The Visual Computer
https://doi.org/10.1007/s00371-023-02913-y

ORIGINAL ARTICLE

A data-centric unsupervised 3D mesh segmentation method

Talya Tümer Sivri[1] · Yusuf Sahillioğlu[2]

Accepted: 15 May 2023

Abstract
In this paper, a novel data-centric approach is proposed for solving the 3D mesh segmentation problem. The method uses node2vec, a semi-supervised learning algorithm, to create vector embedding representations for each node in a 3D mesh graph. This makes the mesh data more compact and easier to process which is important for reducing computation costs. K-Means clustering is then used to cluster each node according to their node embedding information. This data-centric approach is more computationally efficient than other complex models such as CNN and RNN. The main contribution of this study is the development of a data-centric AI framework that combines node2vec embedding, machine learning, and deep learning techniques. The use of cosine similarity is also adapted to compare and evaluate the trained node embedding vectors with different hyperparameters. Additionally, a new algorithm is developed to determine the optimal cluster number using geodesic distance on the 3D mesh. Overall, this approach provides competitive results compared to existing mesh segmentation methods.

Keywords 3D mesh segmentation · Unsupervised learning · Embedding · Node2vec · K-Means · Geodesic distance

1 Introduction

A data-centric approach that is proposed for solving the problem of mesh segmentation is presented in this work. Unlike traditional methods, which rely on labeled data, the proposed algorithm uses unlabeled 3D triangular meshes. We will consider both a data-centric approach and fully unlabeled experiments. The algorithm is applied to 3D human body and object meshes, such as a goblet, guitar, alien, chair, and vase. In addition to the new approach to mesh segmentation, the work also introduces three new methods: geodesic inertia, an evaluation method for 3D mesh data, and an embedding quality evaluation method for finding better vector representations of 3D mesh data. The proposed solution is shown to be effective on 3D human body and object meshes.

Motivation This work takes a data-centric approach to AI, focusing on data preparation rather than complex models. Unlike model-centric AI, which aims to improve models by making them more complex, data-centric AI focuses on improving the raw data itself. This approach has several advantages, including reduced computation costs and improved energy efficiency. A recent study showed [38] that dimensionality reduction can lower the energy requirements of model training by up to 76% while maintaining performance. This approach is particularly effective when the number of data points is reduced. Most machine learning and deep learning techniques rely on labeled data, but in reality, most data are unlabeled [24]. AI pioneers LeCun et al. believe that unsupervised learning will be increasingly important in the long term, as it more closely resembles how humans and animals learn. They argue that we discover the world by observing, rather than by labeling objects. In light of this, the authors have developed an unsupervised algorithm that is more efficient at training time, thanks to its data-centric approach. In this study, we have focused on unsupervised learning, using a data-centric and unsupervised approach. They have applied the node2vec embedding algorithm to 3D mesh data, transforming it into a more compact and meaningful form. This reduces the curse of dimensionality and significantly lowers the computation cost, allowing the results to compete with models such as meshWalker [22] and meshCNN [14], which require significant GPU and computation power.

✉ Yusuf Sahillioğlu
ys@ceng.metu.edu.tr

Talya Tümer Sivri
tumer.talya@metu.edu.tr

1 Multimedia Informatics Department, METU, Ankara, Turkey

2 Computer Engineering Department, METU, Ankara, Turkey

2 Related work

A wide range of research contributed 3D mesh segmentation area. The graph embedding technique is vital in the 3D mesh segmentation we implemented in this work. We cover graph embedding techniques in this section in detail. Goyal and Ferrara et al. [12] divided graph embedding techniques into three categories which are factorization, random walk, and deep learning. Factorization-based techniques factorize a matrix representing the connections between nodes to generate the embedding. Locally linear embedding [33] is a method for unsupervised learning that computes neighborhood-preserving, low-dimensional embeddings of high-dimensional inputs, graph factorization [2], graph representation [6] which learns weighted graph vertex representations, and high-order proximity preserved embedding are different kinds of factorization-based techniques. Additionally, Luo et al. [25] provide a novel Cauchy graph embedding that preserves the similarity relationships of the original data in the embedded space by using a new objective.

