2020), we use a PointNet (Qi et al., 2017) as the Global Encoder to extract the global feature $\mathbf{f}_g \in \mathbb{R}^{256}$, which captures the holistic characteristic of the input point cloud. However, using the global feature alone is not sufficient to accurately predict deformations of individual vertices. So we also utilize a Local Encoder as in (Peng et al., 2020), which outputs a 3D Feature Volume from the input point cloud. This encoder divides the input into 32^3 grids and uses PointNet (Qi et al., 2017) to encode feature at each grid point.

To predict the deformed position $v_{out} = (x_{out}, y_{out}, z_{out}, w_{out})$ for each vertex on the deformed 4D template, we design an MLP network (Deformation Net) that takes both the position of the template 4D mesh $v_{in} = (x_{in}, y_{in}, z_{in}, w_{in})$ and the local feature $\mathbf{f}_l$ related to it, i.e. $\mathcal{F}_{Deform} : (v_{in}, \mathbf{f}_l) \rightarrow v_{out}$. The former is independent of the input point cloud, so it is essential to augment it with local feature related to the input. However, $v_{in} \in \mathbb{R}^4$, while the Feature Volume is in the 3D space. To address this, we design another MLP network (Embedding Net) that takes the 4D vertex coordinates along with the global feature $\mathbf{f}_g$ and predicts the 3D embedding (x_l, y_l, z_l), i.e., $\mathcal{F}_{Embed} : (v_{in}, \mathbf{f}_g) \rightarrow (x_l, y_l, z_l)$. Again, v_{in} is independent of the input point cloud, so it is essential to incorporate the global feature for more informative embedding. Once the 3D embedding (x_l, y_l, z_l) is obtained, we use this to obtain $\mathbf{f}_l$ from the 3D Feature Volume through trilinear interpolation. To predict the slicing hyperplane, we simply pass the global feature $\mathbf{f}_g$ through a fully connected network to obtain α.

Without further constraints, when the 4D mesh is deformed, it is only constrained along the slicing hyperplane, so the Deformation Net can be under-constrained. So we further introduce Lipschitz normalization to constrain the movement of 4D vertices, and Laplacian regularization to encourage more regular triangulation.

Lipschitz normalization. Our Deformation Net can be seen as $\mathcal{F} : \mathbb{R}^4 \rightarrow \mathbb{R}^4$, i.e., deforming a 4D template to its new position. To avoid excessive deformations, we bound the deformation by a Lipschitz constant L:

$$\|\mathcal{F}(v_1) - \mathcal{F}(v_2)\| \leq L\|v_1 - v_2\| \tag{2}$$

where $v_1, v_2 \in \mathbb{R}^4$ are two 4D points. Intuitively, this inequality constrains the ratio of output change and input change, so that the deformation is bounded by some range. The upper bound of the Lipschitz constant L in an MLP with 1-Lipschitz activation function (e.g., ReLU) can be estimated by

$$L = \prod_i \|W_i\|_p \tag{3}$$

where $\|W_i\|_p$ is the p-norm of i-th layer MLP weight matrix, and the i-th layer MLP can be formulated as $y = \sigma(W_i x + b_i)$, where x and y are the input and output of the layer, and b_i is the i-layer bias. In practice, we use $p = \infty$, $\|W_i\|_\infty = \max_i \sum_j |w_{ij}|$ because it is efficient and easy to scale (Liu et al., 2022). To constrain the amount of deformation, we enforce the Lipschitz constant of the i-layer to be L_i (a hyper-parameter) and the upper bound of Lipschitz constant in the whole MLP is $L = \prod_i L_i$. To achieve this, we scale the weights of the i-th layer to $\hat{W}_i$:

$$\hat{W}_i = LipNorm(W_i, L_i) = W_i \times \min(1, \frac{L_i}{\|W_i\|_\infty}) \tag{4}$$

Loss Functions. During training, we use Chamfer distance $\mathcal{L}_{cd}$ to penalize the difference between the input point cloud S_I and the output point cloud S_O sampled from the output mesh $\mathcal{M}$. Following previous work, this loss is self-supervised, and does not require ground truth meshes:

$$\mathcal{L}_{cd} = \sum_{x \in S_I} \min_{y \in S_O} \|x - y\|_2^2 + \sum_{y \in S_O} \min_{x \in S_I} \|x - y\|_2^2 \tag{5}$$

To promote more regular triangulation, we further incorporate a Laplacian regularization (Alexa & Wardetzky, 2011):

$$\mathcal{L}_{lap} = \sum_{v_i \in \mathcal{M}} \| \sum_{v_j \in N(v_i)} \frac{1}{d(v_i)} (v_i - v_j) \|_2 \tag{6}$$

where $N(v_i)$ is the 1-ring neighborhood of v_i, $d(v_i)$ is the degree of v_i. We train the overall 3D Reconstruction network with Lipschitz normalization and loss function $\mathcal{L} = \mathcal{L}_{cd} + \lambda_{lap}\mathcal{L}_{lap}$. With Lipschitz normalization and Laplacian regularization, the deformation network can output coarse but regular triangle meshes.

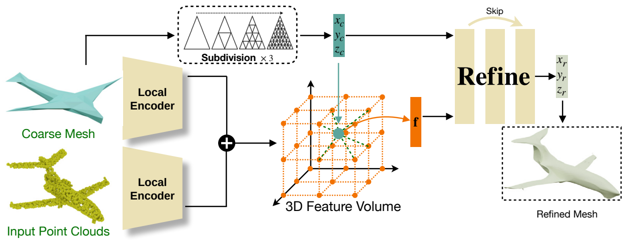


Figure 4. Pipeline of the subdivision-based refinement. We employ subdivision and a refine deformation. (x_c, y_c, z_c) are vertices on the coarse mesh after subdivision. $\mathbf{f}_{refine}$ is the sum of coarse mesh features and input point cloud features. The refinement then predicts the refined output mesh with vertices (x_r, y_r, z_r) as the output vertices in the final mesh.

3.4. Subdivision-based Refinement

With $LipNorm$ and $\mathcal{L}_{lap}$, the sliced meshes tend to be simpler and coarser. To obtain more detailed meshes, we first employ 1-to-4 subdivision three times (with vertices added directly at the edge midpoints), and then put them into a refinement network to refine coarse meshes. In refinement, we employ two encoders. One is to encode features from the coarse mesh, and the other is to encode features from the input point cloud, both of them are the same as the Local Encoder in the reconstruction network. Then we add the two feature volumes together as the final feature volume $\mathbf{f}_{refine}$ for the Refinement Net. The input to Refinement Net is the feature volume $\mathbf{f}_{refine}$ and vertices from the coarse mesh after 3 times subdivision (x_c, y_c, z_c). The output of Refinement Net is the refined vertices in the subdivided mesh $\mathcal{F}_{Refine} : \mathbf{f}_{refine} \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$. It can deform the subdivided coarse mesh to a fine mesh. We also employ loss function as $\mathcal{L} = \mathcal{L}_{cd} + \lambda_{lap}\mathcal{L}_{lap}$ to train the Refinement Net.

