DOI: 10.1111/cgf.14655
Pacific Graphics 2022
N. Umetani, E. Vouga, and C. Wojtan
(Guest Editors)

Volume 41 (2022), Number 7

MeshFormer: High-resolution Mesh Segmentation with Graph Transformer

Yuan Li[1], Xiangyang He[1], Yankai Jiang[1], Huan Liu[1], Yubo Tao[1*], Lin Hai[1*]

[1]State Key Lab of CAD&CG, Zhejiang University

Abstract

Graph transformer has achieved remarkable success in graph-based segmentation tasks. Inspired by this success, we propose a novel method named MeshFormer for applying the graph transformer to the semantic segmentation of high-resolution meshes. The main challenges are the large data size, the massive model size, and the insufficient extraction of high-resolution semantic meanings. The large data or model size necessitates unacceptably extensive computational resources, and the insufficient semantic meanings lead to inaccurate segmentation results. MeshFormer addresses these three challenges with three components. First, a boundary-preserving simplification is introduced to reduce the data size while maintaining the critical high-resolution information in segmentation boundaries. Second, a Ricci flow-based clustering algorithm is presented for constructing hierarchical structures of meshes, replacing many convolutions layers for global support with only a few convolutions in hierarchy structures. In this way, the model size can be reduced to an acceptable range. Third, we design a graph transformer with cross-resolution convolutions, which extracts richer high-resolution semantic meanings and improves segmentation results over previous methods. Experiments show that MeshFormer achieves gains from 1.0% to 5.8% on artificial and real-world datasets.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ *Neural networks; Shape analysis;*

1. Introduction

High-resolution meshes have been widely used in various real-world applications [MPS14, TST96]. The demand for automated analysis and understanding of high-resolution meshes is increasing. As a crucial part of understanding high-resolution meshes, the semantic segmentation of high-resolution meshes becomes a key task [LHMR08]. Nevertheless, different from general mesh segmentation, high-resolution mesh segmentation has to consider the unstable mesh quality due to the difficulties of repairing high-resolution meshes [ACK13]. The unstable mesh quality of high-resolution meshes means these meshes may not be closed manifolds, which makes methods based on convolution neural networks (CNNs) unworkable and urges us to develop graph-based methods without closed manifold constraints [HHF*19, HLG*22]. Compared with other graph-based methods, the graph transformer [YJK*19], which offers state-of-the-art performance on graph-based segmentation [ZZH*22], is a promising option. Consequently, we develop a new method named MeshFormer for applying the graph transformer to high-resolution mesh segmentation.

Direct application of the graph transformer to high-resolution meshes does not work because of the requirements of unacceptable computing resources. A feasible solution is to reduce the data size: simplify raw meshes, apply the mesh segmentation methods to the simplified meshes, and map the segmentation results back to raw meshes. However, general simplification may lead to the loss of high-resolution information in segmentation boundaries. The immediate consequences of this information loss are ragged segmentation boundaries in simplified meshes, leading to prior errors for learning and mapping, as shown in Fig. 1(a, b). To avoid ragged segmentation boundaries, we develop a boundary-preserving simplification algorithm that prioritizes protecting segmentation boundaries over meaningless details during simplification. The simplified mesh in Fig. 1(d) demonstrates the utility of boundary-preserving simplification: the data size is reduced while the high-resolution information in segmentation boundaries is preserved, i.e., avoiding prior learning and mapping errors.

Despite the reduction in data size, the graph transformer's large model size still makes it unusable within limited computing resources. In other words, directly applying the graph transformer to simplified meshes necessitates many convolution layers to obtain the global support of meshes, which may not be acceptable given limited computing resources. The standard solution is to use clustering algorithms [HJZS20, TPT17, SYW*22] to build hierarchical structures of meshes so that a few convolutions in hierarchical structures can obtain global support. Therefore, we devise a new clustering algorithm based on Ricci flow [CLN06, NLLG19] to construct hierarchical structures, i.e., different resolutions of the same mesh. Ricci flow is used because it has adaptive clustering pa-

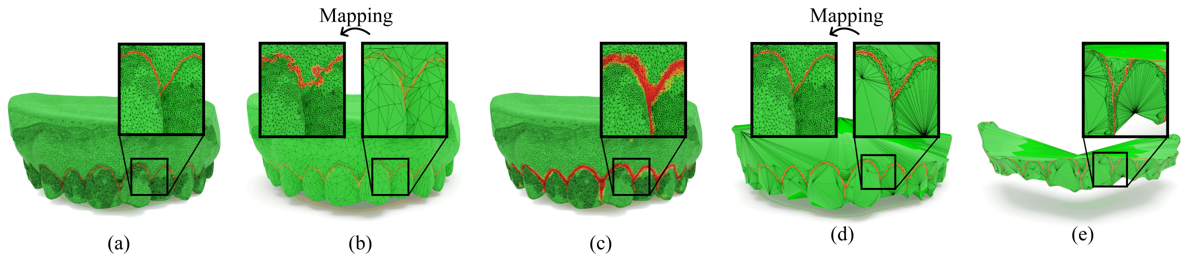


Figure 1: *The segmentation boundaries between tooth and gums (red areas). (a) raw tooth mesh, (b) simplified mesh by general simplification (10000 vertices), (c) prediction result of segmentation boundaries, (d) simplified mesh by boundary-preserving simplification (10000 vertices), (e) same mesh with (d) but removing high-frequency information (suggested by [SACO22]).*

rameters for diverse meshes and excellent noise resistance in clustering [ZSG10, NLLG19]. The excellent noise resistance helps our algorithm work reliably in the presence of noise caused by simplification, e.g., the high-frequency noise (unreal sharp shapes) in Fig. 1(d). The adaptive clustering parameters make our algorithm robust to diverse meshes. Experiments show that our clustering results are closer to ground truth than other clustering algorithms, contributing to better segmentation results. With the above two components of MeshFormer, the data and model sizes can be controlled within an acceptable range, so massive computing resources are no longer required. Furthermore, critical high-resolution information in segmentation boundaries is retained with the boundary-preserving principle.

The final component is designed to mine the mesh semantic meanings by convolutions in hierarchical structures. Applying graph attention convolutions into same resolutions of hierarchical structures separately, i.e., **convolutions within same resolutions**, is a popular choice for mining semantic meanings, such as graphUnets [HJZS20, GJ19]. However, this choice is not enough for high-resolution segmentation because it ignores the relationship between the semantic meanings of different resolutions. Recent studies [WSC*20, ZGZ*21] found that introducing cross-resolution convolutions can leverage the relationship between the semantic meanings of different resolutions and enrich high-resolution semantic meanings, which contributes to better high-resolution segmentation results. In other words, without the cross-resolution convolutions, i.e., **convolutions between different resolutions**, some semantic meanings may be overlooked, resulting in inaccurate segmentation results. Consequently, we propose a new graph transformer architecture with cross-resolution convolutions, i.e., representing hierarchical structures as heterogeneous directed graphs, adding extra cross-resolution edges with different types, and applying different convolutions to different edge types. In this way, cross-resolution convolutions are implemented by the convolutions on cross-resolution edges.

In a nutshell, a new method named MeshFormer is proposed to segment high-resolution meshes using the graph transformer. Three components are designed to overcome three challenges, including a boundary-preserving simplification to remove unimportant details for data size reduction, a clustering algorithm to build hierarchical structures for model size reduction, and a graph transformer to achieve cross-resolution convolutions for richer semantic meanings. In this way, MeshFormer can achieve better segmentation results using limited computing resources. The following is a summary of our contributions.

- A new graph transformer architecture with cross-resolution convolutions is proposed for high-resolution mesh segmentation. Richer high-resolution semantic meanings are extracted, and better segmentation results are achieved. Experiments show MeshFormer achieves gains ranging from 1.0% to 5.8%.
- A boundary-preserving simplification algorithm is introduced, retaining important high-resolution information in segmentation boundaries during simplification.
- A new clustering algorithm based on Ricci flow is developed for building hierarchical structures, contributing to the model size reduction and great segmentation results.

2. Related work

2.1. High-resolution segmentation for 3D data

High-resolution mesh segmentation can be transformed into a similar segmentation problem on other data, such as point clouds, voxels, and images. However, converting high-resolution meshes to other data presents more intractable issues. When meshes are converted to point clouds, all topology information is lost, making it difficult for point cloud-based methods [QYSG17, TQD*19] to achieve the same performance as mesh-based methods [SACO22, HLG*22]. Another option is the voxel, which is a shape representation of the entire 3D space. Nevertheless, in high-resolution volumetric data, extra 3D representations beyond surfaces will result in unacceptably high computation consumption [ÇAL*16, FBD*21]. Some researchers prefer to project meshes as images [LBD17, SBR16, SBZB15] so that mesh segmentation can be implemented by 2D image segmentation. Unfortunately, these methods are sensitive to viewpoints due to occlusion and distortion in projections. Mesh-based methods are free from the aforementioned issues and have shown potential, so developing semantic segmentation methods based on high-resolution meshes would be preferable.

