Differentiable Micro-Mesh Construction

Yishun Dou[2] Zhong Zheng[2] Qiaoqiao Jin[1] Rui Shi[1] Yuhan Li[1] Bingbing Ni[1,2*]
[1]Shanghai Jiao Tong University, Shanghai 200240, China [2] Huawei
yishun.dou@gmail.com nibingbing@sjtu.edu.cn

Abstract

Micro-mesh (μ-mesh) is a new graphics primitive for compact representation of extreme geometry, consisting of a low-polygon base mesh enriched by per micro-vertex displacement. A new generation of GPUs supports this structure with hardware evolution on μ-mesh ray tracing, achieving real-time rendering in pixel level geometric details.

In this article, we present a differentiable framework to convert standard meshes into this efficient format, offering a holistic scheme in contrast to the previous stage-based methods. In our construction context, a μ-mesh is defined where each base triangle is a parametric primitive, which is then reparameterized with Laplacian operators for efficient geometry optimization. Our framework offers numerous advantages for high-quality μ-mesh production: (i) end-to-end geometry optimization and displacement baking; (ii) enabling the differentiation of renderings with respect to μ-mesh for faithful reprojectability; (iii) high scalability for integrating useful features for μ-mesh production and rendering, such as minimizing shell volume, maintaining the isotropy of the base mesh, and visual-guided adaptive level of detail. Extensive experiments on μ-mesh construction for a large set of high-resolution meshes demonstrate the superior quality achieved by the proposed scheme.

1. Introduction

Micro-Mesh (μ-mesh) [38] is a promising graphics primitive for compact storage and efficient ray tracing rendering, supported by a new generation of GPUs. A μ-mesh can be seen as a special displacement-mapped mesh, consisting of a low-polygon base mesh and per-μ-vertex scalar displacement (see Fig. 1 (a)), where the μ-vertex is produced by subdivision at rendering time. The μ-mesh construction, *i.e.* converting a standard mesh to this format, is critical to μ-mesh accuracy and rendering efficiency.

A straightforward choice is to combine existing polygon mesh processing methods [5], including mesh decimation [12], remeshing [24], displacement baking [10, 11] and so on, which is adopted in [29, 38]. This forms a stage-based pipeline, where each stage independently minimizes some local objective, such as the quadric error metric for decimation, or the centroidal voronoi tessellation energy for remeshing. However, both of these objective functions are designed for specific mesh processing tasks, while the ultimate goals of building a μ-mesh, *i.e.* faithful reproduction and minimal bounding shell volume, are not directly considered. The most recent state-of-the-art method proposed by [28] significantly improves the μ-mesh quality, but they still continue with the staged pipeline and pose risks in error accumulation and suboptimal construction.

[*]Corresponding author: Bingbing Ni.

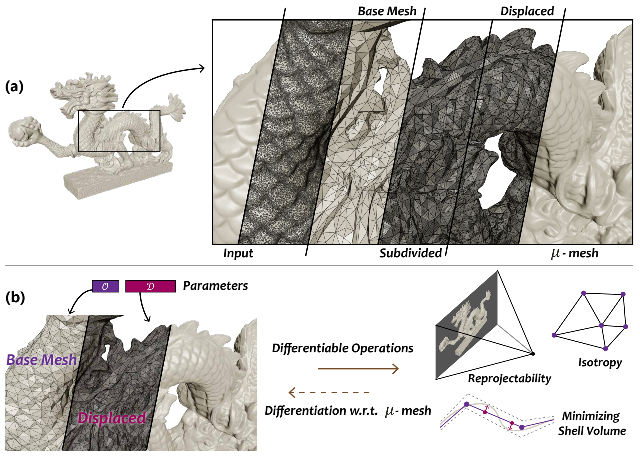


Figure 1. **(a) A μ-mesh** produced by our method in $1:64$ decimation. **(b) Overview** of our differentiable μ-mesh construction.

We propose to alleviate these limitations through the end-to-end optimization, taking the ultimate goals of μ-mesh as the objective for both the base mesh optimization and displacement baking. This is made possible by ***(i)*** a holistic μ-mesh construction system that is full of differentiable operations for end-to-end backpropagation, ***(ii)*** a curriculum optimization strategy exploiting the level of detail nature of μ-mesh, and ***(iii)*** several novel objectives that well instantiate the goals of producing a high quality μ-mesh. See Fig. 1 (b) for an illustration. Our system is designed according to the following principles:

- **Reprojectability.** The primary requirement for building a μ-mesh is to reproduce the original mesh. Since the μ-

mesh is eventually presented to users in 2D, the construction system must consider not only the geometry error but also the rendering reproduce error.

- **Scalability.** μ-mesh makes additional requirements to maximize its strengths, either in memory efficiency or rendering cost. In contrast to the less flexible stage-based methods, our goal is to develop a holistic framework, embracing new functionality in a plug-and-play fashion.
- **Robustness.** Meshes vary dramatically in topology, frequency and mesh resolution, which poses challenges to the design of an automatic construction system. Case-specific design eventually leads to brittle systems. We strive for a solution that is robust to mesh variations.

Reprojectability. The first principle necessitates a differentiable renderer to support the inverse rendering of μ-mesh. Existing differentiable renderers [14, 20, 44], however, are designed for standard meshes, for which we introduce a differentiable μ-mesh pre-tessellation to utilize the existing techniques. Still, naïve application of differentiable rendering often results in tangled mesh, as indicated by Nicolet et al. [34]. To this end, a μ-mesh reparameterization is introduced to bias gradient steps towards smooth solutions without requiring the final μ-mesh to be smooth [34]. In other views, instantiating *reprojectability* with images error instead of geometry error (*e.g.*, Chamfer Distance, Quadric Error Metric [8]) gains in: ***(i)*** better *robustness* to mesh resolutions due to the efficient rasterization and no sampling required on geometry surface, and ***(ii)*** higher μ-mesh rendering quality since imperceptible rendering degeneration is more likely achieved by construction with visual-guidance.

Scalability. An end-to-end differentiable system enjoys greater *scalability* than stage-based pipelines, especially for functionalities that involve multiple stages. For instance, the μ-mesh bounding shell volume, which is key to rendering performance [38], should be minimized during construction. For the staged pipeline, the shell volume is fixed right after producing the base mesh, while the exact volume value is calculated after baking displacements. As a result, multiple stages are involved to minimize the bounding shell volume. Recent method [28] makes compromises and tackles this by introducing a post processing after baking displacements. In contrast, our holistic system handle this via simultaneous optimization of base mesh and displacements, guided by a novel shell volume regularization.

Robustness. One of the benefits of μ-mesh is the self-governed nature of base triangles. In our system, the key requirements for a high quality μ-mesh are all realized upon a base triangle or one-ring neighboring base triangles of a base vertex, including *minimizing shell volume*, *isotropy*, and *adaptive level of detail*. These local operations are delicately designed to maintain robustness.

In summary, we highlight the following contributions:

- We present a novel end-to-end system for μ-mesh construction, together with extensive experiments to demonstrate the superior quality achieved by our method.
- Several differentiable operations dedicated to efficient μ-mesh construction are proposed to support this system.
- Critical features for producing high quality μ-mesh are elegantly and effectively devised.