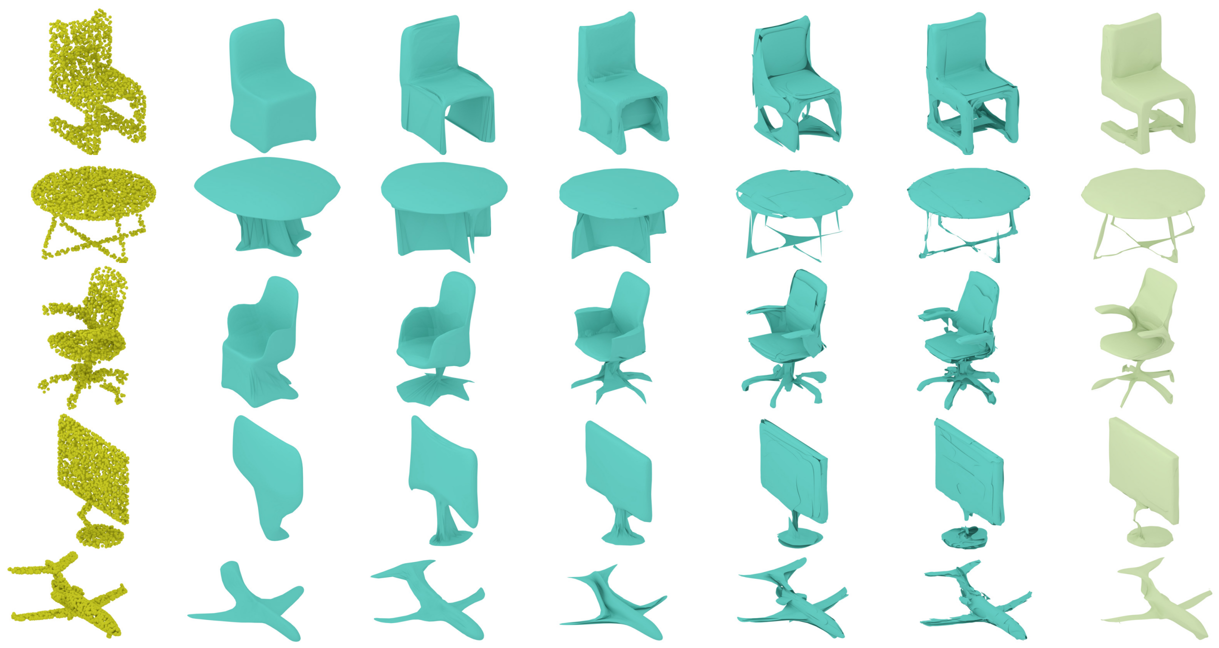


Figure 5. 3D reconstruction results of our method compared with explicit methods. Our method has advantages in both topology and accuracy. Patch-based methods (MNA-25, Atlas-25) may cause cracks. Sphere-based methods cannot represent different topologies.

4. Experiments

After describing the experiment settings, we first evaluate our method on reconstructing 3D meshes from point clouds, and compare it to state-of-the-art methods. We then conduct ablation studies to verify the effectiveness of each design choice. Finally, we analyze the limitations and failure cases of our method.

Our method is implemented using PyTorch (Paszke et al., 2019). We use the Adam optimizer (Kingma & Ba, 2015) to train our NEURALSLICE without the refinement net for 150 epochs with batch size of 32 and learning rate of 10^{-3}. We then train NEURALSLICE with the refinement net for 150 epochs with batch size of 16 and learning rate of 10^{-4}. For hyper-parameters, we set $\lambda_{lap} = 0.1$, $L_i = 1.12$ $(\forall i)$ empirically. Our experiments were carried out on a computer with an Intel 10700 CPU and an RTX 3090 GPU. We use the ShapeNet (Chang et al., 2015) dataset for training and evaluation, utilizing the 13 categories and the same dataset split as in (Gupta & Chandraker, 2020).

4.1. Surface Reconstruction from Point Clouds

In this experiment, we conduct surface reconstruction from point clouds. The input to this experiment is a point cloud with 2500 points, and the output is a 3D mesh. We compared NEURALSLICE with explicit methods and implicit methods separately. We compare with the following explicit baselines: NFM-3D, NFM-6D (Gupta & Chandraker, 2020), Atlas-O, Atlas-25(Groueix et al., 2018), TearingNet (Pang et al., 2021), DSP-3, DSP-25(Bednarik et al., 2020), MNA-3, MNA-25 (Low & Lee, 2022); and implicit baselines: Occ(Mescheder et al., 2019), ConvOcc (Peng et al., 2020). All meshes and input point clouds are scaled into $[-1, 1]^3$. To evaluate the quality of reconstructed meshes, we sample $25,000$ points on the output mesh and calculate the Chamfer distance between it and the ground truth point cloud with the same number of points.

Comparison with explicit methods. Our method outperforms state-of-the-art explicit methods in both topology and accuracy. Specifically, NFM and Atlas-O only deform one template, which causes the generated 3D meshes to be homeomorphism to a sphere. Although patch-based methods such as AtlasNet (Groueix et al., 2018) and MNA (Low & Lee, 2022) can represent different topologies, there is no connection between the 25 patches, making the final mesh contain many cracks. MNA uses an occupancy network to modify the topology according to the occupancy values, which may result in irregular patches. Moreover, the meshes generated by patch-based methods contain many patches and are not watertight.

Comparison with implicit methods. We compare with implicit methods ConvOcc (Peng et al., 2020) and Occ (Park et al., 2019). Following the existing protocol, we add a Gaussian noise with zero mean and 0.005 standard deviation for points in all input clouds, which tests the robustness of methods w.r.t. noise. As shown in Figure 6 and Table 2, NEURALSLICE outperforms ConvOcc (Peng et al., 2020) and Occ (Park et al., 2019) both qualitatively and quantita-

Table 1. 3D reconstruction results comparing our method with explicit methods. The performance of our method on Chamfer distance is better than others. '#P/T' represents the number of patches (for patch-based methods) or templates. 'Top.' means whether the method can represent different topologies. NEURALSLICE outperforms other explicit methods on Chamfer distance by using only 1 template.

Methods	NMF-3D	NMF-6D	Atlas-O	TearingNet	MNA-3	DSP-3	DSP-25	Atlas-25	MNA-25	NEURALSLICE
#P/T	**1**	**1**	**1**	**1**	3	3	25	25	25	**1**
Top.	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
CD	4.253	2.443	1.483	2.001	0.665	1.198	0.861	0.743	0.671	**0.537**

Table 2. 3D reconstruction results comparing our method with implicit methods in Chamfer distance $\times 10^3$. Input point clouds are with added Gaussian noise, where each vertex coordinate is added with an offset randomly chosen from a Gaussian with zero mean and 0.005 standard deviations. Our method outperforms Occ and ConvOcc in Chamfer distance for certain categories and overall on average. Our method is more efficient than implicit methods as our method does not require dense signed distance prediction and iso-surface extraction.