2.2. High-resolution mesh segmentation

Traditional shape descriptor-based segmentation methods [BKR*16, BK10] can be easily extended to high-resolution mesh segmentation, but these methods cannot achieve good results due to the lack of semantic meanings for 3D shapes. Although deep learning-based methods [XLZ*20] can access semantic meanings, it is difficult for these methods to implement high-resolution mesh segmentation due to limited computing resources. Mesh simplification, especially boundary-preserving simplification, is nearly an unavoidable compromise in this case. Consequently, we apply these deep learning-based methods to simplified meshes and discuss their benefits and drawbacks. In addition, deep learning-based mesh segmentation is a hot topic in computer graphics. For an overview of previous methods, we refer the readers to recent surveys [WZ22, XLZ*20]. This section briefly reviews the state-of-the-art methods that are most relevant to our work. These methods fall into two categories: methods based on regular structures and methods based on irregular structures.

Methods based on regular structures. Motivated by the phenomenal success of CNNs in image segmentation, some researchers prefer to introduce CNNs to mesh segmentation. The critical challenge is that CNNs require inputs with regular structure, but meshes are irregular naturally. Pioneering works [TPKZ18, HSBH*19] in this direction mainly depend on parameterization to map meshes to a regular domain so that CNNs can work. However, these methods are insensitive to shape information due to parameterization. Recently, some studies [HHF*19, HLG*22] avoid parameterization by defining the convolutions on meshes directly and achieve state-of-the-art performance, but these methods rely on the closed manifold surfaces of meshes. Because many high-resolution meshes do not have closed manifold surfaces, these methods may not be suitable for high-resolution mesh segmentation. MeshFormer is superior to these methods for high-resolution mesh segmentation because it is free from parameterization and closed manifold constraints.

Methods based on irregular structures regard meshes as irregular graphs, avoiding the requirement of regular structures. These methods can be divided into two main types: methods based on Graph Laplacians and methods based on spatial networks.

Methods based on Graph Laplacians. The Graph Laplacians [MACO91, SHKVL09] are powerful tools for shape analysis, inspiring various spectral-based mesh segmentation methods [Sha08, LZ04]. However, there are two issues with these methods. On the one hand, since the Laplacian eigenfunctions are not the same between different meshes, transferring the coefficients or learned filters from one mesh to another is difficult. On the other hand, such processing (e.g., the truncation of high-frequency information and diffusion) may lead to high-frequency information loss. As shown in Fig. 1(e), once high-frequency information in simplified meshes is overlooked, shape information will suffer severe distortion, e.g., flattened segmentation boundaries, potentially resulting in more segmentation errors. Recently, Sharp et al. [SACO22] and Smirnov et al. [SS21] presented their solutions to tackle these two issues. However, these solutions cannot eliminate high-frequency information loss in such processing. Unlike these methods, MeshFormer does not involve the Laplacian eigenfunctions and the above processing resulting in high-frequency information loss. In addition, MeshFormer considers contextual and hierarchical information of meshes.

Methods based on spatial networks treat meshes as irregular graphs from the spatial perspective. Recent segmentation methods in this direction can be divided into two kinds: random walk methods and graph convolution network (GCN)-based methods. For the first kind of method, Lahav et al. [LT20] proposed MeshWalker to build shape understanding based on the shape changes along 1D walking paths and implement mesh segmentation with this shape understanding. Nevertheless, noise in simplified meshes interferes with shape changes, resulting in inaccurate shape understanding and segmentation results. For the second kind of method, GCN-based methods, such as CurvaNet [HJZS20], combine GCNs and U-Nets to implement mesh segmentation. There are two flaws in these methods. Firstly, because GCNs are low-pass filters, they rapidly lose most high-frequency information. Second, recent studies [WSC*20, WYC*21] suggest that U-Nets may not be ideal for high-resolution mesh segmentation because they ignore the cross-resolution convolutions. MeshFormer is also a method based on spatial networks. However, MeshFormer is built with attention mechanisms [VSP*17, VCC*18] and Ricci flow [Kap03], which means it has strong anti-noise capabilities and alleviates high-frequency information loss efficiently [ZSG10, FLL22, LSM*21]. Thus, MeshFormer is more suitable for simplified meshes than previous methods. MeshFormer also overcomes the drawbacks of U-Nets with cross-resolution convolutions and achieves better segmentation results with richer semantic meanings.

3. Method

As shown in Fig. 2, MeshFormer is made up of three components. To begin with, we apply boundary-preserving simplification on high-resolution meshes to reduce data size while preserving critical high-resolution information. Secondly, we execute Olliver Ricci flow [Oll09, BLL*20] on simplified meshes to implement clustering on meshes and obtain hierarchical structures. Finally, we create a new graph transformer called mesh transformer to implement semantic segmentation.

3.1. Boundary-preserving simplification

A mesh can be defined as $M = (V, E)$, where $V \in \mathbb{R}^{\lambda\times 3}$ is the coordinate matrix of λ vertices and $E \in \mathbb{R}^{\lambda\times\lambda}$ is the corresponding adjacent matrix of M. If faces containing this vertex have different labels, this vertex is considered the *segmentation boundary*. The goal of boundary-preserving simplification is to simplify the mesh while protecting high-resolution segmentation boundaries from simplification to avoid ragged segmentation boundaries. The following is a description of our algorithm in detail. **First**, we encode the shape information of M to a shape descriptor matrix $F = (F_1, ..., F_i, ..., F_\lambda)^T$, where F_i is the shape descriptor of i-th vertex v_i. The key challenge is encoding rich shape information to F costlessly. A natural solution is to form each descriptor by concatenating a set of shape features with low calculation consumption and comprehensive scales. Therefore, we select features with cheap calculations from both local and global perspective and fuse these features as shape descriptor. The local features include coordinate after principal component analysis correction, normal vector, shape diameter [BKR*16], fast point feature histogram [RBB09], and four kinds of curvature (mean, maximum, minimum, average). The

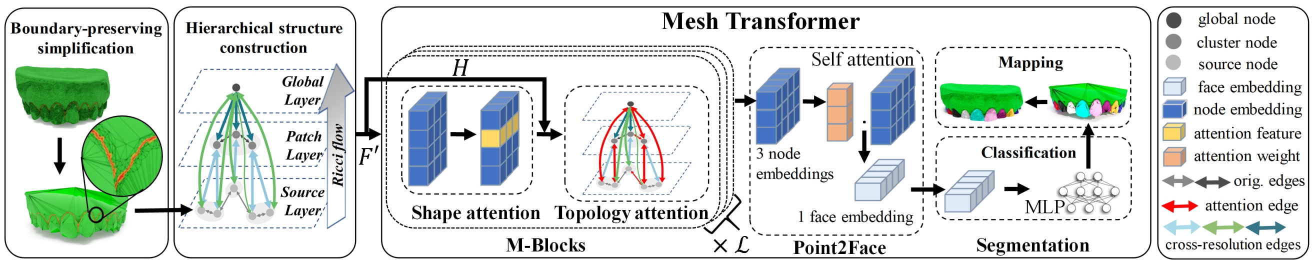


Figure 2: *The structure of MeshFormer: boundary-preserving simplification, clustering algorithm for building hierarchical structures, mesh transformer.*

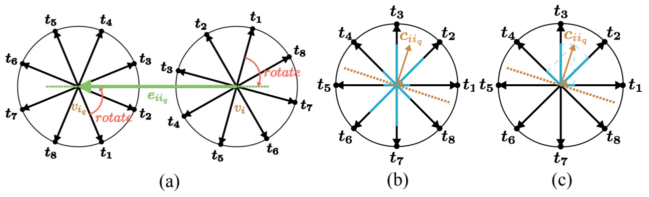


Figure 3: *(a) Alignment of two vertices on the edge e_{ii_q}. (b) Projection from $\mathbf{c}_{ii_q}$ to k different directions in [HJZS20]. (c) Projection from $\mathbf{c}_{ii_q}$ to k different directions in MeshFormer. The blue lines represent the projected curvatures.*

global features include heat kernel signatures and scale-invariant heat kernel signatures [BK10]. **Second**, we use shape descriptor matrix F as the input and train a XGBoost [CHB*15] to calculate the probability matrix $P = (P_1, ..., P_i, ..., P_\lambda)^T$, where P_i is the probability that v_i belong to segmentation boundaries. **Finally**, we apply the QEM algorithm [GH97,OKV15] to simplify meshes by collapsing edges and protecting vertices with high probabilities being segmentation boundaries in simplification. Specifically, given an edge e_{ij} between v_i and v_j, the error of collapsing e_{ij} is redefined from the quadratic error Q_{ij} to $Q_{ij} \cdot exp(max(P_i, P_j))$. By introducing the weight $exp(max(P_i, P_j))$, the error of collapsed edges on segmentation boundaries exceeds non-boundary parts, so QEM prefers to collapse edges on non-boundary parts. When v_i and v_j collapse to a new vertex v_{new}, P_{new} can be calculated as $max(P_i, P_j)$.

In a word, boundary-preserving simplification increases the errors of collapsed edges on segmentation boundaries by assigning large weights, thereby protecting segmentation boundaries from simplification and avoiding ragged segmentation boundaries. We choose XGBoost rather than deep learning-based methods in this process because of the faster speed and less computing consumption of decision tree algorithms [TGI18]. In this way, the data size can be reduced to an acceptable range, and high-resolution information in segmentation boundaries is retained.