Symbol	**Domain**	**Definition**
M		standard mesh
$\mathcal{M}$		μ-mesh
$\mathcal{V}$	$\mathbb{R}^{N_{\mathcal{V}} \times 3}$	base vertex
$\mathcal{F}$	$\mathbb{N}^{N_{\mathcal{F}} \times 3}$	base face
$\ell; \ell_{\text{max}}$	$\mathbb{N}$	level of detail
$\mathcal{V}^{\ell}$	$\mathbb{R}^{N_{\mathcal{V}}^{\ell} \times 3}$	μ-vertex at level ℓ
$\mathcal{F}^{\ell}$	$\mathbb{N}^{4^{\ell} N_{\mathcal{F}} \times 3}$	μ-face at level ℓ
$\mathcal{O}$	$\mathbb{R}^{N_{\mathcal{V}} \times 3}$	base vertex offset
$\mathcal{D}$	$\mathbb{R}^{N_{\mathcal{V}}^{\ell_{\text{max}}}}$	μ-vertex displacement
$\mathcal{D}^{\ell}$	$\mathbb{R}^{N_{\mathcal{V}}^{\ell}}$	μ-vertex displacement at level ℓ
$\overline{\mathcal{O}}$	$\mathbb{R}^{N_{\mathcal{V}} \times 3}$	reparameterization of $\mathcal{O}$
$\overline{\mathcal{D}}^{\ell}$	$\mathbb{R}^{N_{\mathcal{V}}^{\ell}}$	reparameterization of $\mathcal{D}^{\ell}$
$\mathcal{P}$		parametric primitive
$\mathsf{Tess}(\cdot)$	$\mathcal{M} \rightarrow M$	μ-mesh tessellation

2. Related Work

Our method is related to μ-mesh construction, differentiable rendering, and Laplacian operator.

2.1. Micro-Mesh Construction

Given a standard mesh, the process of constructing a μ-mesh [28, 29, 38] can be roughly divided into three stages: ***(i)*** Firstly, an edge collapse decimation and a remeshing are applied on the high-polygon input mesh to get an isotropic simplification, called base mesh, where the base triangle is taken as the elementary primitive at rendering time. ***(ii)*** Secondly, the base triangles are then stationarily subdivided into μ-faces in a 1-to-4 manner, and recursively repeat this until reach the maximum level or meet some stopping criteria. ***(iii)*** Lastly, the geometric surface details are baked into the base surface via ray-casting at μ-vertices along normal directions, where the resulting scalar displacements are then quantized and packed. After finishing these steps, a compact μ-mesh consisting of a base mesh and displacements is saved and ready for rendering. We refer to the white paper [38] for more details about μ-mesh structure, compression, and rendering.

This stage-based construction poses risks in error accumulation, oversize shell volumes stemmed from the suboptimal base mesh, and the lack of flexibility in further geometry processing that involves multiple stages. In contrast, we strive for an end-to-end solution that bridges as much intermediate processes as possible.

2.2. Differentiable Rendering

Differentiable rendering has proven to be a powerful tool in the realm of computer vision and graphics, including geometry optimization [13, 34, 49], uv and texture optimization [19], 3D reconstruction [32, 36], and many inverse rendering tasks [9, 54]. Volumetric renderings are inherently differentiable and often used in the inverse rendering tasks defined upon volume representation [30, 31]. Instead, rasterization rendering pipeline processes geometry surface, and thus is more suitable for inverse rendering on meshes. Due to the discrete rasterization, several approaches [7, 16, 20, 26] employ various techniques like minor image blurring or antialiasing operations to enable differentiation. Differentiable rasterization is computationally efficient, offering great feasibility in optimization-based problems, such as the recent 3D generations [6] relying on 2D supervision [45], radiance field reconstruction [17], and shape optimization [34]. Similarly, we can efficiently optimize a μ-mesh with the rendering guidance, *i.e.* towards similar renderings with the original mesh.

There is also an active research area focusing on physics-based differentiable rendering [3, 22, 27, 35, 37, 48–53], aiming at solving the inverse rendering with shadow, indirect illumination, and other high-order effects. However, instead of recovering the whole rendering scene, where many local minima exist, we optimize the geometry with frozen material and illumination, for which a computationally efficient rasterization renderer is sufficient for our purpose.

2.3. Laplacian Operator in Mesh Processing

The Laplacian is a elementary differentiable operator of geometry processing. Consider a mesh with n vertices $\mathbf{V}$, its Laplacian representation $\Delta \in \mathbb{R}^{n\times 3}$ is defined as $\Delta = \mathbf{LV}$, where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a sparse matrix, referred to as the discrete Laplacian operator. $\mathbf{L}$ is defined locally on derivatives along edge; Δ of a vertex encodes its local neighborhood on surface. We refer to [5] for a review of Laplacian mesh processing. Built upon Laplacian, Nicolet et al. [34] show an application in inverse rendering of geometry. We step further in exploiting the usage in μ-mesh.

3. Methodology

Our goal is to develop a fully differentiable framework for μ-mesh construction. In this section, we first parameterize the μ-mesh in Sec. 3.1, which is then reparameterized with differentiable Laplacian operators in Sec. 3.2. We then present the optimization algorithm instantiated within a differentiable rendering framework in Sec. 3.3. Finally, Sec. 3.4 demonstrates that broad requirements and features can be easily realized upon such a differentiable framework, which is critical either for μ-mesh compression or rendering. Figure 2 illustrates the overall construction pipeline.

3.1. Micro-Mesh Parameterization

We denote a μ-mesh as $\mathcal{M} = (\mathcal{V}, \mathcal{F}, \mathcal{O}, \mathcal{D})$ with base vertices $\mathcal{V} \in \mathbb{R}^{N_\mathcal{V}\times 3}$ and base triangle faces $\mathcal{F} \in \mathbb{N}^{N_\mathcal{F}\times 3}$, where $N_\mathcal{V}$ and $N_\mathcal{F}$ are the number of vertices and faces respectively. The base mesh $M(\mathcal{V}, \mathcal{F})$ is initialized with mesh decimation and remeshing [1, 12].

Subdivision. Building a μ-mesh necessitates subdivision. We use Loop [25] 1-to-4 subdivision without vertex update to obtain finer topology from the base mesh. From the view of level of detail, the base mesh has the initial level $\ell = 0$, and the finest topology level $\ell_{\max}$ is obtained by applying subdivision $\ell_{\max}$ times. Regardless of the various granularities at different levels, for brevity, we denote the subdivided faces as micro-face (μ-face) $\mathcal{F}^\ell \in \mathbb{N}^{4^\ell N_\mathcal{F}\times 3}$, as well as vertices as micro-vertex (μ-vertex) $\mathcal{V}^\ell \in \mathbb{R}^{N_\mathcal{V}^\ell\times 3}$ for level $\ell \in \{0, 1, \ldots, \ell_{\max}\}$, where $N_\mathcal{V}^\ell$ is the μ-vertex number that depends on the mesh topology [1]. Indeed, $\mathcal{V}^0$ is the initial base vertex $\mathcal{V}$ and μ-vertices at level ℓ are a subset of those at level $\ell + 1$ (*i.e.* $\mathcal{V}^0 \equiv \mathcal{V}, \mathcal{V}^\ell \subset \mathcal{V}^{\ell+1}$).