Method	plane	bench	cabinet	car	chair	display	lamp	speaker	rifle	sofa	table	phone	vessel	mean	Time↓
ConvOcc	**0.233**	**0.270**	0.824	1.968	**0.435**	**0.323**	6.263	0.984	**0.172**	**0.433**	**0.411**	**0.188**	**0.459**	0.891	0.261
Occ	0.736	0.710	1.370	2.599	1.329	1.011	5.246	3.032	0.760	1.037	1.310	0.452	1.571	1.630	0.316
NEURALSLICE	0.288	0.500	**0.522**	**0.684**	0.653	0.453	**0.906**	**0.848**	0.230	0.481	0.510	0.270	0.518	**0.538**	**0.025**

tively in Chamfer distance. As can be seen, implicit methods have difficulties handling very thin structures and may produce thicker or even broken outputs, whereas our method effectively reconstructs the proper shapes. Such topological incorrectness is picked up in the Chamfer distance measure, as a missing structure can add a large Chamfer distance to relevant points. Another significant advantage of our method compared with implicit methods is efficiency because implicit methods need to sample dense points and extract the iso-surface. Both stages are time-consuming. In Table 2, With a similar network parameter size, NEURALSLICE is about 10× faster than implicit method ConvOcc.

Compared with the performance of NEURALSLICE without added noise, as reported in Table 1 to be consistent with the testing protocol as used in existing explicit methods, the performance of our method is stable, demonstrating its robustness to noisy inputs.

Overall, our method can explicitly adapt to the topology with manifold meshes using a single template. The reconstruction of NEURALSLICE outperforms state-of-the-art template-based methods and implicit methods in Chamfer distance. NEURALSLICE can also handle thin structures that implicit methods fail and are more efficient than implicit surface representations.

4.2. Ablation Study

We conduct an ablation study of our design choices. Specifically, we evaluate the necessity of $\mathcal{L}_{lap}$, $LipNorm$, and subdivision-based refinement. We use Chamfer distance and normal consistency (Guillard et al., 2022) for measuring accuracy and smoothness respectively. The normal consistency from (Guillard et al., 2022) uses unsigned cosine-similarity between the normals of pairs of the closest point to evaluate NC for a mesh.

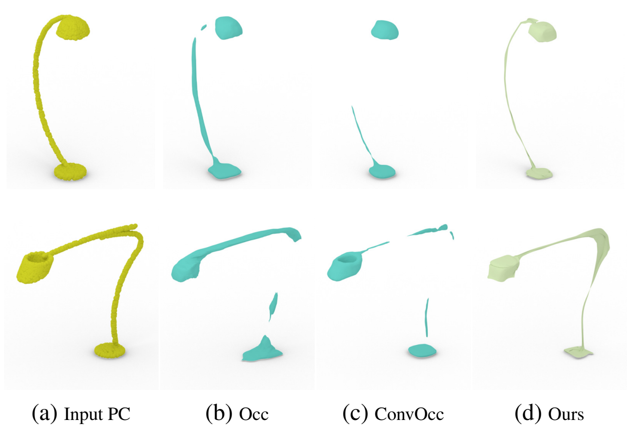


Figure 6. 3D reconstruction results comparing our method with implicit methods. Implicit methods need occupancy or signed distance values to train the networks, and it is hard for them to fit very thin surfaces: thin structures may be reconstructed too thick (Occ method) or lost entirely (ConvOcc method).

Necessity of $\mathcal{L}_{lap}$. We first verify the usefulness of $\mathcal{L}_{lap}$ for mesh quality. $\mathcal{L}_{lap}$ is a regularization loss acting on output meshes which guides the deformation to learn to produce smooth meshes. As shown in Figure 7 and Table 3. Without $\mathcal{L}_{lap}$, NEURALSLICE is difficult to reconstruct regular triangulation, with poor normal consistency. As shown in Figure 7, although without L_{lap}, the reconstructed meshes have lower CD, and the visual quality is much worse with irregular triangles and visual artifacts.

Necessity of $LipNorm$. $LipNorm$ is a normalization affecting network parameters to force the deformation to be limited within some range. It makes the optimizer more easily learn a proper deformation to reconstruction meshes and achieve higher performance. Meanwhile, it can avoid visual artifacts on meshes due to extreme deformations as shown in Figure 7.

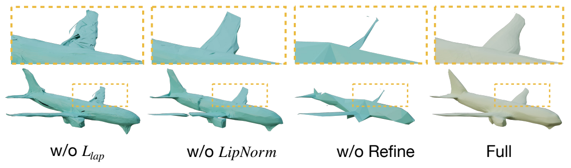


Figure 7. Ablation of NEURALSLICE. From left to right: without Laplacian regularization, without Lipschitz normalization, without refinement and our full model.

Necessity of Refine. Subdivision-based refinement is a very important block. With only $\mathcal{L}_{lap}$ and $LipNorm$. The final meshes are very coarse and simple. This is because $\mathcal{L}_{lap}$ and $LipNorm$ guide the deformation net to learn to have small, constrained deformations. The refinement step is effective to obtain high-quality, smooth meshes.

Table 3. Ablation study regarding regularization and refinement. Although adding these may lead to a somewhat higher Chamfer distance, the reconstructed mesh becomes smoother with better Normal Consistency (NC), and more regular triangles. L_{lap} is necessary for regular triangulation and $LipNorm$ improves the performance. Without refinement, both CD and NC become much worse.

Variant	w/o Refine	w/o $\mathcal{L}_{lap}$	w/o $LipNorm$	Full
CD$\times 10^3 \downarrow$	1.819	**0.449**	0.784	0.538
NC $\uparrow$	0.206	0.193	0.451	**0.602**
Regular Surf	✔	✗	✔	✔

4.3. Representation Power

NEURALSLICE can represent different 3D shapes and topologies in one 4D tetrahedral mesh. We can learn a Deformation Net and learnable hyper-planes with fixed feature volume. It means we have a fixed deformed 4D tetrahedral mesh and different 3D meshes from different slices. In DMTet (Shen et al., 2021), one tetrahedron grid only represents a single 3D mesh, as shown in Figure 8.

Another intriguing feature of our method is its ability to utilize the same 4D tetrahedral mesh with varying hyperplanes to represent a deformable shape sequence—a capability that is not possible with DMTet (Shen et al., 2021). To showcase this ability, we conduct an experiment on sphere deformation, as depicted in Figure 9. The experiment inputs a starting shape and an ending shape, and we employ NEURALSLICE to construct a deformation sequence. Additionally, we perform a more realistic deformation on an animal, as illustrated in Figure 9. For this experiment, we input three shapes representing the start, middle and end positions, and fit the entire deformation using a single 4D tetrahedral mesh.