3.2. Hierarchical structure construction

Pooling layers [HHF*19, HLG*22] and clustering algorithms [HJZS20, SYW*22] are two popular ways to construct hierarchical structures. We use clustering algorithms to build hierarchical structures because previous pooling layers in mesh segmentation require uniform vertex distributions or closed manifolds. However, the hierarchical structures produced by different clustering algorithms may produce different segmentation results. Suitable hierarchical structures have clustering results closer to ground truth, similar to a better pre-segmentation. Previous clustering algorithms [LZ04, HJZS20] may struggle to build suitable hierarchical structures due to the noise in simplified meshes and fixed clustering parameters (e.g., the same cluster number for five-teeth mesh and 32-teeth mesh). Motivated by Ricci flow's anti-noise ability [ZSG10] and convergence guarantee [Tao10], we develop an adaptive clustering algorithm based on Ricci flow.

Ricci flow, especially graph Ricci flow, is a mature tool for clustering on graph structures. By shrinking the areas with positive Ricci curvature to a point and spreading out the areas with negative Ricci curvature, Ricci flow can implement the adaptive clustering on meshes naturally. The key is how to define the positive and negative Ricci curvatures, i.e., a Riemann metric for clustering on meshes [Tao10]. Previous Riemann metrics [NLLG19, NLGG18] based only on topology produce a clustering process guided by topology similarity, which is not suitable for clustering on meshes. Therefore, one of our main contributions is a new Riemann metric based on shape information to achieve the clustering guided by shape similarity.

Our clustering algorithm includes three steps: (a) We encode the shape information to edge weights of meshes. Specifically, we calculate the difference between two connected vertices' directional curvatures as edge weight. (b) We define a new Riemann metric based on shape information (i.e., edge weights) and implement clustering with Ricci flow so that vertices with similar directional curvature can be clustered as one patch. (c) We build a three-layer hierarchical structure based on the clustering results of Ricci flow. The specifics of these three steps are as follows:

(a) Edge weights based on shape information. Firstly, we calculate the original directional curvatures of each vertex. Given an edge between i-th vertex v_i and the q-th adjacent vertex v_{i_q} of v_i, the curvature c_{ii_q} along the edge $\boldsymbol{e}_{ii_q}$ can be calculated as $c_{ii_q} = 2\boldsymbol{n}_i \cdot \frac{\boldsymbol{e}_{ii_q}}{||\boldsymbol{e}_{ii_q}||^2}$, where $\boldsymbol{e}_{ii_q} = \boldsymbol{v}_{i_q} - \boldsymbol{v}_i$, $\boldsymbol{v}_i$ represents the coordinate of vertex v_i, and $\boldsymbol{n}_i$ is the normal vector of v_i. Calculating the curvatures along all edges containing v_i, we can obtain the original directional curvature $C_i = (c_{i1}, ..., c_{im})^T$ of v_i, where m is the degree of v_i. Similarly, we can get the original directional curvature of any vertex.

Secondly, we define the difference between two connected vertices' directional curvatures as edge weight. For two connected vertices v_i and v_{i_q}, directly calculating the difference of C_i and C_{i_q} is not feasible, because the edge directions of C_i and C_{i_q} are not aligned. A suitable choice is to project the C_i and C_{i_q} to k aligned directions $\boldsymbol{t}_1 \sim \boldsymbol{t}_k$, where k is 8 suggested by CurvaNet [HJZS20]. Thus, we generate k directions for connected vertices randomly and rotate the k directions until their k directions are aligned, as shown in Fig. 3(a). Then, the aligned directional curvatures $D_i = (d_{i1}, ..., d_{ik})^T$ of vertex v_i can be calculated as $D_i = A^i \cdot C_i$, where $A^i \in \mathbb{R}^{k \times m}$ is a projection matrix and A^i_{pq} is $max(0, \boldsymbol{t}_p \cdot \frac{\boldsymbol{e}_{ii_q}}{||\boldsymbol{e}_{ii_q}||})$. As shown in Fig. 3(b,c), A^i is designed to project each curvature $c_{ii_q} \in C_i$ to k-aligned directions. Compared to the previous projection in Fig. 3(b), only the positive projections of curvature are preserved in our method, so our directional curvature is more sensitive to curvature changes and closer to the real distribution of curvatures. Similarly, we can get the aligned directional curvature D_{i_q} of v_{i_q}. Then, the directional curvatures' difference S_{ii_q} between v_i and v_{i_q} can be calculated as $\sum_{o=1}^{k} |d_{io} - d_{i_q o}|$. Applying the above operations to all edges, we encode the shape information to edge weight matrix $S \in \mathbb{R}^{\lambda \times \lambda}$.

(b) Adaptive clustering based on Ricci flow. We define a new Riemann metric based on shape information (i.e., edge weights) to implement clustering on meshes. Similar to previous Riemann metrics [JKLG08, NLLG19], our Riemann metric W is also based on the Wasserstein distance [Lot06]:

$$W(v_i, v_j) = \inf\{\int_{X \times Y} \mathsf{Cost}(x,y) d\gamma(x,y) | \gamma \in \Gamma(v_i, v_j)\}, \quad (1)$$

where $W(v_i, v_j)$ represents the Wasserstein distance between vertex v_i and vertex v_j, X and Y are the neighbors of v_i and v_j with similar directional curvature, $X \times Y$ represents all the vertex pairs including vertices from $x \in X$ and $y \in Y$, Cost is a function to sum all edge weights in the shortest path along edges from x to y, $\gamma(x,y)$ is a probability measure for the shortest path along edges from x to y, $\Gamma(v_i, v_j)$ is the collection of all possible probability measures of the shortest path along edges between X and Y, and inf() represents the infimum cost from X and Y. Different from previous Riemann metrics [NLLG19,NLLG19], our Riemann metric introduces the shape information, i.e., defines the Cost function with edge weights. In this way, our clustering is guided by shape similarity rather than topology similarity.

We use a fast seed expansion algorithm to find neighbors with similar directional curvature (i.e., X and Y in Eq. 1) for any vertex x_1. In the beginning, we initialize two seed sets: $\overline{X} = \{x_1\}$ for vertices and $\overline{U} = \varnothing$ for edges. Then, we collect neighbors of all vertices in $\overline{X}$ as $\overline{X}_{adj}$ and calculate the distance between $v_x \in (\overline{X}_{adj} \setminus \overline{X})$ and $\overline{X}$. The distance can be calculated effectively by aggregating the weights of edges between v_x and $v_y \in \overline{X}$. We add the nearest vertex $v_{nearest} \in (\overline{X}_{adj} \setminus \overline{X})$ to $\overline{X}$ and all edges between $v_{nearest}$ and v_y to $\overline{U}$. In the end, repeat the last step until the $\sum_{(x,y) \in \overline{U}} S_{xy}$ is more than En, where En is a constant threshold discussed in Sec. 4.5. In this way, by continuously adding the nearest vertex to $\overline{X}$, the neighbors with similar directional curvature to any vertex x_1 are extracted, which makes our Riemann metric W workable.

Next, based on the Riemann metric W, the Olliver Ricci curvature $\kappa(v_i, v_j)$ of e_{ij} can be described as:

$$\kappa(v_i, v_j) = 1 - \frac{W(v_i, v_j)}{S_{ij}}. \quad (2)$$

If two connected vertices have the similar directional curvature, they will share the same neighbors almost, and $W(v_i, v_j)$ is less than edge weight S_{ij} and close to 0. Therefore, the Ricci curvature $\kappa(v_i, v_j) > 0$, and then v_i and v_j will be clustered. In contrast, if two vertices have completely different directional curvatures, their neighbors are very different and $W(v_i, v_j)$ is much larger than S_{ij}, so the Ricci curvature $\kappa(v_i, v_j) < 0$, and then v_i and v_j will be pulled apart. By updating all edge weights as follows:

$$S_{ij}^{\alpha} = (1 - \kappa^{\alpha-1}(v_i, v_j)) \cdot S_{ij}^{\alpha-1}, \quad (3)$$

the vertices with the similar directional curvature will shrink to one point in space, and the vertices with different directional curvatures will separate gradually after several iterations, where S_{ij}^{α} and $S_{ij}^{\alpha-1}$ represent the edge weights in α-th and $(\alpha-1)$-th iterations, respectively, and S_{ij}^{0} is the edge weight S_{ij}. Following the mature workflow of Ricci flow [NLLG19], we use the maximum modularity to extract clusters after 40 iterations. Experiments in Sec. 4.2 show that our clustering results are closer to ground truth than other clustering methods. A specific example of our clustering algorithm can be found in Appendix A.

(c) The construction of hierarchical structures. We divided meshes into patches after the adaptive clustering based on Ricci flow. Naturally, a three-layer hierarchical structure for each mesh can be built, as shown in Fig. 2: a source layer including vertices, a patch layer including patches, and a global layer including the representation of the total mesh. To represent this 3-layer hierarchical structure, we construct a heterogeneous directed graph consisting of three types of nodes, i.e., source, patch, and global nodes.