Trainable Parameters. We define the following trainable parameters: ***(i)*** 3-dimensional vectors on base vertices, which we refer to offset $\mathcal{O} \in \mathbb{R}^{N_\mathcal{V}\times 3}$, and ***(ii)*** scalar displacements $\mathcal{D} \in \mathbb{R}^{N_\mathcal{V}^{\ell_{\max}}}$ on μ-vertices, from which we can extract the displacements for level ℓ via slicing the first $N_\mathcal{V}^\ell$ values in $\mathcal{D}$ (*i.e.* $\mathcal{D}[:N_\mathcal{V}^\ell]$), denoted as $\mathcal{D}^\ell \in \mathbb{R}^{N_\mathcal{V}^\ell}$.

Parametric Primitive. Particularly, the set of parameters belonging to a base triangle at level ℓ is deemed as an elementary *parametric primitive* $\mathcal{P}$:

$$\mathcal{P} = (o \in \mathbb{R}^{3\times 3},\ d \in \mathbb{R}^{\sum_{i=1}^{2^\ell+1} i}), \tag{1}$$

where $o \subset \mathcal{O}$ indicates the vector offsets of the three base vertices and $d \subset \mathcal{D}$ is the scalar displacements of the inclusive μ-vertices. Notably, we also make the position of base vertex optimizable in contrast to the staged μ-mesh construction methods that freeze the base vertex once it is initialized [38] or rely on further post process after displacement baking [28]. This enhances robustness to sub-optimal simplification and provides scalability for further extensions, *e.g.*, minimizing the shell volume.

In practice, only the updated base mesh $M(\mathcal{V} + \mathcal{O}, \mathcal{F})$ and the displacements $\mathcal{D}$ are asked to be saved to disk after our optimization-based construction. The remaining components, *i.e.* topology and μ-vertex positions, are discarded and generated on the fly during rendering.

3.2. Laplacian on Micro-Mesh

With the above parameterization, the μ-mesh is immediately ready for optimizing within a differentiable geometry optimization framework, given an objective such as a rendering consistency loss, and a gradient-based optimizer.

[1]A base triangle contains $\sum_{i=1}^{2^\ell+1} i$ μ-vertices at level ℓ. μ-vertices along base mesh edges are shared between two base triangles.

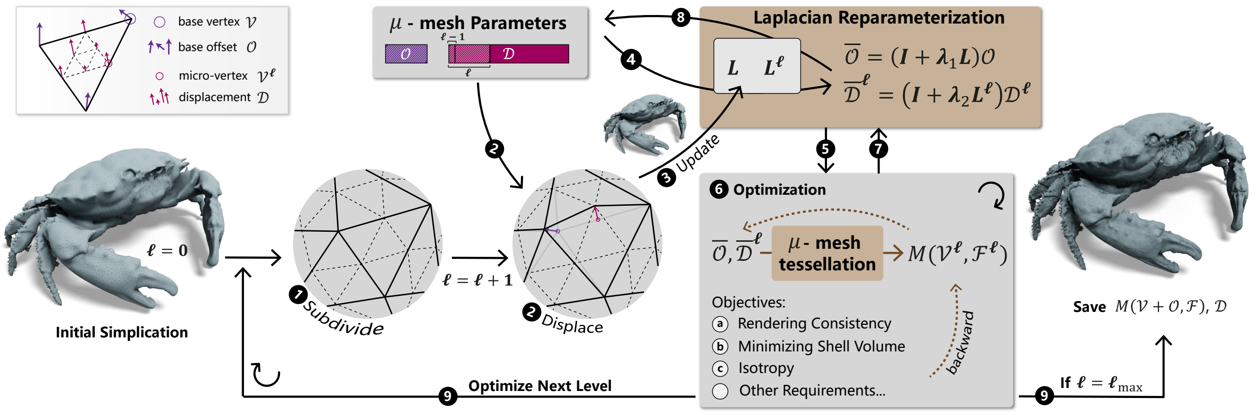

base vertex $\mathcal{V}$
base offset $\mathcal{O}$
micro-vertex $\mathcal{V}^{\ell}$
displacement $\mathcal{D}$
μ - mesh Parameters
$\mathcal{O}$
$\ell=1$
$\mathcal{D}$
ℓ
Laplacian Reparameterization
L L^{ℓ}
$\overline{\mathcal{O}} = (I + \lambda_1 L)\mathcal{O}$
$\overline{\mathcal{D}}^{\ell} = (I + \lambda_2 L^{\ell})\mathcal{D}^{\ell}$
8
4
2
3 Update
5
7
$\ell = 0$
Initial Simplification
1 Subdivide
$\ell = \ell + 1$
2 Displace
6 Optimization
$\overline{\mathcal{O}}, \overline{\mathcal{D}}^{\ell}$
μ- mesh tessellation
$M(\mathcal{V}^{\ell}, \mathcal{F}^{\ell})$
Objectives:
a Rendering Consistency
b Minimizing Shell Volume
c Isotropy
Other Requirements...
backward
Save $M(\mathcal{V}+\mathcal{O},\mathcal{F}), \mathcal{D}$
9 If $\ell = \ell_{\max}$
9 Optimize Next Level

Figure 2. **Illustration of our μ-mesh construction pipeline.** Starting from an initial simplification, we progressively optimize the μ-mesh parameters $\mathcal{O}$ and $\mathcal{D}$. After each 1-to-4 Loop subdivision ❶, μ-mesh reach the level ℓ with newly introduced μ-vertices, whose displacements are initialized with interpolation. We then displace base vertices and μ-vertices ❷, resulting in a new base mesh and a new subdivided mesh, which are used to update the discrete Laplacian operators $\mathbf{L}$ and $\mathbf{L}^{\ell}$ ❸. Afterwards, μ-mesh parameters are reparameterized ❹ to $\overline{\mathcal{O}}$ and $\overline{\mathcal{D}}$, which are optimized ❻ in a fully differentiable system, with the objectives of rendering and important functionalities for μ-mesh. We then update ❼ ❽ μ-mesh parameters $\mathcal{O}, \mathcal{D}$. These steps will be repeated until reach the maximum level ❾.

However, as indicated by pioneer works in mesh deformation and editing [33, 47], learning on the vertex positions is unstable and often lead to tangled μ-mesh. A conventional approach to tame distortion in geometry processing is to append a Laplacian smooth regularizer [46]. Still, the convergence result must now compromise between solving the reprojectability and being smooth [34].

To this end, inspired by the diffusion reparameterization on meshes [34], we develop a reparameterization method tailored for μ-mesh. Specifically, two cotangent Laplacian operators [41] are computed after each subdivision, among which one for updated base mesh and another for *tessellated μ-mesh* at level ℓ, denoted as $\mathbf{L}$ and $\mathbf{L}^{\ell}$. Note that we employ progressive learning (Sec. 3.3) and the shape after each subdivision is different. The vector-valued offset $\mathcal{O}$ and the scalar-valued displacement $\mathcal{D}$ are reparameterized following [34]:

$$\overline{\mathcal{O}} = (\mathbf{I} + \lambda_1 \mathbf{L})\mathcal{O}, \quad \overline{\mathcal{D}}^{\ell} = (\mathbf{I} + \lambda_2 \mathbf{L}^{\ell})\mathcal{D}^{\ell}, \tag{2}$$

where λ can be seen as the temporal duration of diffusion (like the smooth regularizer weight). The reparameterized offset $\overline{\mathcal{O}} \in \mathbb{R}^{N_{\mathcal{V}} \times 3}$ and displacement $\overline{\mathcal{D}}^{\ell} \in \mathbb{R}^{N_{\mathcal{V}}^{\ell}}$ bear some resemblance to the differentiable coordinate [23, 34, 46]. Note that the Laplacian can be applied to any functions that takes values at every vertex, such as our case that involves both vector-valued and scalar-valued functions.