4.4. Limitations and Future Work

One limitation of NEURALSLICE is that for some challenging input point clouds, it may not be able to recover all the

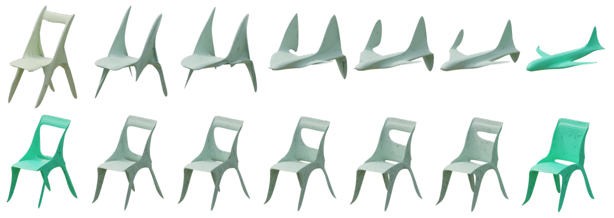


Figure 8. 3D meshes in each row are sliced from the same 4D tetrahedral mesh, they have different shapes and topologies.

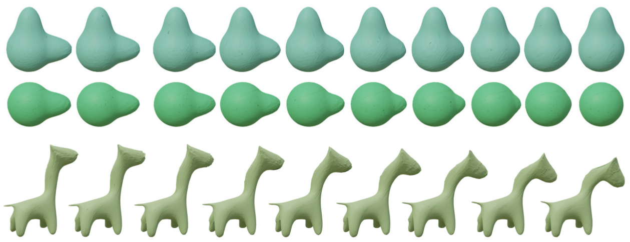


Figure 9. NEURALSLICE is capable of fitting sphere deformation sequences and animal deformation sequences.

detailed structures. This is because it is difficult to learn details in a pure explicit method without the supervision signal of SDF. We show one failure case in Appendix C.

The 4D tetrahedral mesh is consistent with recent 4D games (Bosch, 2023a;b). However, in these games, the shapes are built by hand, which are quite simple. As a future work, we would like to extend our method to generate 4D shapes for 4D games.

5. Conclusion

This paper presents a novel representation called NEURALSLICE for 3D mesh reconstruction. The central concept revolves around deforming a 4D tetrahedral mesh and subsequently slicing it to obtain a 3D mesh. In comparison to existing explicit representations, NEURALSLICE demonstrates superior reconstruction accuracy and offers flexible topology. Moreover, our method exhibits higher efficiency compared to implicit methods and greater robustness for thin structures. Notably, NEURALSLICE has the capability to represent various shapes and topologies within a single tetrahedral mesh.

Acknowledgement This work was supported by grants from the National Natural Science Foundation of China (No. 62061136007), the Beijing Municipal Natural Science Foundation for Distinguished Young Scholars (No. JQ21013), and the Royal Society Newton Advanced Fellowship (No. NAF\R2\192151). We thank Tong Wu for his valuable assistance with illustration. Additionally, we extend our appreciation to Bo Yang and Tianyi Sang for their valuable advice on this project.

References

Abbott, E. *Flatland: a romance of many dimensions*. Domaine public, 1884.

Alexa, M. and Wardetzky, M. Discrete Laplacians on general polygonal meshes. *ACM Transactions on Graphics (TOG)*, pp. 1–10, 2011.

Atzmon, M. and Lipman, Y. SAL: Sign agnostic learning of shapes from raw data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2565–2574, 2020a.

Atzmon, M. and Lipman, Y. SALD: Sign agnostic learning with derivatives. In *International Conference on Learning Representations (ICLR)*, 2020b.

Badki, A., Gallo, O., Kautz, J., and Sen, P. Meshlet priors for 3D mesh reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2849–2858, 2020.

Bednarik, J., Parashar, S., Gundogdu, E., Salzmann, M., and Fua, P. Shape reconstruction by learning differentiable surface representations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4716–4725, 2020.

Berger, M. and Gostiaux, B. *Differential Geometry: Manifolds, Curves, and Surfaces: Manifolds, Curves, and Surfaces*, volume 115. Springer Science & Business Media, 2012.

Bhaniramka, P., Wenger, R., and Crawfis, R. Isosurfacing in higher dimensions. *Proceedings of the IEEE Visualization Conference*, 05 2000. doi: 10.1109/VISUAL.2000.885704.

Bosch, M. T. N-dimensional rigid body dynamics. *ACM Transactions on Graphics (TOG)*, 39(4):55:1–55:6, jul 2020. doi: 10.1145/3386569.3392483. URL `http://dx.doi.org/10.1145/3386569.3392483`.

Bosch, M. T. 4D Toys. *An interactive toy for 4D children. [WWW Document]. URL* `https://4dtoys.com`, 2023a.

Bosch, M. T. Miegakure. *A Puzzle-Platformer in Four Dimensions. [WWW Document]. URL* `https://miegakure.com`, 2023b.

Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., and Lévy, B. *Polygon mesh processing*. CRC press, 2010.

Boulch, A., Langlois, P.-A., Puy, G., and Marlet, R. NeeDrop: Self-supervised shape representation from sparse point clouds using needle dropping. In *2021 International Conference on 3D Vision (3DV)*, pp. 940–950. IEEE, 2021.

Boyd, S., Boyd, S. P., and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

Chang, A. X., Funkhouser, T. A., Guibas, L. J., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., S. Song, H. S., Xiao, J., Yi, L., and Yu., F. ShapeNet: An information-rich 3D model repository. *arXiv.org, 1512.03012*, 2015.

Chen, Z. and Zhang, H. Learning implicit fields for generative shape modeling. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Chen, Z., Tagliasacchi, A., and Zhang, H. BSP-Net: Generating compact meshes via binary space partitioning. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Chibane, J., Pons-Moll, G., et al. Neural unsigned distance fields for implicit function learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:21638–21652, 2020.

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 *European Conference on Computer Vision (ECCV)*, 2016.

Chu, A., Fu, C.-W., Hanson, A., and Heng, P.-A. GL4D: A GPU-based architecture for interactive 4D visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1587–1594, 2009. doi: 10.1109/TVCG.2009.147.

Deng, B., Genova, K., Yazdani, S., Bouaziz, S., Hinton, G., and Tagliasacchi, A. CvxNet: Learnable convex decomposition. June 2020a.

Deng, Z., Bednařík, J., Salzmann, M., and Fua, P. Better patch stitching for parametric surface reconstruction. In *2020 International Conference on 3D Vision (3DV)*, pp. 593–602. IEEE, 2020b.

Emmer, M. and Banchoff, T. F. Beyond the third dimension: Geometry, computer graphics, and higher dimensions. *Computers & Graphics*, 25(3):385, 1990.

Fan, H., Hao, S., and Guibas, L. A point set generation network for 3D object reconstruction from a single image. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Gao, J., Chen, W., Xiang, T., Tsang, C. F., Jacobson, A., McGuire, M., and Fidler, S. Learning deformable tetrahedral meshes for 3D reconstruction. In *Advances In Neural Information Processing Systems (NeurIPS)*, 2020.

Gao, L., Yang, J., Wu, T., Yuan, Y.-J., Fu, H., Lai, Y.-K., and Zhang, H. SDM-NET: Deep generative network for structured deformable mesh. *ACM Transactions on Graphics (TOG)*, 38(6):243:1–243:15, 2019.

