MWFormer: Mesh Understanding with Window-based Transformer

Hao-Yang Peng [a], Meng-Hao Guo [a], Zheng-Ning Liu [b], Yong-Liang Yang [c], Tai-Jiang Mu [a,*]

[a] *Key Laboratory of Pervasive Computing, Ministry of Education, Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China*
[b] *Fitten Tech Co., Ltd., Beijing, 100084, China*
[c] *Department of Computer Science, University of Bath, Bath, BA2 7AY, United Kingdom*

ARTICLE INFO

ABSTRACT

Polygonal mesh has been proven to be a powerful representation of 3D shapes, given its efficiency in expressing shape surface while maintaining geometric and topological information. Increasing efforts have been made to design elaborate deep convolutional neural networks for meshes. However, these methods naturally ignore the global connectivity among mesh primitives due to the locality nature of convolution operations. In this paper, we introduce a transformer-like self-attention mechanism with down-sampling architectures for mesh learning to capture both the global and local relationships among mesh faces. To achieve this, we propose BFS-Pooling, which can convert a connected mesh into discrete tokens (i.e., a set of adjacent faces) with breath-first-search (BFS) and naturally build hierarchical architectures for mesh learning by pooling mesh tokens. Benefiting from BFS-Pooling, we design a hierarchical transformer architecture with a window-based local attention mechanism, Mesh Window Transformer (MWFormer). Experimental results demonstrate that MWFormer achieves the best or competitive performance in both mesh classification and mesh segmentation tasks. Code will be available.

1. Introduction

Mesh is one of the most common representations of 3D shapes with three basic elements, i.e., vertices, edges, and faces. It naturally conveys more topological and geometric information than point cloud and can represent 3D objects' surfaces effectively. Thus, mesh data are widely used in the field of modeling and rendering. Due to its wide range of applications, how to extract representative features from meshes has become an attractive topic in deep geometric learning.

Inspired by the great success of convolutional neural networks (CNNs) in image processing [1–5], many recent works focus on how to adapt CNN-based methods to 3D meshes. MeshCNN [6] and SubdivNet [7] are representative works. MeshCNN [6] treats edges as the input units and aggregates features from two adjacent faces. SubdivNet [7] designs convolution and pooling operations on faces directly, which can adapt popular 2D CNN architectures to 3D mesh such as ResNet [1] and DeepLabv3+ [8]. However, the above methods naturally ignore the global connectivity among mesh primitives due to the locality nature of convolution operations.

Unlike CNNs, the transformer architecture, originating from the natural language processing (NLP) field [9], can extract the global relationships among all the input tokens with the self-attention mechanism and makes great success in the domains of 2D vision [10–12] and point cloud [13,14]. Though transformers can be naturally applied to meshes by treating each mesh face as a token, analogous to a pixel/patch in image or a point in point cloud, the original self-attention has a quadratic computational complexity w.r.t. the number of input tokens. Hence it is urgent to design a new way to generate tokens with reduced quantity while maintaining the original information for practical deep mesh learning.

Many influential transformer models in image processing domains have a down-sampling hierarchical architecture to achieve both effectiveness and efficiency. However, meshes do not own an inherent hierarchical structure like images. So it is challenging to process a mesh with straightforward down-sampling deep networks due to its irregularity. SubdivNet [7] can only process meshes with Loop subdivision sequence connectivity. Other works like MeshCNN [6] adopt edge collapse pooling, which can only aggregate features of adjacent mesh edges. Inspired by ViT [10] which splits image grids into non-overlapping patches, we propose a pooling operation based on breadth-first search (BFS) to generate the input tokens for the transformer-based neural networks. Specifically, we first sample faces as the initial

* Corresponding author.
E-mail addresses: phy22@mails.tsinghua.edu.cn (H.-Y. Peng), gmh20@mails.tsinghua.edu.cn (M.-H. Guo), lzhengning@gmail.com (Z.-N. Liu), y.yang@cs.bath.ac.uk (Y.-L. Yang), taijiang@tsinghua.edu.cn (T.-J. Mu).

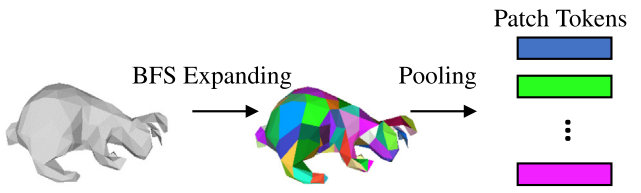

Fig. 1. The proposed pooling operation to convert irregular mesh data into discrete tokens while maintaining all face features and locality. Features of faces in the same color will be integrated into a new patch. The patch tokens can then be processed by various transformer architectures.

patches and iteratively expand them in a BFS way until every face is assigned to a certain patch. Then we aggregate all face features within each patch to produce new features of patch, i.e., token. The process is shown in Fig. 1. To add adjacency to patches, we introduce neighborhood relationships to merge patches from the origin adjacent relationships at the previous level. We regard two patches as neighbors if any associated faces are adjacent.

The BFS pooling operations enable the use of hierarchical transformer structures in the mesh domain. To demonstrate this, we propose the Mesh Window Transformer(MWFormer). Inspired by Swin-Transformer [11], MWFormer employs a window-based transformer architecture capable of processing both global and local information of given meshes based on the attention mechanism. Furthermore, We adopt a re-sample strategy to enhance MWFormer's ability to extract global connections across different patch windows. We evaluate the effectiveness of MWFormer on mesh classification and mesh segmentation tasks, all achieving the best or competitive performance compared to existing methods.

The main contributions of this paper can be summarized as follows:

(a) We designed a general pooling method, BFS-Pooling for mesh learning, which can efficiently convert connected meshes of various sizes into discrete tokens and naturally build hierarchical neural network architectures.
(b) Based on the BFS-Pooling, we present MWFormer, a transformer-based method for mesh processing that effectively learns both the local and global representations for mesh faces.
(c) We evaluated MWFormer's performance on mesh classification and segmentation tasks and achieve competitive results.

2. Related works

2.1. Deep learning on 3D representation

Numerous methods have been proposed to apply deep learning to 3D data. Some surveys [15,16] collected and categorized these methods. Roughly, they can be divided into four branches according to different data types including voxels, point clouds, multi-view images, and meshes. There are also some methods based on implicit functions.

The most direct way is to project 3D object into multiple 2D images and adopt CNNs to process 2D images directly [17,18]. GVCNN [19] adopted a view-group-shape architecture to better represent 3D shape information. Recently, SimpleView [20] proposed a simple projection method with strong data augmentation and achieved great success. However, the projected images are from limited views, thus may lose key information of 3D shapes, which may affect the final performance.

For voxel data, Voxnet [21] and 3D shapenets [22] directly applied 3D CNNs on 3D volumetric data. Voxelnet [23] treated point cloud data as voxel units and achieved prominent results on 3D object detection. However, the high computation and memory overhead of volumetric data limit their capability.