The process in which randomly moving objects depart from their starting point is known as a random walk. Node centrality [29] and similarity [11] are some of the application areas for approximating graph properties. For instance, the authors applied the random walk method in order to feed the RNN model in [22]. DeepWalk [31] which uses truncated random walks, node2vec, HARP [8] which presents a method to enhance the solution and prevent local optima, and walklet [32] where explicit modeling and random walks are combined are algorithms for obtaining node representations in a graph. This method is scalable and preserves high-order proximity between nodes. node2vec [13] is a framework for developing continuous feature representations for nodes in a graph. It is an advanced version of DeepWalk controlling the path and weighted random behavior. Grover and Leskovec et al. developed BFS and DFS algorithms for controlling randomness rather than walking randomly. In [23] implemented a random walk strategy to segment 3D meshes.

Deep learning algorithms are being developed for graph embeddings and scene processing [1] as a growing research area. SDNE, DNGR, GCN, and VGAE are DL-based methods. SDNE [39] is a semi-supervised model that has layers with nonlinear functions for capturing nonlinearity while preserving the local and global structures. Cao et al. [7] developed a model for learning graph representations that creates a low-dimensional vector representation for each vertex by encoding the graph's structural information. GCN [20] is different from SDNE and DNGR algorithms. While inputs are the global neighborhood of each node which is expensive and non-optimal for large sparse graphs in SDNE and DNGR, the GCN model takes inputs via a graph convolutional operator, which solves the problems in other models. For example, Sever et al. [34] used GCN for a 3D mesh segmentation problem using sparse face labels of a 3D object. VGAE [21] is an unsupervised learning technique in which a variational autoencoder is applied to the graph. In [21], Kipf and Welling et al. demonstrated their architecture, GCN as an encoder, and an inner product as a decoder. Moreover, GCN is used to learn the higher-level dependencies between nodes from the input adjacency matrix. In [14], the authors developed a new mesh convolution layer. They used it in the segmentation problem. Lahav and Tal et al. [22] proposed a new algorithm for segmentation that uses random walks for exploring mesh and RNN, a deep learning algorithm for learning temporal sequences in order to train a supervised segmentation model.

Unsupervised learning is often used in the field of mesh segmentation. K-Means is applied for 3D mesh segmentation in [17]. Using geodesic distance and convexity was a strategy for decomposing object meshes into segmented parts developed by Katz and Tal [17]. Also, [18] applied K-Means and fuzzy C-Means algorithms to output direct clustering results. Sidi et al. [37] performed a descriptor space study, and using the results, they produced a spectral clustering for segmentation. In [26], the labeling is done by adding the energy term of unlabeled data to the conditional random fields. Shu et al. [35] developed scribble-based segmentation with weakly labeled objects. Additionally, Shu et al. [36] developed another algorithm using soft density peak clustering and semi-supervised learning. They proposed a network that significantly reduces the necessity for not preparing fully labeled 3D shapes. In [16], the authors developed a self-supervised algorithm that uses KNN graph and shape descriptor two autoencoders: classical and improved graph.

3 Methodology

In recent years, the field of deep learning has seen a growth in the development of algorithms for data representation techniques, such as embedding vectors. Embedding is a way of converting data into numerical form to make it easier to process and reduce its size, which can help prevent the challenge of working with high-dimensional data known as the curse of dimensionality. In this way, embedding extracts important information from high-dimensional data and creates a low-dimensional representation that retains enough information to be useful for solving a particular problem. Our proposed algorithm mainly focuses on the statement that an unsupervised algorithm works best for train time with the data-centric approach. Hence, we design an algorithm with node2vec and K-Means to find an answer to our research question. In this chapter, the details of our segmentation algorithm are presented. Our 3D mesh segmentation algorithm is divided into two parts, which are node embedding representation and node clustering.

