DMesh: A Differentiable Representation for General Meshes

Sanghyun Son[1], Matheus Gadelha[2], Yang Zhou[2], Zexiang Xu[2],
Ming C. Lin[1], and Yi Zhou[2]

[1] University of Maryland, College Park
[2] Adobe Research

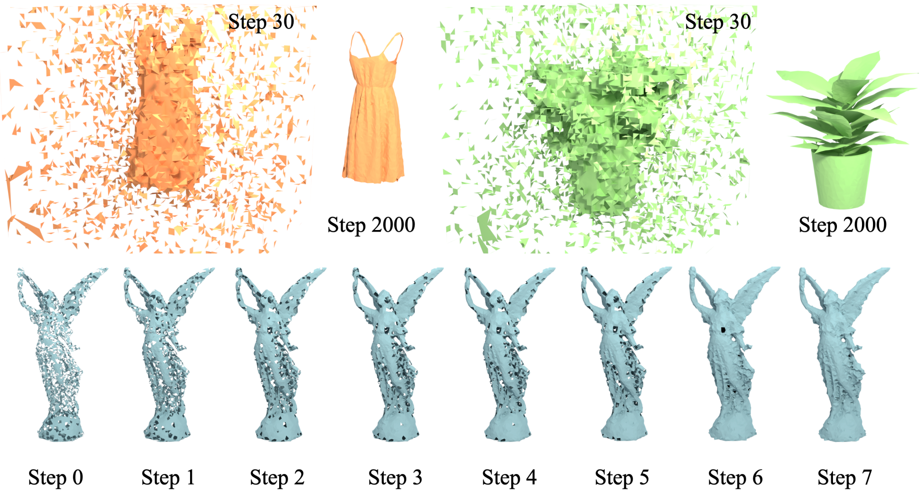


Fig. 1: (→) **Optimization process.** We can start from either random state (up) or initialization based on sample points (down) for faster convergence. Mesh connectivity changes dynamically during the optimization. To make this topology change possible, we compute existence probability for an arbitrary set of faces in a differentiable manner.

Abstract. We present a differentiable representation, **DMesh**, for general 3D triangular meshes. **DMesh** considers both the geometry and connectivity information of a mesh. In our design, we first get a set of convex tetrahedra that compactly tessellates the domain based on *Weighted Delaunay Triangulation* (WDT), and formulate probability of faces to exist on our desired mesh in a differentiable manner based on the WDT. This enables **DMesh** to represent meshes of various topology in a differentiable way, and allows us to reconstruct the mesh under various observations, such as point cloud and multi-view images using gradient-based optimization. The source code and full paper is available at: https://sonsang.github.io/dmesh-project [3].

Keywords: Differentiable Mesh · 3D reconstruction

[3] This paper was last modified at Apr 9, 2024

1 Introduction

Polygonal meshes are widely used in modeling and animation due to their diverse, compact and explicit configuration. Recent AI progress has spurred efforts to integrate mesh generation into machine learning, but challenges like varying topology hinder suitable differentiable mesh representations. This limitation leads to reliance on differentiable intermediates like implicit functions, and subsequent iso-surface extraction for mesh creation [16,25,36,43,44]. However, meshes generated by such approaches can be unnecessarily dense and misaligned at sharp regions [44], and struggle with open surfaces due to their reliance on the volumetric representation.

The fundamental challenge in creating a differentiable mesh representation lies in formulating both the vertices' geometric features and their connectivity, defined as edges and faces in a differentiable way. Given a vertex set, predicting their connectivity in a free-form way using existing machine learning data-structures can cost significant amount of computation and be difficult to avoid irregular and intersecting faces. Consequently, most studies on differentiable meshes simplify the task by using a mesh with a *pre-determined* topology and modifying it through various operations [17,38,40,54]. This work, on the contrary, ambitiously aims to establish a general 3D mesh representation, named as DMesh, where both mesh topology and geometric features (e.g. encoded in vertex location) can be simultaneously optimized through gradient-based techniques.

Our core insight is to use differentiable Weighted Delaunay Triangulation (WDT) to divide a convex domain, akin to amber encapsulating a surface mesh, into tetrahedra to form a mesh. To create a mesh with arbitrary topology, we select only a subset of triangular faces from the tetrahedra, termed the "real part", as our final mesh. The other faces, the "imaginary part", *support* the real part but are *not* part of the final mesh. We introduce a method to assess the probability of a face being part of the mesh based on weighted points that carry positional and inclusiveness information. Optimization is then focused on the *points' features*, using a dual power diagram of WDT [3] to generate the triangular mesh. The probability determination allows us to compute geometric losses and rendering losses during gradient-based optimization. This method is essentially a 3D, differentiable extension of $\mathcal{A}$-shape [33,34], and a differentiable solution to the problem addressed by constrained Delaunay Triangulation [15, 45,46].

The key contributions of our work can be summarized as follows.

- We present a novel mesh representation, **DMesh**, which is versatile to accommodate various mesh types for both open surfaces and closed surfaces. The generated meshes are always face-intersection-free.
- We provide efficient reconstruction algorithms for DMesh, which are designed for 3d point cloud and multi-view image inputs. For multi-view reconstruction, we present a differentiable renderer that meets our needs.
- We provide effective regularization methods for DMesh, which can be used for mesh simplification, or triangle quality enhancement.

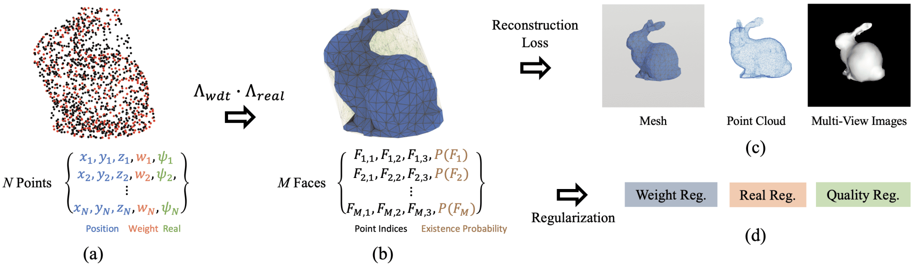

$\Lambda_{wdt} \cdot \Lambda_{real}$
Reconstruction Loss
Mesh
Point Cloud
Multi-View Images
(c)
N Points
$x_1, y_1, z_1, w_1, \psi_1$
$x_2, y_2, z_2, w_2, \psi_2,$
$x_N, y_N, z_N, w_N, \psi_N$
Position
Weight
Real
(a)
M Faces
$F_{1,1}, F_{1,2}, F_{1,3}, P(F_1)$
$F_{2,1}, F_{2,2}, F_{2,3}, P(F_2)$
$F_{M,1}, F_{M,2}, F_{M,3}, P(F_M)$
Point Indices
Existence Probability
(b)
Regularization
Weight Reg.
Real Reg.
Quality Reg.
(d)

Fig. 2: Our overall framework to optimize mesh according to the given observations. **(a)**: Each point is defined by a 5-dimensional feature vector, which includes position, weight, and real value. Points with larger real values are rendered in red. **(b)**: Given a set of points, we can gather possible faces to exist in our mesh and evaluate their existence probability in differentiable manner. **(c)**: We can compute reconstruction loss by comparing our mesh with given observations, such as mesh, point cloud, or multi-view images. **(d)**: To facilitate the optimization process and enhance the mesh quality, we can use additional regularizations.

- To overcome prohibitively large computational cost of the exact formulation, we propose an efficient relaxation that computes the face existence probabilities with a practical computational cost.

Additionally, to further accelerate the algorithm, we implemented our main algorithm and differentiable renderer in CUDA, which is made available for further research.

2 Related Work

2.1 Shape Representations for Optimization

Neural Implicit Functions The trend of modeling 3D objects as differentiable neural representations has gained popularity in graphics and vision applications, primarily for 3D reconstruction and novel view synthesis, allowing shape optimization through gradient descent and backpropagation [7, 8, 26, 35, 47, 51, 52]. Many methods, inspired by NeRF [35], express scene geometry using volume density and differentiable volume rendering. However, these density-based volumetric approaches don't always result in accurate 3D geometry. To improve this, several approaches [39, 47, 48, 50] model surface functions as neural signed distance functions (SDFs), converting them to density for rendering and optimization. More recently, neural unsigned distance functions (UDFs) have been developed to model open surfaces, which SDFs can't describe [28, 29]. While these implicit surface representations show promise in reconstruction, they require iso-surface extraction algorithms like Marching Cubes [30] to convert implicit functions to explicit high-poly meshes, introducing geometric errors. In contrast, our explicit representation can directly output a mesh that can also represent open surfaces, avoiding these issues.

Mesh Representations Previous methods have tried optimizing meshes directly, but often with the assumption of a fixed overall mesh topology [9, 23, 27, 38]. While local connectivity can be altered through remeshing [40], the fundamental geometric topology remains unchanged. Learning-based approaches like BSP-Net [11] allow for topological variation, yet their meshing process isn't differentiable. Recently, differentiable iso-surface extraction techniques have been developed, resulting in high-quality geometry reconstruction of various topology when combined with Neural or discrete Signed Distance Functions (SDFs) [25,36,43,44,49]. Some methods even demonstrate backpropagating gradients from mesh vertices to SDF values using non-differentiable techniques like Marching Cubes [32]. However, these surface extraction methods, reliant on SDFs and uniform grids, often need high-poly meshes for accurate reconstruction, regardless of the actual surface's complexity. Our approach does not have to concern about these issues, because we explicitly define faces and their existence probabilities. See Table 3 for more detailed comparisons to these other methods.

2.2 Shape Representation using Delaunay Triangulation

Delaunay Triangulation (DT) in $\mathbb{R}^d$ connects points whose Voronoi cells share a boundary [3], making it useful for reconstructing shapes from unorganized point sets. It's been shown that DT of dense samples on a smooth 2D curve includes the curve within its edges [1,5]. This idea of using DT to approximate shape has been successfully extended to 3D, to reconstruct three-dimensional shapes [2] for point sets that satisfy certain constraints. Our method can be thought of as a differentiable version of these approaches.

Additionally, [42] focused on this DT's property to connect points and tessellate the domain, and proposed a differentiable WDT algorithm to compute smooth inclusion, or existence score of 2-simplexes (triangles) in 2 dimensional WDT. Our approach develops this approach to compute that score for 2-simplexes in 3 dimensional WDT, which faces different computational challenges than the previous work (Section 3.3). More recently, VoroMesh [31] used similar approach to ours using Voronoi diagram for point cloud reconstruction, but it cannot handle open surfaces and is only confined to point clouds (Section 4).

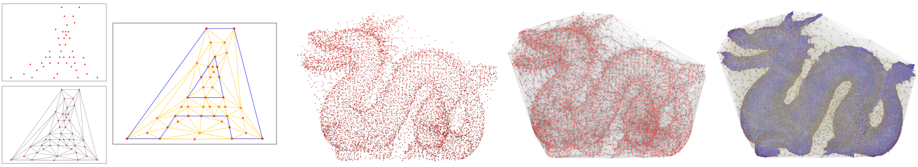
3 Formulation

In this section, we start with the definition of our new mesh representation. Then we introduce its differentiable formulation, which evaluates the probability of a face to exist in the mesh. Finally we explain how to conquer the computational difficulties posed in our formulation.

3.1 Overall definition

In this work, we take a flexible approach to define a d-dimensional mesh as a set of $(d-1)$-simplexes [4], and **propose to represent a mesh as a subset**

[4] They become line segments when $d = 2$, and triangles when $d = 3$.



(a) 2D Font

(b) 3D Dragon

Fig. 3: Illustration of our mesh representation for 2D and 3D cases. **(a):** Our representation in 2D for a letter "A". **(b):** Our representation in 3D for a dragon model. Blue faces are *"real part"* and yellow ones are *"imaginary part"*.

of WDT. To elaborate, for a given set of d-dimensional points $\mathbb{P} \in \mathbb{R}^d$ and their weights $\mathbb{W} \in \mathbb{R}$, we first obtain the WDT from the weighted points, which tessellates the convex hull of the given points into a compact set of d-simplexes. Then, we extract the desirable $(d-1)$-simplexes from the tessellation to define our mesh. Without losing generality, we call the $(d-1)$-simplexes as faces here. Among the entire set of faces, we refer the desirable faces as *"real part"*, and the others as *"imaginary part"*. Figure 3 illustrates the cases for $d = 2$ and $d = 3$. Note that the imaginary part is used to sustain the tessellation, even though it is not included in the mesh.

Now let us assume there is a face F that we want to know if it exists in the final mesh or not. Based on the above scheme, we notice that there are two layers of "existence" for F. First, we have to check if F exists in the WDT or not. Formally, we say $F \in \text{WDT}(\mathbb{P}, \mathbb{W})$ if there is a d-simplex in the tessellation induced by WDT that has F as one of its faces. Second, if F exists in WDT, we have to find out if it is included in the *"real part"*. Therefore, we define two predicates, $\mathbb{I}_{wdt}$ and $\mathbb{I}_{real}$, to evaluate the existence of F in the mesh.

$$\mathbb{I}_{wdt}(F) = \begin{cases} 1 \quad \text{if} \quad F \in \text{WDT}(\mathbb{P}, \mathbb{W}) \\ 0 \text{ else} \end{cases}$$

$$\mathbb{I}_{real}(F) = \begin{cases} 1 \quad \text{if} \quad F \in \text{Mesh when } F \in \text{WDT}(\mathbb{P}, \mathbb{W}) \\ 0 \text{ else} \end{cases}$$

Unlike $\mathbb{I}_{wdt}$, there are various formulations we can use for $\mathbb{I}_{real}$. In this work, we opt to formulate it using point-wise value $\Psi \in \{0, 1\}$ for the convenience of inference and optimization. When $d = 3$, given a face $F = (p_i, p_j, p_k)$ in $WDT(\mathbb{P}, \mathbb{W})$, we define $\mathbb{I}_{real}(F)$ as:

$$\mathbb{I}_{real}(F) = \min(\Psi_i, \Psi_j, \Psi_k).$$

Note that all of the three points should have a value of 1 to make F to be considered in the *"real part"*. Finally, we can define the complete face existence function to determine if a face F exists in the final mesh or not as

$$\mathbb{I}(F) = \mathbb{I}_{wdt}(F) \wedge \mathbb{I}_{dist}(F).$$

Differentiable Approach To evaluate the existence of a face F in a differentiable manner, we take a probabilistic approach. That is, we define differentiable functions $\varLambda_{wdt}$ and $\varLambda_{real}$ that evaluate the following probabilities,

$$\varLambda_{wdt}(F) = P(F \in WDT(\mathbb{P}, \mathbb{W})) \tag{1}$$
$$\varLambda_{real}(F) = P(F \in \text{Mesh} \,|\, F \in WDT(\mathbb{P}, \mathbb{W})), \tag{2}$$

which produce the following function to determine the final probability of F to exist in mesh:

$$\varLambda(F) = P(F \in Mesh) = \varLambda_{wdt}(F) \cdot \varLambda_{real}(F).$$

Not only this probabilistic interpretation is important to our differentiable formulation, but also to the downstream tasks that we solve (Section 3.4). In the following section, we discuss the details of $\varLambda_{wdt}$ and $\varLambda_{real}$.