Gkioxari, G., Malik, J., and Johnson, J. Mesh R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 9785–9795, 2019.

Groueix, T., Fisher, M., Kim, V. G., Russell, B., and Aubry, M. AtlasNet: A papier-mâché approach to learning 3D surface generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Guillard, B., Stella, F., and Fua, P. MeshUDF: Fast and differentiable meshing of unsigned distance field networks. In *European Conference on Computer Vision (ECCV)*, 2022.

Gupta, K. and Chandraker, M. Neural mesh flow: 3D manifold mesh generation via diffeomorphic flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Hilbert, D. and Cohn-Vossen, S. Anschauliche geometrie. *Mathematical Gazette*, 36(317), 1932.

Hui, Z. and Hanson, A. J. Physically interacting with four dimensions. In *Advances in Visual Computing, Second International Symposium, ISVC 2006, Lake Tahoe, NV, USA, November 6-8, 2006 Proceedings, Part I*, 2006.

Jiang, C., Huang, J., Tagliasacchi, A., and Guibas, L. J. ShapeFlow: Learnable deformation flows among 3D shapes. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:9745–9757, 2020.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *Proceedings of International Conference on Machine Learning (Proceedings of International Conference on Machine Learning (ICML))*, 2015.

Lee, J. *Introduction to topological manifolds*, volume 202. Springer Science & Business Media, 2010.

Lee, J. M. Submanifolds. In *Introduction to smooth manifolds*, pp. 98–124. Springer, 2013.

Li, J., Xu, K., Chaudhuri, S., Yumer, E., Zhang, H., and Guibas, L. GRASS: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

Liu, H. and Zhang, H. A flip-book of knot diagrams for visualizing surfaces in 4-space. In *Computer Graphics Forum*, volume 41, pp. 345–354. Wiley Online Library, 2022.

Liu, H.-T. D., Williams, F., Jacobson, A., Fidler, S., and Litany, O. Learning smooth neural functions via Lipschitz regularization. *Special Interest Group for Computer Graphics (SIGGRAPH)*, 2022.

Liu, S.-L., Guo, H.-X., Pan, H., Wang, P.-S., Tong, X., and Liu, Y. Deep implicit moving least-squares functions for 3d reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, May 2021.

Lorensen, W. E. and Cline, H. E. Marching cubes: A high resolution 3D surface construction algorithm. *ACM Transactions on Graphics (TOG)*, pp. 163–169, 1987.

Low, W. F. and Lee, G. H. Minimal neural atlas: Parameterizing complex surfaces with minimal charts and distortion. In *European Conference on Computer Vision (ECCV)*, 2022.

Ma, B., Han, Z., Liu, Y.-S., and Zwicker, M. Neural-pull: Learning signed distance functions from point clouds by learning to pull space onto surfaces. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139, 2021.

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. Occupancy networks: Learning 3D reconstruction in function space. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Mildenhall, B., Srinivasan, P., Tancik, M., Barron, J., Ramamoorthi, R., and Ng, R. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision (ECCV)*, 2020.

Mo, K., Guerrero, P., Yi, L., Su, H., Wonka, P., Mitra, N. J., and Guibas, L. J. StructureNet: hierarchical graph networks for 3D shape generation. *ACM Transactions on Graphics (TOG)*, 38(6):1–19, 2019.

Morreale, L., Aigerman, N., Kim, V. G., and Mitra, N. J. Neural surface maps. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4639–4648, 2021.

Mount, D. M. Cmsc 754 computational geometry. *Lecture Notes, University of Maryland*, pp. 1–122, 2002.

Oliver, N. *Calculus in Hyperspace*. Lecture Notes, Harvard University, MATH 22A, Unit 40, 2018 Fall.

Pan, J., Han, X., Chen, W., Tang, J., and Jia, K. Deep mesh reconstruction from single RGB images via topology modification networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9964–9973, 2019.

Pang, J., Li, D., and Tian, D. TearingNet: Point cloud autoencoder to learn topology-friendly representations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7453–7462, 2021.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. Deepsdf: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Park, K., Sinha, U., Hedman, P., Barron, J. T., Bouaziz, S., Goldman, D. B., Martin-Brualla, R., and Seitz, S. M. HyperNeRF: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Transactions on Graphics (TOG)*, 40(6), dec 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Peng, S., Niemeyer, M., Mescheder, L., Pollefeys, M., and Geiger, A. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, 2020.

Prokudin, S., Lassner, C., and Romero, J. Efficient learning on point clouds with basis point sets. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. PointNet: Deep learning on point sets for 3D classification and segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Ricky, T. Q. C., Yulia, R., Jesse, B., and David, D. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Shen, T., Gao, J., Yin, K., Liu, M.-Y., and Fidler, S. Deep marching tetrahedra: a hybrid representation for high-resolution 3D shape synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Tang, J.-H., Chen, W., Yang, J., Wang, B., Liu, S., Yang, B., and Gao, L. OctField: Hierarchical implicit functions for 3D modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Wang, N., Zhang, Y., Li, Z., Fu, Y., and Jiang, Y. G. Pixel2Mesh: 3D mesh model generation via image guided deformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE TAPMI)*, PP(99):1–1, 2020.

Wang, P.-S., Liu, Y., and Tong, X. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Transactions on Graphics (TOG)*, 41(4), 2022.

Wang, W., Ceylan, D., Mech, R., and Neumann, U. 3DN: 3D deformation network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1038–1046, 2019.

Wen, C., Zhang, Y., Li, Z., and Fu, Y. Pixel2Mesh++: Multi-view 3d mesh generation via deformation. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

Williams, F., Schneider, T., Silva, C., Zorin, D., Bruna, J., and Panozzo, D. Deep geometric prior for surface reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10130–10139, 2019.

Wu, J., Zhang, C., Xue, T., Freeman, W. T., and Tenenbaum, J. B. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Xiao, Y., Lai, Y., Zhang, F., Li, C., and Gao, L. A survey on deep geometry learning: From a representation perspective. *Comput. Vis. Media*, 6(2):113–133, 2020. doi: 10.1007/s41095-020-0174-8. URL https://doi.org/10.1007/s41095-020-0174-8.

Yan, X., Fu, C. W., and Hanson, A. J. Multitouching the fourth dimension. *Computer*, 45(9):80–88, 2012.

Yang, J., Mo, K., Lai, Y.-K., Guibas, L. J., and Gao, L. DSG-Net: Learning disentangled structure and geometry for 3D shape generation. *ACM Transactions on Graphics (TOG)*, 2022.

Yang, Y., Feng, C., Shen, Y., and Tian, D. FoldingNet: Point cloud auto-encoder via deep grid deformation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 206–215, 2018.

Appendix

A. Overview

We first introduce methods to construct the 4D tetrahedral template mesh and perform analysis for different templates. Then we provide details for the "slice" operation in 4D, including the formulae and code. After that, we prove our sliced mesh is guaranteed to be a manifold in 3D. Finally, we show more results for surface reconstruction using our method.

