Mesh Variational Autoencoders with Edge Contraction Pooling

Yu-Jie Yuan\textsuperscript{a,b}, Yu-Kun Lai\textsuperscript{c}, Jie Yang\textsuperscript{a,b}, Qi Duan\textsuperscript{d}, Hongbo Fu\textsuperscript{e} and Lin Gao\textsuperscript{a,b}

\textsuperscript{a}Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, CAS
\textsuperscript{b}University of Chinese Academy of Sciences
\textsuperscript{c}School of Computer Science and Informatics, Cardiff University, UK
\textsuperscript{d}SenseTime Research
\textsuperscript{e}City University of Hong Kong

Abstract

3D shape analysis is an important research topic in computer vision and graphics. While existing methods have generalized image-based deep learning to meshes using graph-based convolutions, the lack of an effective pooling operation restricts the learning capability of their networks. In this paper, we propose a novel pooling operation for mesh datasets with the same connectivity but different geometry, by building a mesh hierarchy using mesh simplification. For this purpose, we develop a modified mesh simplification method to avoid generating highly irregularly sized triangles. Our pooling operation effectively encodes the correspondence between coarser and finer meshes in the hierarchy. We then present a variational auto-encoder (VAE) structure with the edge contraction pooling and graph-based convolutions, to explore probability latent spaces of 3D surfaces and perform 3D shape generation. Our network requires far fewer parameters than the original mesh VAE and thus can handle denser models thanks to our new pooling operation and convolutional kernels. Our evaluation also shows that our method has better generalization ability and is more reliable in various applications, including shape generation and shape interpolation.

1. Introduction

In recent years, 3D shape datasets have been increasingly available on the Internet. Consequently, data-driven 3D shape analysis has been an active research topic in computer vision and graphics. Apart from traditional data-driven works such as\cite{7}, recent works attempted to generalize deep neural networks from images to 3D shapes such as\cite{30, 31, 18} for triangular meshes,\cite{24} for point clouds,\cite{35, 21} for voxel data, etc. In this paper, we concentrate on deep neural networks for triangular meshes. Unlike images, 3D meshes have complex and irregular connectivity. Most existing works tend to keep mesh connectivity unchanged from layer to layer, thus losing the capability of increased receptive fields when pooling operations are applied.

As a generative network, the Variational Auto-Encoder (VAE)\cite{16} has been widely used in various kinds of generation tasks, including generation, interpolation and exploration on triangular meshes\cite{31}. The original MeshVAE\cite{31} uses a fully connected network that requires a huge number of parameters and its generalization ability is often weak. Although the fully connected layers allow changes of mesh connectivity between layers, due to irregular changes, such approaches cannot be directly generalized to convolutional layers. Some works\cite{18, 9} adopt convolutional layers in the VAE structure. However, such convolution operations cannot change the connectivity of the mesh. The work\cite{26} introduces sampling operations in convolutional neural networks on meshes, but their sampling strategy does not aggregate all the local neighborhood information when reducing the number of vertices. Therefore, in order to deal with denser models and enhance the generalization ability of the network, it is necessary to design a pooling operation for meshes similar to the pooling for images to reduce the number of network parameters. Moreover, it is desired that the defined pooling operation can support further convolutions and allow recovery of the original resolution through a relevant de-pooling operation.

In this paper we propose a VAE architecture with newly defined pooling operations. Our method uses mesh simplification to form a mesh hierarchy with different levels of details, and achieves effective pooling by keeping track of the mapping between coarser and finer meshes. To avoid generating highly irregular triangles during mesh simplification, we introduce a modified mesh simplification approach based on\cite{11}. The input to our network is a vertex-based deformation feature representation\cite{8}, which unlike 3D coordinates, encodes deformations using deformation

1
gradients defined on vertices. Our framework uses a collection of 3D shapes with the same connectivity to train the network. Such meshes can be easily obtained through consistent remeshing. Also, we adopt graph convolutions [6] in our network. In all, our network follows a VAE architecture where pooling operations and graph convolutions are applied. As we will show later, our network not only has better generalization capabilities but also can handle much higher resolution meshes, benefiting various applications, such as shape generation and interpolation.