Point Features Before moving on to the next section, we'd like to point out that the introduced face existence solely depends on the configuration of the weighted points. Thus, our representation features can be defined purely on the point set. In our representation, each point is defined as a $(d+2)$-dimensional vector, d of which represents the spatial position, 1 stands for the weight for WDT, and the remaining 1 is used as ψ, which corresponds to the differentiable version of $\varPsi$ (Section 3.2). Note that we set the range of weight and ψ to be $[0, 1]$ in all of our experiments. Our overall framework to optimize our mesh according to the given observations based on these point features is shown in Figure 2.

3.2 Probability Functions

$\boldsymbol{\varLambda_{wdt}}$ estimates probability of a face F to exist in WDT (Eq. 1). Our formulation leverages the dual structure of WDT, or Power Diagram (PD) to compute it, following [42]. Note that we develop our theoretical reasoning mainly in 2D for ease of understanding, but it can be extended to 3D easily. To avoid confusion, we denote 1-simplex (line segment) and 2-simplex (triangle) as F_2 and F_3 in this section. Please see Appendix 7 for more detailed discussions.

To start with, given a set of points $\mathbb{P} \in \mathbb{R}^2$ and their weights $\mathbb{W} \in \mathbb{R}$, we call Power cell of p_i in the (dual) PD as C_i. In Figure 4(a), we can see points p_1, p_2, and p_3 and their corresponding C_1, C_2, and C_3 in PD. In Figure 4(b, d), C_1 is marked with orange lines. Now, we consider a face F_2, which connects two points p_i and p_j. Then we can construct its dual line L_F in PD as the intersection of two half spaces defined by the two points. In Figure 4, faces and their dual lines are rendered as solid and dotted blue lines, respectively. In Figure 4(b, d), we can observe that F_2 exists if and only if the two Power cells C_i and C_j share a common edge, and it is a subset of L_F, which holds in general.

Based on this observation, we can measure the unsigned minimum distance between L_F and Power cells C_i and C_j, and use it to identify the existence of F_2. However, note that the distance stays at 0 when F_2 exists, which means that

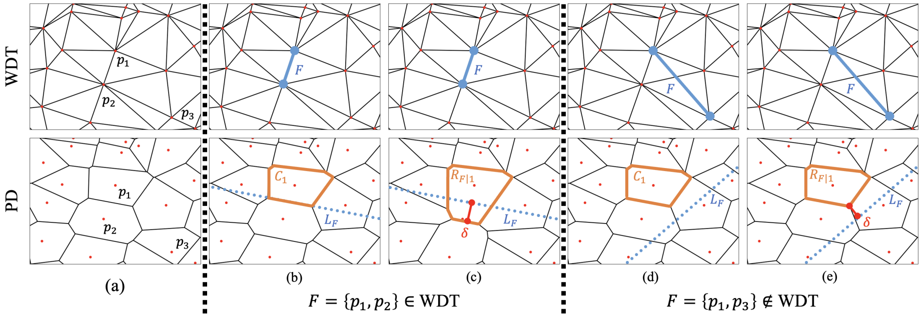


Fig. 4: To compute probability of a $(d-1)$-simplex F's existence in WDT (upper row), we investigate its dual PD (lower row). For given F (solid blue), we measure the signed distance δ (red) between its dual L_F (dotted blue) and reduced Power cell (orange) for the estimation. If F exists as shown in (b) and (c), δ becomes positive. In contrast, it evaluates to negative when F does not exist as shown in (d) and (e).

we cannot measure how "stable" F_2 is when it exists. Thus, it is not suitable for measuring differentiable existence probability of F_2.

To amend this issue, we adopt the concept of reduced Power cell [42]. Reduced Power cell, denoted as $R_{F|i}$, is a Power cell of p_i when ignoring the other point p_j in F_2. In Figure 4(c, e), we render reduced Power cell $R_{F|1}$ for two different F_2s in orange lines. Note that when F_2 exists, $R_{F|1}$ gets bigger than C_1 and L_F goes through it, rather than lying on its boundary. When F_2 does not exist, $R_{F|1}$ is just same as C_1, and thus L_F does not have contact with it.

Now, we newly define a signed distance between L_F and $R_{F|i}$. To that end, we define a signed distance between a random point $P \in \mathbb{R}^2$ and a random reduced Power cell R as follows,

$$\tau_1(P, R) = d(P, R) \cdot (-1)^{1-I(P \in R)},$$

where $d(P, R)$ is the minimum (unsigned) distance between P and R, and $I(\cdot)$ is an indicator function. Then, based on τ_1, we can define a signed distance between a random line L and R as

$$\tau_2(L, R) = \max_{P \in L} \tau_1(P, R). \tag{3}$$

Observe that the sign of τ_2 is positive when L goes through R, and negative when L does not have contact with R.

Noting that $R_{F|i}$ can exist only when C_i exists [5], we define the signed distance between the dual line L_F and a reduced Power cell $R_{F|i}$ as

$$\delta(L_F, R_{F|i}) = \begin{cases} \tau_2(L_F, R_{F|i}) & \text{if } \exists C_i \\ -\infty \text{ else} \end{cases} \tag{4}$$

[5] If the weight of p_i is lower than its neighboring points, there is a chance that C_i does not exist.

Then, the following relationship holds,

$$\delta(L_F, R_{F|i}) > 0 \Leftrightarrow \mathbb{I}_{wdt}(F) = 1.$$

which means, when F exists in WDT, its dual line has positive signed distance to the reduced Power cell of its two ends, and vice versa. Note that this relationship holds for any $x \in \{i, j\}$, because the sign of every $\delta(L_F, R_{F|x})$ is the same. In the right columns of Figure 4(b, c), we can see pink line segments that represent $\delta(L_F, R_{F|1})$.

Then, coming back to $d = 3$, we define a function

$$\varDelta(F_3) = \frac{1}{3}(\delta(L_F, R_{F|i}) + \delta(L_F, R_{F|j}) + \delta(L_F, R_{F|k})), \tag{5}$$

which satisfies $\varDelta(F_3) > 0 \Leftrightarrow \mathbb{I}_{wdt}(F) = 1$, because the sign of every δ is the same.

Note that this function goes to $-\infty$ if any one of the points in F_3 loses its Power cell. When all of the three points have Power cell, but F_3 does not exists, the function evaluates to a negative value. Finally, it becomes a positive value when F_3 exists. Therefore, we can define a differentiable probability function for the face F_3 to exist in WDT as follows,

$$\varLambda_{wdt}(F_3) = \sigma(\alpha_{wdt} \cdot \varDelta(F_3)), \tag{6}$$

where σ is a sigmoid function parameterized by α_{wdt}. In our experiments, we set $\alpha_{wdt} = 1000$.

$\boldsymbol{\varLambda_{real}}$ evaluates the existence probability of $F_3 = \{p_i, p_j, p_k\}$ in our mesh when it exists in WDT. To define it, we modify per-point discrete value $\varPsi$ to ψ, which can have a continuous value in $[0, 1]$. Then, we define $\varLambda_{real}$ as,

$$\varLambda_{real}(F_3) = dmin(\psi_i, \psi_j, \psi_k, \alpha_{real}),$$

where $dmin$ is a differentiable min operator (Appendix 7), and α_{real} is a hyperparameter for it. We set it as $\alpha_{real} = 100$ in our experiments.

3.3 Computational Difficulties

Although Eq. 4 plays a vital role, it is not trivial to compute it, especially in 3-dimensional space that we are dealing with. For instance, when C_i exists and we have to evaluate Eq. 3, it is not trivial to find an answer to the optimization problem. Moreover, it is hardly possible to compute every reduced Power cell, $R_{F|i,j,k}$, for every possible F_3.

To overcome these computational difficulties, we propose to leverage lower bound of Eq. 4, which can be efficiently found without constructing any reduced Power cell explicitly. To that end, we treat two cases, $F_3 \in WDT(\mathbb{P})$ and $F_3 \notin WDT(\mathbb{P})$, differently. To be specific, when $F_3 \in WDT(\mathbb{P})$, we define δ_1 as

$$\delta_1(L_F, R_{F|i}) = \tau_1(P_{mid}, R_{F|i}) \geq 0, \tag{7}$$

where P_{mid} is the middle point of the line segment $L_{F|i} = L_F \cap C_i$. The existence of P_{mid} is guaranteed, because L_F is on the boundary of C_i if $F_3 \in WDT(\mathbb{P})$. Note that we can compute Eq. 7 efficiently by projecting P_{mid} to the planes that comprise $R_{F|i}$, because of convexity. This alone reduces a lot of computational burden, because we only have to gather planes that would possibly comprise $R_{F|i}$, instead of explicitly constructing it [6]. Also, note that δ_1 is a lower bound of δ by definition at Eq. 3.

When $F_3 \notin WDT(\mathbb{P})$, we use following δ_2:

$$\delta_2(L_F, R_{F|i}) = \tau_2(L_F, C_i) \leq 0. \tag{8}$$

Note that this is lower bound of δ when F_3 does not exist, because C_i is a subset of $R_{F|i}$. Since we can readily obtain C_i from current Power diagram, we can compute minimum distances between L_i and line segments on the boundary of C_i to evaluate Eq. 8.

To sum up, we redefine $\delta(L_F, R_{F|i})$ as follows.

$$\delta(L_F, R_{F|i}) = \begin{cases} \delta_1(L_F, R_{F|i}) & \text{if} \quad \exists F_3 \\ \delta_2(L_F, R_{F|i}) & \text{else if } \exists C_i \wedge \nexists F_3 \\ -\infty & \text{else} \end{cases} \tag{9}$$

Even though this formulation gives a lower bound of Eq. 4, note that when the original function evaluates to 0, this relaxation also evaluates to 0. Therefore, we can still use sigmoid function of Eq. 6 to get differentiable existence probability. Note that in these relaxations, we need to obtain every Power cell C_i, which can be achieved by computing WDT for current point configuration. Please see Appendix 7 and 9 for more details about our formulation, and how it is used in the real optimization process.

3.4 Loss Functions

DMesh could be reconstructed from various types of inputs, such as meshes, point clouds and multi-view images. Given those inputs, we optimize it by minimizing the specific energy functions leveraging the existence probabilities $\Lambda(F)$ of faces F. Here we briefly introduce how we define the reconstruction losses and additional regularization losses that we use in the optimization process. Please see Appendix 8 for more detailed explanations for these loss functions.

Reconstruction Loss ($\boldsymbol{L_{recon}}$) First, we assume that we are given a ground truth mesh, which is comprised of points $\mathbb{P}$ and faces $\mathbb{F}$, and we need to represent it with our representation. In this case, we can easily see that we should maximize $\Lambda(\mathbb{F})$, as we already know that they exist in the mesh. In contrast, if we say $\bar{\mathbb{F}}$ as the remaining set of faces that can be defined on $\mathbb{P}$, we notice that we should

[6] In our experiments, during optimization process, we keep a set of planes that were on Power cell C_i for each point, and update it during optimization.

minimize $\Lambda(\bar{\mathbb{F}})$. Likewise, the reconstruction loss for mesh input can be defined by this explicit connectivity information (Appendix 8.1).

However, when it comes to mesh reconstruction from point clouds or multi-view images, we need to use another form of reconstruction loss. Commonly, we exploit the probabilistic nature of our formulation in defining reconstruction loss for these inputs. For instance, for point cloud, we formulate our loss mainly based on Chamfer Distance (CD) loss, and compute the "expected" CD using our face probabilities (Appendix 8.2). For multi-view images, we define our loss based on rendering loss, which can be computed as L_1 loss between the images of our models rendered by a differentiable renderer and the given images. Here we interpret the face probabilities as face opacities in the rendering process. To allow gradients to flow across the face opacities, we implemented efficient differentiable renderers. Please see Appendix 8.3 for details about them.

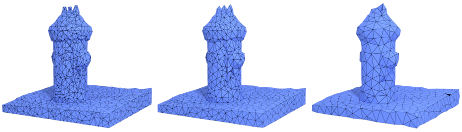

(a) $\lambda_{weight} = 10^{-8}$ (b) $\lambda_{weight} = 10^{-5}$ (c) $\lambda_{weight} = 10^{-4}$

Fig. 5: Results with different λ_{weight}.

Regularizations During optimization, we can employ various regularizations to facilitate the process and enhance the final mesh quality. The first regularization that we introduce is **weight regularization** (L_{weight}), which works on the the dual Power Diagram of WDT (Appendix 8.4). Using this regularization, we intend to reduce the structural complexity of WDT, and discard unnecessary points that are not required to represent our mesh. Note that we can use this regularization because we use WDT, not DT. Using this regularization, we can control the final mesh complexity, as shown in Figure 5.

The next regularization is designed to guide *real* values of points, which is called as **real regularization** (L_{real}). This regularization aims at enforcing nearby points to have similar real values. At the same time, it increases real values of points that are adjacent to the points of high *real* values (Appendix 8.5). This regularization facilitates the optimization process by removing holes or inner structures of the mesh (Appendix 9), and making the faces near current surface to be considered with higher probabilities than the others.

The final regularization aims at improving the quality of the triangle faces on the mesh, which we name as **quality regularization** (L_{qual}). To be specific, we minimize the average expected aspect ratio of the faces (Appendix 8.6). Using this regularization, we intend to remove thin triangles on the mesh.

Total Loss To sum up, our final loss function can be written as follows:

$$L = L_{recon} + \lambda_{weight} \cdot L_{weight} + \lambda_{real} \cdot L_{real} + \lambda_{qual} \cdot L_{qual},$$

where λ values are hyperparameters. In Appendix 10, we provide values for these hyperparameters for every experiment. Also, in Appendix 10.3, we present ablation studies for these regularizations.

4 Experiments and Applications

In this section, we provide experimental results that show the efficacy of our approach. First, when we are given a ground truth mesh, we optimize the point attributes to restore the mesh. With this experiment, we directly prove the differentiability of our design and show the representation power of DMesh. Next, we conduct experiments about 3D reconstruction from point clouds and multi-view images to show how our differentiable formulation can be used in downstream applications. We also show how the regularization affects the reconstruction results through ablation studies.

For the first mesh reconstruction problem, we used three models from Stanford 3D scanning repository [13]. For point cloud and multi-view reconstruction tasks, we used 4 closed-surface models from Thingi32 dataset [53], 4 open-surface models from DeepFashion3D dataset [18], and 3 additional general models that are comprised of both closed and open surfaces from Objaverse dataset [14] and Adobe Stock, to accommodate meshes of various topology. Each of these kinds of models is denoted as “closed”, “open”, and “mixed” model in this section.

We implemented our main algorithm for computing face existence probabilites and differentiable renderer used for multi-view image reconstruction in CUDA [37]. Since we need to compute WDT before running the CUDA algorithm, we used WDT implementation of CGAL [19]. On top of that, we implemented the rest of logic with Pytorch [41]. All of the experiments were run on a system with AMD EPYC 7R32 CPU and Nvidia A10 GPU.