3.3. Optimization

We choose to instantiate the optimization procedure within a differentiable rendering framework, because of the rendering consistency immediately reflects the reprojectability quality of a μ-mesh. The rendering-guided optimization problem is modeled as:

$$\underset{\mathcal{O},\mathcal{D}}{\arg\min} \;\; \Phi(R(\mathcal{M})), \tag{3}$$

where Φ is a loss function (we use a L_1 loss in our experiments) measuring the reconstruction accuracy of images produced by a differentiable renderer R with known cameras surrounding the mesh.

Solving this problem involves μ-mesh rendering. Different from the μ-mesh rendering pipeline in a deployment version [38], in which the tessellation is deferred, we instead *pre-tessellate* μ-mesh before rendering to make use of the off-the-shelf differentiable renderer [21].

Micro-Mesh Tessellation. We denote a tessellation of μ-mesh up to level ℓ as $\mathsf{Tess}^{\ell}(\mathcal{M}) = M(\mathcal{V}^{\ell}, \mathcal{F}^{\ell})$, where $\mathcal{V}^{\ell}$ is the μ-vertices position after tessellation. The μ-vertices are firstly assigned with the subdivided positions by applying subdivision ℓ times on the offset base mesh $M(\mathcal{V} + \mathcal{O}, \mathcal{F})$. The μ-vertices, for now, lie on the plane of the belonging base triangle. Meanwhile, we compute the base vertex normal $\mathcal{N} \in \mathbb{R}^{N_{\mathcal{V}} \times 3}$ on mesh $M(\mathcal{V} + \mathcal{O}, \mathcal{F})$. The normals are then interpolated at every μ-vertices' barycentric coordinate, resulting in μ-normals, which are further used as the moving directions for scalar-valued displacements. In summary, the μ-vertex after tessellation is given as:

$$\mathcal{V}^{\ell} = \mathsf{subdiv}(M(\mathcal{V} + \mathcal{O}, \mathcal{F})) + \mathsf{interp}(\mathcal{N}) \cdot \mathcal{D}^{\ell}, \tag{4}$$

where $\mathsf{subdiv}(\cdot)$ and $\mathsf{interp}(\cdot)$ are functions of subdivision and barycentric interpolation respectively, both of which are naturally differentiable. By substituting $\mathcal{O}$ and $\mathcal{D}$ in Eq. (4) with the corresponding reparameterizations formulated in Eq. (2), we reach the following μ-mesh tessellation form:

$$\begin{aligned} \mathcal{V}^{\ell} \;=\; & \mathsf{subdiv}(M(\mathcal{V} + (\mathbf{I} + \lambda_1 \mathbf{L})^{-1}\overline{\mathcal{O}}, \mathcal{F})) \\ & + \mathsf{interp}(\mathcal{N}) \cdot (\mathbf{I} + \lambda_2 \mathbf{L}^{\ell})^{-1}\overline{\mathcal{D}}^{\ell}. \end{aligned} \tag{5}$$

These steps constitute a fully differentiable μ-mesh tessellation computation graph, as depicted in Fig. 3. Afterwards,

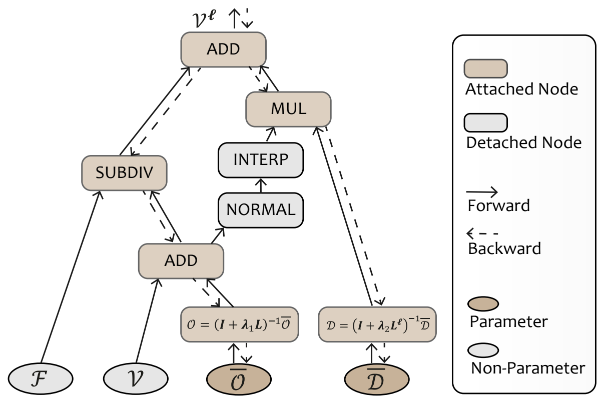


Figure 3. μ**-mesh tessellation computation graph.** μ-normals for displacing μ-vertex are detached from the computation graph. As a special note, the normal of *tessellated* μ*-mesh* fed into (differentiable) renderer R is not detached, because of the gradient of the normal consumed during shading would boost the gradients.

with the above reparameterized $\mathcal{V}^\ell$, the optimization problem at target level ℓ is given as:

$$\underset{\mathcal{O},\mathcal{D}}{\arg\min}\ \Phi(R(M(\mathcal{V}^\ell, \mathcal{F}^\ell))). \tag{6}$$

Both of the tessellation, reparameterization, and renderer introduced above are differentiable, which then constitute a fully differentiable μ-mesh construction framework. Next, we show the derivatives with respect to μ-mesh under the reparameterized regime.

Differentiation w.r.t. Micro-Mesh. Differentiating the rendering loss with respect to μ-mesh is actually to the reparameterizations $\overline{\mathcal{O}}$ and $\overline{\mathcal{D}}^\ell$. Considering the Eq. (5) and Eq. (6), the update rule for reparameterized offset $\overline{\mathcal{O}}$ and displacement $\overline{\mathcal{D}}^\ell$ are given as:

$$\overline{\mathcal{O}} \ \leftarrow\ \overline{\mathcal{O}} - \eta_1 \frac{\partial_{\mathcal{V}^\ell}}{\partial_{\overline{\mathcal{O}}}} \frac{\partial_\Phi}{\partial_{\mathcal{V}^\ell}}, \tag{7}$$

$$\overline{\mathcal{D}}^\ell \ \leftarrow\ \overline{\mathcal{D}}^\ell - \eta_2 \frac{\partial_{\mathcal{V}^\ell}}{\partial_{\overline{\mathcal{D}}^\ell}} \frac{\partial_\Phi}{\partial_{\mathcal{V}^\ell}}, \tag{8}$$

where η_1 and η_2 are learning rates. This is sufficient for training a μ-mesh now, whose original parameters can be calculated according to Eq. (2). In order to get an intuition of how the original μ-mesh parameters are updated, we first rewrite the above Eq. (7) by applying chain rule:

$$\overline{\mathcal{O}} \ \leftarrow\ \overline{\mathcal{O}} - \eta_1 \frac{\partial_{\mathcal{O}}}{\partial_{\overline{\mathcal{O}}}} \frac{\partial_{\mathcal{V}^\ell}}{\partial_{\mathcal{O}}} \frac{\partial_\Phi}{\partial_{\mathcal{V}^\ell}}, \tag{9}$$

where $\partial_{\mathcal{O}}/\partial_{\overline{\mathcal{O}}} = (\mathbf{I}+\lambda_1\mathbf{L})^{-1}$ is the Jacobian of offset $\mathcal{O}$ as a function of the reparameterization $\overline{\mathcal{O}}$. The scalar-valued displacement $\overline{\mathcal{D}}$ also follows the similar update rule, which can be found in supplementary material. We can then derive the following update rule for original parameters $\mathcal{O}$:

$$\begin{aligned} \mathcal{O} \ &\leftarrow\ (\mathbf{I}+\lambda_1\mathbf{L})^{-1}(\overline{\mathcal{O}} - \eta_1 \frac{\partial_{\mathcal{O}}}{\partial_{\overline{\mathcal{O}}}} \frac{\partial_{\mathcal{V}^\ell}}{\partial_{\mathcal{O}}} \frac{\partial_\Phi}{\partial_{\mathcal{V}^\ell}}) \\ &= \mathcal{O} - \eta_1 (\mathbf{I}+\lambda_1\mathbf{L})^{-2} \frac{\partial_{\mathcal{V}^\ell}}{\partial_{\mathcal{O}}} \frac{\partial_\Phi}{\partial_{\mathcal{V}^\ell}}. \end{aligned} \tag{10}$$