Each vertex of mesh is converted into a source node in the source layer, each edge of the mesh is preserved, and the difference between two connected vertices' directional curvatures is used as edge weight. We convert each patch into a patch node in the patch layer, and the edge weight between two patch nodes is calculated as the sum of the weights of all edges between source nodes across these two patches. In the global layer, a global node is introduced to provide global support.

In the heterogeneous directed graph, we also add the edges between different layers (i.e., the cross-resolution edges) to enable convolutions between different resolutions. According to the clustering relationship provided by Ricci flow, we build the bi-directional edges between source nodes and patch nodes. The weight of each edge between the source node and patch node is initialized as the Euclidean distance from the source node to the centroid of the patch. The global node has bi-directional edges to any other node, and the initial edge weight is $1/\mathcal{N}$, where $\mathcal{N}$ is the number of nodes. Directed edges are assigned different types if their origin nodes or destination nodes have different node types. As shown in Fig. 2, we build a heterogeneous directed graph H for each mesh in this manner.

3.3. Mesh transformer

Motivation. As the core of transformer-based methods, attention mechanisms [VSP*17, VCC*18] have revolutionized graph-based segmentation tasks [MCB*22]. By learning a latent importance distribution of features, attention mechanisms selectively aggregate important features, contribute to more accurate and richer information, and help to achieve better segmentation results [MCB*22]. Thus, we develop the mesh transformer for mesh segmentation.

Input. Both topology and shape information are essential for mesh segmentation. Consequently, we specify our model's inputs from both topology and shape perspectives. We pick the heterogeneous directed graph H constructed in Sec. 3.2 as the input for topology information. We reuse the shape descriptor matrix in Sec. 3.1 as source nodes' features and obtain the features of patch nodes and global nodes by the averaging operation. In this way, we obtain the node feature matrix $F' \in \mathbb{R}^{\mathcal{N} \times \mathcal{D}}$ as the input for shape information.

Model design. As shown in Fig. 2, the mesh transformer is composed of three components: multiple M-Blocks, a Point2Face layer, and a multilayer perceptron. M-Blocks capture the semantic meanings of meshes to generate node embeddings. Following M-Blocks, a Point2Face layer is designed to convert node embeddings to face embeddings. Finally, based on the face embeddings, a multilayer perceptron is used to predict the face labels.

(a) **M-Block**. It is a challenging problem to capture rich semantic meanings of meshes. As we discussed in Sec. 1, the convolutions between different resolutions need to be considered because they can help capture richer high-resolution semantic meanings with the help of low-resolution semantic meanings and contribute to better segmentation results. In addition, the shape noise in simplified mesh needs to be taken into account. Regardless of noise, directly applying convolutions to noise features is not a good choice. Thus, we divide the M-Block into two modules: shape attention for handling noise and topology attention for convolution between different resolutions.

Shape attention. Mapping features to multiple subspaces and extracting important features in these subspaces, multi-head attention can effectively enhance important information for segmentation and inhibit noise [VSP*17, ZGZ*20]. Thus, we propose the shape attention with a feature enhancement operation and a multi-head self-attention layer. The feature enhancement operation prepares rich shape information for feature extraction, and the multi-head self-attention layer is responsible for extracting features and inhibiting noise. Let $F'_l = (f'_1, ..., f'_j, ..., f'_{\mathcal{N}})^T$ be the input of the l-th M-Block and set F'_0 as the shape information input F'. The feature enhancement operation $\mathsf{Enhance}$ on j-th node' feature $f'_j \in \mathbb{R}^{\mathcal{D}}$ can be described as:

$$\mathsf{Enhance}(f'_j) = \rho(f'_j \oplus \sum_{i \in \mathsf{adj}(j)} (f'_j - f'_i)), \tag{4}$$

where $\mathsf{adj}(j)$ represents the adjacent nodes' indexes of j-th node, $\oplus$ represents the concatenation operation of features, ρ means the trainable linear projections from $\mathbb{R}^{2\mathcal{D}}$ to $\mathbb{R}^{\mathcal{D}}$. Through this feature enhancement, richer shape features, especially the shape changes, are involved in feature extraction. Then, employing a multi-head self-attention layer $\mathsf{M\text{-}Att}$ for feature extraction [VSP*17], the shape attention SA can be described as:

$$\mathsf{SA}(f'_j) = f'_j + \mathsf{M\text{-}Att}(\mathsf{Enhance}(f'_j)). \tag{5}$$

Applying SA to all node features, our model emphasizes important shape features and inhibits the noises.

Topology attention. Taking H and $\mathsf{SA}(F'_l)$ as inputs, the topology attention is designed to obtain the rich semantic meanings of shapes, i.e., achieve the convolutions within same resolutions and the convolutions between different resolutions. In fact, we have designed cross-resolutions edges and assigned them different types in the construction of heterogeneous directed graphs. Therefore, topology attention only needs to apply different convolutions for different edge types, and then both the convolutions within same resolutions and the convolutions between different resolutions can be achieved. Let the shape attention result $\mathsf{SA}(f'_j)$ be f''_j, the heterogeneous directed edge from i-th node to j-th node be e_{ij}, the edge type of e_{ij} be R_{ij}, the node types of i-th node and j-th node be T_i and T_j, respectively. The topology attention TA on e_{ij} can be defined as:

$$\mathsf{TA}(\mathsf{e}_{ij}) = \mathsf{V}^{T_i}(f''_i) \cdot \mathsf{Softmax}(\theta^{R_{ij}}(\mathsf{Q}^{T_i}(f''_i) \cdot \mathsf{K}^{T_j}(f''_j)) \cdot (1 - \mathcal{W}_{ij})), \tag{6}$$

where V^{T_i}, Q^{T_i}, and K^{T_j} are trainable linear projections varying with node types from $\mathbb{R}^{\mathcal{D}}$ to $\mathbb{R}^{\mathcal{D}}$, $\theta^{R_{ij}}$ is the trainable linear projections varying with edges types from $\mathbb{R}^{\mathcal{D}}$ to $\mathbb{R}^{\mathcal{D}}$, and $\mathcal{W}_{ij}$ represents the edge weight of e_{ij}. Similar to heterogeneous mutual attention [HDWS20], our topology attention varies with edge types, so the convolutions between different resolutions can be implemented without interference to the convolutions within same resolutions. Even though topology attention has a similar implementation as the heterogeneous mutual attention [HDWS20], our method does not ignore the information of edge weights, as shown in Eq. 6. With TA, we can get the formula of M-Block as:

$$\mathsf{M\text{-}Block}(f'_j) = f'_j + \mathsf{Gelu}(\sum_{i \in \mathsf{adj}(j)} \mathsf{TA}(\mathsf{e}_{ij})) \tag{7}$$

where Gelu represents the activation function. Feeding our inputs to $\mathsf{M\text{-}Block}s$, our method captures semantic meanings as the node embeddings.

(b) **Point2Face Layer**. Mesh segmentation is based on faces, but $\mathsf{M\text{-}Block}s$ generate embedding for each node, i.e., each vertex of meshes. To tackle this problem, we propose the Point2Face layer to convert node embeddings to face embeddings. First, we concatenate the embeddings of three connected nodes from different resolutions as the new node embedding $f^{new} \in \mathbb{R}^{3\mathcal{D}}$. Second, given three node embeddings $(f_1^{new}, f_2^{new}, f_3^{new})^T$ in any face, we employ a self-attention layer to calculate the face embedding $f_{face} \in \mathbb{R}^{3\mathcal{D}}$:

$$f_{face} = \sum_{i=1}^{3} \mathsf{V}(f_i^{new}) \cdot \mathsf{Softmax}(\mathsf{Q}(f_i^{new}) \cdot \mathsf{K}(f_i^{new})) \tag{8}$$

where V, Q, and K are the trainable linear projections from $\mathbb{R}^{3\mathcal{D}}$ to $\mathbb{R}^{3\mathcal{D}}$. Applying the above operations on each face, we obtain all face embeddings.

(c) Finally, we use a **multilayer perceptron** to predict the labels of faces based on face embeddings. Because our segmentation result is based on the simplified meshes, we map our segmentation

Table 1: *Statistical information of datasets and results of XGBoost.*

Dataset	Face number			Mesh number	Ratio of boundaries	XGBoost	
	Average	Maximum	Minimum			Recall	Accuracy
Telealiens	133.6k	471.5k	97.5k	195	0.027	0.961	0.912
Vases	213.6k	503.2k	24.1k	297	0.046	0.985	0.861
Chairs	88.2k	386.1k	41.9k	393	0.032	0.957	0.909
Tooth (Upper/Lower)	227.01k / 213.88k	514.64k / 374.29k	67.40k / 68.18k	100	0.020 / 0.020	0.991 / 0.989	0.857 / 0.849

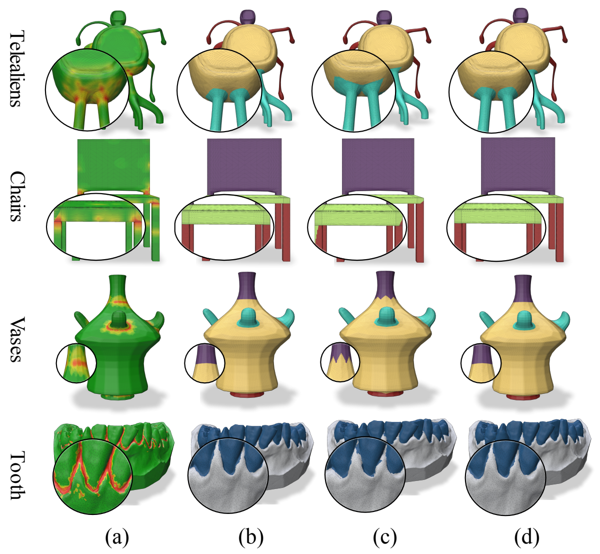


Figure 4: *(a) The probabilistic heatmaps of segmentation boundaries provided by XGBoost. All probability values are mapped to a linear colour space from green (0.0) to red (1.0). (b) The mapped ground truth of meshes after boundary-perserving simplification. (c) The mapped ground truth of meshes after general simplification. (d) The ground truth.*

results to raw meshes using the nearest neighbour query. The label of each face i in raw meshes is the same as the nearest face of simplified meshes. We use the area of the mispredicted faces as the training loss.