2. Related Work

Deep Learning for 3D Shapes. Deep learning on 3D shapes has received increasing attention. Boscaini et al. [2, 3] generalize CNNs from the Euclidean domain to the non-Euclidean domain, which is useful for 3D shape analysis such as establishing correspondences. Bronstein et al. [5] give an overview of utilizing CNNs on non-Euclidean domains, including graphs and meshes. Masci et al. [20] propose the first mesh convolutional operations by applying filters to local patches represented in geodesic polar coordinates. Maron et al. [19] parameterize a surface to a planar flat-torus to define a natural convolution operator for CNNs on surfaces. Wang et al. [33, 34] propose octree-based convolutions for 3D shape analysis. Unlike local patches, planar flat-tori, or octrees, our work performs convolutional operations using vertex features [8] as input.

To analyze meshes with the same connectivity but different geometry, the work [31] first introduced the VAE architecture to 3D mesh data, and demonstrates its usefulness using various applications. Tan et al. [30] use a convolutional auto-encoder to extract localized deformation components from mesh datasets with large-scale deformations. Gao et al. [9] propose a network that combines convolutional mesh VAEs with CycleGAN [37] for automatic unpaired shape deformation transfer. Their follow-up work [10] further proposes a two-level VAE for generating 3D shapes of man-made objects with fine geometry details and complex structures. The works [30, 9] apply convolutional operations to meshes in the spatial domain, while the works of [6, 13] extend CNNs to irregular graphs by construction in the spectral domain, and show superior performance when compared with spatial convolutions. Following [6, 36], our work also performs convolutional operations in the spectral domain.

While pooling operations have been widely used in deep networks for image processing, existing mesh-based VAE methods either do not support pooling [31, 9], or use a simple sampling process [26], which is not able to aggregate all the local neighborhood information. In fact, the sampling approach in [26], although also based on a simplification algorithm, directly drops vertices, and uses the barycentric coordinates in triangles of the coarse mesh to recover the lost vertices by interpolation. In contrast, our pooling operations can aggregate local information by recording the simplification procedure, and support direct reversal of the pooling operation to effectively achieve de-pooling. More recently, Hanocka et al. [12] proposed MeshCNN, containing a dynamic mesh pooling operation, which conducts mesh simplification according to specific tasks. On the contrary, we define our pooling operation based on a static mesh simplification algorithm, aiming for generating high quality mesh models. The static algorithm ensures consistent hierarchies, so better preserves geometric details and is more robust.

Uniform Sampling or Pooling Methods. Taking point clouds as input, PointNet++ [25] proposes a uniform sampling method for point cloud based neural networks. Using the same idea, TextureNet [14] also conducts uniform sampling on the vertices of a mesh. This kind of sampling method destroys the connection between vertices, turning mesh data into a point cloud, which cannot support further graph convolutions. In contrast, simplification methods can build mesh hierarchies, so can help us perform mesh pooling operations. However, most simplification methods, such as [11], are shape-preserving, but vertices on the simplified meshes can be highly non-uniform. Remeshing operations such as [4], on the other hand, can build uniform simplified meshes, but lose the correspondence between meshes in the hierarchy. We propose a modified mesh simplification method based on the classic method [11] to simplify meshes more uniformly and record the correspondences between the coarse and dense meshes for newly defined mesh pooling and de-pooling operations.

3. Our Framework

In this section we introduce the basic operations and network architecture used in our framework.

3.1. Mesh Simplification

We use mesh simplification to help build reliable pooling operations. For this purpose, mesh simplification not only creates a mesh hierarchy with different levels of details, but also ensures the correspondences between coarser and finer meshes. Our simplification process is based on the classical method [11], which performs repeated edge contraction in an order based on a metric measuring shape changes. However, the original approach cannot guarantee that the simplified mesh contains evenly distributed triangles. To achieve more effective pooling, each vertex in the coarser mesh should correspond to a similarly sized region.