4.1 Mesh to DMesh

Table 1: Mesh reconstruction results.

-	Bunny	Dragon	Buddha
RE	99.78%	99.72%	99.64%
FP	0.00%	0.55%	0.84%

In this experiment, we demonstrate that we can preserve most of the faces in the original ordinary mesh after converting it to DMesh using the mesh reconstruction loss introduced in Section 3.4. Please see Appendix 9.1 to learn about the details of the entire optimization process.

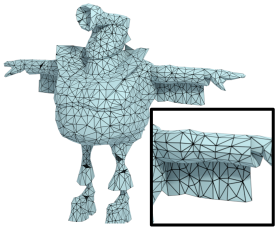
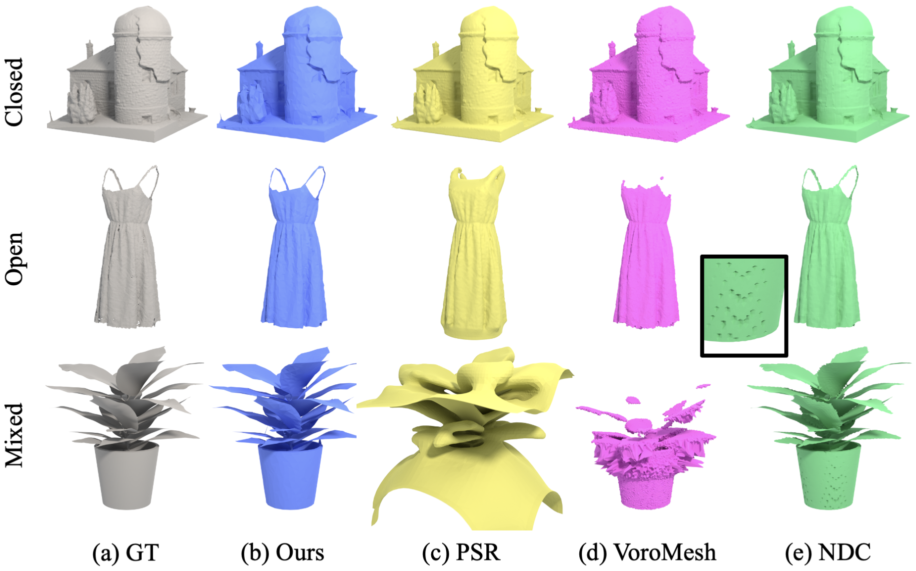


Fig. 6: Reconstruction result that has mesh pattern adaptive to local geometry.

In Table 1, we show the recovery ratio (RE) and false positive ratio (FP) of faces in our reconstructed mesh. Note that we could recover over 99% of faces in the original mesh, while only having under 1% of false faces. Please see Appendix 10.1 for more details. This result shows that our differentiable formulation is correct, but also tells us that there is a limitation in converting the original mesh into DMesh using connectivity information. To overcome this limitation, we can reconstruct mesh using other reconstruction losses as discussed in next section. Interestingly, under some occasions, we could observe that our optimized mesh exhibits artificial quad-mesh like pattern (Figure 6),



(a) GT (b) Ours (c) PSR (d) VoroMesh (e) NDC

Fig. 7: Point cloud reconstruction results. For a given point cloud sampled from ground truth mesh in (a), **our method (b) successfully restores the original shape without losing much much detail.** In contrast, PSR [20] (c) and VoroMesh [31] (d) fail for open and mixed surface models. NDC [10] (e) exhibits artifacts from grids.

even if we optimize our mesh without ground truth connectivity information, which shows potential ability of our method.

4.2 Point Cloud & Multi-View Reconstruction

Table 2: Statistics for Point Cloud (PC) and Multi-View (MV) Reconstruction. Best results are highlighted in bold.

	Methods	CD $(10^{-3})\downarrow$			Time (sec) $\downarrow$
		Closed	Open	Mixed	
PC	Ours	7.42	6.87	**8.06**	775.05
	PSR	**7.15**	26.94	67.18	10.61
	VoroMesh	7.30	26.31	99087.64	12.18
	NDC	7.30	**6.83**	8.25	**3.48**
MV	Ours	**15.56**	**11.11**	**18.33**	1434
	Flexicube	31.23	34.91	25.15	**56.47**
	NIE	31.54	67.37	43.05	6696.43

In this experiment, we aim to reconstruct a mesh from partial geometric data, such as (oriented) point clouds or multi-view images. For point cloud reconstruction, we sampled 100K points from the ground truth mesh. Even though our formulation can use normal information for better reconstruction (Figure 9), we only use point positions for fair comparison. For multi-view reconstruction, we rendered diffuse and depth images of the ground truth mesh from 64 view points. In Appendix 10, we illustrated the example inputs for these experiments. Also, please see Appendix 9 to see the initialization and densification strategy we took in these experiments.

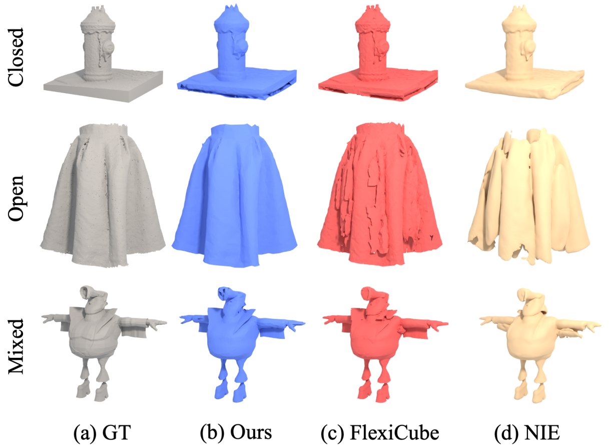


Fig. 8: Multi-view Reconstruction results. For given images captured at multiple viewpoints around the ground truth mesh in (a), our mesh (b) succeeds in reconstructing overall shapes for every model, with small artifacts. However, since (c) Flexicube [44] and (d) NIE [32] rely on volumetric principles, they produce wrong meshes for open and mixed mesh models.

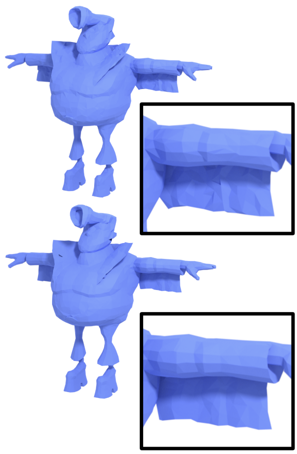


Fig. 9: Point cloud reconstruction results from oriented points. (Up) Reconstruction with $\lambda_{normal} = 0.001$ (Down) Reconstruction with $\lambda_{normal} = 0.01$.

To validate our approach, we compare our results with various approaches. When it comes to point cloud reconstruction, we first compare our result with classical Screened Poisson Surface Reconstruction (PSR) method [20] [7]. Then, to compare our method with optimization based approach, we use recent VoroMesh [31] method, which shares similar principles with us. Note that these two methods are essentially volumetric approach, and thus are not tailored for open surfaces. To compare our method also for the open surfaces, we use Neural Dual Contouring (NDC) [10], even though it is learning-based approach. Finally, for multi-view reconstruction task, we compare our results with Flexicube [44] and Neural Implicit Evolution (NIE) [32], which correspond to volumetric approaches that can directly produce mesh of varying geometric topology for given visual inputs.

In Figure 7 and 8, we visualize the reconstruction results along with the ground truth mesh. In general, volumetric approaches like PSR, VoroMesh, and Flexicube, capture fine details better than our methods for closed models. This is mainly because we currently have limitation in the mesh resolution that we can produce with our method. NIE, which is also based on volumetric principles, generates overly smoothed reconstruction results. However, when it comes to open or mixed mesh models, we can observe that these methods fail, usually with false internal structures or self-intersecting faces (Appendix 10.2). Since NDC leverages unsigned information, it can handle these cases without much

[7] We also feed in point orientations for PSR, which is optional for our method.

problem as ours. However, we can observe step-like visual artifacts coming from its usage of grid in the final output, which requires post-processing.

Table 2 presents quantitative comparisons with other methods. Chamfer Distance (CD) based on L_1-norm is computed between the reconstructed mesh and the ground truth mesh, along with an average for different types of meshes. Additionally, we report the average running time of each method. In the table, we observe that CD error generally aligns with the visual renderings. Compared to the other methods, our method exhibits generally better, or comparable results across every model for both point cloud and multi-view reconstruction. However, notice that our method has clear limitation in computation time in the current implementation. This is partially because we run too many steps (Appendix 10.2) for the sake of completeness of every model, but many models converge very fast in practice, as shown in Figure 1 when we use sample points for initialization.

5 Conclusion and Future Directions

Our method achieves a more effective and complete representation of meshes of various topology than existing methods, but opens up areas for future research.

- Computational cost: Currently, the resolution of DMesh is largely constrained by computational cost. Even though we succeeded in decreasing computational burden through our theoretical relaxation and CUDA implementation, it costs more than a second to process over 100K vertices, mainly because we run WDT for the entire points at every step (Appendix 9.2).
- Non-manifoldness: As we have claimed so far, DMesh shows much better generalization than the other methods as it does not have any constraints on the mesh connectivity. However, due to this relaxation of constraint, small holes or "ears" in the reconstruction can appear as "non-manifoldness". They become more evident when there is no strong supervision or appropriate regularization. Multi-view image reconstruction with occlusions is a typical example. It is possible to eliminate them up to some extent by using additional measures (Appendix 9.2). But, more structured mechanism to eliminate them completely and generate geometric entities that align with more formal definition of "mesh" [4] would be a natural extension.

To address the aformentioned limitations, it is possible to accelerate the main algorithm by carefully constraining the points to update or imposing some bounds on the step size to minimize costly WDT at every iteration. Also, we can investigate if GPU acceleration is possible for WDT [6]. Next, additional geometric constraints can be imposed to remove non-manifold edges. Adopting regularizations like Eikonal loss could be one possible approach, as we can encode unsigned distance information in the points.

Further research can also extend this work to solve other challenging problems (e.g. 3D reconstruction from real world images) or other related applications (e.g. 3D mesh generative model) in the future.

Acknowledgements We thank Zhiqin Chen and Matthew Fisher for helpful advice. This research is a joint collaboration between Adobe and University of Maryland at College Park. This work has been supported in part by Adobe, IARPA, UMD-ARL Cooperate Agreement, and Dr. Barry Mersky and Capital One Endowed E-Nnovate Professorships.

References

1. Amenta, N., Bern, M., Eppstein, D.: The crust and the β-skeleton: Combinatorial curve reconstruction. Graphical models and image processing **60**(2), 125–135 (1998)
2. Amenta, N., Bern, M., Kamvysselis, M.: A new voronoi-based surface reconstruction algorithm. In: Proceedings of the 25th annual conference on Computer graphics and interactive techniques. pp. 415–421 (1998)
3. Aurenhammer, F., Klein, R., Lee, D.T.: Voronoi diagrams and Delaunay triangulations. World Scientific Publishing Company (2013)
4. Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., Lévy, B.: Polygon mesh processing. CRC press (2010)
5. Brandt, J.W., Algazi, V.R.: Continuous skeleton computation by voronoi diagram. CVGIP: Image understanding **55**(3), 329–338 (1992)
6. Cao, T.T., Nanjappa, A., Gao, M., Tan, T.S.: A gpu accelerated algorithm for 3d delaunay triangulation. In: Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. pp. 47–54 (2014)
7. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: Tensorf: Tensorial radiance fields. In: European Conference on Computer Vision (ECCV) (2022)
8. Chen, A., Xu, Z., Wei, X., Tang, S., Su, H., Geiger, A.: Dictionary fields: Learning a neural basis decomposition. ACM Trans. Graph. (2023)
9. Chen, W., Ling, H., Gao, J., Smith, E., Lehtinen, J., Jacobson, A., Fidler, S.: Learning to predict 3d objects with an interpolation-based differentiable renderer. Advances in neural information processing systems **32** (2019)
10. Chen, Z., Tagliasacchi, A., Funkhouser, T., Zhang, H.: Neural dual contouring. ACM Transactions on Graphics (TOG) **41**(4), 1–13 (2022)
11. Chen, Z., Tagliasacchi, A., Zhang, H.: Bsp-net: Generating compact meshes via binary space partitioning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 45–54 (2020)
12. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G., et al.: Meshlab: an open-source mesh processing tool. In: Eurographics Italian chapter conference. vol. 2008, pp. 129–136. Salerno, Italy (2008)
13. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. pp. 303–312 (1996)
14. Deitke, M., Schwenk, D., Salvador, J., Weihs, L., Michel, O., VanderBilt, E., Schmidt, L., Ehsani, K., Kembhavi, A., Farhadi, A.: Objaverse: A universe of annotated 3d objects. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13142–13153 (2023)
15. Diazzi, L., Panozzo, D., Vaxman, A., Attene, M.: Constrained delaunay tetrahedrization: A robust and practical approach. ACM Transactions on Graphics (TOG) **42**(6), 1–15 (2023)

16. Guillard, B., Remelli, E., Lukoianov, A., Richter, S.R., Bagautdinov, T., Baque, P., Fua, P.: Deepmesh: Differentiable iso-surface extraction. arXiv preprint arXiv:2106.11795 (2021)
17. Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., Cohen-Or, D.: Meshcnn: a network with an edge. ACM Transactions on Graphics (ToG) **38**(4), 1–12 (2019)
18. Heming, Z., Yu, C., Hang, J., Weikai, C., Dong, D., Zhangye, W., Shuguang, C., Xiaoguang, H.: Deep fashion3d: A dataset and benchmark for 3d garment reconstruction from single images. In: Computer Vision – ECCV 2020. pp. 512–530. Springer International Publishing (2020)
19. Jamin, C., Pion, S., Teillaud, M.: 3D triangulations. In: CGAL User and Reference Manual. CGAL Editorial Board, 5.6 edn. (2023), `https://doc.cgal.org/5.6/Manual/packages.html#PkgTriangulation3`
20. Kazhdan, M., Hoppe, H.: Screened poisson surface reconstruction. ACM Transactions on Graphics (ToG) **32**(3), 1–13 (2013)
21. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics **42**(4) (2023)
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Laine, S., Hellsten, J., Karras, T., Seol, Y., Lehtinen, J., Aila, T.: Modular primitives for high-performance differentiable rendering. ACM Transactions on Graphics (TOG) **39**(6), 1–14 (2020)
24. Lee, J.: Introduction to topological manifolds, vol. 202. Springer Science & Business Media (2010)
25. Liao, Y., Donne, S., Geiger, A.: Deep marching cubes: Learning explicit surface representations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2916–2925 (2018)
26. Liu, L., Gu, J., Lin, K.Z., Chua, T.S., Theobalt, C.: Neural sparse voxel fields. NeurIPS (2020)
27. Liu, S., Li, T., Chen, W., Li, H.: Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7708–7717 (2019)
28. Liu, Y.T., Wang, L., Yang, J., Chen, W., Meng, X., Yang, B., Gao, L.: Neudf: Leaning neural unsigned distance fields with volume rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 237–247 (2023)
29. Long, X., Lin, C., Liu, L., Liu, Y., Wang, P., Theobalt, C., Komura, T., Wang, W.: Neuraludf: Learning unsigned distance fields for multi-view reconstruction of surfaces with arbitrary topologies. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 20834–20843 (2023)
30. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: Seminal graphics: pioneering efforts that shaped the field, pp. 347–353 (1998)
31. Maruani, N., Klokov, R., Ovsjanikov, M., Alliez, P., Desbrun, M.: Voromesh: Learning watertight surface meshes with voronoi diagrams. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 14565–14574 (2023)
32. Mehta, I., Chandraker, M., Ramamoorthi, R.: A level set theory for neural implicit evolution under explicit flows. In: European Conference on Computer Vision. pp. 711–729. Springer (2022)
33. Melkemi, M.: A-shapes of a finite point set. In: Proceedings of the thirteenth annual symposium on Computational geometry. pp. 367–369 (1997)