To solve the above problem, some works aim to process point clouds directly. PointNet [24] is the pioneering work that used simple multi-layer perceptron (MLP) and pooling operation to process raw point cloud directly. After that, many following works focused on processing raw point cloud data. PointNet++ [25] introduced the hierarchical architecture and local information by using furthest point sampling(FPS) and neighborhood aggregation. DGCNN [26], PointCNN [27] and KPConv [28] defined various convolution operations on points based on their geometric features. Besides, PCT [13] utilized the strong capability of transformer for point cloud. Sun et al. [29] introduced a multilevel consistent semi-supervised method to leverage unlabeled data in segmentation tasks.

More recently, implicit function methods have emerged in 3D deep learning. Deepsdf [30] and Local Implicit Grid [31] utilized the signed distance function (SDF) to represent and reconstruct 3D scenes. Occupancy networks [32] learned classifiers with the continuous occupancy function to represent mesh surfaces. NeRF [33] used MLPs to reconstruct density and color information from sparse input images. Following works NSVF [34] and FastNeRF [35] greatly reduced the time consumption of training original NeRF with better performance. These implicit methods mainly focus on 3D reconstruction and view synthesis instead of 3D classification and segmentation tasks for now.

2.2. Deep learning on mesh

Mesh structure consists of three parts: vertices, edges, and faces. Various deep learning methods are proposed by considering these three primitives as the primary data, while some other works focus on the spectral domain [36]. Some methods focus on how to properly process features of certain vertices and their neighbors. Masci et al. [37], Boscaini et al. [38] and MoNet [39] parameterized each geodesic patch into 2D domains. Based on the geodesic methods, PFCNN [40] selected tangent planes to extend standard convolutions. TextureNet [41] designed a 4-Rosy parameterization to carry out convolutions on mesh surfaces. DiffusionNet [42] adopted local diffusion operations based on the Laplacian operator to better represent surfaces. HodgeNet [43] also extended the Laplacian operators and designs an end-to-end architecture with sparse operators to extract mesh features.

One mesh edge connects two vertices and is adjacent to two triangular faces and four neighboring edges in a 2-manifold mesh. Based on this, MeshCNN [6] used relative edge features as inputs and designed a permutation-invariant convolution operation on adjacent edges. PD-MeshNet [44] performed attention-based convolutions and pooling operations on a pair of graphs built on the input mesh. Other than convolution operations, MeshWalker [45] adopted Recurrent Neural Network (RNN) to extract geometric features on edges based on multiple random walks.

Face-based methods treat mesh faces as input units and focus on how to capture and aggregate local face features more effectively. MeshNet [46] proposed a convolution operation that extracts both spatial and structural features from faces while aggregating features of adjacent faces by concatenation. SubdivNet [7] introduced subdivision sequences and general convolution operations on mesh faces. DNF-Net [47] extracted face patches with multi-scale embedding units to denoise meshes. The above methods mainly focus on how to efficiently extract local features and ignore the global relationship among mesh primitives. Moreover, some methods such as SubdivNet have strict requirements, i.e., the input mesh must be watertight and have Loop subdivision sequence connectivity, which restricts its

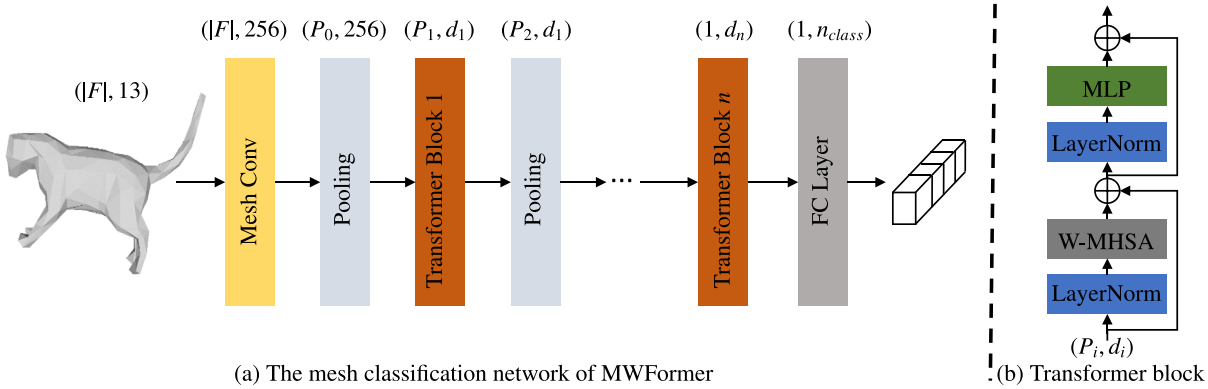

Fig. 2. (a) The architecture of the MWFormer's classification model with n transformer blocks. The input mesh is encoded into a tensor in the shape of $|F| \times 13$ and then sent to MWFormer. The output of the model is in the shape of n_{class}. (b) The architecture of our transformer block.

performance on low-quality meshes. With our BFS-Pooling operation, our MWFormer can handle more general meshes without such constraints. The transformer architecture also enables MWFormer to capture the global relationships among all mesh faces.

2.3. Transformer in deep learning

Self-attention is a special attention mechanism which is proposed by Lin et al. [48] for visualizing and interpreting sentence embeddings. After that, many self-attention models (transformers) [9,49–51] have quickly made great achievements in various natural language processing tasks. Recently, Dosovitskiy et al. [10] introduced the first transformer structure called vision transformer (ViT) into computer vision, and obtained excellent performance in image classification. Then some variants [11,13,52] showed big potential to replace convolutional neural networks (CNNs), making transformer the state-of-the-art framework in computer vision. Recently, transformer-based models also demonstrated their strong performance on 3D vision tasks [13,14,53]. Graphormer [54] introduced a general transformer architecture with effective graph encoding into Graph Neural Network (GNN). Katam [55] stacked transformer blocks to process mesh vertices in mesh segmentation tasks. Transformesh [56] encoded a sequence of hippocampi meshes and applied transformers to process them. Compared to these methods, our proposed generic mesh pooling operations can be incorporated into different variants of transformers in MWFormer, which can be applied to both mesh classification and segmentation tasks. Readers are referred to the recent surveys [57–59] for a more comprehensive review.

3. Mesh transformers with BFS pooling

In this section, we first introduce how we process raw input meshes. Then we detail how to tokenize a mesh into a patch-like representation by sampling and pooling. Finally, we describe the design of our MWFormer with a hierarchical architecture, and a CSA module for segmentation tasks.

3.1. Overall design

Given a connected triangular mesh $M = (V, E, F)$ as input, we treat its faces F as our initial input units because mesh faces contain rich topological and geometric information. Drawn from SubdivNet [7] and MeshNet [46], we extract both spatial descriptors and structure descriptors to better characterize each face. The initial face feature consists of the following components:

- Face area $A \in \mathbb{R}$: the area of a triangular face;
- Face normal $\mathbf{n} \in \mathbb{R}^3$: the normal of a triangular face;
- Face center $\mathbf{c} \in \mathbb{R}^3$: the center of a triangular face;
- Face angles $(a_1, a_2, a_3) \in \mathbb{R}^3$: the combination of three angles of a triangular face;
- Face curvatures $(\kappa_1, \kappa_2, \kappa_3) \in \mathbb{R}^3$: the inner products of three vertex normal and the face normal.