This is similar to the update rule for coordinate positions in [34], thus allowing stable training and anti-distortion.

Progressive Subdivision and Optimization. The above derivation are formulated in the level of detail context intentionally, as we design the training procedure under a curriculum learning paradigm [4], shown in next:

1. $\ell = 0$: Warmup with optimizing base mesh, which is similar to the mesh deformation under fixed topology.
2. $\ell \in \{1, \dots, \ell_{\max}\}$: ***Subdivision and solving Eq. (6) alternately***, forming a progressive learning paradigm. Right after each subdivision ($\ell - 1$ to ℓ), **(i)** the new involved μ-vertices are initialized with interpolation, *i.e.* $\mathcal{D}[N_{\mathcal{V}}^{\ell-1} : N_{\mathcal{V}}^{\ell}] = \texttt{interp}(\mathcal{D}[: N_{\mathcal{V}}^{\ell-1}])$; **(ii)** cotangent Laplacian operators $\mathbf{L}$ and $\mathbf{L}^\ell$ are recomputed.
3. Finally, we freeze the base offset $\overline{\mathcal{O}}$ and tune the displacement $\overline{\mathcal{D}}$ of all levels simultaneously.

Figure 2 omits the warmup and the last tuning for brevity. This scheme provides better construction results, validated in our ablation studies. Furthermore, it also offers the possibilities for manipulation at each level separately.

3.4. Beyond the Reconstruction

The fully differentiable nature is useful for developing novel features beyond the reprojectability. We demonstrate the most important features supported by our differentiable system, minimizing shell volume and isotropic base mesh. We also show a better adaptive level of detail achieved by the visual-guided subdivision stopping criterion.

3.4.1 Minimizing Shell Volume

In contrast to the AABB for conventional meshes, a new generation GPUs ray trace a μ-mesh with a tighter and more efficient BLAS (bottom level acceleration structure), *i.e.* a *prismoid* for each base triangle. Connecting the top and bottom triangles of all prismoids results in a *shell* that envelopes the geometry surface, as shown in Fig. 4 (a).

The *shell volume*, *i.e.* sum of the volume of all prismoids, is a crucial factor that affects the rendering time cost. Specifically, a μ-mesh with smaller shell volume gains in: *(i)* effective occlusion culling [40], and *(ii)* lower ray-μ-face miss ratio within the prismoid. However, the shell volume of a μ-mesh built from previous methods is almost determined right after the initial decimation [29, 38], or is slightly adjustable within a post processing after baking μ-vertex displacement [28].

Recall that the positions of base vertices is optimized together with displacements, which is key to minimize shell volume. Given a μ-mesh, the bounding prismoid for each base triangle is computed by min/max fitting [38]. A prismoid containing non-parallel side edges could be divided into three tetrahedrons [42], and thus the shell volume is the sum of all tetrahedrons volume. Although we can directly differentiate the tetrahedron volume w.r.t. the μ-mesh parameters, it is deficient since the involved parameters to be optimized are only those corresponding to the minimum and maximum, according to the min/max fitting strategy.

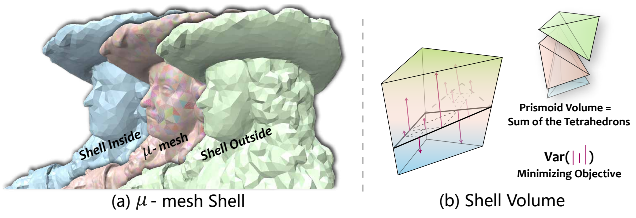


Figure 4. **(a) μ-mesh shell.** Shell inside, μ-mesh surface, and shell outside. **(b) Shell volume.** A shell volume is the sum of all prismoids; a prismoid volume is the sum of three tetrahedrons.

We instead propose a more elegant and effective way: ***(i)*** Given a base vertex v, we find its adjacent base faces. ***(ii)*** Then the corresponding parameter of v is $\mathcal{P}^{\mathsf{v}} = \mathsf{unique}(\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_t\})$, where $\mathcal{P}_i$ is parametric primitive of each adjacent base face, t denotes the neighbors number, and $\mathsf{unique}(\cdot)$ is a function removing the duplicated parameters along edges. ***(iii)*** We then take the variance of the displacements as the minimizing objective. ***(iv)*** Lastly, repeat ***(i) (ii) (iii)*** for all base vertices and take the variance summation. Figure 4 (b) depicts this strategy. In short, the shell volume regularization is the sum of the displacement variances of all base vertex:

$$\mathcal{L}_{sv} = \sum_{\mathsf{v}} \mathsf{var}(\mathcal{P}^{\mathsf{v}}.d), \tag{11}$$

where $\mathsf{var}(\cdot)$ denotes a function computing variance. Although this objective does not directly constraint the offset $\overline{\mathcal{O}}$, minimizing the variance of local displacements facilitates the base mesh fitting the original geometry surface. As a bonus, $\mathcal{L}_{sv}$ leads to smaller displacement, for more accurate low-bit quantization [28, 38] of μ-mesh.

3.4.2 Isotropy

Meshes with roughly equilateral triangles are often obtained via isotropic remeshing [1, 2], which is a longstanding task that plays an important role in geometry processing. The roughly equilateral-shaped base triangle allows more efficient subdivision and baking [28]. Consequently, previous μ-mesh construction methods [28, 38] always employ isotropic remeshing before baking displacement.

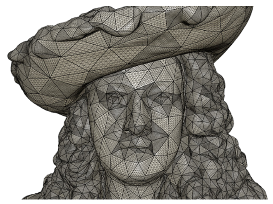
To maintain the isotropic of the base mesh during evolution, we impose *as-equilateral-as-possible* constraint. This is also achieved by applying regularizer upon our parametric primitive $\mathcal{P}$: given a base triangle face f that is offset by the corresponding $\mathcal{P}^{\mathsf{f}}.o$, we formulate the regularization as the variance of edge lengths

$$\mathcal{L}_{iso} = \sum_{\mathsf{f}} \mathsf{var}(\mathsf{e}_1^{\mathsf{f}}, \mathsf{e}_2^{\mathsf{f}}, \mathsf{e}_3^{\mathsf{f}}), \tag{12}$$

where e denotes lengths between the offset base vertices. In contrast to the naïve global edge length regularizer that is often used for preventing distortions, our design takes effects locally and may less distract other global targets, such as the rendering and shell volume. This allows us to make fewer assumptions about the isotropy of initial decimation.