34. Melkemi, M., Djebali, M.: Weighted a-shape: a descriptor of the shape of a point set. Pattern Recognition **34**(6), 1159–1170 (2001)
35. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM **65**(1), 99–106 (2021)
36. Munkberg, J., Hasselgren, J., Shen, T., Gao, J., Chen, W., Evans, A., Müller, T., Fidler, S.: Extracting triangular 3d models, materials, and lighting from images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8280–8290 (2022)
37. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? Queue **6**(2), 40–53 (2008)
38. Nicolet, B., Jacobson, A., Jakob, W.: Large steps in inverse rendering of geometry. ACM Transactions on Graphics (TOG) **40**(6), 1–13 (2021)
39. Oechsle, M., Peng, S., Geiger, A.: Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In: International Conference on Computer Vision (ICCV) (2021)
40. Palfinger, W.: Continuous remeshing for inverse rendering. Computer Animation and Virtual Worlds **33**(5), e2101 (2022)
41. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
42. Rakotosaona, M.J., Aigerman, N., Mitra, N.J., Ovsjanikov, M., Guerrero, P.: Differentiable surface triangulation. ACM Transactions on Graphics (TOG) **40**(6), 1–13 (2021)
43. Shen, T., Gao, J., Yin, K., Liu, M.Y., Fidler, S.: Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. Advances in Neural Information Processing Systems **34**, 6087–6101 (2021)
44. Shen, T., Munkberg, J., Hasselgren, J., Yin, K., Wang, Z., Chen, W., Gojcic, Z., Fidler, S., Sharp, N., Gao, J.: Flexible isosurface extraction for gradient-based mesh optimization. ACM Transactions on Graphics (TOG) **42**(4), 1–16 (2023)
45. Shewchuk, J.R.: Constrained delaunay tetrahedralizations and provably good boundary recovery. IMR **193**, 204 (2002)
46. Si, H.: Constrained delaunay tetrahedral mesh generation and refinement. Finite elements in Analysis and Design **46**(1-2), 33–46 (2010)
47. Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., Wang, W.: Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. arXiv preprint arXiv:2106.10689 (2021)
48. Wang, Y., Han, Q., Habermann, M., Daniilidis, K., Theobalt, C., Liu, L.: Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2023)
49. Wei, X., Xiang, F., Bi, S., Chen, A., Sunkavalli, K., Xu, Z., Su, H.: Neumanifold: Neural watertight manifold reconstruction with efficient and high-quality rendering support. arXiv preprint arXiv:2305.17134 (2023)
50. Yariv, L., Gu, J., Kasten, Y., Lipman, Y.: Volume rendering of neural implicit surfaces. In: Thirty-Fifth Conference on Neural Information Processing Systems (2021)
51. Yariv, L., Kasten, Y., Moran, D., Galun, M., Atzmon, M., Ronen, B., Lipman, Y.: Multiview neural surface reconstruction by disentangling geometry and appearance. Advances in Neural Information Processing Systems **33** (2020)

52. Zhang, K., Riegler, G., Snavely, N., Koltun, V.: Nerf++: Analyzing and improving neural radiance fields. arXiv:2010.07492 (2020)
53. Zhou, Q., Jacobson, A.: Thingi10k: A dataset of 10,000 3d-printing models. arXiv preprint arXiv:1605.04797 (2016)
54. Zhou, Y., Wu, C., Li, Z., Cao, C., Ye, Y., Saragih, J., Li, H., Sheikh, Y.: Fully convolutional mesh autoencoder using efficient spatially varying kernels. Advances in neural information processing systems **33**, 9251–9262 (2020)

Table 3: Traits of different optimization-based shape reconstruction methods.

Methods	Closed	Open	Diff. Mesh	Diff. Render.	Geo. Topo.	Mesh Topo.	Manifold
Template Mesh [38,40]	O	O	O	O	X	X	O
Neural SDF [47,48]	O	X	X	O	O	X	O
Neural UDF [28,29]	O	O	X	O	O	X	△
Diff. Isosurface [36,43,44]	O	X	O	O	O	X	O
DMesh (Ours)	O	O	O	O	O	O	X

6 Comparison to Other Shape Reconstruction Methods

Here we provide conceptual comparisons between our approach and the other optimization-based 3D reconstruction algorithms, which use different shape representations. To be specific, we compared our method with mesh optimization methods starting from template mesh [38, 40], methods based on neural signed distance fields (SDF) [47,48], methods based on neural unsigned distance fields (UDF) [28, 29], and methods based on differentiable isosurface extraction [36,43,44]. We used following criteria to compare these methods.

- Closed surface: Whether or not the given method can reconstruct, or represent closed surfaces.
- Open surface: Whether or not the given method can reconstruct, or represent open surfaces.
- Differentiable Meshing: Whether or not the given method can produce gradients from the loss computed on the final mesh.
- Differentiable Rendering: Whether or not the given method can produce gradients from the loss computed on the rendering results.
- Geometric topology: Whether or not the given method can change geometric topology of the shape. Here, geometric topology defines the continuous deformation of Euclidean subspaces [24]. For instance, genus of the shape is one of the traits that describe geometric topology.
- Mesh topology: Whether or note the given method can produce gradients from the loss computed on the mesh topology, which denotes the structural configuration, or edge connectivity of a mesh.
- Manifoldness: Whether or not the given method guarantees manifold mesh.

In Table 3, we present a comparative analysis of different methods. Note that our method meets all criteria, only except manifoldness. It is partially because our method does not assume volume, which is the same for methods based on neural UDF. However, because our method does not leverage smoothness prior of neural network like those methods, it could exhibit high frequency noises in the final mesh. Because of this reason, we gave △ to the neural UDF methods, while giving X to our approach.

Likewise, DMesh shows promise in addressing the shortcomings found in previous research. Nonetheless, it has its own set of limitations (Section 5). Identifying and addressing these limitations is crucial for unlocking the full potential of our method.

7 Details about Section 3.2

7.1 Mathematical Definitions

Here, we provide formal mathematical definitions of the terms used in Section 3.2. Please refer to [3] for further discussion on this particular topic.

Half Plane Given a set of points $\mathbb{P} \in \mathbb{R}^d$ and their weights $\mathbb{W} \in \mathbb{R}$, we denote an i-th weighted point as (p_i, w_i). Then, we can define a hyperplane $H(i, j)$, which we call as a half plane, that divides the domain into two half spaces $H_{i<j}$ and $H_{j<i}$ that satisfy the following relationship:

$$H_{i<j} = \{x \in \mathbb{R}^d \,|\, ||x - p_i||^2 - w_i \leq ||x - p_j||^2 - w_j\}.$$

Note that this half plane becomes an infinite line when $d = 2$, and an infinite plane when $d = 3$.

Power Cell Based on this definition, the Power cell C_i of a point p_i can be described as an intersection of half spaces between p_i and the other points p_j.

$$C_i = \{x \in \mathbb{R}^d \,|\, x \in H_{i<j}, \forall j \neq i\}.$$

Note that there could be no Power cell C_i for a certain point p_i if its weight w_i is relatively smaller than those of its neighboring points.

Reduced Power Cell As mentioned in Section 3.2, we adopt the concept of "reduced" Power cell from [42]. Unlike the ordinary Power cell, reduced Power cell depends on the face F. That is, if we define $F = \{p_1, p_2, ..., p_d\}$, the reduced Power cell $R_{F|i}$ for a point $p_i \in F$ is defined as

$$R_{F|i} = \{x \in \mathbb{R}^d \,|\, x \in H_{i<j}, \forall j \notin F\}.$$

Therefore, $R_{F|i}$ can vary depending on F. We consider only the points not included in F to define this reduced Power cell, as opposed to considering all points other than p_i. Consequently, the following relationship holds:

$$C_i \subseteq R_{F|i}, \forall F.$$

Dual Line Now, for a given $(d-1)$-simplex face $F = \{p_1, p_2, ..., p_d\}$, we can construct its dual line L_F as an intersection of the half spaces that are defined by the points of F.

$$L_F = \bigcap_{(i,j) \in F} H(i, j).$$

Note that L_F becomes an infinite line in both $d = 2$ and $d = 3$ cases. As discussed in Section 3.2, when the face F exists in WDT, a subset of L_F exists as an edge in Power diagram. This relationship forms the bases of our differentiable formulation.

Differentiable Min Operation In Section 3.2, we used a differentiable min operator *dmin* to define Λ_{real}. Formally, we define *dmin* as follows.

$$dmin(x_1, x_2, ..., x_k, \alpha) = \frac{\sum_{i=1,...,k} \exp(-x_i \cdot \alpha) \cdot x_i}{\sum_{i=1,...,k} \exp(-x_i \cdot \alpha)}.$$

Note that it becomes more non-differentiable as α becomes larger.

7.2 Differentiable Power Cell Existence

Even though our formulation in Section 3.2 is differentiable, we wish to highlight a potential issue here.

According to our formulation, there is a scenario in which a point might lose its Power cell because its updated weight becomes relatively smaller compared to those of its neighbors. In such a case, the existence probability of a face containing that point may abruptly decrease from a value greater than 0.5 to 0, which is not desirable.

To address this problem, we can introduce an additional constraint regarding the existence of Power cells. Specifically, a Power cell C_i exists if and only if there is a face F that includes the point p_i. Consequently, the following relationship is established:

$$\exists C_i \Leftrightarrow \exists F \in \mathbb{I}_{wdt}(F) \Leftrightarrow \delta(L_F, R_{F|i}) > 0.$$

Based on this observation, we can define a differentiable Power cell existence probability function, which is parameterized by a threshold ϵ_{pc} and sigmoid parameter α_{pc},

$$\Lambda_{pc}(C_i) = \sigma(\alpha_{pc} \cdot (\sum_{F \in \bar{F}} \delta(L_F, R_{F|i}) - \epsilon_{pc})),$$

where $\bar{F}$ is a set of faces that has point p_i. Using this function, we can newly define Λ_{wdt} in Eq. 6 for a triangular face F_3 as

$$\Lambda^*_{wdt}(F_3) = \Lambda_{wdt}(F_3) \cdot \min(\Lambda_{pc}(C_i), \Lambda_{pc}(C_j), \Lambda_{pc}(C_k)). \tag{10}$$

Here, we used the min operation because F_3 ceases to exist when any of its points loses its Power cell. With this operation, the probability function becomes fully differentiable, even when points gain or lose their Power cells during the optimization process.

However, when we implemented this new formulation and compared it with the existing one, we observed no significant difference between the two. We hypothesize that this is because, in most cases, faces that need to exist are updated to increase Eq. 6, which, in turn, reinforces the existence of Power cells for the points constituting the face. Therefore, although we omitted this formulation in the final version, we introduce it here to ensure the completeness of our discussion.

8 Loss Functions

Here we provide formal definitions for the loss functions that we use in the paper.

8.1 Mesh to DMesh

In this section, we explore the loss function used to transform the ground truth mesh into our DMesh representation. As previously mentioned in Section 3.4, the explicit definition of ground truth connectivity in the provided mesh allows us to establish a loss function based on it.

Building on the explanation in Section 3.4, if the ground truth mesh consists of vertices $\mathbb{P}$ and faces $\mathbb{F}$, we can construct an additional set of faces $\bar{\mathbb{F}}$. These faces are formed from vertices in $\mathbb{P}$ but do not intersect with faces in $\mathbb{F}$.

$$\bar{\mathbb{F}} = \mathbb{F}^* - \mathbb{F}, \text{ where } \mathbb{F}^* = \text{ every possible face combination on } \mathbb{P}.$$

Then, we notice that we should maximize the existence probabilities of faces in $\mathbb{F}$, but minimize those of faces in $\bar{\mathbb{F}}$. Therefore, we can define our reconstruction loss function as

$$L_{recon} = -\sum_{F \in \mathbb{F}} \Lambda(F) + \sum_{F \in \bar{\mathbb{F}}} \Lambda(F). \tag{11}$$

If the first term of the loss function mentioned above is not fully optimized, it could lead to the omission of ground truth faces, resulting in a poorer recovery ratio (Section 4.1). Conversely, if the second term is not fully optimized, the resulting DMesh might include faces absent in the ground truth mesh, leading to a higher false positive ratio (Section 4.1). Refer to Appendix 9.1 for details on how this reconstruction loss is integrated into the overall optimization process.

8.2 Point Cloud Reconstruction

In the task of point cloud reconstruction, we reconstruct the mesh by minimizing the (L_1-norm based) expected Chamfer Distance (CD) between the given point cloud ($\mathbb{P}_{gt}$) and the sample points ($\mathbb{P}_{ours}$) from our reconstructed mesh. We denote the CD from $\mathbb{P}_{gt}$ to $\mathbb{P}_{ours}$ as CD_{gt}, and the CD from $\mathbb{P}_{ours}$ to $\mathbb{P}_{gt}$ as CD_{ours}. The final reconstruction loss is obtained by combining these two distances.

$$L_{recon} = CD_{gt} + CD_{ours}. \tag{12}$$

Sampling $\mathbb{P}_{ours}$ To compute these terms, we start by sampling $\mathbb{P}_{ours}$ from our current mesh. First, we sample a set of faces that we will sample points from. We consider the areas of the triangular faces and their existence probabilities. To be specific, we define $\eta(F)$ for a face F as

$$\bar{\eta}(F) = \Lambda(F), \quad \eta(F) = F_{area} \cdot \bar{\eta}(F),$$

and define a probability to sample F from the entires faces $\mathbb{F}$ as

$$P_{sample}(F) = \frac{\eta(F)}{\sum_{F' \in \mathbb{F}} \eta(F')}.$$

We sample N faces from $\mathbb{F}$ with replacement and then uniformly sample a single point from each selected face to define $\mathbb{P}_{ours}$. In our experiments, we set N to $100K$.

In this formulation, we sample more points from faces with a larger area and higher existence probability to improve sampling efficiency. However, we observed that despite these measures, the sampling efficiency remains low, leading to slow convergence. This issue arises because, during optimization, there is an excessive number of faces with very low existence probability.