We sort the face angles and curvatures for alignment. Other features, such as face colors, can be added for different tasks. After this process, each mesh face is expressed in the form of a 13-dimensional vector if we choose to use all features and the input mesh can then be represented as a tensor of size $(N, 13)$, where $N = |F|$ is the number of faces of the input mesh. The input tensor is then fed into stacked convolution layers to obtain a more semantic representation and larger receptive fields by aggregating local information. Afterward, the faces are divided into multiple patches by using the proposed BFS-Pooling algorithm. Each patch contains a bundle of adjacent faces and inherits the connection of faces with other patches, making it possible to apply convolution kernels to them. The discrete patches will then be sent to the transformer backbones to generate mesh features. The architecture of our transformer blocks is shown in Fig. 2(b). It follows the conventional transformer structure which consists of a self-attention module and a feed-forward module. Before the start of the next transformer stage, MWFormer will generate new patches in a down-sampling way, which forms a hierarchical architecture. The whole network architecture for mesh classification is shown in Fig. 2(a). MWFormer can also handle other mesh-related tasks such as mesh segmentation with minor modifications.

3.2. Mesh tokenization with BFS-Pooling

Transformer is a powerful architecture that is capable of adapting to different input data and capturing the global connections among input tokens. However, transformer requires the input data in the shape of a discrete token sequence. As for 2D vision, it is easy to directly apply transformers because 2D images consist of ordered discrete RGB pixels and have a clear boundary, which simplifies the work to patch the input images with rectangles of fixed size. But for 3D meshes, it is difficult to decide how to embed the meshes into tokens effectively with a down-sampling hierarchical architecture. To this end, we propose a BFS-Pooling operation to generate mesh patches by sampling faces and tokenizing them with pooling.

3.2.1. Sampling

We sample N_p faces as patch centers and expand them by BFS to generate all the patches. To enable effective learning, we try to generate a uniform size distribution for mesh patches like grid-patches in 2D where patches are embedded with the same feature size. We thus sample the initial faces uniformly.

We propose Furthest Face-center Sampling (FFS), which operates similarly to Furthest Point Sampling (FPS) widely used in 3D point cloud processing. FFS calculates the distances among face centers and maintains the regularity of sampling, with the center of the initial face serving as the center of the patch. Like FPS, FFS maintains a distance table. FFS iteratively selects the face farthest from already sampled faces by querying the table. The sampled face is added to the initial face list, and then FFS updates the distance table accordingly. Similar to FPS, FFS can also be applied to patches.

3.2.2. BFS-Pooling for mesh patches

To achieve down-sampling aggregation for meshes, we propose a non-overlapping pooling operation based on the BFS algorithm. Analogous to the patches split uniformly in 2D images, the initial face of each patch is generated by uniform sampling and then BFS is employed to fill each patch, making all patches have a similar size.

Formally, we propose our BFS algorithm in Algorithm 1 with python-like pseudo codes. The BFS algorithm starts from the initial faces F_{init} sampled by the FFS algorithm proposed in Section 3.2.1 and the face adjacency matrix M_{adj}. Faces not in F_{init} are set unmarked. N_{remain} represents the number of unexpanded faces and is initialized as the number of faces. Each patch expands an unmarked face in turns and we use queues Q_{faces} to store expanded faces of each patch, where f_{head} indicates the head face of the queue. f_{head} will be set as the next recorded face in the queue if all its adjacent faces f_{adj} have already been marked. The BFS-pooling process continues until every face has been associated with a patch.

Algorithm 1 BFS Patch Expanding

Input: The list of sampled initial face of each patch F_{init}, the matrix used to describe faces' adjacency M_{adj}, the number of patches N_p.
Output: Q_{faces} recording the affiliation of faces and patches.

Set all Q_{faces} patch queues to empty and append faces in F_{init} to each Q_{faces} respectively.
Assign f_{head} to the initial face in each patch and N_{remain} to the number of faces.
while $N_{remain} > 0$ **do**
  for i in range(N_p) **do**
    Found = False
    while f_{head}[i] != NONE (consider expandable patches only) **do**
      for f_{adj} in $M_{adj}[f_{head}]$ **do**
        if f_{adj} has not been expanded **then**
          mark f_{adj} expanded
          Q_{faces}[i].append(f_{adj})
          N_{remain} -= 1
          Found = True
          break
        end if
      end for
      if *Found* is True **then**
        break (expand the next patch)
      end if
      if there is no succeeding face in the queue Q_{faces}[i] **then**
        f_{head}[i] = NONE
      else
        f_{head}[i] = the next face recorded in Q_{faces}[i]
      end if
    end while
  end for
end while

3.2.3. Patch tokenization with feature aggregation

After we obtain which patch the faces belong to, we need to aggregate all the face features to generate a new patch token representing the feature of the patch. Due to different face numbers of patches, we cannot adopt concatenation-based aggregation in our method. In practice, we use the average pooling and addition to aggregate all the face features.

3.3. Mesh Window TransFormer - MWFormer

After being processed by the pooling operations proposed in Section 3.2, the features are organized in a discrete token form $F_p \in \mathbb{R}^{N_p \times d}$, which can be directly processed by traditional transformer architecture.

A conventional transformer block contains two modules: one is the self-attention module and the other is the feed-forward module. Given an input feature F_{in} in the shape of $N_t \times C$ where N_t stands for the number of tokens and C stands for the channel length of each token feature, the transformer block first sends it to the self-attention (SA) module. The SA module is permutation-invariant and generates the *query (Q), key (K)*, and *value (V)* matrices from F_{in} with linear layers. Then the query and key matrices can be used to calculate an attention map. After normalization, the attention map can be treated as a weight matrix A. The output of the SA module F_{out} is the product of the attention map and value matrix. All the processes can be described using the terminology below [9]:

$$Q, K, V = F_{in} W_{qkv} \tag{1}$$

$$\hat{A} = (QK^T)/\sqrt{d_a} \tag{2}$$

$$A = softmax(\hat{A}) \tag{3}$$

$$F_{out} = AV \tag{4}$$

where $\sqrt{d_a}$ is a scaling factor. The SA module is self-adaptive and capable of capturing the global relationships among all the input tokens.

A very important learning strategy of transformers in the image vision domain is to combine pixel values with the positional encoding (PE) to distinguish different patches. In practice, both relative PE and absolute PE can improve transformer models' performance on various tasks. In our MWFormer, we already initialize our input tensor with the center coordinates of faces, which can provide intrinsic positional knowledge. Hence we do not apply additional PE to the proposed MWFormer.

3.3.1. Hierarchical MWFormer

Many neoteric transformer architectures are presented in a hierarchical design by down-sampling the tokens, which requires neighborhood information of tokens. While down-sampled 2D image grids still remain token-wise connectivity, the connectivity of our embedded mesh patch tokens is ambiguous. Using the face adjacency information, we define that two newly generated patches are adjacent when any face or patch belonging to them is adjacent. An example of hierarchical down-sampling is shown in Fig. 3. However, it is hard to decide the k-nearest neighbors of the patches. As for 3D point clouds, PCT [13] uses Euclidean distance to determine the nearest neighbors of a certain point

Fig. 3. The hierarchical architecture. The face patches inherit connection relationships from original meshes, which enables them to be continuously processed by the pooling operation to generate new patches from previous face patches.