3.4.3 Adaptive Level of Detail

μ-mesh supports per-base triangle subdivision level, *i.e.* adaptive level of detail (LOD), benefiting from the self-governed management nature of the base triangle. Intuitively, we can assign lower level for flat surface regions. In order to meet the water-tightness between displaced base faces, the level difference among adjacent base faces is limited to a maximum of 1. Two possible avenues to achieve adaptive LOD: ***(i)*** applying subdivision early stop during optimization; ***(ii)*** uniformly subdividing all triangles, then pruning the overkill subdivision in a post process.



Both of which necessitate a stopping criterion. One can employ the usual geometry metric, such as the common used Chamfer Distance and those proposed in [8, 12, 15]. Although these metrics perform well for most cases, they may fail to handle the fine-grained details since they rely on surface sampling. Accordingly, we devise a visual-guided stopping criterion that can be used independently or as a supplement to other metrics.

The visual-guided subdivision metric is shown in Algorithm 1. It measures the visual improvements bring from a finer level. Given a base face f, a *hard* rasterizer is employed to get the projected pixels for each camera c view. Then the image space error is measured at two adjacent levels. With this metric, the above two avenues can be achieved with further neighborhood level adjustment to meet water-tightness (full algorithms are shown in supplementary material). We use the first strategy since we find they achieve similar results but the first one is more efficient.

ALGORITHM 1: Visual Metric for Adaptive LOD

```
Function SubdivisionCriterion($\mathcal{M}$, $\ell$, $\mathsf{f}$):
    $\delta^{\ell} \leftarrow 0;\ \delta^{\ell-1} \leftarrow 0$
    for each camera $\mathsf{c}$ do
        $\mathsf{pix}^{\ell} \leftarrow \mathtt{Rasterize}(\mathsf{Tess}^{\ell}(\mathcal{M}), \mathsf{f}, \mathsf{c})$
        $\delta^{\ell} \leftarrow \delta^{\ell} + \Phi(R(\mathsf{Tess}^{\ell}(\mathcal{M}), \mathsf{c}), \mathsf{pix}^{\ell})$
        $\mathsf{pix}^{\ell-1} \leftarrow \mathtt{Rasterize}(\mathsf{Tess}^{\ell-1}(\mathcal{M}), \mathsf{f}, \mathsf{c})$
        $\delta^{\ell-1} \leftarrow \delta^{\ell-1} + \Phi(R(\mathsf{Tess}^{\ell-1}(\mathcal{M}), \mathsf{c}), \mathsf{pix}^{\ell-1})$
    end
    Return $(\delta^{\ell-1} - \delta^{\ell}) / \delta^{\ell-1}$
End Function
```

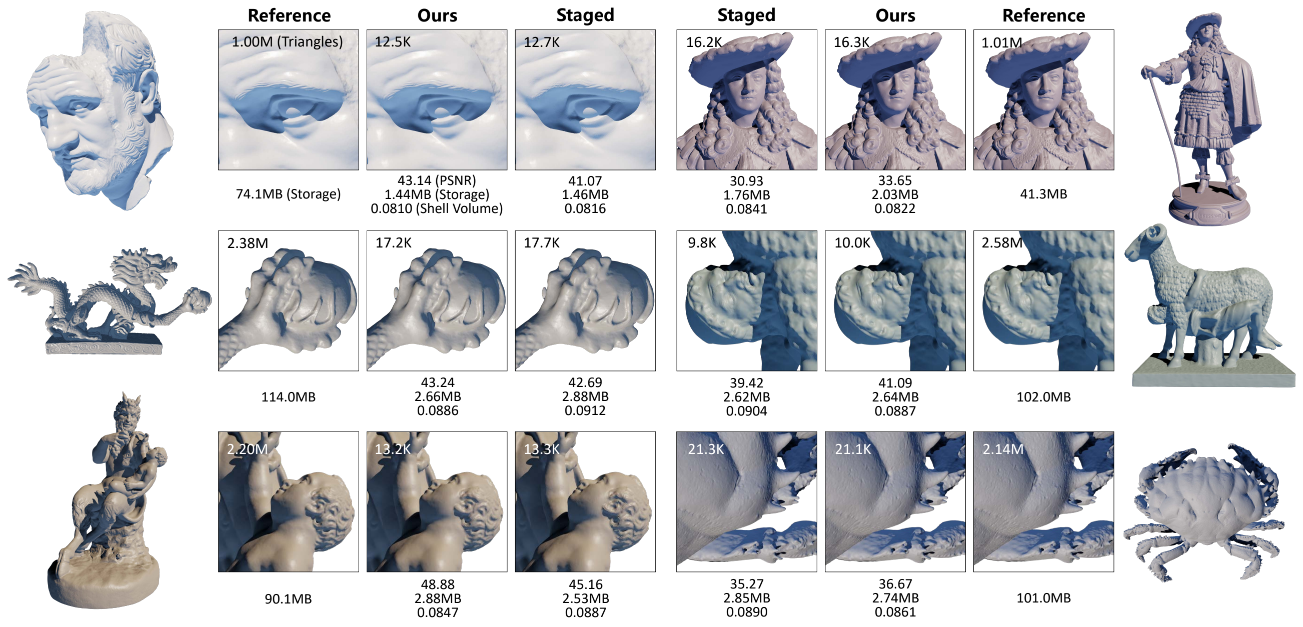


Figure 5. μ**-mesh construction results.** "Staged" denotes the previous state-of-the-art method proposed in [28]. With the similar number base triangles (upper-left corner), our method exhibits better visual quality, and significant numeric improvements in terms of PSNR of rendered μ-mesh. Sizes of μ-mesh storage are affected by subdivision stopping criteria. The similar storages and the better visual effects together suggest the effectiveness of our visual-based criterion. Moreover, our method also performs better in terms of shell volume.

4. Experiments

All experiments were performed on a Linux workstation (i9 12900k and RTX 3090). The parametric μ-mesh is implemented as a PyTorch [39] neural module, rendered with nvdiffrast differentiable rasterizer [21], along with a spherical harmonics shading model [43]. In order to meet a minimum requirement of a scene for rendering, we set all the mesh surfaces to smooth diffuse material, and adopt *Kloppenheim (©3DHEVEN)* as the environment map.

4.1. Implementation Details

Optimization. We optimize the parameters by applying UNIFORMADAM optimizer, a modified ADAM [18] by Nicolet et al. [34], with the same learning rate 2e-3 for offsets $\overline{\mathcal{O}}$ and displacements $\overline{\mathcal{D}}$. All the models in our experiments are trained for 1600 iterations, in which the 50, 150, 400, and 800 iterations are milestones that will trigger the subdivision. The first 50 iteration is warmup and the last 200 iteration is the displacement tuning with frozen base vertex. After each subdivision, the learning rates are decayed by 0.8 and 0.95 for $\overline{\mathcal{O}}$ and $\overline{\mathcal{D}}$ respectively; we apply cosine annealing for last 200 iteration, lowering the learning rate of $\overline{\mathcal{D}}$ to 0 at the end of optimization. To balance the losses, regularizer terms $\mathcal{L}_{sv}$ and $\mathcal{L}_{iso}$ are weighted by 10 and 100.