To overcome this limitation, we decided to do stratified sampling based on point-wise real values and cull out faces with very low existence probabilities. To be specific, we define two different η functions:

$$\begin{aligned} \bar{\eta_1}(F) &= \Lambda_{wdt}(F) \cdot \min(\psi_i, \psi_j, \psi_k), \quad \eta_1(F) = F_{area} \cdot \bar{\eta_1}(F) \\ \bar{\eta_2}(F) &= \Lambda_{wdt}(F) \cdot \max(\psi_i, \psi_j, \psi_k), \quad \eta_2(F) = F_{area} \cdot \bar{\eta_2}(F) \end{aligned}$$

where (ψ_i, ψ_j, ψ_k) are the real values of the points that comprise F. Note that η_1 is the same as η [8].

For the faces in $\mathbb{F}$, we first calculate the $\bar{\eta_1}$ and $\bar{\eta_2}$ values and eliminate faces with values lower than a predefined threshold ϵ_η. We denote the set of remaining faces as $\mathbb{F}_1$ and $\mathbb{F}_2$. Subsequently, we sample $\frac{N}{2}$ faces from $\mathbb{F}_1$ and the other $\frac{N}{2}$ faces from $\mathbb{F}_2$, using the following two sampling probabilities:

$$P_{sample,1}(F) = \frac{\eta_1(F)}{\sum_{F' \in \mathbb{F}_1} \eta_1(F')}, \quad P_{sample,2}(F) = \frac{\eta_2(F)}{\sum_{F' \in \mathbb{F}_2} \eta_2(F')}.$$

The rationale behind this sampling strategy is to prioritize (non-existing) faces closer to the current mesh over those further away. In the original $\eta = \eta_1$ function, we focus solely on the minimum real value, leading to a higher sampling rate for existing faces. However, to remove holes in the current mesh, it's beneficial to sample more points from potential faces—those not yet existing but connected to existing ones. This approach, using η_2, enhances reconstruction results by removing holes more effectively. Yet, there's substantial potential to refine this importance sampling technique, as we haven't conducted a theoretical analysis in this study.

Moreover, when sampling a point from a face, we record the face's existence probability alongside the point. Additionally, if necessary, we obtain and store the face's normal. For a point $\mathbf{p} \in \mathbb{P}_{ours}$, we introduce functions $\Lambda_{pt}(\cdot)$ and $Normal(\cdot)$ to retrieve the face existence probability and normal, respectively:

[8] We do not use differentiable min operator, as we do not require differentiability in the sampling process.

$$\Lambda_{pt}(\mathbf{p}) = \Lambda(F(\mathbf{p})), \quad Normal(\mathbf{p}) = F(\mathbf{p})_{normal},$$
$$F(\mathbf{p}) = \text{the face where } \mathbf{p} \text{ was sampled from.}$$

CD_{gt} Now we introduce how we compute the CD_{gt}, which is CD from $\mathbb{P}_{gt}$ to $\mathbb{P}_{ours}$. For each point $\mathbf{p} \in \mathbb{P}_{gt}$, we first find k-nearest neighbors of $\mathbf{p}$ in $\mathbb{P}_{ours}$, which we denote as $(p_1, p_2, ..., p_k)$. Then, we define a distance function between the point $\mathbf{p}$ and the k-nearest neighbors as follows, to accommodate the orientation information:

$$\begin{aligned} \bar{D}(\mathbf{p}, p_i) &= ||\mathbf{p} - p_i||_2 + \lambda_{normal} \cdot \bar{D}_n(\mathbf{p}, p_i), \\ \text{where } \bar{D}_n(\mathbf{p}, p_i) &= 1 - | < \mathbf{p}_{normal}, Normal(p_i) > |, \end{aligned} \tag{13}$$

where λ_{normal} is a parameter than determines the importance of point orientation in reconstruction. If $\lambda_{normal} = 0$, we only consider the positional information of the sampled points.

After we evaluate the above distance function values for the k-nearest points, we reorder them in ascending order. Then, we compute the following expected minimum distance from $\mathbf{p}$ to $\mathbb{P}_{ours}$,

$$\begin{aligned} D(\mathbf{p}, \mathbb{P}_{ours}) &= \sum_{i=1,...,k} \bar{D}(\mathbf{p}, p_i) \cdot P(p_i) \cdot \bar{P}(p_i), \\ P(p_i) &= \Lambda_{pt}(p_i) \cdot \mathbb{I}_{prev}(F(p_i), \\ \bar{P}(p_i) &= \Pi_{i=1,...,k-1}(1 - P(p_i)), \end{aligned}$$

where $\mathbb{I}_{prev}$ is an indicator function that returns 1 only when the given face has not appeared before in computing the above expected distance. For instance, if the face ids for the reordered points were $(1, 2, 3, 2, 3, 4)$, the $\mathbb{I}_{prev}$ function evaluates to $(1, 1, 1, 0, 0, 1)$. This indicator function is needed, because if we select p_i as the nearest point to $\mathbf{p}$ with the probability $\Lambda_{pt}(\mathbf{p})$, it means that we interpret that the face corresponding to p_i already exists, and then we would select p_i on the face as the nearest point to $\mathbf{p}$ rather than the other points that were sampled from the same face, but have larger distance than p_i and thus come after p_i in the ordered points.

Note that we dynamically change k during runtime to get a reliable estimation of $D(\mathbf{p}, \mathbb{P}_{ours})$. That is, for current k, if most of $\bar{P}(p_k)$s for the points in $\mathbb{P}_{gt}$ are still large, it means that there is a chance that the estimation could change a lot if we find and consider more neighboring points. Therefore, in our experiments, if any point in $\mathbb{P}_{gt}$ has $\bar{P}(p_k)$ larger than 10^{-4}, we increase k by 1 for the next iteration. However, if there is no such point, we decrease k by 1 to accelerate the optimization process.

Finally, we can compute CD_{gt} by summing up the point-wise expected minimum distances.

$$CD_{gt} = \sum_{\mathbf{p} \in \mathbb{P}_{gt}} D(\mathbf{p}, \mathbb{P}_{ours}).$$

CD_{ours} In computing CD_{ours}, which is CD from $\mathbb{P}_{ours}$ to $\mathbb{P}_{gt}$, we also find k-nearest neighbors for each point $\mathbf{p} \in \mathbb{P}_{ours}$, which we denote as $(p_1, p_2, ..., p_k)$. Then, for a point $\mathbf{p}$, we use the same distance function $\bar{D}$ in Eq. 13 to find the distance between $\mathbf{p}$ and $(p_1, p_2, ..., p_k)$. After that, we select the minimum one for each point, multiply the existence probability of each point, and then sum them up to compute CD_{ours}.

$$D(\mathbf{p}, \mathbb{P}_{gt}) = \min_{i=1,\dots,k} \bar{D}(\mathbf{p}, p_i),$$

$$CD_{ours} = \sum_{\mathbf{p} \in \mathbb{P}_{ours}} \Lambda_{pt}(\mathbf{p}) \cdot D(\mathbf{p}, \mathbb{P}_{gt}).$$

Finally, we can compute the final reconstruction loss for point clouds as shown in Eq. 12.

8.3 Multi-View Reconstruction

When we are given multi-view images, we reconstruct the mesh by minimizing the L_1 difference between our rendered images and the given images. In this work, we mainly use both diffuse and depth renderings to reconstruct the mesh.

If we denote the (N_{img}) ground truth images of N_{pixel} number of pixels as $\mathcal{I}_i^{gt} (i = 1, ..., N_{img})$, and our rendered images as $\mathcal{I}_i^{ours}$, we can write the reconstruction loss function as

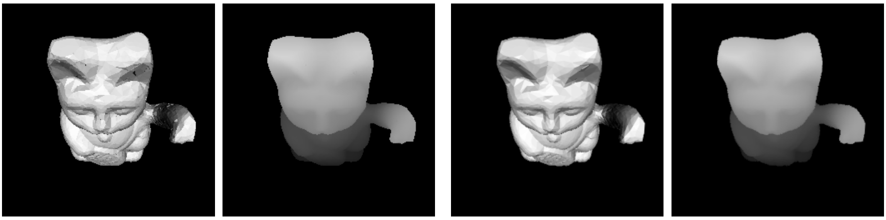
$$L_{recon} = \frac{1}{N_{img} \cdot N_{pixel}} \sum_{i=1,\dots,N_{img}} ||\mathcal{I}_i^{gt} - \mathcal{I}_i^{ours}||.$$

Then, we can define our rendered image as follows:

$$I_i^{ours} = \mathcal{F}(\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), \mathbf{MV_i}, \mathbf{P_i}).$$

where $\mathcal{F}$ is a differentiable renderer that renders the scene for the given points $\mathbb{P}$, faces $\mathbb{F}$, face existence probabilities $\Lambda(\mathbb{F})$, i-th modelview matrix $\mathbf{MV_i} \in \mathbb{R}^{4\times4}$, and i-th projection matrix $\mathbf{P_i} \in \mathbb{R}^{4\times4}$. The differentiable renderer $\mathcal{F}$ has to backpropagate gradients along $\mathbb{P}$, $\mathbb{F}$, and $\Lambda(\mathbb{F})$ to update our point attributes. Specifically, here we interpret $\Lambda(\mathbb{F})$ as opacity for faces to use in the rendering process. This is because opacity means the probability that a ray stops when it hits the face, which aligns with our face existence probability well. For this reason, we ignore faces with very low existence probability under some threshold to accelerate the reconstruction, as they are almost transparent and do not contribute to the rendering a lot.

To implement $\mathcal{F}$, we looked through previous works dedicated for differentiable rendering [23, 27]. However, we discovered that these methods incur substantial computational costs when rendering a large number of (potentially) semi-transparent triangles, as is the case in our scenario. Consequently, we developed two efficient, partially differentiable renderers that meet our specific requirements. These renderers fulfill distinct roles within our pipeline—as detailed in Appendix 9, our optimization process encompasses two phases within a single epoch. The first renderer is employed during the initial phase, while the second renderer is utilized in the subsequent phase.



(a) Rendered Images from $\mathcal{F}_A$ **(b)** Rendered Images from $\mathcal{F}_{A'}$

Fig. 10: Rendered images from two differentiable renderers, $\mathcal{F}_A$ and $\mathcal{F}_{A'}$. Left and right image corresponds to diffuse and depth rendering, respectively. (a) $\mathcal{F}_A$ is our (partially) differentiable renderer based on tile-based approach. (b) Since $\mathcal{F}_A$ does not produce visibility-related gradients, we additionally use $\mathcal{F}_{A'}$ [23] to render images and integrate with ours.

$\mathcal{F}_A$ If there are multiple semi-transparent faces in the scene, we have to sort the faces that covers a target pixel with their (view-space) depth values, and iterate through them until the accumulated transmittance is saturated to determine the color for the pixel. Conducting this process for each individual pixel is not only costly, but also requires a lot of memory to store information for backward pass.

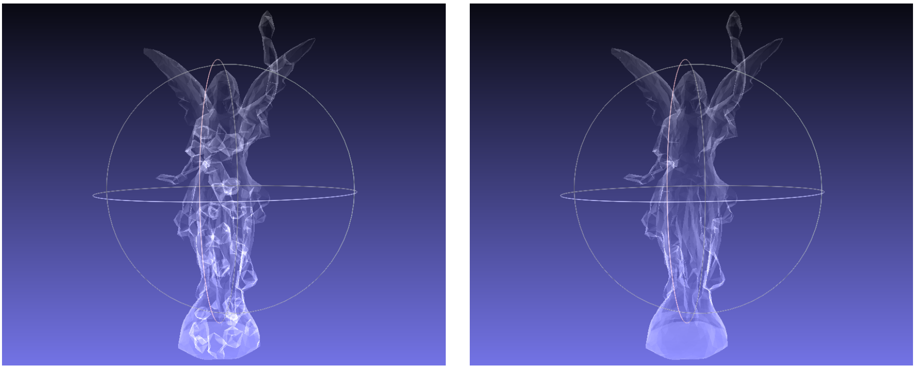
Recently, 3D Gaussian Splatting [21] overcame this issue with tile-based rasterizer. We adopted this approach, and modified their implementation to render triangular faces, instead of gaussian splats. To briefly introduce its pipeline, it first assigns face-wise depth value by computing the view-space depth of its center point. Then, after subdividing the entire screen into 16 × 16 tiles, we assign faces to each tiles if they overlap. After that, by using the combination of tile ID and the face-wise depth as a key, we get the face list sorted by depth value in each tile. Finally, for each tile, we iterate through the sorted faces and determine color and depth for each pixel as follows.

$$C = \sum_{i=1,\dots,k} T_i \cdot \ \alpha_i \cdot C_i, \quad (T_i = \varPi_{j=1,\dots,i-1}(1-\alpha_j)),$$

where T_i is the accumulated transmittance, α_i is the opacity of the i-th face, and C_i is the color (or depth) of the i-th face. Note that $\alpha_i = \varLambda(F_i)$, as mentioned above.

Even though this renderer admits an efficient rendering of large number of semi-transparent faces, there are still two large limitations in the current implementation. First, the current implementation does not produce visibility-related gradients (near face edges) to update point attributes. Therefore, we argue that this renderer is partially differentiable, rather than fully differentiable. Next, since it does not compute precise view-point depth for each pixel, its rendering result can be misleading for some cases, as pointed out in [21].

To amend the first issue, we opt to use another differentiable renderer of [23], which produces the visibility-related gradients that we lack. Since this renderer cannot render (large number of) transparent faces as ours does, we only render



(a) Extracted Mesh after phase 1

(b) Extracted Mesh after phase 2

Fig. 12: Reconstructed mesh from multi-view images, rendered in MeshLab's [12] x-ray mode to see inner structure. In multi-view reconstruction, we divide each epoch in two phases. (a) After the first phase ends, where we do inaccurate depth testing, lots of false inner faces are created. (b) To remove these inner faces, we require a renderer that does the exact depth testing, which we use in the second phase. Also see Appendix 9.2 for details about post-processing step to remove the inner structure.

the faces with opacity larger than 0.5. Also, we set the faces to be fully opaque. If we call this renderer as $\mathcal{F}_{A'}$, our final rendered image can be written as follows.

$$\mathcal{I}_i^{ours} = \frac{1}{2}(\mathcal{F}_A(\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), \mathbf{MV}_i, \mathbf{P}_i) + \mathcal{F}_{A'}(\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), \mathbf{MV}_i, \mathbf{P}_i)).$$

In Figure 10, we illustrate rendered images from $\mathcal{F}_A$ and $\mathcal{F}_{A'}$.

Acknowledging that this formulation is not theoretically correct, we believe that it is an intriguing future work to implement a fully differentiable renderer that works for our case. However, we empirically found out that we can reconstruct a wide variety of meshes with current formulation without much difficulty.