4. Experiments

We devise three experiments to evaluate the superiority of MeshFormer. Firstly, by comparing our boundary-preserving simplification with general simplification, we can demonstrate that the boundary-preserving simplification can avoid the ragged segmentation boundaries, i.e., the prior errors for learning and mapping. Secondly, by comparing our clustering algorithm with popular clustering algorithms on mesh segmentation, we can prove that our clustering algorithm works better in simplified meshes, i.e., it provides the clustering results closer to ground truth. Thirdly, by comparing MeshFormer with the state-of-the-art methods, we can present the advantages of MeshFormer. For each experiment, we use three high-resolution coseg datasets generated by subdivision and a real high-resolution tooth dataset. The specific parameters of the dataset can be found in Tab. 1. For each trainable component, i.e., boundary-preserving simplification and semantic segmentation, we perform ten-fold cross-validation (90% for training, 10% for testing) and report the average performances. All experiments are repeated ten times on one machine, including two RTX 3090 (GPUs) and AMD 5950 (CPU).

4.1. Boundary-preserving simplification

According to the statistical results of face numbers in Tab. 1, each dataset involves some meshes containing over 370k faces, implying that applying previous deep learning-based methods directly will consume unacceptable computing resources. As a result, simplification is a necessary choice. However, if we use general simplification without protecting segmentation boundaries, some prior and mapping errors at segmentation boundaries would be raised. As shown in Fig. 4, we simplify meshes and map ground truth on simplified meshes back to the raw meshes. The general simplification causes massive ragged label errors in segmentation boundaries, i.e., the imprecise labels for learning and inevitable errors in mapping. Therefore, to solve the imprecise labels, our algorithm uses XGBoost to predict the boundary-probability of each vertex independently and protects vertices with high probabilities being segmentation boundaries in simplification. We increase the samples of segmentation boundaries to 50% to address the issue of unbalanced labels between segmentation boundaries and non-boundary parts on meshes. With an initial learning rate of 0.05 and a maximum depth of 6, we train the XGBoost for 100 epochs.

As shown in Tab. 1, our algorithm maintains over 95% recall and over 80% accuracy on multiple datasets, which can be attributed to the scale-comprehensive features. The excellent recalls and good-enough accuracies mean that most segmentation boundaries are identified and protected in simplification. The results in Fig. 4 verify our analysis: compared with general simplification, the simplified results of our algorithm effectively eliminate the inaccurate labels of boundaries, i.e., the prior errors for neural networks and the mapping mistakes.

4.2. Hierarchical structure construction

We compare our clustering algorithm with two popular clustering algorithms in mesh segmentation: hierarchical clustering [HJZS20] and spectral clustering [TPT17, LZ04]. To illustrate the superiority of our algorithm, we use two metrics to measure the difference

Table 2: *Accuracy of different methods.* ***g*** *represents the accuracies in general simplified meshes,* ***s*** *represents the accuracies in boundary-preserving simplified meshes, and* ***point cloud*** *represents the accuracies in point cloud data.*

Datasets	MeshCNN (g / s)	PD-MeshNet (g / s)	SubdivNet (g / s)	CurvaNet (g / s)	MeshWalker (g / s)	DiffusionNet (g / s)	DiffusionNet (point cloud)	Ours
Telealiens	96.5 / 94.7	96.9 / 96.0	97.2 / 98.1	94.7 / 92.3	96.7 / 91.3	95.8 / 92.7	91.6	**99.1**
Vases	95.0 / 93.1	95.6 / 94.7	96.2 / 97.1	93.2 / 91.5	95.3 / 89.0	93.4 / 90.4	89.1	**99.1**
Chairs	95.7 / 93.3	96.0 / 94.9	96.1 / 97.0	96.9 / 93.2	95.8/ 89.4	90.9 / 89.3	88.8	**98.9**
Tooth (Upper)	−/−	−/−	−/−	93.3 / 91.4	92.2 / 89.5	94.8 / 91.7	89.2	**99.6**
Tooth (Lower)	−/−	−/−	−/−	92.8 / 90.9	91.6 / 88.2	93.6 / 91.0	88.8	**99.4**

Table 3: *The number of parameters in different methods (millions). Only the parameter settings with the best performance are reported.*

Datasets	MeshCNN	PD-MeshNet	SubdivNet	CurvaNet	MeshWalker	DiffusionNet	Ours
Telealiens	2.277	**1.354**	10.752	3.56	12.638	2.76	1.814
Vases	2.278	**1.354**	10.752	1.46	12.638	2.76	1.814
Chairs	2.278	**1.354**	10.752	1.459	12.638	2.76	1.814
Tooth	−	−	−	14.674	40.98	2.76	**2.665**

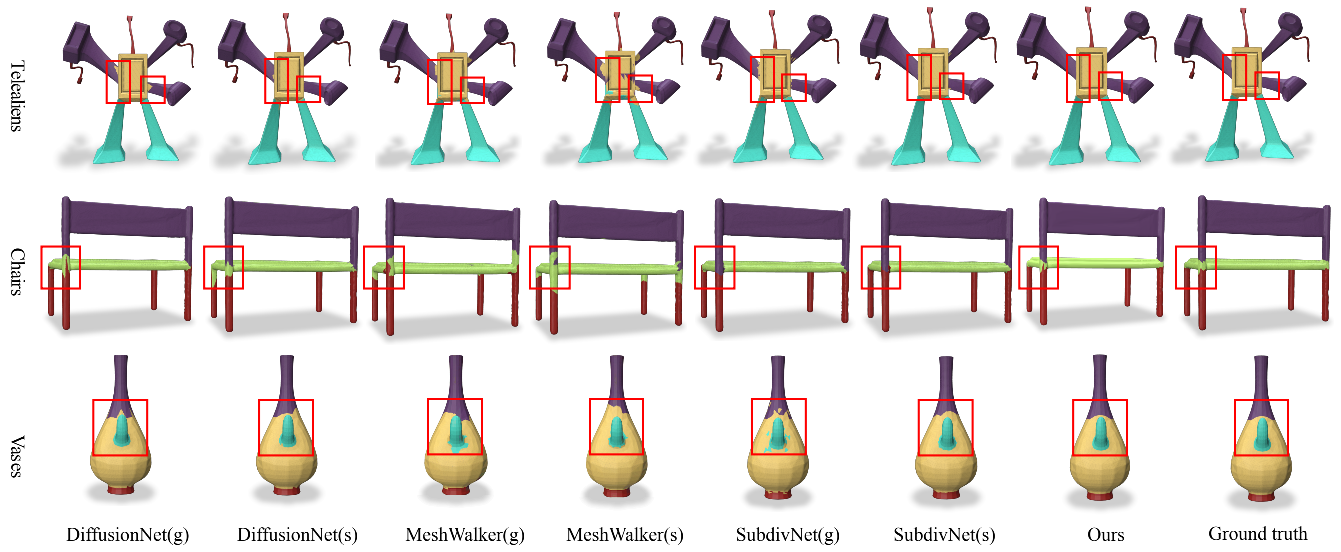


Figure 5: *The segmentation results of state-of-the-art methods in coseg datasets.* ***g*** *represents the result in general simplified meshes, and* ***s*** *represents the result in boundary-preserving simplified meshes.*

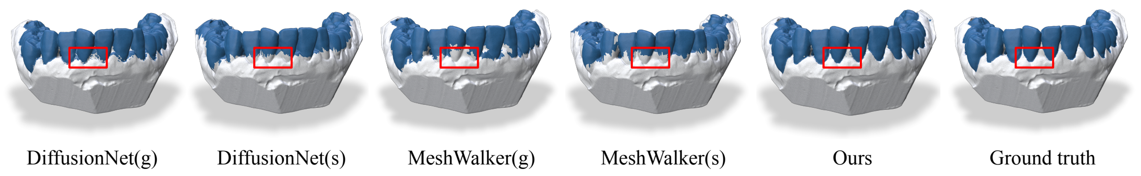


Figure 6: *The segmentation results of different methods in tooth datasets.* ***g*** *represents the result in general simplified meshes, and* ***s*** *represents the result in boundary-preserving simplified meshes.*