B. 4D Tetrahedral Mesh

This section will introduce constructing 4D templates, design choices of different 4D templates, details about slicing a 4D object, and proves our sliced mesh is guaranteed to be a manifold in 3D.

B.1. Analysis of Different 4D Templates

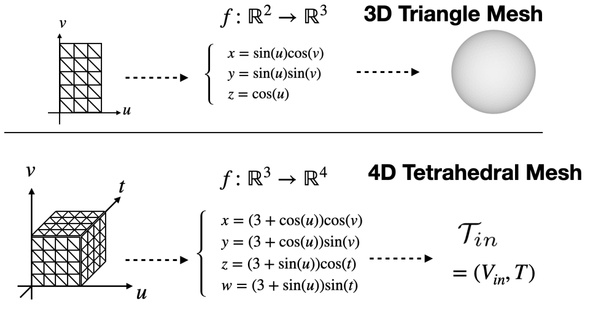

$f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$
3D Triangle Mesh
v
u
$x = \sin(u)\cos(v)$
$y = \sin(u)\sin(v)$
$z = \cos(u)$
$f : \mathbb{R}^3 \rightarrow \mathbb{R}^4$
4D Tetrahedral Mesh
v
t
u
$x = (3 + \cos(u))\cos(v)$
$y = (3 + \cos(u))\sin(v)$
$z = (3 + \sin(u))\cos(t)$
$w = (3 + \sin(u))\sin(t)$
$\mathcal{T}_{in} = (V_{in}, T)$

Figure 10. Method to construct 4D tetrahedral mesh.

Constructing 4D Tetrahedral Mesh. Similar to constructing 3D meshes with parametric equations, we can also generate 4D templates with parametric equations. The process of constructing 3D meshes can be seen as mapping a 2D grid to a 3D space through parametric equations. Similarly, constructing a 4D template can be seen as mapping a 3D tetrahedral grid to a 4D space through parametric equations. As shown in Figure 10. We analyze 3 different 4D tetrahedral meshes ($\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$) as the template for surface reconstruction.

Here are the parametric equations of $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ from (Oliver, 2018 Fall). All of these map 3D space $u \times v \times t$ to 4D space $x \times y \times z \times w$. $f : \mathbb{R}^3 \rightarrow \mathbb{R}^4$.

$\mathcal{T}_1$:

$$\begin{cases} x = \cos(u)\cos(v)\cos(t), \\ y = \cos(u)\cos(v)\sin(t), \\ z = \cos(u)\sin(v), \\ w = \sin(u), \end{cases} \tag{7}$$

$\mathcal{T}_2$:

$$\begin{cases} x = (5 + (2 + 2\cos(t))\cos(v))\cos(u), \\ y = (5 + (2 + 2\cos(t))\cos(v))\sin(u), \\ z = (2 + 2\cos(t))\sin(v), \\ w = 5\sin(t), \end{cases} \tag{8}$$

$\mathcal{T}_3$:

$$\begin{cases} x = (3 + \cos(u))\cos(v), \\ y = (3 + \cos(u))\sin(v), \\ z = (3 + \sin(u))\cos(t), \\ w = (3 + \sin(u))\sin(t), \end{cases} \tag{9}$$

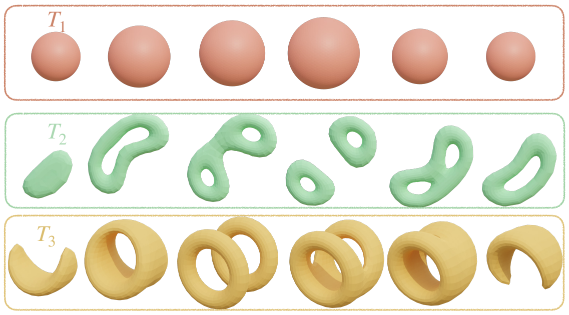


Figure 11. Different slices for 3 different 4D templates. We visualize these 4D templates by using different hyper-planes to slice them to obtain different 3D meshes. As can be seen, this may result in different topologies in different slices.

As 4D templates can be hard to imagine, we visualize these three templates by slicing these templates by different hyper-planes. As shown in Figure 11. $\mathcal{T}_1$ has the same topology in different slices and $\mathcal{T}_2, \mathcal{T}_3$ have different topologies in different slices.

We employ these three templates for surface reconstruction. Visually, there is little difference in the results of the three template reconstructions. Quantitatively, $\mathcal{T}_2$ has a better (smaller) Chamfer distance and $\mathcal{T}_1$ is smoother. As shown in Figure 12 and Table 4. In the main paper, we use $\mathcal{T}_2$ as our 4D template.

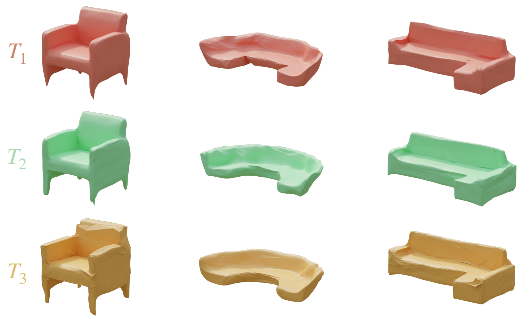


Figure 12. NEURALSLICE reconstruction results with different 4D templates. The accuracy of $\mathcal{T}_3$ is slightly worse, and the overall difference is minor.

CD$\times 10^3 \downarrow$	NC $\uparrow$	Variant
0.578	**0.723**	NEURALSLICE-$\mathcal{T}_1$
0.593	0.614	NEURALSLICE-$\mathcal{T}_3$
0.538	0.602	NEURALSLICE-$\mathcal{T}_2$

Table 4. Analysis of different templates. $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ are different 3-manifolds embedded in $\mathbb{R}^4$. $\mathcal{T}_2$ is better in Chamfer distance and $\mathcal{T}_1$ can reconstruct smoother meshes.