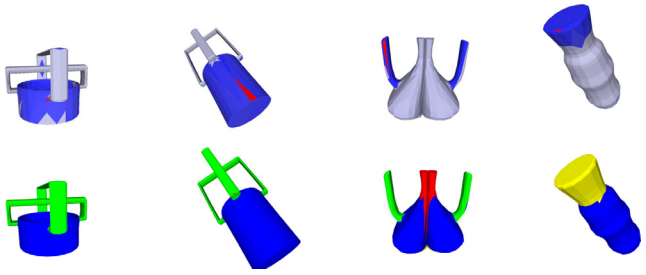

Fig. 4. The attention map (top row) from our MWFormer and corresponding mesh segmentation results (bottom row) of the COSEG-Vases dataset [60]. We normalize the attention map of the red faces and set a threshold to filter faces with low attention weights. We color the faces with high correlation in blue.

to implement a hierarchical architecture. However, simply using Euclidean distance to measure the distance of mesh patches to get their neighbors will lose the topological information provided by original meshes. This is because two face patches close in Euclidean distance may actually be in different parts of the mesh. Instead, we measure the distance $\mathbb{D}$ of two patches by the number of faces along the shortest path (across edges) connecting these two patches. With the distance $\mathbb{D}$, we can get neighbor relationships on mesh surfaces. Also, it is easy to compute $\mathbb{D}$ with the adjacent information of patches.

After this process, the discrete patches are converted into a non-directed graph, which makes the down-sampling in the patch space possible. In this way, we can deploy multiple transformer stages to enhance MWFormer's ability for feature extraction.

By using the pooling operation designed in Section 3.2 and the definition of connectivity among patch tokens, we can finally deploy hierarchical transformers on input meshes.

3.3.2. Window-based local attention mechanism

Although conventional transformer structures are effective in extracting global information of meshes and patches, they lack emphasis on the face locality, which can negatively impact the model's performance because local topological and spatial knowledge is vital in mesh learning. To address this issue, we introduce a window-based local attention mechanism to the mesh domain, which is why our model is named Mesh **Window** Transformer (MWFormer). Specifically, MWFormer will perform self-attention within patches of the same window. When the number of windows is 1, the transformer structure of MWFormer is equivalent to the conventional ViT [10] structure in the image domain. To clearly demonstrate how transformer structures of MWFormer capture global relationship among faces, we visualize the distributions of attention weights in the mesh segmentation task in Fig. 4, where faces belonging to the same part as the selected face should have high attention weights.

In order to generate local windows for face patches, we utilize a sampling approach as it is challenging to achieve both a fixed window size and non-overlapping partitions at the same time.

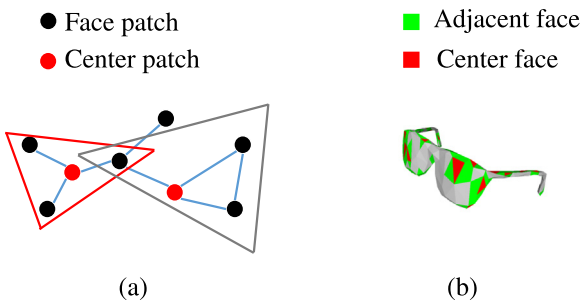

Fig. 5. Structures of MWFormer. (a) MWFormer with the window size being 4. Windows are colored differently. (b) We use adjacent faces (green) in the convolution operation.

Our proposed Furthest Face-center Sampling (FFS) algorithm is used to sample the centers of each attention window, which are then expanded to the given window size. However, since FFS sampling algorithm cannot guarantee absolute uniformity and faces the problem of overlapping, the product of the window's quantity and the window size $k+1$ needs to be larger than the number of patches to better cover all patches. After performing the self-attention operations within each window, we project the features in the windows back to the original patches. Due to the possibility of overlapping windows, we perform an addition aggregation on those overlapped patches. We present how we generate windows in Fig. 5(a) when window size is 4.

It can be noticed that MWFormer can now better utilize local information, but may lose the ability to grasp global connections of all patches, which is the main advantage of transformer-based structures. To tackle this problem, we propose a re-sampling strategy that is performed in each training step. Specifically, we re-sample the centers of windows, allowing MWFormer to learn relationships between patches from different adjacent zones and propagating features of certain patches to all other patches. This approach enables MWFormer to better capture both local and global information, resulting in improved performance in mesh classification and segmentation tasks.

3.4. Cross-Stage Attention Block

As for mesh segmentation tasks, the process of transforming features of patches back to original faces is crucial. To better aggregate information of different transformer stages, we propose a Cross-Stage Attention(CSA) Block. CSA applies attention operations on patches of different levels simultaneously. To avoid high computation overheads, we do not include original face features in the CSA block. Combining features of all patch stages helps MWFormer to better learn their global connections. Following the CSA block, a simple U-Net structure is used to generate the final segmentation results.

3.5. Enhance the locality with convolution

Transformer is powerful at extracting global relationships of face patches. However, simply using transformers will lead to ignorance of neighborhood information, which has been proven to be valuable in MeshCNN [6] and SubdivNet [7]. So we incorporate the convolution operation into our network architecture to enable MWFormer to better extract and aggregate local features for both meshes and patches.

The convolution operation can be described as below:

$$Conv(f_i) = w_0 \cdot e_i + w_1 \cdot \sum_{f_j \in \Omega(f_i)} e_j + w_2 \cdot \sum_{f_j \in \Omega(f_i)} (|e_j - e_i|), \tag{5}$$

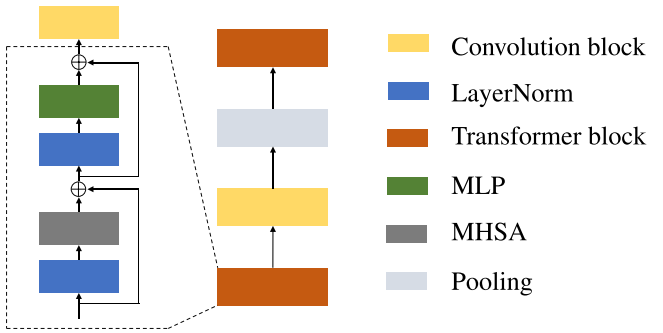

Fig. 6. Convolution layer is inserted after the transformer block to better extract local features.

where $\Omega(f_i)$ means the adjacent faces of the face f_i, and w_0, w_1, w_2 are the learnable kernel parameters. The whole convolution operation is permutation invariant. Since transformer models can effectively learn global geometric features, we just use adjacent faces to extract local information. We show how the convolution operation is carried out on given red faces in Fig. 5(b). We also present the structure of a single stage in Fig. 6.