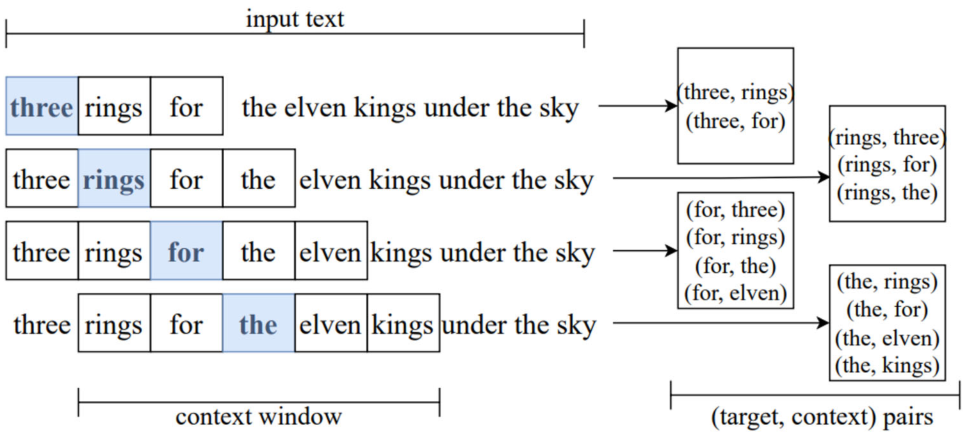

Fig. 1 An example of how to pair target and context data for skip-gram architecture. The passage is taken from the Lord of the Rings, the Fellowship of the Ring by J. R. R. Tolkien

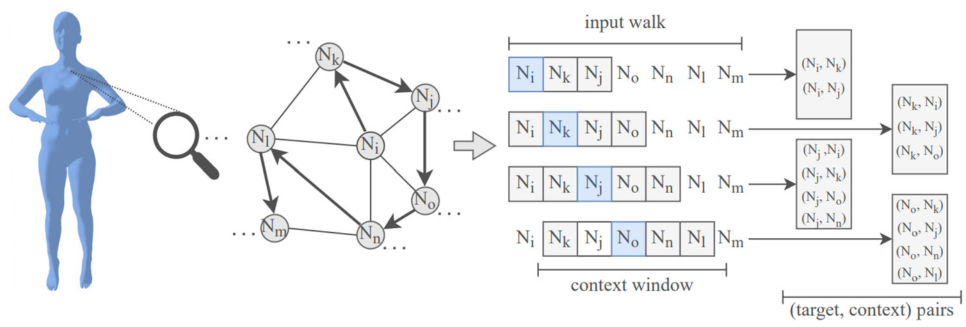

Fig. 2 A diagram of how skip-gram architecture is applied on 3D mesh data

3.1 Node embedding representation

Embedding representation is a way of using mathematical operations on powerful computation tools to extract similar information from objects or data features. In the first step of our 3D mesh segmentation algorithm, we map 3D mesh data to embedding vectors. The 3D mesh data structure creates a relationship between positions in 3D space and the connections between them. We use the node2vec [13] algorithm to capture the neighborhood information because it uses the connections between nodes in a biased random walk. node2vec is a semi-supervised framework that learns continuous vector representations for nodes in graphs, like the 3D meshes in our case. The algorithm maps nodes to a low-dimensional space that maximizes the likelihood of preserving the network neighborhoods of the nodes. The random walk with bias method effectively explores different neighborhoods and creates a flexible representation of a node's connected neighborhood. This method allows for more flexibility in exploring neighborhoods, which leads to broader representations that capture more comprehensive attributes of a node. It also improves upon previous methods that used more rigid conceptions of network neighborhoods.

The node2vec algorithm is based on the idea behind the word2vec/skip-gram [28] algorithm, which is used for word embeddings. In the word2vec algorithm, input and output vectors are one-hot encoded pairs of target and context words, obtained by sliding a window of context words around the target word; see Fig. 1. The algorithm uses a negative sampling strategy and a fully connected deep learning architecture with one hidden layer and one output layer to train the weights, which are then used as the word embedding vectors for each word in the vocabulary; see Fig. 3. In the node2vec algorithm, nodes are used instead of words, and biased random walks are used as input text. This allows the node2vec algorithm to capture the relevant context of the 3D mesh data.