Mesh Decimation. The base mesh in our approach is initialized with a decimated low-polygon mesh, generated by a GPU-based QEM [12] family of decimation [38], which is much lighter compared with [28]. For meaningful comparison, we keep the similar number of base triangles with [28],

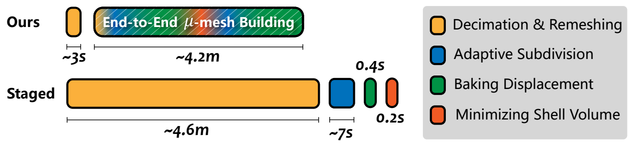


Figure 6. μ**-mesh construction time breakdown.** Evaluated on an extreme mesh (DARK_FINGER_REEF_CRAB, 2.14M triangles).

and consequently the similar μ-mesh compression ratio.

Datasets. We demonstrate the effectiveness of our approach with extensive batch μ-mesh construction for high-resolution meshes. The mesh dataset used in experiments is inherited from [28], including 121 meshes with face numbers ranging from 0.1M (million) to 4M. We divide these meshes into three subsets: medium ($<$ 1M, contains 37 meshes), large (1M $\sim$ 2M, 34), and extreme ($>$ 2M, 50).

4.2. μ-mesh Construction Results

Table 1 shows comparative results and performances. For base mesh (after optimization), we report vertex quadrics geometry error [12] and isotropy; for μ-mesh, we report rendering PSNR that is averaged on 16 evaluation views, shell volume, and construction time. Note that these evaluation views are randomly selected and have no overlap with training views of our method. Simplygon [29] lags a lot behind previous state-of-the-art method proposed by Maggiordomo et al. [28] (also denoted as *Staged* for brevity), while our method exhibits significant improvements. In addition to improvements on the metrics that directly measure μ-mesh quality (underlined), our method also shows comparable results on base mesh. *Nevertheless, we hold that the*

	Metrics	Simplygon μ-mesh	Simplygon Base	Maggiordomo et al. μ-mesh	Maggiordomo et al. Base	Ours μ-mesh	Ours Base
	Geom. Err. ↓	-	4.78	-	4.52	-	**4.46**
	Isotropy ↑	-	0.902	-	**0.747**	-	0.729
Medium	<u>PSNR</u> ↑	37.22	-	41.61	-	<u>**42.93**</u>	-
(< 1M, 37)	<u>Volume</u> ↓	0.114	-	0.085	-	<u>**0.083**</u>	-
	Time (m) ↓	**1.3**	-	2.0	-	3.9	-
	Geom. Err. ↓	-	2.40	-	**2.63**	-	2.65
	Isotropy ↑	-	0.961	-	0.742	-	**0.750**
Large	<u>PSNR</u> ↑	36.63	-	42.42	-	<u>**44.18**</u>	-
(1M ~ 2M, 34)	<u>Volume</u> ↓	0.108	-	0.086	-	<u>**0.082**</u>	-
	Time (m) ↓	**2.5**	-	4.0	-	4.3	-
	Geom. Err. ↓	-	2.60	-	1.84	-	**1.80**
	Isotropy ↑	-	0.954	-	**0.751**	-	0.747
Extreme	<u>PSNR</u> ↑	35.02	-	42.61	-	<u>**44.89**</u>	-
(> 2M, 50)	<u>Volume</u> ↓	0.124	-	0.089	-	<u>**0.083**</u>	-
	Time (m) ↓	**3.1**	-	5.3	-	4.8	-

Table 1. **μ-mesh construction on 121 high-resolution models.** Both methods have similar reduction ratio. (Geom. Err. $\times 10^{-5}$)

base mesh's geometry error and isotropy only have rough correlations to the final μ-mesh quality, a good practice is to focus on final/global objectives on μ-mesh. We qualitatively compare the constructed μ-mesh in Fig. 5. Our visual-based subdivision criterion performs well in determining subdivisions according to surface details. We show the visual effect and the storage together since they jointly validate the effectiveness of our adaptive level of detail strategy.

Quality over Decimation Ratio. We experiment μ-mesh construction at different decimation ratios, demonstrated in Fig. 7. Our method shows stronger competitiveness as the decimation ratio increases, because of a large one could raise more demands on global optimization.

Time Breakdown. Figure 6 demonstrates the time breakdown of converting an extreme mesh to μ-mesh. The previous staged method [28], as indicated, devotes most of the time in decimation, pursuing a base mesh that can well reproduce the input surface, with properties of small shell volume and isotropy. For staged scheme, we believe it's a good practice to concentrate on the base mesh, due to the final μ-mesh quality (reprojectability, shell volume) are almost determined right after decimation. Instead, we consider the construction stages as a whole, which is optimized end-to-end with some global objectives. Consequently, our scheme has large potential to achieve greater efficiency and quality.

4.3. Ablation Studies

Table 2 shows the impact of key components in our system, including progressive scheme, reparameterization, objectives, and visual-guided adaptive LOD. ***(i)*** Experiments without progressive scheme and reparameterization degenerate the quality most, which we attribute to the lack of anti-distortion during geometry evolution. ***(ii)*** The underlined values indicate the effectiveness of the two introduced objectives. Without any of them, the rendering PSNR of the μ-mesh degrades. ***(iii)*** By replacing our visual-guided subdivision stopping criterion with Chamfer Distance, the μ-mesh quality also deteriorates, where the increase of the shell volume may stem from inappropriate stopping.

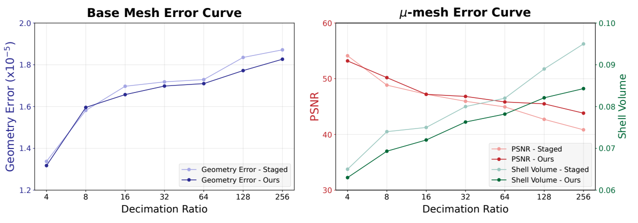


Figure 7. **μ-mesh quality over decimation ratio**, evaluated on extreme subset. Our method exhibits consistent improvements.

We further investigate the effect of $\mathcal{L}_{sv}$ since the shell volume is the most critical factor besides reprojectability, as shown in Fig. 8. Although the rendering loss during training is almost the same, the μ-mesh rendering PSNR (of this mesh) decreases from 33.65 to 33.40 after removing $\mathcal{L}_{sv}$, suggesting $\mathcal{L}_{sv}$ leads to smaller displacements, which then reduce the μ-mesh quantization losses.

					μ-mesh		Base mesh	
Prog.	Reparam.	$\mathcal{L}_{sv}$	$\mathcal{L}_{iso}$	Adap. LOD	PSNR↑	Volume↓	Geom. Err.↓	Iso.↑
✗	✓	✓	✓	✓	43.65	0.084	1.80	0.745
✓	✗	✓	✓	✓	42.25	0.087	1.85	0.740
✓	✓	✗	✓	✓	44.60	<u>0.088</u>	1.79	0.746
✓	✓	✓	✗	✓	44.52	0.082	1.83	<u>0.730</u>
✓	✓	✓	✓	✗	44.78	0.085	1.81	0.747
✓	✓	✓	✓	✓	44.89	0.083	1.80	0.747

Table 2. **Ablation studies of key components** on extreme subset.