As mentioned before, this renderer is used at the first phase of the optimization process, where all of the point attributes are updated. However, in the second phase, we fix the point positions and weights, and only update point-wise real values (Appendix 9.2). In this case, we can leverage the tessellation structure to implement an efficient differentiable renderer. As the second renderer does a precise depth testing unlike the first one, it can be used to modify the errors incurred by the second limitation of the first renderer (Figure 12).

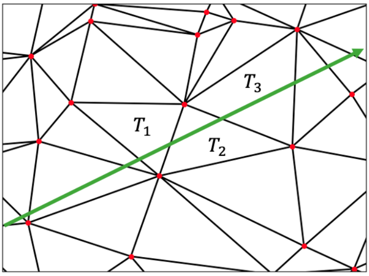


Fig. 11: $\mathcal{F}_B$ uses tessellation structure to efficiently render overlapped faces in the correct order.

$\mathcal{F}_B$ The second renderer performs precise depth ordering in an efficient way, based on the fixed tessellation structure that we have. In Figure 11, we illustrate a 2D diagram that explains our approach. When the green ray, which corresponds to a single ray to determine the color of a single pixel, goes through the tessellation, we can observe that it goes through a sequence of triangles (tetrahedron in 3D), which are denoted as T_1, T_2, and T_3. When the ray enters a triangle T_i through one of its three edges, we can see that it moves onto the other adjacent triangle T_{i+1} only through one of the other edges of T_i, because of compact tessellation. Therefore, when the ray hits one edge of T_i, it can only examine the other two edges of T_i to find the next edge it hits. Note that we do not have to do depth testing explicitly in this approach. Also, unlike the first approach, this renderer does not have to store all the possible faces that a ray collides for the backward pass, because it can iterate the same process in the opposite way in the backward pass to find the edge that it hit before the last edge. If we only store the last edge that each hits at the forward pass, we can start from the last edge and find the previous edges that it hit to compute gradients. Therefore, this second renderer requires much less memory than the first one, and also performs precise depth testing naturally. However, note that this renderer is also partilly differentiable, because it cannot update point positions and weights.

To sum up, we implemented two partially differentiable renderers to solve multi-view reconstruction problem with DMesh. They serve different objectives in our reconstruction process, and we empirically found out that they are powerful enough to reconstruct target meshes in our experiments. However, we expect that we can simplify the process and improve its stability, if we can implement a fully differentiable renderer that satisfy our needs. We leave it as a future work.

8.4 Weight Regularization

Weight regularization aims at reducing the complexity of WDT, which supports our mesh. By using this regularization, we can discard unnecessary points that do not contribute to representing our mesh. Moreover, we can reduce the number of points on the mesh, if they are redundant, which ends up in the mesh simplification effect (Appendix 10.3).

We formulate the complexity of WDT as the sum of edge lengths in its dual Power diagram. Formally, we can write the regularization as follows,

$$L_{weight} = \sum_{i=1,\ldots,N} Length(E_i),$$

where E_i are the edges in the dual Power diagram, and N is the number of edges.

8.5 Real Regularization

Real regularization is a regularization that is used for maintaining the real values of the connected points in WDT as similar as possible. Also, we leverage this regularization to make real values of points that are connected to the points with high real values to become higher, so that they can be considered in reconstruction more often than the points that are not connected to those points. To be specific, note that we ignore faces with very low existence probability in the reconstruction process. By using this regularization, it can remove holes more effectively.

This real regularzation can be described as

$$\begin{aligned}
L_{real} &= \frac{1}{\sum_{i=1,...,N} \Lambda(F_i)} \sum_{i=1,...,N} \Lambda(F_i) \cdot (\sigma_1(F_i) + \sigma_2(F_i)), \\
\sigma_1(F_i) &= \frac{1}{3} \sum_{j=1,2,3} |\psi_j - \frac{(\psi_1 + \psi_2 + \psi_3)}{3}|, \\
\sigma_1(F_i) &= \frac{1}{3} \sum_{j=1,2,3} |1 - \psi_j| \cdot \mathbb{I}(\max_{j=1,2,3}(\psi_j) > \delta_{high}).
\end{aligned}$$

Here $\psi_{1,2,3}$ represent the real values of points that comprise F_i, and δ_{high} is a threshold to determine "high" real value, which is set as 0.8 in our experiments. Note that the faces with higher existence probabilities are prioritized over the others.

8.6 Quality Regularization

After reconstruction, we usually want to have a mesh that is comprised of triangles of good quality, rather than ill-formed triangles. We adopt the aspect ratio as a quality measure for the triangular faces, and minimize the sum of aspect ratios for all faces during optimization to get a mesh of good quality. Therefore, we can write the regularization as follows.

$$\begin{aligned}
L_{qual} &= \frac{1}{\sum_{i=1,...,N} \Lambda(F_i)} \sum_{i=1,...,N} AR(F_i) \cdot E_{max}(F_i) \cdot \Lambda(F_i), \\
AR(F_i) &= \frac{E_{max}(F_i)}{H_{min}(F_i)} \cdot \frac{\sqrt{3}}{2}, \\
E_{max}(F_i) &= \text{Maximum edge length of } F_i, \\
H_{min}(F_i) &= \text{Minimum height of } F_i.
\end{aligned}$$

Note that we prioritize faces with larger maximum edge length and higher existence probability than the others in this formulation. In Appendix 10.3, we provide ablation studies for this regularization.

9 Optimization Process

In this section, we explain the optimization processes, or exact reconstruction algorithms, in detail. First, we discuss the optimization process for the experiment in Section 4.1, where we represent the ground truth mesh with DMesh.

Algorithm 1 Mesh to DMesh

```
$\mathbb{P}_{gt}, \mathbb{F}_{gt} \leftarrow$ Ground truth mesh vertices and faces
$\mathbb{P}, \mathbb{W}, \psi \leftarrow$ Initialize point attributes for DMesh
$\bar{\mathbb{F}} \leftarrow$ Empty set of faces
while Optimization not ended do
    $\mathbb{P}, \mathbb{W}, \psi \leftarrow$ Do point insertion, with $\mathbb{P}, \bar{\mathbb{F}}$
    $WDT, PD \leftarrow$ Run WDT algorithm, with $\mathbb{P}, \mathbb{W}$
    $\bar{\mathbb{F}} \leftarrow$ Update faces to exclude, with $WDT$
    $\Lambda(\mathbb{F}_{gt}), \Lambda(\bar{\mathbb{F}}) \leftarrow$ Compute existence probability for faces, with $\mathbb{P}, \psi, WDT, PD$
    $L_{recon} \leftarrow$ Compute reconstruction loss, with $\Lambda(\mathbb{F}_{gt}), \Lambda(\bar{\mathbb{F}})$
    Update $\mathbb{P}, \mathbb{W}, \psi$ to minimize $L_{recon}$
    Bound $\mathbb{P}$
end
$M \leftarrow$ Get final mesh from DMesh
```

Then, we discuss the overall optimization process for point cloud or multi-view reconstruction tasks in Section 4.2, from initialization to post processing.

9.1 Mesh to DMesh

Our overall algorithm to convert the ground truth mesh into DMesh is outlined in Algorithm 1. We explain each step in detail below.

Point Initialization At the start of optimization, we initialize the point positions ($\mathbb{P}$), weights ($\mathbb{W}$), and real values (ψ) using the given ground truth information ($\mathbb{P}_{gt}$, $\mathbb{F}_{gt}$). To be specific, we initialize the point attributes as follows.

$$\mathbb{P} = \mathbb{P}_{gt}, \quad \mathbb{W} = [1, ..., 1], \quad \psi = [1, ..., 1].$$

The length of vector $\mathbb{W}$ and ψ is equal to the number of points. In Figure 13, we illustrate the initialized DMesh using these point attributes, which becomes the convex hull of the ground truth mesh.

Note that during optimization, we allow only small perturbations to the positions of initial points, and fix weights and real values of them to 1. This is because we already know that these points correspond to the ground truth mesh vertices, and thus should be included in the final mesh without much positional difference. In our experiments, we set the perturbation bound as 1% of the model size.

However, we notice that we cannot restore the mesh connectivity with only small perturbations to the initial point positions, if there are no additional points that can aid the process. Therefore, we periodically perform point insertion to add additional points, which is described below.

Point Insertion The point insertion is a subroutine to add additional points to the current point configurations. It is performed periodically, at every fixed

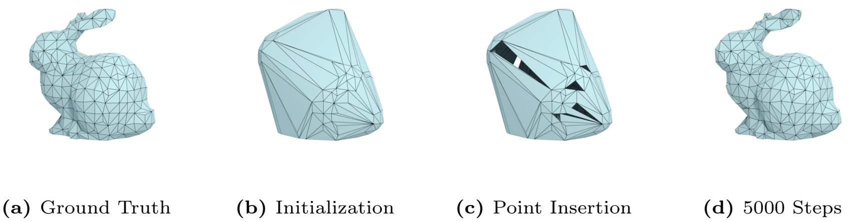

(a) Ground Truth (b) Initialization (c) Point Insertion (d) 5000 Steps

Fig. 13: Intermediate results in converting bunny model to DMesh. For given ground truth mesh in (a), we initialize our point attributes using the mesh vertices. (b) Then, the initial mesh becomes convex hull of the original mesh. (c) To remove undesirable faces that were not in the original mesh, we insert additional points on the undesirable faces. Then, some of them disappear because of the inserted points. (d) After optimizing 5000 steps, just before another point insertion, DMesh recovers most of the ground truth connectivity.

step. The additional points are placed at the random place on the faces in $\bar{\mathbb{F}}$, which correspond to the faces that should not exist in the final mesh. Therefore, these additional points can aid removing these undesirable faces.

However, we found out that inserting a point for every face in $\bar{\mathbb{F}}$ can be quite expensive. Therefore, we use k-means clustering algorithm to aggregate them into $0.1 \cdot N_F$ clusters, where N_F is the number of faces in $\bar{\mathbb{F}}$, to add the centroids of the clusters to our running point set. On top of that, we select 1000 random faces in $\bar{\mathbb{F}}$ to put additional points directly on them. This is because there are cases where centroids are not placed on the good positions where they can remove the undesirable faces.

In Figure 13, we render DMesh after point insertion to the initialized mesh. Note that some of the undesirable faces disappear because of the added points.

Maintaining $\bar{\mathbb{F}}$ In this problem, we minimize the reconstruction loss specified in Eq. 11 to restore the connectivity in the ground truth mesh, and remove faces that do not exist in it. In the formulation, we denoted the faces that are comprised of mesh vertices $\mathbb{P}$, but are not included in the original mesh as $\bar{\mathbb{F}}$. Even though we can enumerate all of them, the total number of faces in $\bar{\mathbb{F}}$ mounts to $O(N^3)$, where N is the number of mesh vertices. Therefore, rather than evaluating all of those cases, we maintain a set of faces $\bar{\mathbb{F}}$ that we should exclude in our mesh during optimization.

To be specific, at each iteration, we find faces in the current WDT that are comprised of points in $\mathbb{P}$, but do not exist in $\mathbb{F}$, and add them to the running set of faces $\bar{\mathbb{F}}$. On top of that, at every pre-defined number of iterations, in our case 10 steps, we compute k-nearest neighboring points for each point in $\mathbb{P}$. Then, we find faces that can be generated by combining each point with 2 of its k-nearest points, following [42]. Then, we add the face combinations that do not belong to $\mathbb{F}$ to $\bar{\mathbb{F}}$. In our experiments, we set $k = 8$.

Algorithm 2 Point cloud & Multi-view Reconstruction

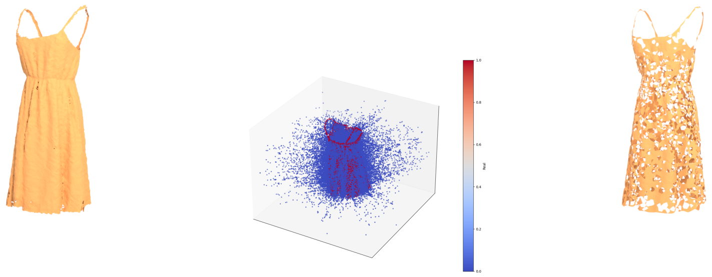
```
$T \leftarrow$ Observation (Point cloud, Multi-view images)
$\mathbb{P}, \mathbb{W}, \psi \leftarrow$ Initialize point attributes for DMesh (using T if possible)
$\mathbb{F} \leftarrow$ Empty set of faces
while epoch not ended do
    $\mathbb{P}, \mathbb{W}, \psi \leftarrow$ (If not first epoch) Initialize point attributes with sample points from
    current DMesh, for mesh refinement
    // Phase 1
    while step not ended do
        $WDT, PD \leftarrow$ Run WDT algorithm with $\mathbb{P}, \mathbb{W}$
        $\mathbb{F} \leftarrow$ Update faces to evaluate existence probability for, with $WDT$
        $\Lambda(\mathbb{F}) \leftarrow$ Compute existence probability for faces in $\mathbb{F}$, with $\mathbb{P}, \psi, WDT, PD$
        $L_{recon} \leftarrow$ Compute reconstruction loss, with $\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), T$
        $L_{weight} \leftarrow$ Compute weight regularization, with $PD$
        $L_{real} \leftarrow$ Compute real regularization, with $\mathbb{P}, \psi, WDT$
        $L_{qual} \leftarrow$ Compute quality regularization, with $\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F})$
        $L \leftarrow L_{recon} + \lambda_{weight} \cdot L_{weight} + \lambda_{real} \cdot L_{real} + \lambda_{qual} \cdot L_{qual}$
        Update $\mathbb{P}, \mathbb{W}, \psi$ to minimize $L$
    end
    // Phase 2
    $WDT, PD \leftarrow$ Run WDT algorithm with $\mathbb{P}, \mathbb{W}$
    $\mathbb{F} \leftarrow$ Faces in $WDT$
    $\Lambda_{wdt}(\mathbb{F}) \leftarrow 1$
    while step not ended do
        $\Lambda(\mathbb{F}) \leftarrow$ Compute existence probability for $\mathbb{F}$, with $\mathbb{P}, \psi, \Lambda_{wdt}(\mathbb{F})$
        $L_{recon} \leftarrow$ Compute reconstruction loss, with $\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), T$
        $L_{real} \leftarrow$ Compute real regularization, with $\mathbb{P}, \psi, WDT$
        $L \leftarrow L_{recon} + \lambda_{real} \cdot L_{real}$
        Update $\psi$ to minimize $L$
    end
end
$M \leftarrow$ Get final mesh from DMesh, after post-processing
```

9.2 Point cloud & Multi-view Reconstruction

In Algorithm 2, we describe the overall algorithm that is used for point cloud and multi-view reconstruction tasks. We explain each step in detail below.

Two Phase Optimization We divide each optimization epoch in two phases. In the first phase (phase 1), we optimize all of the point attributes – positions, weights, and real values. However, in the second phase (phase 2), we fix the point positions and weights, and only optimize the real values.