Our observation is that the edge length is an important indicator for this process. To avoid contracting long edges, we incorporate the edge length as one of the criteria to order pairs of points to be simplified. The original work defines the error at vertex \( v = [u_x, u_y, u_z, 1]^T \) to be a quadratic
form \( v^T Q v \), where \( Q \) is the sum of the fundamental error quadrics introduced in [11]. For a given edge contraction \((v_1, v_2) \rightarrow v\), they simply choose to use \( Q = Q_1 + Q_2 \) to be the new matrix which approximates the error at \( v \). So the error at \( v \) will be \( v^T Q v \). We propose to add the new edge length to the original simplification error metric. Specifically, given an edge \((v_i, v_j)\) to be contracted to a new vertex \( \bar{v}_k \), the total error is defined as:

\[
E = \bar{v}_k^T Q \bar{v}_k + \gamma \max \{|L_{km}, L_{kn}| m \in \mathcal{N}_i, n \in \mathcal{N}_j, m \neq j, n \neq i\},
\]

where \( L_{km} \) (resp. \( L_{kn} \)) is the new edge length between vertex \( k \) and vertex \( m \) (resp. vertex \( n \)). \( \mathcal{N}_i \) (resp. \( \mathcal{N}_j \)) is the set of neighboring vertices of vertex \( i \) (resp. vertex \( j \)), and \( \lambda \) is a weight. Note that we only penalize the maximum edge length around newly created vertices \( \bar{v}_k \) to effectively avoid triangles with too long edges. In our experiments, we contract half of the vertices between adjacent levels of details to support effective pooling. A representative simplification example is shown in Fig. 2, which clearly shows the effect of our modified simplification algorithm. The advantage of our modified simplification algorithm over the original one on pooling and thus shape reconstruction will be discussed in Section 4.1.

### 3.2. Pooling and De-pooling

Mesh simplification is achieved by repeated edge contraction, i.e., contracting two adjacent vertices to a new vertex. We exploit this process to define our pooling operation, in a way similar to image-based pooling. We use average pooling for our framework (and alternative pooling operations can be similarly defined). As illustrated in Fig. 3, following an edge contraction step, we define the feature of a new vertex as the average feature of the contracted vertices. This ensures that the pooling operation effectively operates at relevant simplified regions. This process has some advantages: It preserves a correct topology to support multiple levels of convolutions/pooling, and makes the receptive field well defined.

Since our network has a decoder structure, we also need to properly define a de-pooling operation. We similarly take advantage of simplification relationships, and define de-pooling as the inverse operation: the features of the vertices on the simplified mesh are equally assigned to the corresponding contracted vertices on the dense mesh.

### 3.3. Graph Convolution

To form a complete neural network architecture, we adopt the spectral graph convolutions introduced in [6]. Let \( x \) be the input and \( y \) be the output of a convolution operation. \( x \) and \( y \) are matrices where each row corresponds to a vertex and each column corresponds to a feature dimension. Let \( L \) denote the normalized graph Laplacian. The spectral graph convolution used in our network is then defined as

\[
y = g_0(L)x = \sum_{h=0}^{H-1} \theta_h T_h(\tilde{L})x,
\]

where \( \tilde{L} = 2L/\lambda_{max} - I \), \( \lambda_{max} \) is the largest eigenvalue, \( \theta \in \mathbb{R}^H \) is polynomial coefficients, and \( T_h(L) \in \mathbb{R}^{V \times V} \) is the Chebyshev polynomial of order \( h \) evaluated at \( \tilde{L} \).

### 3.4. Network Structure

As illustrated in Fig. 1, our network is built on our average pooling operation and convolutional operation,
with a variational auto-encoder structure. The input to the encoder is the preprocessed ACAP (As-Consistent-As-Possible) features [8] with each dimension linearly scaled to $[-0.95, 0.95]$ to allow using tanh as activation function, which are shaped as $X \in \mathbb{R}^{V \times 9}$, where $V$ is the number of vertices and 9 is the dimension of the deformation representation. The representation effectively encodes local deformations and copes well with large rotations.

Unlike the original mesh VAE [31], which uses fully connected layers, the encoder of our network consists of two graph convolutional layers and one pooling layer followed by another graph convolutional layer. The output of the last convolutional layer is mapped to a mean vector and a deviation vector by two different fully-connected layers. The mean vector does not have an activation function, and the deviation vector uses $\text{sigmoid}$ as the activation function.