B.2. Slicing 4D Objects

All intersection points between the tetrahedral mesh $\mathcal{T}$ and the hyperplane $\tilde{P}$ can be obtained by intersecting an edge of $\mathcal{T}$ with $\tilde{P}$. For this to happen, the two end vertices of the edge, denoted as $p_0 = (x_0, y_0, z_0, w_0)$, $p_1 = (x_1, y_1, z_1, w_1)$ need to be on opposite sides of $\tilde{P}$, i.e., $\tilde{f}(p_0) > 0, \tilde{f}(p_1) < 0$ (or vice versa), as illustrated in Figure 2 in the main paper (right). The line that passes through p_0 and p_1 is:

$$\begin{cases} x = x_0 + mt, \\ y = y_0 + nt, \\ z = z_0 + pt, \\ w = w_0 + qt, \end{cases} \quad \begin{cases} m = x_0 - x_1, \\ n = y_0 - y_1, \\ p = z_0 - z_1, \\ q = w_0 - w_1, \end{cases} \tag{10}$$

Eliminate t:

$$\frac{x - x_0}{m} = \frac{y - y_0}{n} = \frac{z - z_0}{p} = \frac{w - w_0}{q} \tag{11}$$

Simultaneous equations:

$$\begin{cases} \frac{x - x_0}{m} = \frac{y - y_0}{n} = \frac{z - z_0}{p} = \frac{w - w_0}{q} \\ ax + by + cz + dw + e = 0, \end{cases} \tag{12}$$

Let $w = \alpha$, the solution is :

$$\begin{cases} \hat{x} = \frac{(\alpha - w_0)m}{q} + x_0, \\ \hat{y} = \frac{(\alpha - w_0)n}{q} + y_0, \\ \hat{z} = \frac{(\alpha - w_0)p}{q} + z_0, \\ \hat{w} = \alpha, \end{cases} \tag{13}$$

B.3. Code

Below is the code written in PyTorch(Paszke et al., 2019) style.

```
def slice(vertices, alpha):
  #input vertices's shape is [N,6,2,4]
  #N represents there are N tetrahedra
  #in this tetrahedral mesh.
  #Each tetrahedron has 6 edges,
  #each edge has 2 endpoints,
  #each endpoints are in R^4.
  #It has some redundancy,
  #but easy to implement.
  #alpha represents slice hyper-plane
  #w=alpha.

  #for each edge, if edge<0,
  #this edge intersect hyper-plane
  edge = (vertices[:,:,0,3]-alpha)*
         (vertices[:,:,1,3]-alpha)

  #the index of the tetrahedron
  #intersecting the hyperplane
  idx = torch.nonzero(edge<0)

  # x0,y0,z0,w0,m,n,p,q in equation 10
  m = vertices[idx[:,0],idx[:,1],0,0]-
      vertices[idx[:,0],idx[:,1],1,0]
  n = vertices[idx[:,0],idx[:,1],0,1]-
      vertices[idx[:,0],idx[:,1],1,1]
  p = vertices[idx[:,0],idx[:,1],0,2]-
      vertices[idx[:,0],idx[:,1],1,2]
  q = vertices[idx[:,0],idx[:,1],0,3]-
      vertices[idx[:,0],idx[:,1],1,3]

  x0 = vertices[idx[:,0],idx[:,1],0,0]
  y0 = vertices[idx[:,0],idx[:,1],0,1]
  z0 = vertices[idx[:,0],idx[:,1],0,2]
  w0 = vertices[idx[:,0],idx[:,1],0,3]

  # Sliced 3d points x,y,z in equation 13
  x = (alpha-w0)*m/q + x0
  y = (alpha-w0)*n/q + y0
  z = (alpha-w0)*p/q + z0

  x = x.view(-1,1)
  y = y.view(-1,1)
  z = z.view(-1,1)

  point_3d = torch.cat(
         (x,y,z),dim=1
         ).view(-1,3)

  #for each sliced edges,
  #give a serial number
  ll = torch.arange(len(idx))+1
  edge0 = torch.zeros(
          tetra.size()[1],6
          ).to(device).long()
  edge0[idx[:,0],idx[:,1]] = ll

  #How many edges are sliced
  #for each tetrahedron
  zero = torch.zeros_like(edge).to(device)
  one = torch.ones_like(edge).to(device)
  edgenum = torch.where(
            edge0 > 0, one, zero)
  edgenum = torch.sum(edgenum,dim = 1)

  #idx_edge3 represents the
  #index of the triangle that intersects
  #the tetrahedron and hyperplane
  idx_edge3 = torch.nonzero(edgenum==3)

  #idx_edge4 represents the index of the
  #quadrilateral that intersects
  #the tetrahedron and hyperplane
  idx_edge4 = torch.nonzero(edgenum==4)

  #compute edge index for
  #tetrahedron sliced 3 edges
  edge3 = edge0[idx_edge3,:].view(-1,6)
  idx_3 = torch.nonzero(edge3!=0)

  #compute edge index for
  #tetrahedron sliced 3 edges
  edge4 = edge0[idx_edge4,:].view(-1,6)
  idx_4 = torch.nonzero(edge4!=0)

  #face3 represents
  #the triangle that intersects
  # the tetrahedron and hyperplane
  face3 = edge3[idx_3[:,0],idx_3[:,1]]
          .view(idx_edge3.size()[0],3)
```

```
    #face4 represents
    #the quadrilateral that intersects
    #the tetrahedron and hyperplane
    face4 = edge4[idx_4[:,0],idx_4[:,1]]
            .view(idx_edge4.size()[0],4)
    # we need to triangulate
    #quadrilateral faces
    face4 = triangulation(face4, vertices)

    face = torch.cat((face3,face4),dim=0)

    return point_3d, face
```

B.4. Manifold Mesh

We also prove that the slice of a 4D tetrahedral mesh is guaranteed to be a manifold mesh (based on the way the 4D template is constructed) and we can control whether the 3D triangle mesh is watertight or not via choosing different 4D tetrahedral meshes. We first follow (Botsch et al., 2010)'s definition of manifold triangle meshes. Then we make two reasonable assumptions and prove our theorem.

Definition B.1. A 3D triangle mesh is a manifold if:

- Each edge is adjacent to only one or two triangles.
- Each point is only adjacent to one connected area (i.e, sequence of adjacent triangles).

Assumption B.2. Our 4D tetrahedral mesh satisfies:

- Each face is adjacent to only one or two tetrahedra.
- Each edge is only adjacent to one connected area.
- Each point is only adjacent to one connected area.

Assumption B.3. If the slice hyper-plane is $w = \alpha$, vertices in the 4D tetrahedral mesh is $V = \{x_i, y_i, z_i, w_i\}$, then $\forall i, w_i \neq \alpha$.

Through our construction method and slicing method, these assumptions are guaranteed.

Theorem 1. We have a 4D tetrahedral mesh $\mathcal{T}$ and a hyper-plane P (constructed as described in the paper). The slice of the 4D tetrahedral mesh by hyper-plane P is guaranteed to be a manifold mesh.

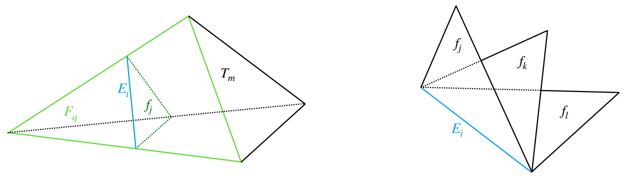


Figure 13. Non-manifold edge: One edge adjacent to more than two faces.

Proof. We make the proof by contradiction.