Table 4: *Errors of hierarchical structures.* ***Spec*** *represents the spectral clustering,* ***Hier*** *represents hierarchical clustering, and* ***Faces*** *represents the face number of simplified meshes.*

Dataset	$error_1$	$error_2$ / Faces			Faces
		Ours	Spec	Hier	
Telealiens	73	**0.086**	0.141	0.174	1.5k
Vases	50	**0.083**	0.118	0.139	1k
Chairs	42	**0.081**	0.121	0.123	1k
Tooth(Upper)	1020	**0.091**	0.115	0.113	20k
Tooth(Lower)	1017	**0.097**	0.117	0.114	20k

between the clustering results and ground truth. Firstly, to avoid vertices with the same label being divided into different clusters, the number of clusters should be close to the number of labels in ground truth. Thus, we set the difference between the number of clusters NC and the number of labels Y as the first metric $error_1 = |NC - Y|$. Secondly, to prevent vertices with different labels from being in the same cluster, we define the second metric as $error_2 = \sum_{n=1}^{NC}(\sum_{i=1}^{u_n} l_{ni} - max(l_{n1}, ..., l_{nu_n}))$, where u_n is the number of labels in the n-th cluster, and l_{ni} is the face number of i-th label in the n-th cluster. Theoretically, if $error_1$ and $error_2$ are 0, the clustering results are ground truth. Since other clustering algorithms need to specify the cluster number for each mesh, we adopt the clustering number generated by our adaptive algorithm as the clustering number of other algorithms on each mesh.

As shown in Tab. 4, the hierarchical structures generated by our method have lower $error_2$ than other methods, i.e., clustering results closer to ground truth. The better performance of our algorithm can be attributed to two aspects. First, the changes of directional curvature are consistent with ground truth: dramatic changes in boundaries and keeping similar in segmentation targets. Second, Ricci flow naturally resists noise, which means stable performance under noise.

4.3. Semantic segmentation

We compare our method with six state-of-the-art methods, including CNN-based methods (MeshCNN [HHF*19], PD-MeshNet [MLR*20], and SubdivNet [HLG*22]), graph laplacian-based methods (Diffusion-Net [SACO22]), and spatial network-based methods (CurvaNet [HJZS20] and MeshWalker [LT20]). For a fair comparison, we look for the optimal hyperparameters for each method within the suggestive ranges and report their best performances on each dataset. The details can be described as follows.

- **MeshCNN.** We search the collapsed edges' proportions and the layer numbers at $\{0.2, 0.3, 0.4\}$ and $\{4, 5, 6, 7\}$, respectively.
- **PD-MeshNet.** Since PD-MeshNet has a similar architecture to MeshCNN, we adopt the same search ranges of hyperparameters as MeshCNN. According to the suggestions in [MLR*20], we select the *add* pooling version for PD-MeshNet.
- **SubdivNet.** We set the base mesh size and subdivision depth as 256 and 3, respectively, and each mesh is remeshed to 10 different meshes for data augmentation as suggested in [HLG*22]. For a fair comparison, each mesh is viewed as an independent mesh, so the original segmentation accuracy is reported rather than the voted accuracy. Additionally, we test both ResNet and DeepVlab structures of SubdivNet and report better accuracy.
- **CurvaNet.** We set the ratio of preserved nodes in pooling layers as 0.25 as suggested in [HJZS20] and search the layer number $\mathcal{L} \in \{3, 4, 5, 6, 7, 8\}$.
- **MeshWalker.** We search the walk length and the number of GRU layers at $300, 600, 900$ and $3, 6, 9$, respectively.
- **DiffusionNet.** We test two versions of DiffusionNet, including *origin* and *hks* versions. Since DiffusionNet is also a point cloud segmentation method, both mesh predictions and point cloud predictions are involved in the comparison. In addition, the number of eigenfunctions is increased to 256 due to more high-frequency information raised by simplification.
- **Our method.** For three coseg datasets, we set the layer number $\mathcal{L}$, the feature dimension D, and the head number for multi-head attention layers as 4, 120, and 8, respectively. We use the same hyperparameters as the coseg datasets for the tooth dataset, but increase the layer number to 6. We train MeshFormer using the Adam optimizer [KB15], setting the initial learning rate as 0.001, decaying the learning rate with a factor 0.5, and running each epoch with batch size 1.

For all methods, we ensure that 600 epochs are conducted in training processes. Additionally, we simplify all meshes to acceptable data sizes because none of the above methods can be directly applied to high-resolution meshes within limited computing resources. For three coseg datasets, we simplify each mesh until the face numbers match the suggestions in MeshCNN [HHF*19]. For the tooth dataset, we simplify tooth meshes to 20k faces. Note that many real-world tooth meshes are not closed manifolds, and converting these meshes with complex topological errors into closed manifolds is still a challenging problem, so the dual graph-based methods cannot be applied to the tooth datasets. In this paper, we define accuracy as the area ratio of faces with correct predictions. To implement a more comprehensive comparison, we test other methods on both general simplified meshes and boundary-preserving simplified meshes. The accuracies of all methods are listed in Tab. 2.

Compared with point cloud-based DiffusionNet, mesh-based methods generally perform better due to complete topological information. Compared with other mesh-based methods, our method achieves gains from 1.0% to 5.8%. The reason can be attributed to three points.

- Firstly, the prior and the mapping errors are reduced by boundary-preserving simplification. In contrast, previous methods with general simplification have to suffer prior and mapping errors. For example, after general simplification, the segmentation labels of meshes become not accurate (i.e., ragged segmentation boundaries) due to inappropriate simplification occurring in boundaries, as shown in Fig. 4(c). These imprecise segmentation labels bring prior errors for learning and mapping errors (Fig 1(b)). Therefore, it is difficult for previous methods with general simplification to obtain the same performance as MeshFormer.
- Secondly, since our method based on Ricci flow and attention mechanisms has good anti-noise ability naturally, our method can handle the noise raised by boundary-preserving simpli-

Table 5: *The time consumption for differents methods (seconds). **g** represents general simplification and **s** represents boundary-preserving simplification.*

Methods	Time (seconds)			
	Simplification	Extra computation	Segmentation	All
MeshCNN	2.54(g)	0	1.31	3.85
CurvaNet	2.54(g)	0.18 (Features)	0.07	2.79
PD-MeshNet	2.54(g)	0	1.47	4.01
SubdivNet	4.99(s)	9.86(Remesh)	0.05	14.9
DiffusionNet	2.54(g)	0	0.02	2.56
MeshWalker	2.54(g)	0.21 (Paths)	0.17	2.92
Ours	4.99(s)	1.74 (Ricci flow)	0.11	6.84

fication. Most previous methods can also use our boundary-preserving simplification in a plug-and-play manner, but they cannot achieve better results in the same way due to the shape noise. The better performance of SubdivNet on boundary-preserving simplified meshes validates our analysis. Although SubdivNet has similar architectures to other CNNs [HHF*19, MLR*20], it employs the extra remeshing operation to alleviate noises, resulting in improved accuracy on boundary-preserving simplified meshes.

- Thirdly, our transformer implements the convolutions between different resolutions that capture richer high-resolution semantic meanings and contribute to better segmentation results. Thus, our method outperforms SubdivNet, especially in segmentation boundaries, as shown in Fig. 5 and Fig. 6. The ablation study in Sec. 4.4 provides precise quantitative gains of convolutions between different resolutions.

As shown in Tab. 3, our method also has fewer parameters than most previous methods. In addition, we also record and analyze the computation time. We select 20 closed tooth meshes with 200k vertices and test the computation time of all methods. As shown in Tab. 5, the computation time is divided into two parts: time for pre-computation (i.e., simplification and extra computation) and time for segmentation. Unlike the segmentation task, the pre-computation does not involve many parameters and an exclusive GPU computation mode, so we employ 10 parallel processes for pre-computation and one single process for segmentation. For pre-computation, boundary-preserving simplification and Ricci flow bring more gains and take more time, but the increase of computation time is acceptable on non-real-time segmentation tasks, i.e., approximately increase two seconds from the average performance of other methods (4.66s). For the segmentation, our method is well above the average performance (0.11s vs 0.52s). In a word, our method achieves the best accuracy, and the increase of computation time is in an acceptable range for non-real-time segmentation.

4.4. Ablation study

The gains of MeshFormer can be attributed to four main factors: the boundary-preserving simplification for fewer prior errors, the clustering algorithm for the clustering results closer to ground truth, the cross-resolution convolutions for the richer high-resolution semantic meanings, and extra shape attention for noise handling. We discuss the gains brought by these factors independently.

Table 6: *Accuracies of different models in Ablation study.*

Dataset	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Ours
Telealiens	97.0	98.7	97.9	97.7	98.7	98.3	99.1
Vases	96.4	97.6	97.1	96.3	98.2	98.1	99.1
Chairs	97.1	98.6	98.4	97.8	98.3	98.2	98.9
Tooth (Upper)	96.2	96.3	96.7	95.2	97.0	97.1	99.6
Tooth (Lower)	95.9	95.4	95.8	94.5	96.1	96.3	99.4

The gains from boundary-preserving simplification. We build Model 1 by replacing the boundary-preserving simplification in MeshFormer with QEM and discuss the ability of MeshFormer to defeat the prior errors. The accuracies of Model 1 in Tab. 6 show that MeshFormer suffers performance loss due to the prior errors but still has better performance (0.8% gains on average) than previous methods, which can be attributed to the strong ability of Ricci flow and mesh transformer.