4. Experiments

In this section, we show the generality of MWFormer by conducting quantitative experiments. We evaluate the performance of our MWFormer on mesh classification and mesh segmentation tasks. The MWFormer's architecture is implemented on the Jittor deep learning framework [61].

4.1. Data process and augmentation

Mesh objects in the same dataset may have different sizes. Following MeshCNN [6] and SubdivNet [7], we normalize the input meshes into the unit cube space. Then we randomly rescale the input meshes with a Gaussian noise whose mean $\mu = 1$ and std $\sigma = 0.1$. Since the orientation of 3D mesh objects also plays an important role and some orientations are missed in the segmentation datasets, we randomly rotate meshes in three axes with an Euler angle in $\{\frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$ for segmentation tasks.

4.2. Implementation details

We adopt the Adam optimizer in all tasks. As for the classification task on the SHREC11 [62] datasets, we set the number of transformer stages in MWFormer to 2 and the window number to 1. For classification tasks on the Manifold40 [7] dataset, we set the stage numbers to 3 with 128 to 32 patches with window size being 8. Following SubdivNet [7], we use area, curvatures, and angles as the input features for the SHREC dataset and all five input features for Manifold40. For segmentation tasks on COSEG [60] datasets, we set the transformer block numbers to 2, 2, 6 with 512, 128, and 32 patches and window size being 8. The patches are divided by 4 following Loop division.

4.3. Mesh classification

We conduct mesh classification experiments on two public 3D mesh datasets: SHREC11 [62] and Manifold40 [7].

Table 1
Mesh classification accuracy on SHREC11 dataset.

Method	Split-16	Split-10	Split-1
MeshCNN [6]	98.6%	91.0%	–
PD-MeshNet [44]	99.7%	99.1%	–
MeshWalker [45]	98.6%	97.1%	–
HodgeNet [43]	99.2%	94.7%	–
DiffusionNet [42]	–	99.7%	–
SubdivNet [7]	100%	99.5%	36.5%
MWFormer	100.0%	100.0%	41.7%

Table 2
Mesh classification accuracy on Manifold40 dataset.

Method	Manifold40
MeshWalker [45]	90.5%
MeshNet [46]	88.4%
SubdivNet [7]	91.2%
MWFormer	92.2%

4.3.1. SHREC11

SHREC11 [62] dataset contains 600 mesh objects of 30 different classes. We use the split settings provided in MeshCNN [6] and evaluate our MWFormer's learning ability on two protocols: Split-10 with half of the samples in the training set and Split-16 with 80% of the samples in the training set. The quantitative results compared with other competitors are shown in Table 1. Our MWFormer can achieve 100% classification accuracy under both protocols, notably without the need of voting strategy like SubdivNet [7]. The results show that by introducing global relationships to mesh learning, the models can better extract features of meshes to categorize them more accurately. Combining the local features with global relationships also helps learning the 3D mesh representation.

We also evaluate our method under the SHREC's Split-1 protocol, where the training dataset only contains 1 sample for each class to show the transfer learning capability of MWFormer in the few-shot scenario. We initialize our network with parameters from models trained on the Manifold40 dataset and compare the result with SubdivNet [7] in Table 1. The results show that with transformer architectures' strong ability in transfer learning and prior global geometric knowledge, MWFormer can produce promising results in few-shot learning.

4.3.2. Manifold40

Manifold40 [7] reconstructs all meshes from ModelNet40 [22] into 2-manifold and watertight meshes with the same number of faces, i.e. 500. The classification task on Manifold40 is challenging due to the reconstruction error and simplification distortion introduced by the remeshing process. We compare the classification results with other networks on the Manifold40 dataset in Table 2. It can be seen that MWFormer can achieve competitive results on this dataset. This shows that the global relationships learned by transformer architectures can help MWFormer to have a more holistic understanding of meshes while methods with small perceptive fields stuck in the reconstruction errors and simplification distortion.

4.4. Mesh segmentation

We also test our MWFormer on 3D mesh shape segmentation tasks. We train MWFormer on three COSEG datasets [60] to predict the part probability of each face in the mesh.

COSEG datasets consist of three large subsets: aliens, vases, and chairs. Following MeshCNN [6], we split the datasets into a train/test split of 85%/15%. We trained our MWFormer on these

Table 3
Mesh segmentation accuracy on three COSEG datasets.

Method	Chairs	Vases	Aliens
MeshCNN [6]	91.1%	90.0%	93.5%
HodgeNet [43]	95.7%	90.3%	96.0%
PD-MeshNet [44]	95.1%	93.4%	96.7%
SubdivNet [7]	96.3%	95.9%	96.3%
MWFormer	99.0%	95.4%	97.5%

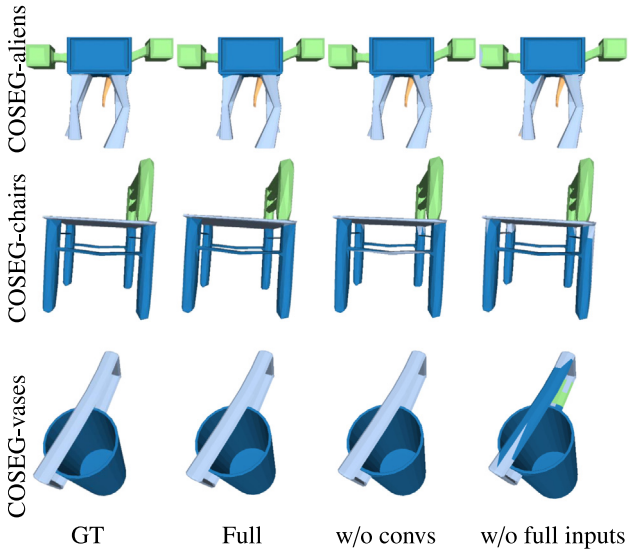

Fig. 7. Comparisons of the segmentation results on COSEG datasets under different settings.

datasets. We re-trained MeshCNN [6], PD-MeshNet [44] and SubdivNet [7] on the new face labels and dataset splits with their settings for comparison. The quantitative results are shown in Table 3 and the visualized results under different settings are presented in Fig. 7, demonstrating that our MWFormer can achieve better or competitive performance. It can be observed that the locality introduced by the local self-attention in MWFormer allows for a better understanding of complex meshes, like those in COSEG-Chairs and COSEG-Aliens datasets. Some meshes in COSEG datasets are not manifold (e.g. with odd face numbers), but with BFS pooling and transformer, our MWFormer can be directly applied on them without preprocessing these meshes like SubdivNet [7]. We also showcase mesh segmentation results along with mesh patch partitions in Fig. 8. It can be observed that faces belonging to patches across different parts can be segmented accurately with geometric and topologic information extracted by MWFormer and the CSA module.

4.5. Ablation study

4.5.1. Input features

As described in Section 3.1, there are five available features that can be added to the input. We conduct experiments on the classification and segmentation tasks to show the influence of different input features. The results are shown in Table 4. We observed that for different datasets, the combination of these features should also vary. Specifically, for small datasets like SHREC11-10, just using parts of the input features can have a better performance because it may ease the overfitting in transformer models. However, for big datasets with more categories like Manifold40 and segmentation tasks requiring face labels, incorporating more input features can improve the model's coverage and better represent the intrinsic information of meshes.