- Each triangle edge E_i in a sliced mesh is a subset of a tetrahedron's face. Assume that this edge has three adjacent faces f_j, f_k, and f_l. A tetrahedron in 4D space is a 3-simplex and an n-simplex is a convex set(Mount, 2002). A hyper-plane is also a convex set. The intersection between two convex sets is guaranteed to be convex (Boyd et al., 2004). So, for each tetrahedron, there is only one face sliced at most. So f_j, f_k and f_l must be sliced from different tetrahedra. So $E_i \subset f_j \subset T_m$, $E_i \subset f_k \subset T_n$, $E_i \subset f_l \subset T_p$. T_m, T_n, T_p are three different tetrahedra. Meanwhile, this edge is in the tetrahedron's face $E_i \subset F_q \subset T_m$. f_j, f_k, f_l share the same edge E_i. So we have $E_i \subset F_q = adj(T_m, T_n, T_p)$. As shown in Figure 13, this violates that each face is adjacent to only one or two tetrahedra. So each edge is adjacent to only one or two triangles in the sliced mesh. $adj(a, b)$ represents the adjacent part between a and b.

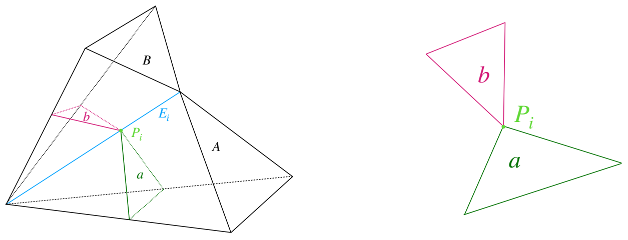


Figure 14. Non-manifold vertex: One vertex adjacent to more than one connected area.

- If there is a point P_i in a sliced mesh that is adjacent to more than one area, i.e., at least two areas a and b. Then P_i is sliced from an edge E_j in 4D tetrahedra. a, b are sliced from A, B. A and B are different one or more tetrahedra. In A and B, these tetrahedra are face-to-face adjacent to each other. And the edge E_i must be adjacent to two areas A and B. So A and B are two connected areas adjacent by one edge $E_i = adj(A, B)$. As shown in Figure 14, this violates that each edge is only adjacent to one connected area. So each point in a sliced mesh is only adjacent to one connected area.

□

The same method can be used to prove that when all the faces in a 4D tetrahedral mesh are each adjacent to two tetrahedra, the sliced mesh is guaranteed to be watertight.

C. More Results

An example of NEURALSLICE's failure case is shown in Figure 15. This is because the input point cloud is discrete and noisy. The supervision of our method during training is only on the input point cloud, and it is difficult for the encoder to discern the correct structure of this point cloud.

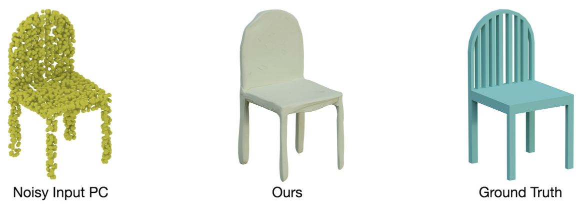


Figure 15. Failure case of NEURALSLICE. As our method takes the discrete point cloud with noise as input, it is hard to discern the correct topology in such a challenging case.

We demonstrated more surface reconstruction results in Figure 16.

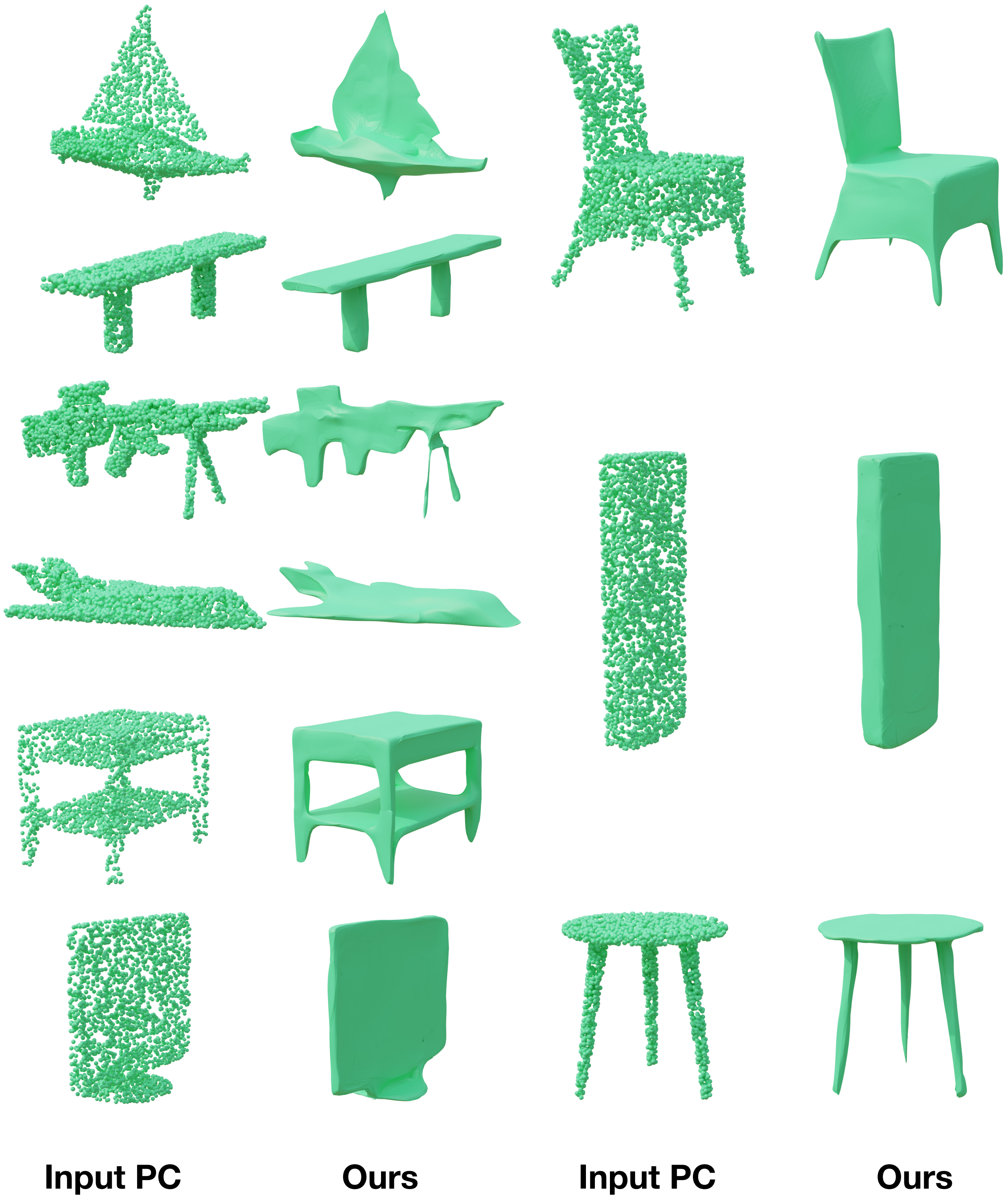


Figure 16. More results of NEURALSLICE.