The gains from Ricci flow-based clustering algorithm. We build new models by replacing our clustering algorithm with the spectral clustering algorithm (Model 2) and the hierarchical clustering algorithm (Model 3), respectively. As shown in Tab. 6, similar to a better pre-segmentation, our clustering results closer to ground truth bring better segmentation results (i.e., more than 0.4% absolute improvements) than Model 2 and Model 3. Similarly, Model 2 outperforms Model 3 since its clustering results are closer to the ground truth than Model 3.

The gains from the cross-resolution convolutions. Model 4 and Model 5 are created by removing all the edges between different resolutions and the directed edges from low resolutions to high resolutions in our model, respectively. All information exchanges between different resolutions are blocked in Model 4 due to the lack of cross-resolution edges, and only part information exchanges are preserved in Model 5 (similar to U-Nets). MeshFormer, which includes complete information exchanges, outperforms these two models by 2.24% on average, as shown in Tab. 6. The main reason is that complete information exchanges between resolutions can broaden the range of convolutions and provide richer high-resolution semantic meanings for segmentation, resulting in better results [WSC*20, WYC*21].

The gains from shape attention. Model 6 is created by removing the shape attention from M-Blocks. Without shape attention from M-blocks, our model loses more than 0.7% accuracy, as shown in Tab. 6. In M-Blocks, shape attention focuses on extracting important shape features and inhibiting the noise, and topology attention is responsible for obtaining semantic meanings of meshes by convolutions. Without shape attention to alleviate noise, Model 6 cannot achieve the same performance as our model.

4.5. Hyperparameter Analysis

We discuss the impact of important hyperparameters in three components of MeshFormer on real-world tooth datasets and report the averaged performance of the upper and lower tooth meshes. In the boundary-preserving simplification, we adjust the sampling ratio of segmentation boundaries as $\{30\%, 40\%, 50\%, 60\%\}$, respectively. The recall of XGBoost increases with the ratio of boundary

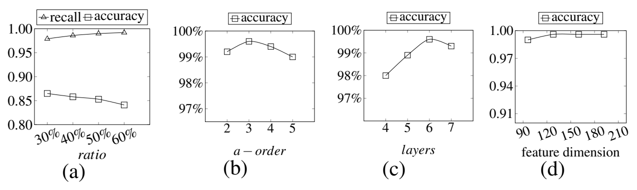


Figure 7: *(a) The performance under different ratio of boundary nodes. (b) The accuracies under different size of neighbors. (c) The accuracies under different $\mathcal{L}$. (d) The accuracies under different D.*

nodes, but the accuracy decreases, as shown in Fig. 7(a); so, the balance between recall and accuracy can be adjusted based on the demand. In the clustering algorithm, we sum the weights in the a-order neighbours for all vertices and set the average value as En. To evaluate the influence of a, we set a as $\{2,3,4,5\}$, respectively, and record the accuracies as shown in Fig. 7(b). The results in Fig. 7(b) show that our method has stable performance with different a. In mesh transformer, we discuss the influence of layer number $\mathcal{L}$ and the feature dimension D. Fig. 7(c) shows the gradually increasing trend of our method's accuracy with a larger layer number $\mathcal{L}$. Similarly, according to the accuracies under different D in Fig. 7(d), our algorithm also performs better with a larger feature dimension and eventually converges.

4.6. Limitations

Although our method achieves the best performance in high-resolution mesh segmentation, it still suffers two limitations. To begin with, our method obtains semantic meanings of meshes with convolutions in connected graphs, similarly to previous methods. Consequently, our method is ineffective for unconnected graphs, i.e., meshes containing multiple isolated parts. Introducing some visual features might help us solve this limitation, but it would over-complicate our method. Second, according to Tab. 5, our method cannot support real-time segmentation, which is also a common drawback of current deep learning-based methods. Using a computer with multiple GPUs to run the algorithm may solve the problem, but it will consume more computing resources. Fortunately, most high-resolution mesh segmentation tasks, such as tooth segmentation, do not require real-time processing.

5. Conclusion and future work

Limited computing resources and insufficient semantic meanings are the directed obstacles to achieving good performance with the graph transformer architecture. We propose MeshFormer containing three components to remove these two obstacles. Experiments prove the effectiveness of MeshFormer and each component of MeshFormer. MeshFormer shows a good generalized performance in the real-world mesh segmentation, i.e., tooth segmentation. Furthermore, our experiments show that previous deep learning-based methods can also be benefited from our method in a plug-and-play manner. For example, SubdivNet achieves better accuracy with boundary-preserving simplification. In the future, we tend to improve MeshFormer by accelerating feature computation and simplification for supporting real-time segmentation tasks.

6. Acknowledge

This research was supported by National Major Scientific Research Instrument Development Project under Grant 81827804, the National Natural Science Foundation of China under Grant 61972343, and the Key Research and Development of Zhejiang Province under Grant 2021C03032.

References

[ACK13] Attene M., Campen M., Kobbelt L.: Polygon mesh repairing: An application perspective. *ACM Computing Surveys 45* (2013), 1–33. 1

[BK10] Bronstein M. M., Kokkinos I.: Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Proceedings of CVPR* (2010), pp. 1704–1711. 2, 4

[BKR*16] Baldacci A., Kamenický R., Riečický A., et al.: Gpu-based approaches for shape diameter function computation and its applications focused on skeleton extraction. *Computers & Graphics 59* (2016), 151–159. 2, 3

[BLL*20] Bai S., Lin Y., Lu L., Wang Z., Yau S.: Ollivier ricci-flow on weighted graphs. *CoRR* (2020). 3

[ÇAL*16] Çiçek Ö., Abdulkadir A., Lienkamp S. S., Brox T., Ronneberger O.: 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention* (2016), Springer, pp. 424–432. 2

[CHB*15] Chen T., He T., Benesty M., Khotilovich V., Tang Y., Cho H., Chen K., et al.: Xgboost: extreme gradient boosting. *R package version 0.4-2 1* (2015), 1–4. 4

[CLN06] Chow B., Lu P., Ni L.: *Hamilton's Ricci flow*, vol. 77. American Mathematical Soc., 2006. 1

[FBD*21] Fayyaz M., Bahrami E., Diba A., Noroozi M., Adeli E., Van Gool L., Gall J.: 3d cnns with adaptive temporal feature resolutions. In *Proceedings of CVPR* (2021), pp. 4731–4740. 2

[FLL22] Fan C., Liu T., Liu K.: Sunet: Swin transformer unet for image denoising. *CoRR* (2022). 3

[GH97] Garland M., Heckbert P. S.: Surface simplification using quadric error metrics. *ACM Transactions on Graphics* (1997), 209–216. 4

[GJ19] Gao H., Ji S.: Graph u-nets. In *Proceedings of ICML* (2019), pp. 2083–2092. 2

[HDWS20] Hu Z., Dong Y., Wang K., Sun Y.: Heterogeneous graph transformer. In *Proceedings of WWW* (2020), pp. 2704–2710. 6

[HHF*19] Hanocka R., Hertz A., Fish N., Giryes R., Fleishman S., Cohen-Or D.: Meshcnn: a network with an edge. *ACM Transactions on Graphics 38* (2019), 1–12. 1, 3, 4, 9, 10

[HJZS20] He W., Jiang Z., Zhang C., Sainju A. M.: Curvanet: Geometric deep learning based on directional curvature for 3d shape analysis. In *Proceedings of KDD* (2020), pp. 2214–2224. 1, 2, 3, 4, 5, 7, 9

[HLG*22] Hu S., Liu Z., Guo M., Cai J., Huang J., Mu T., Martin R.: Subdivision-based mesh convolution networks. *ACM Transactions on Graphics 41* (2022), 1–16. 1, 2, 3, 4, 9

[HSBH*19] Haim N., Segol N., Ben-Hamu H., Maron H., Lipman Y.: Surface networks via general covers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 632–641. 3

[JKLG08] Jin M., Kim J., Luo F., Gu X.: Discrete surface ricci flow. *IEEE Transactions on Visualization and Computer Graphics 14* (2008), 1030–1043. 5

[Kap03] Kapovich M.: *Geometrization conjecture and Ricci flow*. Tech. rep., 2003. 3

[KB15] Kingma D., Ba J.: Adam: A method for stochastic optimization. In *Proceedings of ICLR* (2015). 9

[LBD17] Le T., Bui G., Duan Y.: A multi-view recurrent neural network for 3d mesh segmentation. *Computers & Graphics 66* (2017), 103–112. 2

[LHMR08] Lai Y., Hu S., Martin R., Rosin P.: Fast mesh segmentation using random walks. In *Proceedings of the ACM symposium on Solid and physical modeling* (2008), pp. 183–191. 1