In the node2vec algorithm, random walks are used to construct the input and output vectors, similar to how sentences are used in the word2vec algorithm. A random walk is a sequence of nodes that are randomly chosen from the nodes connected to the current node in the sequence. For example, in a sample graph from 3D mesh data (see Fig. 2), a random walk starting from node N_i might be N_i, N_k, N_j, N_o, N_n, N_l, N_m. In the node2vec algorithm, bias is added to the random walks to make them follow a breadth-first search (BFS) or depth-first search (DFS) strategy. In BFS, only nodes directly adjacent to the source node are allowed in the neighborhood, while in DFS, the neighborhood is formed of nodes that are progressively further from the source node.

After exploring the 3D mesh data using biased random walks, the node2vec algorithm begins the training process. This involves training a shallow, deep learning network with one-hot encoded target and context node pairs, (N_i, N_j), (N_i, N_k), where $N_i, N_j, N_k \in N$ and N is the set

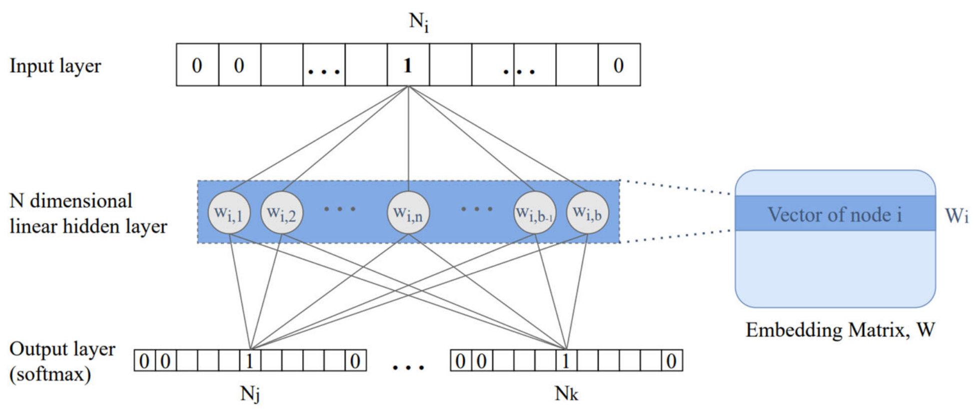

Fig. 3 Skipgram architecture. N_i is the target node and N_j and N_k are two example samples from context node paired with the target

of nodes in the 3D mesh data; see Fig. 3. The hidden layer is linear, and the output layer uses a softmax function to calculate probabilities for each pair. These context-paired nodes are also known as positive samples. In addition, the node2vec algorithm uses a negative sampling method to optimize the training process, similar to the word2vec algorithm. Negative samples are selected using a unigram distribution, which means that the most frequent nodes are not used in the training and their weights are not updated. At the end of the training, the neural network produces a weight matrix, also called an embedding matrix, W, where each W_i corresponds to a node N_i, and the matrix has dimensions $a \times b$, where a is the number of nodes and b is the embedding size hyperparameter.

3.2 Measuring embedding vector quality

The goal of this implementation is to measure the quality of node embedding vectors in a neural network. To do this, we use a cosine similarity metric to compare the similarity of vectors representing the neighbor nodes in the network. This allows us to evaluate the effectiveness of the node embedding vectors in representing the data, as well as the quality of the training process for the neural network. By finding the cosine similarity between the vectors of neighboring nodes, it can be determined how well the vectors capture the relationships between the nodes and thus how well the neural network has been trained.

$$\text{Cosine similarity} = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|}, \text{ where A, B} \in \mathbb{R}^n \tag{1}$$

This implementation uses cosine similarity to evaluate the quality of node embedding vectors in a neural network. In order to do this, the authors first detect and pair the unique neighboring nodes in the network. Next, they calculate the cosine similarity (Eq. 1) between each pair of nodes and append the results to a list. Finally, they use the mean value of the cosine similarity scores to evaluate the quality of the node embedding vectors. If the mean is closer to 1, the authors consider the embedding vectors to be well represented. If the mean is closer to 0 or -1, they consider the vectors to be poorly represented.