Table 4
Different input features will influence MWFormer's performance.

Datasets	Full inputs	Inputs w/o face center and normal
SHREC-10	95.7%	100.0%
Manifold40	92.2%	80.2%
Aliens	97.5%	94.1%
Chairs	99.0%	96.6%
Vases	95.4%	93.0%

Table 5
Segmentation results of MWFormer on the COSEG-chairs dataset. The FLOPs is the amount of computation in transformer blocks processing a single mesh.

Method	Accuracy	MFLOPs
With pooling operation (2 blocks in the last stage)	98.6%	531.24
With pooling operation (4 blocks in the last stage)	98.8%	556.53
Without pooling operation	98.8%	3557.38

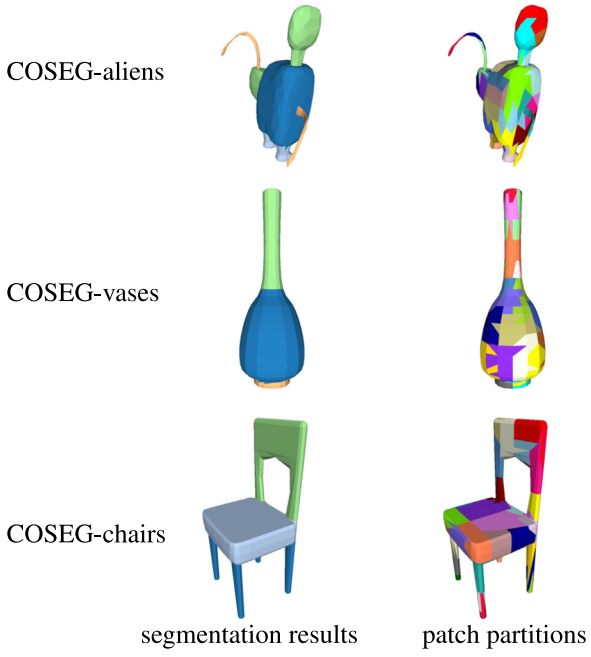

Fig. 8. The visualization of segmentation results along with the patch partitions.

4.5.2. Pooling operation

One main contribution of MWFormer is to present a BFS pooling operation to convert mesh shapes into face patches. Here we consider a special case where each patch only contains one face. In this situation, MWFormer does not perform the pooling operation and can better preserve the geometric information of the original meshes. However, the computational cost and memory consumption is significantly increased. We conduct experiments on mesh segmentation with and without the pooling process to show the efficiency and performance of the BFS pooling operation. We set transformer block numbers of MWFormer without pooling operations all to 2. As for MWFormer with BFS-pooling, we only set different block numbers in the last stage. The results are shown in Table 5.

It can be observed that maintaining the original structure of mesh shapes leads to a marginal improvement in the performance of mesh segmentation tasks with the same network architecture. However, it will consume a lot of memory resources, so MWFormer cannot run with large networks without pooling. With our pooling operation, we can adjust the number of transformer blocks in stages with smaller patch numbers, which can better aggregate high-level mesh features with little computational and memory overheads. It is also worth noting that

Table 6
Performance with and without random sampling in each transformer stage.

Dataset	Fixed sampling	Re-sampling
Manifold	86.7%	92.2%
COSEG-Vases	92.4%	95.4%

Table 7
Segmentation results on COSEG datasets with and w/o convolution operations.

Datasets	With convs	w/o convs
Aliens	97.5%	97.4%
Chairs	99.0%	98.8%
Vases	95.4%	95.3%

Table 8
Accuracy with different window settings.

Dataset	Acc.all patches	Acc.window size = 8
SHREC11–10	100.0%	98.7%
Manifold40	91.2%	92.2%

MWFormer with pooling operations can achieve the same performance with less than 1/6 of the computational cost in transformer blocks.

4.5.3. Random sampling on each stage

As explained above in Section 3.3.2, we re-sample patches before every transformer stage in each training iteration to introduce more global connections to MWFormer and alleviate the loss of face information. To evaluate the effectiveness of our proposed sampling method, we compare the results with fixed sampling points on the same classification and segmentation tasks. Results are shown in Table 6.

It can be observed that, with fixed sampling points in each level of the transformer stage, the performance of MWFormer drops dramatically. This means that the information loss caused by fixed sampling is severe to the training of MWFormer's transformer architecture.

4.5.4. Convolution blocks

To show how convolution blocks can help mesh learning, we conduct experiments on COSEG datasets with and without the convolution blocks after each transformer block and list results in Table 7. The results show that even for MWFormer which already introduces locality in their transformer blocks, incorporating convolution operations can enhance the ability to aggregate mesh features in adjacent patches by adding more locality into the models to help MWFormer learn mesh structures better.

5. Discussion

5.1. Performance with different window number

The window number of the MWFormer is set as a hyperparameter. If the window number is 1, the whole model becomes the same as ViT. When the window number increases, more local information can be extracted in attention steps. We show the performance of MWFormer with different window numbers in Table 8.

As we can observe, the performance of MWFormer indeed varies with the window number. For small datasets like SHREC11 [62], MWFormer with only one window achieves the best performance. This is because MWFormer may capture excessive local information in the local attention stages, leading to overfitting of transformer blocks when trained on small datasets and finally degrading the performance. However, for larger datasets like Manifold40, the strong locality introduced by the local attention assists MWFormer in achieving more competitive results.

Table 9
Running time on Manifold40 dataset.

Method	Python	C++	Speed boost
MWFormer	6.31 s	0.104 s	60.67x

5.2. Running speed

While MWFormer runs models on GPUs, it is hard to use interfaces provided by deep learning frameworks to implement BFS pooling operations. So we try to implement these operations with Python and find out that these operations become the bottleneck of the training and testing phases. To address these problems we implement these operations using C++ instead and get drastic improvement in running speed. We test the time taken to run 1 iteration of training on the Manifold40 datasets with *batch_size* = 32 on a single NVIDIA 3090 GPU and list the time in Table 9 to better demonstrate this.

5.3. Limitations

The distance metric $\mathbb{D}$ we use in BFS-pooling limits MWFormer's performance in processing meshes with multiple disconnected components due to the undefined distance among them. Moreover, as for large scene segmentation like ScanNet [63], the time taken to process meshes with hundreds of thousands of faces is too high to be acceptable because the sampling algorithm and transformer architecture itself are both sensitive to the input face numbers. Besides, training and testing on these meshes require extra processes like scene cropping, which can influence MWFormer's performance because such processes will limit transformers' reception field to small crops.

6. Conclusions

In this paper, we introduce MWFormer, a transformer-based architecture for 3D mesh deep learning that can handle both local and global connections of mesh face patches. We propose a BFS-based pooling operation to convert a connected mesh into discrete patches. The patches inherit the geometric information of the original mesh, which makes them suitable to be processed by hierarchical transformer-based backbones. We conduct extensive experiments on mesh classification and mesh segmentation tasks, showing that MWFormer can produce competitive results on these tasks. We envisage that MWFormer can be helpful in more relative applications such as medical model segmentation and geometric analysis.