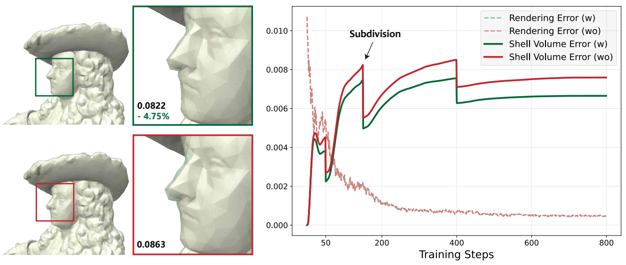


Figure 8. **Ablation studies on the effects of minimizing shell volume.** We show the first 800 iterations for better visualization.

5. Conclusion

In this article, we take steps towards constructing higher quality μ-mesh. We identify the capability of previous methods [28, 29, 38] is mainly limited by their staged behaviour. We also indicate that a rendering guidance is important for μ-mesh construction, since the 3D model is eventually rendered to users. By treating the construction as an end-to-end optimization task, supported with several elaborately designed differentiable operations and objectives that well instantiate the demands of a high quality μ-mesh, our method achieves significant improvements.

6. Acknowledgement

This work was supported by National Science Foundation of China (U20B2072, 61976137). This work was also partly supported by SJTU Medical Engineering Cross Research Grant YG2021ZD18.

References

[1] Pierre Alliez, Eric Colin De Verdire, Olivier Devillers, and Martin Isenburg. Isotropic surface remeshing. In *2003 Shape Modeling International.*, pages 49–58. IEEE, 2003. 3, 6

[2] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. Recent advances in remeshing of surfaces. *Shape analysis and structuring*, pages 53–82, 2008. 6

[3] Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. Unbiased warped-area sampling for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):1–18, 2020. 3

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 5

[5] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, 2010. 1, 3

[6] Rui Chen, Yongwei Chen, Ningxin Jiao, and Kui Jia. Fantasia3d: Disentangling geometry and appearance for high-quality text-to-3d content creation. *arXiv preprint arXiv:2303.13873*, 2023. 3

[7] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Advances in neural information processing systems*, 32, 2019. 3

[8] Joel Daniels, Cláudio T Silva, Jason Shepherd, and Elaine Cohen. Quadrilateral mesh simplification. *ACM transactions on graphics (TOG)*, 27(5):1–9, 2008. 2, 6

[9] Valentin Deschaintre, Yiming Lin, and Abhijeet Ghosh. Deep polarization imaging for 3d shape and svbrdf acquisition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15567–15576, 2021. 3

[10] Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Pierre Poulin, Fabrice Neyret, and Victor Ostromoukhov. Linear efficient antialiased displacement and reflectance mapping. *ACM Transactions on Graphics (TOG)*, 32(6):1–11, 2013. 1

[11] Epic Games. Unreal engine 5: Nanite, 2020. https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5. 1

[12] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997. 1, 3, 6, 7

[13] Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. Appearance-driven automatic 3d model simplification. In *EGSR (DL)*, pages 85–97, 2021. 3

[14] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org. 2

[15] Zhongshi Jiang, Ziyi Zhang, Yixin Hu, Teseo Schneider, Denis Zorin, and Daniele Panozzo. Bijective and coarse high-order tetrahedral meshes. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021. 6

[16] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018. 3

[17] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (ToG)*, 42(4):1–14, 2023. 3

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[19] Julian Knodt, Zherong Pan, Kui Wu, and Xifeng Gao. Joint uv optimization and texture baking. *ACM Transactions on Graphics*, 43(1):1–20, 2023. 3

[20] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020. 2, 3

[21] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):1–14, 2020. 4, 7

[22] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 3

[23] Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, David Levin, Christian Rossi, and Hans-Peter Seidel. Differential coordinates for interactive mesh editing. In *Proceedings Shape Modeling Applications, 2004.*, pages 181–190. IEEE, 2004. 4

[24] Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. On centroidal voronoi tessellation—energy smoothness and fast computation. *ACM Transactions on Graphics (ToG)*, 28(4):1–17, 2009. 1

[25] C Loop. Smooth subdivision surfaces based on triangles, master's thesis. *University of Utah, Department of Mathematics*, 1987. 3

[26] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII 13*, pages 154–169. Springer, 2014. 3

[27] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 3

[28] Andrea Maggiordomo, Henry Moreton, and Marco Tarini. Micro-mesh construction. *ACM Transactions on Graphics (TOG)*, 42(4):1–18, 2023. 1, 2, 3, 5, 6, 7, 8

[29] Microsoft. Simplygon: The standard in 3d games content optimization, 2022. https://www.simplygon.com/. 1, 2, 5, 7, 8

[30] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf:

Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 3

[31] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 3

[32] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting triangular 3d models, materials, and lighting from images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8280–8290, 2022. 3

[33] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Laplacian mesh optimization. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 381–389, 2006. 4

[34] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021. 2, 3, 4, 5, 7

[35] Baptiste Nicolet, Fabrice Rousselle, Jan Novak, Alexander Keller, Wenzel Jakob, and Thomas Müller. Recursive control variates for inverse rendering. *ACM Transactions on Graphics (TOG)*, 42(4):1–13, 2023. 3

[36] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020. 3

[37] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(4):146–1, 2020. 3

[38] NVIDIA. Nvidia ada gpu architecture, 2022. https://www.nvidia.com/it-it/geforce/ada-lovelace-architecture, https://images.nvidia.com/aem-dam/Solutions/geforce/ada/ada-lovelace-architecture/nvidia-ada-gpu-architecture-whitepaper-1.03.pdf. 1, 2, 3, 4, 5, 6, 7, 8

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 7

[40] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming techniques for high-performance graphics and general-purpose computation (gpu gems)*. Addison-Wesley Professional, 2005. 5

[41] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics*, 2(1):15–36, 1993. 4

[42] Serban D Porumbescu, Brian Budge, Louis Feng, and Kenneth I Joy. Shell maps. *ACM Transactions on Graphics (TOG)*, 24(3):626–633, 2005. 5

[43] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 497–500, 2001. 7

[44] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 2

[45] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 3

[46] Olga Sorkine. Laplacian mesh processing. *Eurographics (State of the Art Reports)*, 4(4), 2005. 4

[47] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and H-P Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184, 2004. 4

[48] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Path replay backpropagation: differentiating light paths using constant memory and linear time. *ACM Transactions on Graphics (TOG)*, 40(4):1–14, 2021. 3

[49] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022. 3

[50] Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. Monte carlo estimators for differential light transport. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021.

[51] Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. A differential theory of radiative transfer. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

[52] Cheng Zhang, Bailey Miller, Kan Yan, Ioannis Gkioulekas, and Shuang Zhao. Path-space differentiable rendering. *ACM transactions on graphics*, 39(4), 2020.

[53] Cheng Zhang, Zihan Yu, and Shuang Zhao. Path-space differentiable rendering of participating media. *ACM Transactions on Graphics (TOG)*, 40(4):1–15, 2021. 3

[54] Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. Physg: Inverse rendering with spherical gaussians for physics-based material editing and relighting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5453–5462, 2021. 3