This design aims at removing ambiguity in our differentiable formulation. That is, even though we desire face existence probabilities to converge to either 0 and 1, those probabilities can converge to the values in between. To alleviate this ambiguity, after the first phase ends, we fix the tessellation to make Λ_{wdt} for each face in $\mathbb{F}$ to either 0 or 1. Therefore, in the second phase, we only care



(a) Ground Truth

(b) Initialized DMesh (Points, Extracted Mesh)

Fig. 14: Initialized DMesh using sample points from ground truth mesh. (a) From ground truth mesh, we uniformly sample $10K$ points to initialize DMesh. (b) In the left figure, sample points from the ground truth mesh ($\mathbb{P}_{sample}$) are rendered in red. The points that correspond to $\mathbb{P}_{voronoi}$ are rendered in blue. In the right figure, we render the initial mesh we can get from the points, which has a lot of holes.

about the faces that exist in current WDT, which have Λ_{wdt} value of 1. Then, we can only care about real values.

Note that the two differentiable renderers that we introduced in Appendix 8.3 are designed to serve for these two phases, respectively.

Point Initialization with Sample Points In this work, we propose two point initialization methods. The first initialization method can be used when we have sample points near the target geometry in hand.

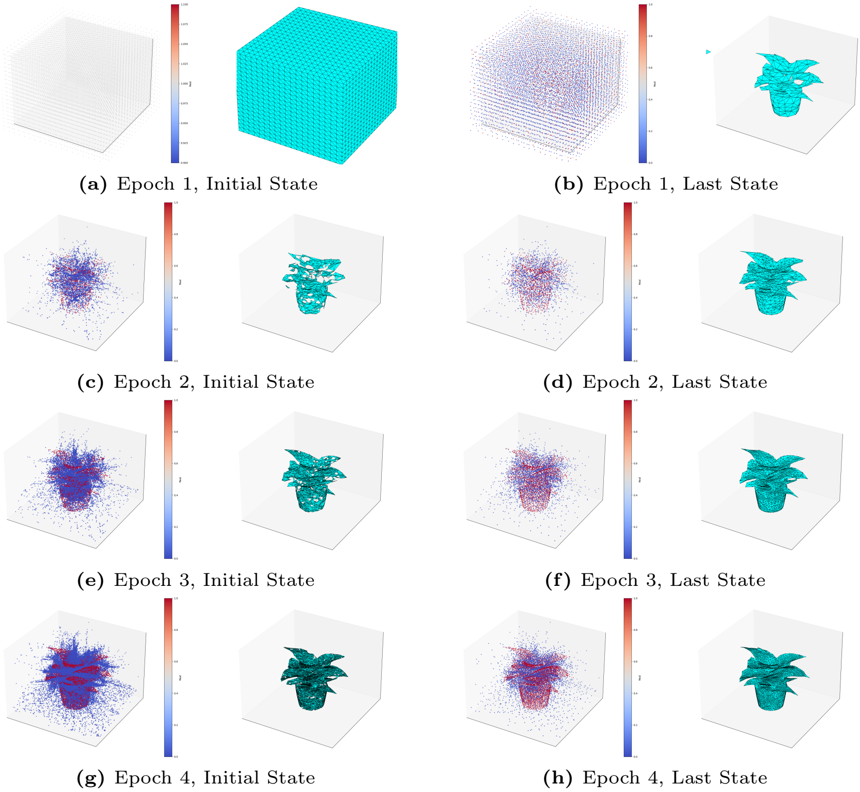
This initialization method is based on an observation that the vertices of Voronoi diagram of a point set tend to lie on the medial axis of the target geometry [1,2]. Therefore, for the given sample point set $\mathbb{P}_{sample}$, we first build Voronoi diagram of it, and find Voronoi vertices $\mathbb{P}_{voronoi}$. Then, we merge them to initialize our point set $\mathbb{P}$:

$$\mathbb{P} = \mathbb{P}_{sample} \cup \mathbb{P}_{voronoi},$$

all of which weights are initialized to 1. Then, we set the real values (ψ) of points in $\mathbb{P}_{sample}$ as 1, while setting those of points in $\mathbb{P}_{voronoi}$ as 0.

In Figure 14, we render the mesh that we can get from this initialization method, when we use $10K$ sample points. Note that the initial mesh has a lot of holes, because there could be Voronoi vertices that are located near the mesh surface, as pointed out by [2]. However, we can converge to the target mesh faster than the initialization method that we discuss below, because most of the points that we need are already located near the target geometry.

Point Initialization without Sample Points If there is no sample point that we can use to initialize our points, we initialize our points with N^3 points regularly distributed on a grid structure that encompasses the domain, all of



(a) Epoch 1, Initial State

(b) Epoch 1, Last State

(c) Epoch 2, Initial State

(d) Epoch 2, Last State

(e) Epoch 3, Initial State

(f) Epoch 3, Last State

(g) Epoch 4, Initial State

(h) Epoch 4, Last State

Fig. 15: Optimization process for multi-view reconstruction for Plant model. At each row, we present the initial state (left) and the last state (right) of each epoch. For each figure, the left rendering shows the point attributes color coded based on real values, while the right one shows the extracted mesh. (a), (b) In the first epoch, we initialize DMesh without sample points. At the end of each epoch, we sample points from the current mesh, and use them for initialization in the next epoch.

which has weight 1 and ψ value of 1. We set $N = 20$ for every experiment (Figure 15a). Then, we optimize the mesh to retrieve a coarse form of the target geometry (Figure 15b). Note that we need to refine this mesh in the subsequent epochs, as explained below.

Point Initialization for Different Inputs Until now, we introduced two point initialization techniques. When the input is a point cloud, we sample subset of the point cloud to initialize our mesh (Figure 14). However, when the input is multi-view images, we start from initialization without sample points (Figure 15), because there is no sample point cloud that we can make use of.

Maintaining $\mathbb{F}$ We maintain the running set of faces to evaluate probability existence for in $\mathbb{F}$. At each iteration, after we get WDT, we insert every face in WDT to $\mathbb{F}$, as it has a high possibility to persist in the subsequent optimization steps. Also, as we did int mesh to DMesh conversion (Appendix 9.1), at every 10 optimization step, we find k-nearest neighbors for each point, and form face combinations based on them. Then, we add them to $\mathbb{F}$.

Mesh Refinement At start of each epoch, if it is not the first epoch, we refine our mesh by increasing the number of points. To elaborate, we refine our mesh by sampling N number of points on the current DMesh, and then initialize point attributes using those sample points as we explained above. We increase N as number of epoch increases. For instance, in our multi-view reconstruction experiments, we set the number of epochs as 4, and set $N = (1K, 3K, 10K)$ for the epochs excluding the first one. In Figure 15, we render the initial and the last state of DMesh of each epoch. Note that the mesh complexity increases and becomes more accurate as epoch proceeds, because we use more points. Therefore, this approach can be regarded as a coarse-to-fine approach.

Post-Processing When it comes to multi-view reconstruction, we found out that it is helpful to add one more constraint in defining the face existence. In our formulation, in general, a face F has two tetrahedra (T_1, T_2) that are adjacent to each other over the face. Then, we call the remaining point of T_1 and T_2 that is not included in F as P_1 and P_2. Our new constraint requires at least one of P_1 and P_2 to have ψ value of 0 to let F exist.

This additional constraint was inspired by the fact that F is not visible from outside if F exists in our original formulation, and both of P_1 and P_2 have ψ value of 1. That is, if it is not visible from outside, we do not recognize its existence. This constraint was also adopted to accommodate our real regularization, which increases the real value of points near surface. If this regularization makes the real value of points inside the closed surface, they would end up in internal faces that are invisible from outside. Because of this invisibility, our loss function cannot generate a signal to remove them. In the end, we can expect all of the faces inside a closed surface will exist, because of the absence of signal to remove them. Therefore, we choose to remove those internal faces by applying this new constraint in the post-processing step.

Note that this discussion is based on the assumption that our renderer does a precise depth testing. If it does not do the accurate depth testing, internal faces can be regarded as visible from outside, and thus get false gradient signal. In Figure 12a, the final mesh after phase 1 is rendered, and we can see therer are lots of internal faces as the renderer used in phase 1 does not support precise depth testing. However, we can remove them with the other renderer in phase 2, as shown in Figure 12b, which justifies our implementation of two different renderers.

Finally, we note that this constraint is not necessary for point cloud reconstruction, because if we minimize CD_{ours} in Appendix 8.2, the internal faces will be removed automatically.

10 Experimental Details

In this section, we provide experimental details for the results in Section 4, and visual renderings of the our reconstructed mesh. Additionally, we provide the results of ablation studies about regularizations that we suggested in Section 3.4.

10.1 Mesh to DMesh

As shown in Table 1, we reconstruct the ground truth connectivity of Bunny, Dragon, and Buddha model from Stanford dataset [13]. For all these experiments, we optimized for $20K$ steps, and used an ADAM optimizer [22] with learning rate of 10^{-4}. For Bunny model, we inserted additional points at every 5000 step. For the other models, we inserted them at every 2000 step.

In Figure 16, we provide the ground truth mesh and our reconstructed mesh. We can observe that most of the connectivity is preserved in our reconstruction, as suggested numerically in Table 1. However, note that the appearance of the reconstructed mesh can be slightly different from the ground truth mesh, because we allow 1% of positional perturbations to the mesh vertices.

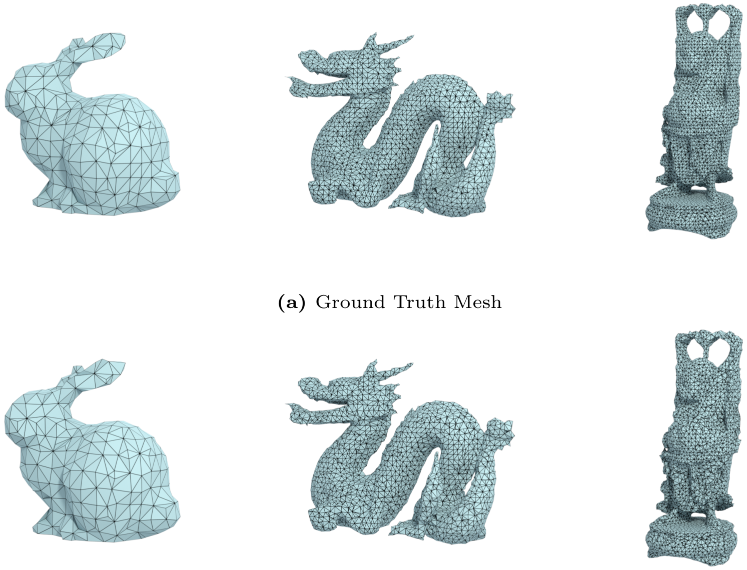
10.2 Point Cloud & Multi-view Reconstruction

Hyperparameters for Point Cloud Reconstruction

- Optimizer: ADAM Optimizer, Learning rate = 10^{-4} for open surface meshes and two mixed surface meshes (Bigvegas, Raspberry) / $3 \cdot 10^{-4}$ for closed surface meshes, and one mixed surface mesh (Plant).
- Regularization: $\lambda_{weight} = 10^{-8}, \lambda_{real} = 10^{-3}, \lambda_{qual} = 10^{-3}$ for every mesh.
- Number of epochs: Single epoch for every mesh.
- Number of steps per epoch: 1000 steps for phase 1, 500 steps for phase 2 for every mesh.

Hyperparameters for Multi-view Reconstruction

- Optimizer: ADAM Optimizer, Learning rate = 10^{-3} in the first epoch, and $3 \cdot 10^{-4}$ in the other epochs for every mesh.
- Weight Regularization: $\lambda_{weight} = 10^{-8}$ for every mesh.
- Real Regularization: $\lambda_{real} = 10^{-3}$ for the first 100 steps in every epoch for open surface meshes and one mixed surface mesh (Plant) / 10^{-2} for the first 100 steps in every epoch for closed surface meshes and two mixed surface meshes (Bigvegas, Raspberry).
- Quality Regularization: $\lambda_{qual} = 10^{-3}$ for every mesh.



(a) Ground Truth Mesh

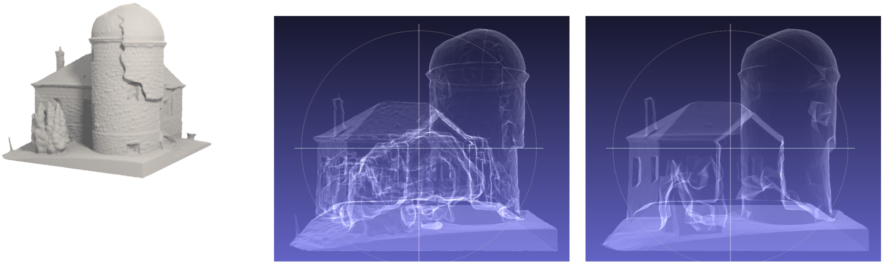
(b) Reconstructed DMesh

Fig. 16: Reconstruction results for mesh to DMesh experiment. From Left: Bunny, Dragon, and Buddha. We can observe that most of the edge connectivity is perserved in the reconstruction, even though the appearance is slightly different from the ground truth mesh because of small perturbations of vertex positions.

- Normal Coefficient: $\lambda_{normal} = 0$ for every mesh (Eq. 13).
- Number of epochs: 4 epochs for every mesh. In the first epoch, use 20^{-3} regularly distributed points for initialization. In the subsequent epochs, sample $1K, 3K$, and $10K$ points from the current mesh for initialization.
- Number of steps per epoch: 500 steps for phase 1, 500 steps for phase 2 for every mesh.
- Batch size: 64 for open surface meshes, 16 for the other meshes.

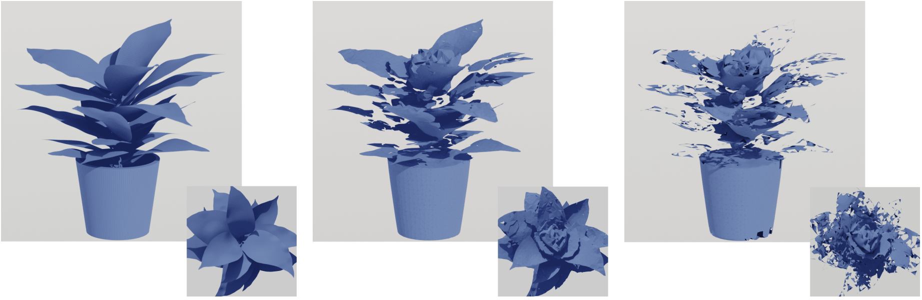
Visual Renderings In Figure 21, 22, and 23, we provide visual renderings of our point cloud and multi-view reconstruction results with ground truth mesh. We also provide illustration of input point cloud and diffuse map. Note that we also used depth renderings for multi-view reconstruction experiments.

Additional Discussion Generally, we can observe that reconstruction results from both point cloud and multi-view images capture the overall topology well. However, we noticed that the multi-view reconstruction results are not as good as point cloud reconstruction results. In particular, we can observe small holes in the multi-view reconstruction results. We assume that these artifacts are coming



(a) Ground Truth Mesh **(b)** Flexicube **(c)** Ours

Fig. 17: Reconstruction results for a closed surface model in Thingi32 dataset. Flexicube [44] can generate internal structures, while our approach removes them through post-processing.