As for future research, we aim to improve MWFormer's efficiency and performance. Currently, the BFS pooling operation and k-NN searching are implemented on the CPU. This will lead to frequent data transfers between GPU and CPU, which slows down the training process. Additionally, the presented sampling algorithm cannot guarantee the sampled faces to be an ideal uniform distribution. By adopting a more uniform sampling method, such as Lloyd's algorithm, we can perform the BFS pooling on more balanced patches which may further improve MWFormer's performance.

Finally, given the recent progress in transformer architectures and the flexibility of the BFS pooling operation, it is straightforward to extend these new transformer models in image domains for 3D mesh learning. This may significantly enhance the understanding of 3D shapes and benefit complex downstream tasks in this field, such as open-vocabulary and large-scene segmentation tasks.

Authorship contribution statement

Hao-Yang Peng: Conceptualization, Methodology, Software, Visualization, Data curation, Writing – original draft. **Meng-Hao Guo:** Conceptualization, Methodology, Writing – original draft. **Zheng-Ning Liu:** Conceptualization, Validation. **Yong-Liang Yang:** Validation, Writing – review & editing. **Tai-Jiang Mu:** Conceptualization, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (2021ZD0112902), and the National Natural Science Foundation of China (Grant No. 62220106003, 61902210).

References

[1] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: IEEE conference on computer vision and pattern recognition (CVPR). 2016, p. 770–8.
[2] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: International conference on learning representations (ICLR). 2015.
[3] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In: IEEE conference on computer vision and pattern recognition (CVPR). 2015, p. 1–9.
[4] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely connected convolutional networks. In: IEEE conference on computer vision and pattern recognition (CVPR). 2017, p. 4700–8.
[5] Guo M-H, Lu C-Z, Liu Z-N, Cheng M-M, Hu S-M. Visual attention network. 2022, arXiv preprint arXiv:2202.09741.
[6] Hanocka R, Hertz A, Fish N, Giryes R, Fleishman S, Cohen-Or D. Meshcnn: a network with an edge. ACM Trans Graph 2019;38(4):1–12.
[7] Hu S-M, Liu Z-N, Guo M-H, Cai J-X, Huang J, Mu T-J, Martin RR. Subdivision-based mesh convolution networks. ACM Trans Graph 2022;41(3):25:1–25:16.
[8] Chen L-C, Papandreou G, Schroff F, Adam H. Rethinking atrous convolution for semantic image segmentation. 2017, arXiv preprint arXiv:1706.05587.
[9] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Advances in neural information processing systems (NeurIPS). 2017, p. 5998–6008.
[10] Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, Uszkoreit J, Houlsby N. An image is worth 16x16 words: Transformers for image recognition at scale. In: International conference on learning representations (ICLR). 2021, URL: https://openreview.net/forum?id=YicbFdNTTy.
[11] Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, Lin S, Guo B. Swin transformer: Hierarchical vision transformer using shifted windows. In: IEEE/CVF international conference on computer vision (ICCV). 2021, p. 9992–10002.
[12] Wang W, Xie E, Li X, Fan D-P, Song K, Liang D, Lu T, Luo P, Shao L. PVT v2: Improved baselines with pyramid vision transformer. Comput Vis Media 2022;8(3):415–24.
[13] Guo M-H, Cai J, Liu Z-N, Mu T-J, Martin RR, Hu S-M. PCT: Point cloud transformer. Comput Vis Media 2021;7(2):187–99.
[14] Zhao H, Jiang L, Jia J, Torr PH, Koltun V. Point transformer. In: IEEE/CVF international conference on computer vision (CVPR). 2021, p. 16259–68.
[15] Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P. Geometric deep learning: going beyond euclidean data. IEEE Signal Process Mag 2017;34(4):18–42.
[16] Xiao Y-P, Lai Y-K, Zhang F-L, Li C, Gao L. A survey on deep geometry learning: From a representation perspective. Comput Vis Media 2020;6(2):113–33.
[17] Su H, Maji S, Kalogerakis E, Learned-Miller E. Multi-view convolutional neural networks for 3d shape recognition. In: IEEE international conference on computer vision (ICCV). 2015, p. 945–53.
[18] Qi CR, Su H, Nießner M, Dai A, Yan M, Guibas LJ. Volumetric and multi-view cnns for object classification on 3d data. In: IEEE conference on computer vision and pattern recognition (CVPR). 2016, p. 5648–56.
[19] Feng Y, Zhang Z, Zhao X, Ji R, Gao Y. GVCNN: Group-view convolutional neural networks for 3D shape recognition. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2018, p. 264–72.
[20] Goyal A, Law H, Liu B, Newell A, Deng J. Revisiting point cloud shape classification with a simple and effective baseline. In: International conference on machine learning (ICML). 2021, p. 3809–20.
[21] Maturana D, Scherer S. Voxnet: A 3d convolutional neural network for real-time object recognition. In: IEEE/RSJ international conference on intelligent robots and systems (IROS). 2015, p. 922–8.
[22] Wu Z, Song S, Khosla A, Yu F, Zhang L, Tang X, Xiao J. 3D shapenets: A deep representation for volumetric shapes. In: IEEE conference on Computer Vision and Pattern Recognition (CVPR). 2015, p. 1912–20.
[23] Zhou Y, Tuzel O. Voxelnet: End-to-end learning for point cloud based 3d object detection. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2018, p. 4490–9.
[24] Qi CR, Su H, Mo K, Guibas LJ. Pointnet: Deep learning on point sets for 3d classification and segmentation. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2017, p. 652–60.
[25] Qi CR, Yi L, Su H, Guibas LJ. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in neural information processing systems (NeurIPS), Vol. 30. 2017, p. 5099–108.
[26] Wang Y, Sun Y, Liu Z, Sarma SE, Bronstein MM, Solomon JM. Dynamic graph cnn for learning on point clouds. ACM Trans Graph (TOG) 2019;38(5):146:1–146:12.
[27] Li Y, Bu R, Sun M, Wu W, Di X, Chen B. Pointcnn: Convolution on x-transformed points. In: Advances in neural information processing systems (NeurIPS), Vol. 31. 2018, p. 828–38.
[28] Thomas H, Qi CR, Deschaud J-E, Marcotegui B, Goulette F, Guibas LJ. Kpconv: Flexible and deformable convolution for point clouds. In: IEEE/CVF international conference on computer vision (ICCV). 2019, p. 6411–20.
[29] Sun C-Y, Yang Y-Q, Guo H-X, Wang P-S, Tong X, Liu Y, Shum H-Y. Semi-supervised 3D shape segmentation with multilevel consistency and part substitution. Comput Vis Media 2023;9(2):229–47.
[30] Park JJ, Florence P, Straub J, Newcombe R, Lovegrove S. Deepsdf: Learning continuous signed distance functions for shape representation. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2019, p. 165–74.
[31] Jiang C, Sud A, Makadia A, Huang J, Nießner M, Funkhouser T, et al. Local implicit grid representations for 3d scenes. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2020, p. 6001–10.
[32] Mescheder L, Oechsle M, Niemeyer M, Nowozin S, Geiger A. Occupancy networks: Learning 3d reconstruction in function space. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2019, p. 4460–70.
[33] Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R. Nerf: Representing scenes as neural radiance fields for view synthesis. In: European conference on computer vision (ECCV). 2020, p. 405–21.
[34] Liu L, Gu J, Zaw Lin K, Chua T-S, Theobalt C. Neural sparse voxel fields. In: Advances in neural information processing systems (NeurIPS), Vol. 33. 2020, p. 15651–63.
[35] Garbin SJ, Kowalski M, Johnson M, Shotton J, Valentin J. Fastnerf: High-fidelity neural rendering at 200fps. In: IEEE/CVF international conference on computer vision (CVPR). 2021, p. 14346–55.
[36] Lemeunier C, Denis F, Lavoué G, Dupont F. Representation learning of 3D meshes using an Autoencoder in the spectral domain. Comput Graph 2022;107:131–43.
[37] Masci J, Boscaini D, Bronstein M, Vandergheynst P. Geodesic convolutional neural networks on riemannian manifolds. In: IEEE international conference on computer wision workshops. 2015, p. 37–45.
[38] Boscaini D, Masci J, Rodolà E, Bronstein M. Learning shape correspondence with anisotropic convolutional neural networks. In: Advances in neural information processing systems (NeurIPS), Vol. 29. 2016, p. 3189–97.
[39] Monti F, Boscaini D, Masci J, Rodola E, Svoboda J, Bronstein MM. Geometric deep learning on graphs and manifolds using mixture model cnns. In: IEEE conference on computer vision and pattern recognition (CVPR). 2017, p. 5115–24.
[40] Yang Y, Liu S, Pan H, Liu Y, Tong X. PFCNN: Convolutional neural networks on 3D surfaces using parallel frames. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2020, p. 13578–87.
[41] Huang J, Zhang H, Yi L, Funkhouser T, Nießner M, Guibas LJ. Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2019, p. 4440–9.