The decoder mirrors the encoder steps. However, we use different convolutional weights from the corresponding layers in the encoder, with all layers using the tanh as activation function. Corresponding to the pooling operation, the de-pooling operation as described in Section 3.2 maps features in a coarser mesh to a finer mesh. The output of the whole network is $X \in \mathbb{R}^{V \times 9}$, which has the identical dimension as the input, and can be rescaled back to the deformation representation and used for reconstructing the deformed shape.

In order to train our VAE network, we use the mean squared error (MSE) as the reconstruction loss. Combined with the KL-divergence [17], the total loss function for the model is defined as

$$L = \frac{1}{2M} \sum_{i=1}^{M} \| X^i - \hat{X}^i \|^2_F + \alpha D_{KL}(q(z|X)||p(z)), \quad (3)$$

where $X^i$ and $\hat{X}^i$ represent the preprocessed features of the $i$th model and the output of the network, $\| \cdot \|_F$ is the Frobenius norm of matrix, $M$ is the number of shapes in the dataset, $\alpha$ is a parameter to adjust the priority between the reconstruction loss and KL-divergence. $z$ is the latent vector, $p(z)$ is the prior probability, $q(z|X)$ is the posterior probability, and $D_{KL}$ is the KL-divergence.

### 3.5. Conditional VAE

When the VAE is used for shape generation, it is often preferred to allow the selection of shape types to be generated, especially for datasets containing shapes from different categories (such as men and women, thin and fat, see [23] for more examples). To achieve this, we refer to [28] and add labels to the input and the latent vectors to extend our framework. In this case, our loss function is changed to

$$L_{ce} = \frac{1}{2M} \sum_{i=1}^{M} \| X^i_c - \hat{X}^i_c \|^2_F + \alpha D_{KL}(q(z|X,c)||p(z|c)),$$

where $\hat{X}$ is the output of the conditional VAE, and $p(z|c)$ and $q(z|X,c)$ are conditional prior and posterior probabilities, respectively.

### 3.6. Implementation Details

In our experiments, we contract half of the vertices with $\gamma = 0.001$ in Eq. 1 and set the hyper-parameter $H = 3$ in graph convolutions, $\alpha = 0.3$ in the total loss function. The latent space dimension is 128 for all our experiments. We also use $L_2$ regularization on the network weights to avoid over-fitting. We use Adam optimizer [15] with the learning rate set to 0.001.

### 4. Experiments

#### 4.1. Framework Evaluation

To compare different network structures and settings, we use several shape deformation datasets, including SCAPE dataset [1], Swing dataset [32], Face dataset [22], Horse and Camel dataset [29], Fat (ID:50002) from the MPI Dyna dataset [23], and Hand dataset. For each dataset, it is randomly split into halves for training and testing. We test the capability of the network to generate unseen shapes, and report the average RMS (root mean squared) errors.

**Effect of Pooling.** In Table 1 (Columns 3 and 8) we compare the RMS errors of reconstructing unseen shapes with and without pooling. The RMS error is lower by an average of 6.92% with pooling. The results show the benefit of our pooling and de-pooling operations.

**Ablation Study.** We compare spectral graph convolutions with alternative spatial convolutions, both with the network as shown in Fig. 1. The comparison results are shown in Table 1 (Columns 2 and 3). One can easily find that spectral graph convolutions give better results. Moreover, to demonstrate the benefit of our simplification-based pooling operation, we compare our pooling with the original simplification algorithm [11] for pooling, a representative uniform remeshing method [4] for pooling, the existing graph pooling method [27], and the mesh sampling operation [26]. Our method aims for a uniform, but also shape-preserving simplification, which leads to better generalization ability. The results are shown in Table 1.

**Comparison with State-of-the-Art.** In Table 2, we compare our method with the state-of-the-art mesh-based auto-encoder architectures [9, 26, 31] in terms of RMS errors of reconstructing unseen shapes. We also compare with MeshCNN [12] in Table 3. We modify the segmentation network of MeshCNN for the encoding-decoding task. Thanks to spectral graph convolutions and our pooling, our method consistently reduces the reconstruction errors of unseen data, showing superior generalizability. We further show qualitative reconstruction comparison with [9] and [26] in Fig. 4. It shows that our method leads to more ac-
Table 1. Comparison of RMS (root mean square) reconstruction errors for unseen data using our network with pooling (‘Our Method’), without pooling (‘Only Spectral Conv.’), without pooling and with an alternative spatial convolution operator (‘Only Spatial Conv.’), with original simplification [11]-based pooling, with uniform remeshing [4], with graph pooling [27] and with mesh sampling [26].