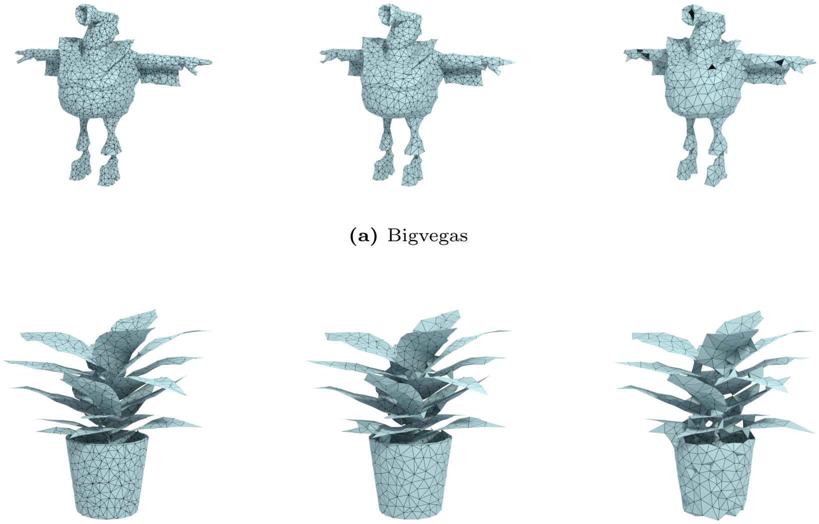


(a) Ground Truth **(b)** Flexicube **(c)** Flexicube, self-intersecting faces removed

Fig. 18: Reconstruction results for the Plant model. Flexicube [44] can generate redundant, self-intersecting faces for open surfaces, in this case, leaves. To better capture the redundant faces, we rendered the models from upper side, which is shown in the bottom right figures.

from relatively weaker supervision of multi-view images than dense point clouds. Also, we believe that we can improve these multi-view reconstruction results with more advanced differentiable renderer, and better mesh refinement strategy. In the current implementation, we lose connectivity information at the start of each epoch, which is undesirable. We believe that we can improve this approach by inserting points near the regions of interest, rather than resampling over entire mesh.

Also, regarding comparison to Flexicube [44] in Table 2, we tried to found out the reason why ours give better results than Flexicube in terms of CD to the ground truth mesh for closed surfaces in thingi32 dataset. We could observe that Flexicube's reconstruction results capture fine geometric details on the surface mesh, but also observed that they have lots of false internal structure (Figure 17). Note that this observation not only applies to closed surfaces, but also



(a) Bigvegas

(b) Plant

Fig. 19: Point cloud reconstruction results with different λ_{weight}. From Left: $\lambda_{weight} = 10^{-6}, 10^{-5}$, and 10^{-4}.

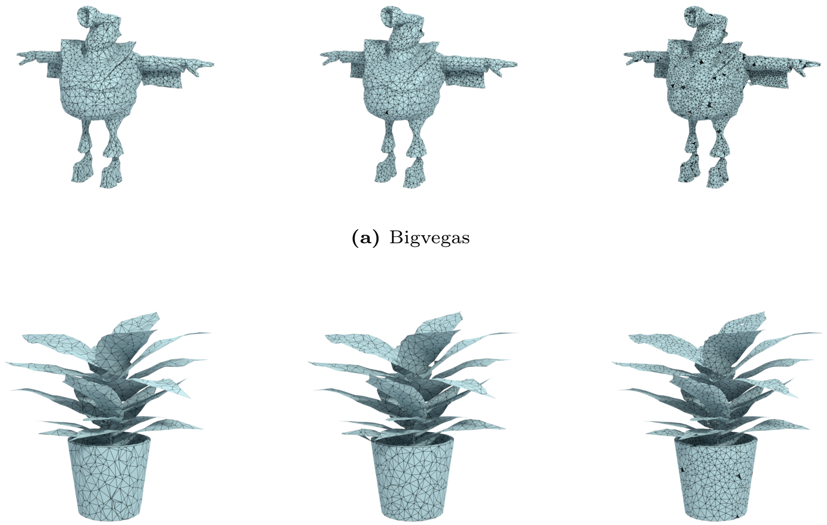
to open surfaces, where it generates lots of false, self-intersecting faces (Figure 18). Our results do not suffer from these problems, as we do post-processing (Appendix 9.2) to remove inner structure, and also our method can represent open surfaces better than the volumetric approaches without self-intersecting faces.

10.3 Ablation studies

In this section, we provide ablation studies for the regularizations that we proposed in Section 3.4. We tested the effect of the regularizations on the point cloud reconstruction task.

Weight Regularization We tested the influence of weight regularzation in the final mesh, by choosing λ_{weight} in $(10^{-6}, 10^{-5}, 10^{-4})$. Note that we set the other experimental settings as same as described in Section 10.2, except $\lambda_{quality}$, which is set as 0, to exclude it from optimization.

In Table 4, we provide the quantitative results for the experiments. For different λ_{weight}, we reconstructed mesh from point clouds, and computed average Chamfer Distance (CD) and average number of faces across every test data. We can observe that there exists a clear tradeoff between CD and mesh complexity.



(a) Bigvegas

(b) Plant

Fig. 20: Point cloud reconstruction results with different $\lambda_{quality}$. From Left: $\lambda_{real} = 10^{-4}, 10^{-3}$, and 10^{-2}.

To be specific, when $\lambda_{weight} = 10^{-6}$, the CD is not very different from the results in Table 2, where we use $\lambda_{weight} = 10^{-8}$. However, when it increases to 10^{-5} and 10^{-4}, we can observe that the mesh complexity (in terms of number of faces) decreases, but CD increases quickly.

The renderings in Figure 19 support these quantitative results. When $\lambda_{weight} = 10^{-6}$, we can observe good reconstruction quality. When $\lambda_{weight} = 10^{-5}$, there are small artifacts in the reconstruction, but we can get meshes of generally good quality with fewer number of faces. However, when it becomes 10^{-4}, the reconstruction results deteriorate, making holes and bumpy faces on the smooth surface. Therefore, we can conclude that weight regularization contributes to reducing the mesh complexity. However, we need to choose λ_{weight} carefully, so that it does not harm the reconstruction quality. The experimental results tell us setting λ_{weight} to 10^{-6} could be a good choice to balance between these two contradictory objectives.

Table 4: Ablation study for weight regularization, quantitative results.

λ_{weight}	10^{-6}	10^{-5}	10^{-4}
CD	7.48	8.08	10.82
Num. Face	4753	2809	1786

Quality Regularization As we did in the previous section, we test the influence of quality regularization in the final mesh by selecting λ_{real} among

$(10^{-4}, 10^{-3}, 10^{-2})$. We also set the other experimental settings as same as before, except $\lambda_{weight} = 0$.

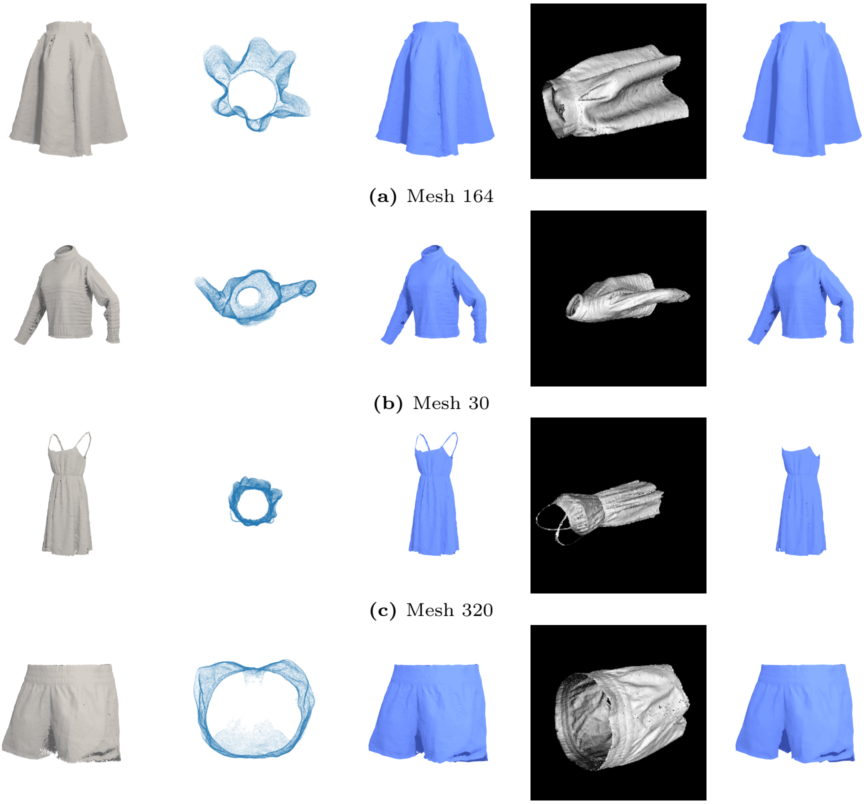
In Table 5 and Figure 20, we present quantitative and qualitative comparisons between the reconstruction results. We provide statistics about average CD, average number of faces, and average aspect ratio of faces. Interestingly, unlike weight regularization, we could not observe tradeoff between CD and aspect ratio. Rather than that, we could find that CD decreases as aspect ratio gets smaller, and thus the triangle quality gets better.

Table 5: Ablation study for quality regularization, quantitative results.

λ_{qual}	10^{-4}	10^{-3}	10^{-2}
CD	7.60	7.42	7.28
Num. Face	8266	8349	10806
Aspect Ratio	2.33	2.06	1.55

We find the reason for this phenomenon in the increase of smaller, good quality triangle faces. Note that there is no significant difference between the number of faces between $\lambda_{qual} = 10^{-4}$ and 10^{-3}. Also, we cannot find big difference between visual renderings between them, even though the aspect ratio was clearly improved. However, when λ_{qual} becomes 10^{-2}, the number of faces increase fast, which can be observed in the renderings, too. We believe that this increase stems from our quality constraint, because it has to generate more triangles to represent the same area, if there is less degree of freedom to change the triangle shape. Since it has more triangle faces, we assume that they contribute to capturing fine details better, leading to the improved CD.

However, at the same time, note that the number of holes increase as we increase λ_{qual}, which lead to visual artifacts. We assume that there are not enough points to remove these holes, by generating quality triangle faces that meet our needs. Therefore, as discussed before, if we can find a systematic way to prevent holes, or come up with a better optimization scheme to remove them, we expect that we would be able to get accurate mesh comprised of better quality triangles.



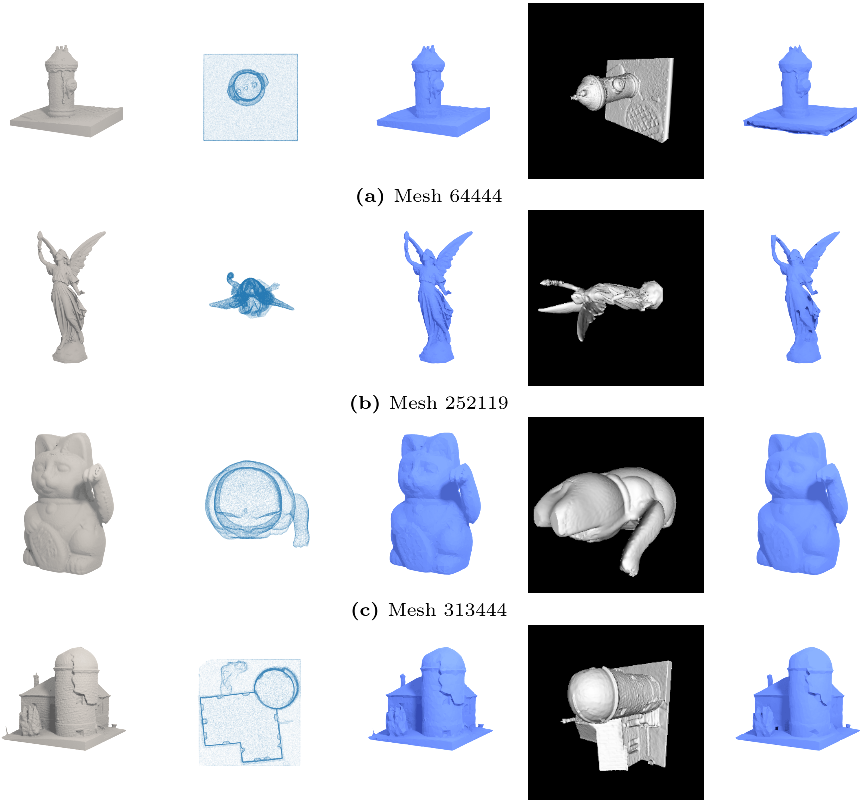
(a) Mesh 164

(b) Mesh 30

(c) Mesh 320

(d) Mesh 448

Fig. 21: Point cloud and Multi-view Reconstruction results for open surface models. From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.



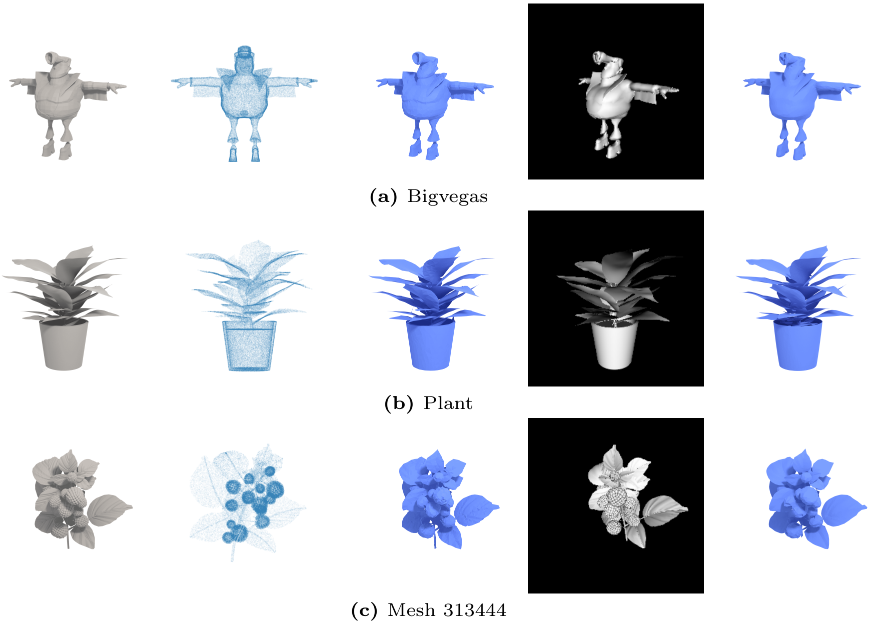
(a) Mesh 64444

(b) Mesh 252119

(c) Mesh 313444

(d) Mesh 527631

Fig. 22: Point cloud and Multi-view Reconstruction results for closed surface models. From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.



(a) Bigvegas

(b) Plant

(c) Mesh 313444

Fig. 23: Point cloud and Multi-view Reconstruction results for mixed surface models. From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.