[42] Sharp N, Attaiki S, Crane K, Ovsjanikov M. DiffusionNet: Discretization agnostic learning on surfaces. ACM Trans Graph (TOG) 2022;41(3):27:1–27:16.
[43] Smirnov D, Solomon J. HodgeNet: learning spectral geometry on triangle meshes. ACM Trans Graph 2021;40(4):1–11.
[44] Milano F, Loquercio A, Rosinol A, Scaramuzza D, Carlone L. Primal-dual mesh convolutional neural networks. In: Advances in neural information processing systems (NeurIPS), Vol. 33. 2020.
[45] Lahav A, Tal A. Meshwalker: Deep mesh understanding by random walks. ACM Trans Graph 2020;39(6):1–13.
[46] Feng Y, Feng Y, You H, Zhao X, Gao Y. Meshnet: Mesh neural network for 3d shape representation. In: AAAI conference on artificial intelligence (AAAI), Vol. 33. 2019, p. 8279–86.
[47] Li X, Li R, Zhu L, Fu C-W, Heng P-A. DNF-Net: A deep normal filtering network for mesh denoising. IEEE Trans Vis Comput Graphics (TVCG) 2021;27(10):4060–72.
[48] Lin Z, Feng M, dos Santos CN, Yu M, Xiang B, Zhou B, Bengio Y. A structured self-attentive sentence embedding. In: International conference on learning representations (ICLR). 2017, URL: https://openreview.net/forum?id=BJC_jUqxe.
[49] Devlin J, Chang M-W, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Conference of the North American chapter of the association for computational linguistics: Human language technologies (NAACL-HLT). 2019, p. 4171–86.
[50] Yang Z, Dai Z, Yang Y, Carbonell J, Salakhutdinov RR, Le QV. Xlnet: Generalized autoregressive pretraining for language understanding. In: Advances in neural information processing systems (NeurIPS), Vol. 32. 2019, p. 5754–64.
[51] Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, et al. Language models are few-shot learners. In: Advances in neural information processing systems (NeurIPS), Vol. 33. 2020, p. 1877–901.
[52] Wang W, Xie E, Li X, Fan D, Song K, Liang D, Lu T, Luo P, Shao L. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In: IEEE/CVF international conference on computer vision (ICCV). 2021, p. 548–58.
[53] Lin K, Wang L, Liu Z. End-to-end human pose and mesh reconstruction with transformers. In: IEEE/CVF conference on computer vision and pattern recognition (CVPR). 2021, p. 1954–63.
[54] Ying C, Cai T, Luo S, Zheng S, Ke G, He D, Shen Y, Liu T-Y. Do transformers really perform badly for graph representation? In: Advances in neural information processing systems (NeurIPS), Vol. 34. 2021, p. 28877–88.
[55] Katam H. 3D mesh segmentation using transformer based graph operations. Tech. Rep., Technische Universität München; 2021, URL: https://elib.dlr.de/140566/.
[56] Sarasua I, Pölsterl S, Wachinger C, Neuroimaging AD, et al. Transformesh: A transformer network for longitudinal modeling of anatomical meshes. In: International workshop on machine learning in medical imaging. Springer; 2021, p. 209–18.
[57] Guo M-H, Xu T-X, Liu J-J, Liu Z-N, Jiang P-T, Mu T-J, Zhang S-H, Martin RR, Cheng M-M, Hu S-M. Attention mechanisms in computer vision: A survey. Comput Vis Media 2022;8(3):331–68.
[58] Han K, Wang Y, Chen H, Chen X, Guo J, Liu Z, Tang Y, Xiao A, Xu C, Xu Y, Yang Z, Zhang Y, Tao D. A survey on vision transformer. IEEE Trans Pattern Anal Mach Intell (PAMI) 2022;1. http://dx.doi.org/10.1109/TPAMI.2022.3152247.
[59] Guo M-H, Liu Z-N, Mu T-J, Liang D, Martin RR, Hu S-M. Can attention enable MLPs to catch up with CNNs? Comput Vis Media 2021;7(3):283–8.
[60] Wang Y, Asafi S, Van Kaick O, Zhang H, Cohen-Or D, Chen B. Active co-analysis of a set of shapes. ACM Trans Graph 2012;31(6):1–10.
[61] Hu S-M, Liang D, Yang G-Y, Yang G-W, Zhou W-Y. Jittor: a novel deep learning framework with meta-operators and unified graph execution. Sci China Inf Sci 2020;63(12):1–21.
[62] Lian Z, Godil A, Bustos B, Daoudi M, Hermans J, Kawamura S, Kurita Y, Lavoué G, Van Nguyen H, Ohbuchi R, et al. SHREC'11 track: Shape retrieval on non-rigid 3D watertight meshes. In: 3DOR@ Eurographics. 2011, p. 79–88.
[63] Dai A, Chang AX, Savva M, Halber M, Funkhouser T, Nießner M. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: IEEE conference on computer vision and pattern recognition. CVPR, 2017, p. 5828–39.