Table 2. Comparison of RMS reconstruction errors for unseen data using different auto-encoder frameworks proposed by Tan et al. [31], Gao et al. [9], and Ranjan et al. [26]. ‘-’ means the corresponding method runs out of memory (largely due to the use of fully connected networks).

Table 3. Comparison of MAE (mean absolute error) reconstruction errors with MeshCNN [12]. We use MAE of the five edge features, which are the inputs of MeshCNN, as the metric.

Figure 4. Qualitative comparison of reconstruction results for unseen data with [9] (left) and [26] (right). Reconstruction errors are color-coded on the left and the results on the right also show close-up views for more details. It can be seen that our method leads to more accurate reconstructions and the method of [26] suffers from easily noticeable artifacts.

Figure 5. Comparison of MAE (mean absolute error) reconstruction errors with the method of [26]. MeshVAE needs 129, 745, 920 parameters, while ours only needs 7, 941, 042.

4.2. Generation of Novel Models

Once our network is trained, we can use the latent space and decoder to generate new shapes. We use the standard normal distribution $z \sim N(0, I)$ as the input to the trained decoder. It can be seen from Fig. 5 that our network is capable of generating reasonable new shapes. To prove that the generated shapes do not exist in the model dataset, we find the nearest shapes based on the average per-vertex Euclidean distance in the original datasets for visual comparison. It can be seen that the generated shapes are indeed new and different from any existing shape in the datasets. To show our conditional random generation ability, we train the network on the DYNA dataset from [23]. We use BMI+gender and motion as the condition to train the network. As shown in Fig. 6, our method is able to randomly generate models that are conditioned on the body shape ‘50007’ – a male model with BMI 39.0 and conditioned on the action with the label ‘One Leg Jump’ including lifting a leg.

4.3. Mesh Interpolation

Our method can also be used for shape interpolation. This is also a way to generate new shapes. We linearly interpolate between two latent vectors of two shapes and the probabilistic decoder outputs a 3D deformation sequence. We compare our method on the SCAPE dataset [1] with a state-of-the-art data-driven deformation method [7], as shown in Fig. 7. We can see that the results by the data-driven method of [7] tend to follow the movement sequences from the original dataset which has similar start and end states, leading to redundant motions such as the swing of right arm. In contrast, our interpolation results give more reasonable motion sequences. We show more interpolation results in Fig. 9, including sequences between newly generated models and models beyond human bodies.

We compare our network with MeshVAE [31] to show the ability of our network for processing denser meshes. A comparison example for interpolation is shown in Fig. 8.
Figure 5. Randomly generated new shapes using our framework, along with their nearest neighbors (NN) in the original datasets.

Figure 6. Conditional random generation of new shapes using our framework.

Figure 7. Comparison of mesh interpolation results with [7] (1st row). The models in the leftmost and rightmost columns are the input models to be interpolated.

Figure 8. Interpolation comparison between Mesh VAE [31] and our method. The original elephant model [29] has 42,321 vertices, which cannot be handled by Mesh VAE due to memory restriction and therefore a simplified mesh with 5,394 vertices is used instead. Our method operates on the original mesh model and produces results with more details.

Figure 9. More interpolation results. (a)(b) more diverse shapes other than human bodies. (c) results interpolated between newly generated shapes.

5. Conclusions

In this paper we introduced a newly defined pooling operation based on a modified mesh simplification algorithm and integrated it into a mesh variational auto-encoder architecture, which uses per-vertex feature representations as inputs, and utilizes graph convolutions. Through extensive experiments we demonstrated that our generative model has better generalization ability. Compared to the original Mesh VAE, our method can generate high quality deformable models with richer details. Our experiments also show that our method outperforms the state-of-the-art methods in various applications including shape generation and shape interpolation. One of the limitations of our method is that it can process only homogeneous meshes. As future work, it is desirable to develop a framework capable of handling shapes with different topology as input.
References


[18] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 1


[27] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Minimum point cloud local structures by kernel correlation and graph pooling. In *CVPR*, volume 4, 2018. 4, 5