[Lot06] Lott J.: Some geometric calculations on wasserstein space. *arXiv preprint math/0612562* (2006). 5

[LSM*21] Luthra A., Sulakhe H., Mittal T., Iyer A., Yadav S.: Eformer: Edge enhancement based transformer for medical image denoising. *CoRR* (2021). 3

[LT20] Lahav A., Tal A.: Meshwalker: Deep mesh understanding by random walks. *ACM Transactions on Graphics 39* (2020), 1–13. 3, 9

[LZ04] Liu R., Zhang H.: Segmentation of 3d meshes through spectral clustering. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference* (2004), pp. 298–305. 3, 4, 7

[MACO91] Mohar B., Alavi Y., Chartrand G., Oellermann O.: The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications 2* (1991). 3

[MCB*22] Min E., Chen R., Bian Y., Xu T., Zhao K., Huang W., Zhao P., Huang J., Ananiadou S., Rong Y.: Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455* (2022). 6

[MLR*20] Milano F., Loquercio A., Rosinol A., Scaramuzza D., Carlone L.: Primal-dual mesh convolutional neural networks. *Advances in Neural Information Processing Systems 33* (2020), 952–963. 9, 10

[MPS14] Marinacci F., Pakmor R., Springel V.: The formation of disc galaxies in high-resolution moving-mesh cosmological simulations. *Monthly Notices of the Royal Astronomical Society 437* (2014), 1750–1775. 1

[NLGG18] Ni C. C., Lin Y. Y., Gao J., Gu X.: Network alignment by discrete ollivier-ricci flow. In *Graph Drawing and Network Visualization* (2018), pp. 447–462. 4

[NLLG19] Ni C., Lin Y., Luo F., Gao J.: Community detection on networks with ricci flow. *Scientific reports 9* (2019), 1–12. 1, 2, 4, 5

[OKV15] Odaker T., Kranzlmueller D., Volkert J.: Gpu-accelerated real-time mesh simplification using parallel half edge collapses. In *International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science* (2015), pp. 107–118. 4

[Oll09] Ollivier Y.: Ricci curvature of markov chains on metric spaces. *Journal of Functional Analysis 256* (2009), 810–864. 3

[QYSG17] Qi C., Yi L., Su H., Guibas L.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems 30* (2017). 2

[RBB09] Rusu R. B., Blodow N., Beetz M.: Fast point feature histograms (fpfh) for 3d registration. In *Proceedings of ICRA* (2009), pp. 3212–3217. 3

[SACO22] Sharp N., Attaiki S., Crane K., Ovsjanikov M.: Diffusionnet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics 41* (2022), 1–16. 2, 3, 9

[SBR16] Sinha A., Bai J., Ramani K.: Deep learning 3d shape surfaces using geometry images. In *Proceedings of ECCV* (2016), pp. 223–240. 2

[SBZB15] Shi B., Bai S., Zhou Z., Bai X.: Deeppano: Deep panoramic representation for 3-d shape recognition. *IEEE Signal Processing Letters 22* (2015), 2339–2343. 2

[Sha08] Shamir A.: A survey on mesh segmentation techniques. *Computer Graphics Forum 27* (2008), 1539–1556. 3

[SHKVL09] Sharma A., Horaud R. P., Knossow D., Von Lavante E.: Mesh segmentation using laplacian eigenvectors and gaussian mixtures. In *Proceedings of AAAI* (2009). 3

[SS21] Smirnov D., Solomon J.: Hodgenet: learning spectral geometry on triangle meshes. *ACM Transactions on Graphics 40* (2021), 1–11. 3

[SYW*22] Shu Z., Yang S., Wu H., Xin S., Pang C., Kavan L., Liu L.: 3d shape segmentation using soft density peak clustering and semi-supervised learning. *Computer-Aided Design 145* (2022), 103181. 1, 4

[Tao10] Tao T.: Iii. 78 ricci flow. In *Princeton Companion to Mathematics*. 2010, pp. 279–281. 4

[TGI18] Treboux J., Genoud D., Ingold R.: Decision tree ensemble vs. nn deep learning: efficiency comparison for a small image dataset. In *International Workshop on Big Data and Information Security* (2018), pp. 25–30. 4

[TPKZ18] Tatarchenko M., Park J., Koltun V., Zhou Q.: Tangent convolutions for dense prediction in 3d. In *CVPR* (2018), pp. 3887–3896. 3

[TPT17] Theologou P., Pratikakis I., Theoharis T.: Unsupervised spectral mesh segmentation driven by heterogeneous graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence 39* (2017), 397–410. 1, 7

[TQD*19] Thomas H., Qi C., Deschaud J., Marcotegui B., Goulette F., Guibas L.: Kpconv: Flexible and deformable convolution for point clouds. In *ICCV* (2019), pp. 6411–6420. 2

[TST96] Thompson P., Schwartz C., Toga A.: High-resolution random mesh algorithms for creating a probabilistic 3d surface atlas of the human brain. *Neuroimage 3* (1996), 19–34. 1

[VCC*18] Veličković P., Cucurull G., Casanova A., Romero A., Liò P., Bengio Y.: Graph attention networks. In *Proceedings of ICLR* (2018). 3, 6

[VSP*17] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., , et al.: Attention is all you need. *Advances in neural information processing systems 30* (2017). 3, 6

[WSC*20] Wang J., Sun K., Cheng T., Jiang B., et al.: Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence 43* (2020), 3349–3364. 2, 3, 10

[WYC*21] Wang W., Yao L., Chen L., Cai D., He X., Liu W.: Crossformer: A versatile vision transformer based on cross-scale attention. *CoRR* (2021). 3, 10

[WZ22] Wang H., Zhang J.: A survey of deep learning-based mesh processing. *Communications in Mathematics and Statistics 10*, 1 (2022), 163–194. 3

[XLZ*20] Xiao Y., Lai Y., Zhang F., Li C., Gao L.: A survey on deep geometry learning: From a representation perspective. *Computational Visual Media 6*, 2 (2020), 113–133. 3

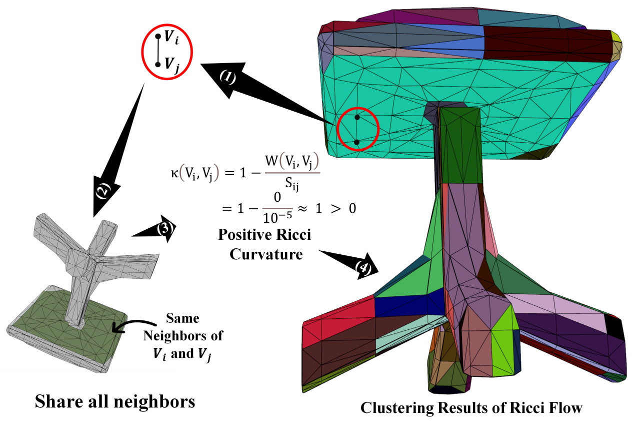
[YJK*19] Yun S., Jeong M., Kim R., Kang J., Kim H.: Graph transformer networks. *Advances in neural information processing systems 32* (2019). 1

[ZGZ*20] Zhang Y., Gong Y., Zhu H., Bai X., Tang W.: Multi-head enhanced self-attention network for novelty detection. *Pattern Recognition 107* (2020), 107486. 6

[ZGZ*21] Zhou H., Guo J., Zhang Y., Yu L., Wang L., Yu Y.: nnformer: Interleaved transformer for volumetric segmentation. *CoRR* (2021). 2

[ZSG10] Zeng W., Samaras D., Gu X.: Ricci flow for 3d shape analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32* (2010), 662–677. 2, 3, 4

[ZZH*22] Zhou Y., Zheng H., Huang X., Hao S., Li D., Zhao J.: Graph neural networks: Taxonomy, advances, and trends. *ACM Transactions on Intelligent Systems and Technology 13* (2022), 1–54. 1


V_i
V_j
(1)
(2)
(3)
(4)
$\kappa(V_i, V_j) = 1 - \frac{W(V_i, V_j)}{S_{ij}} = 1 - \frac{0}{10^{-5}} \approx 1 > 0$
Positive Ricci Curvature
Same Neighbors of V_i and V_j
Share all neighbors
Clustering Results of Ricci Flow

(a)

Figure 8: *An example of Ricci flow.*

Appendix A: An example of Ricci flow

We provide an example to elaborate that our clustering algorithm adaptively divides meshes into patches. As shown in Fig. 8, the details of the clustering algorithm can be explained as follows. Firstly, take two connected vertices in meshes, such as V_i and V_j in Fig. 8. Then, we sampling the neighbors with similar directional curvature of V_i and V_j based on the fast seed expansion algorithm in Sec. 3.2. In this case, the sampling results show that V_i and V_j have the same neighbors, so W is 0, which is less than the edge weight $S_{ij} = 10^{-5}$. Therefore, the Ricci curvature κ is positive, and these two vertices will be clustered into one patch by running Ricci flow. Following the same process, vertices with similar directional curvature will be clustered into one patch, and the whole mesh will be split into patches having similar directional curvature, as shown in Fig. 8. The implementation of MeshFormer can be found at https://github.com/MeshFormer/MeshFormer.