3.3 Node clustering

In the second part of our 3D mesh segmentation algorithm, we use a clustering algorithm on the node embedding. We use the K-Means [27] algorithm, which is a centroid-based method that iteratively partitions the dataset into k non-overlapping clusters. Selecting the initial centroids is important for the convergence of the K-Means algorithm, and there are two common methods for initializing them: random initialization and K-Means++ initialization. Random initialization involves running the algorithm multiple times with different random centroids and selecting the best solution, while K-Means++ [4] initialization reduces the risk of suboptimal solutions and improves convergence speed by choosing centroids that are far from each other.

3.3.1 Choosing the cluster number

The 3D mesh data are segmented into parts and the borders of these segments are joint together. We use the elbow method, which is a way of evaluating the within-cluster sum of squares (WCSS, Eq. 2) to determine the optimal number of clusters for our problem. The K-Means algorithm is used to determine the inertia value when the optimal centroid locations have been found. WCSS evaluates the sum of the squared distance for each sample in a cluster to its centroid. The highest inertia value is obtained when there is only one cluster, and as the number of clusters increases, the inertia value begins to decrease. This is because the distance between the data points and their cluster centers gets smaller as the number of centers increases. The optimal number of clusters is determined by looking for the point at which the rate of decrease in inertia slows down, which is often called the elbow point.

$$\text{inertia} = \sum_{k=1}^{k_c} \sum_{i=1}^{N} (x_i - C_k)^2 \tag{2}$$

3.4 Geodesic distance inertia method

The computation of shortest paths and geodesic distance on curved domains is important for various applications in fields such as digital geometry, scientific computing, computer graphics, and computer vision. These tasks are more challenging than calculating Euclidean distance because of the influence of curvature on shortest path behavior and the potential imprecision of the domain representation. Geodesic distance is calculated by taking into account the shape of the dataset, whereas Euclidean distance ignores it. This makes geodesic distance useful for evaluating the effectiveness of clustering algorithms that use Euclidean distance on raw 3D data.

To evaluate geodesic inertia, we first determine the centroid nodes for each cluster. To do this, we use the cluster centroids on node embedding vectors, as we are experimenting with clustering on node embeddings. Since the indexes of the nodes are the same as the indexes of the corresponding embeddings, we can easily find the cluster centroid node by applying the minimum distance on the 3D axes. Next, we separate the nodes according to their cluster. Then, we evaluate the geodesic distances between the cluster members and the centroid nodes using the shortest paths between them. Since inertia is the sum of squared distances, we square the evaluated geodesic distances and sum the squared distances of all cluster members. After performing these steps for all clusters, we compute the final inertia score by summing the scores for all clusters. The final score gives the geodesic inertia value for the k clusters.

4 Experiments and results

In this section, we will explain the steps we experimented with and show the results of 3D mesh segmentation experiments. We processed the 3D mesh segmentation algorithm experiments in two distinct stages. Firstly, we trained the nodes in 3D meshes with different hyperparameters to find the optimal hyperparameter values according to embedding representation quality using cosine similarity between neighboring pairs. Secondly, we implemented a clustering algorithm on node embeddings in order to cluster each node. While experimenting with clustering, we used the elbow method to find the optimal k value and compared the obtained results with the other elbow method, which uses the inertia values we developed and computed with geodesic distance instead of Euclidean distance.

4.1 Dataset

In 3D mesh segmentation, there are widely used datasets with many different shapes. In this work, we experimented with and tested our new segmentation algorithm on different datasets. The first one is FAUST [5] dataset. The dataset has 100 scans of 10 distinct humans with 10 different poses, which are high-resolution triangulated meshes. The FAUST dataset is used because it has high-resolution meshes, and our solution to the 3D mesh segmentation problem is related to training embedding for each node. The second dataset is SCAPE [3], which has 71 different poses and high-resolution mesh objects. Another dataset is that we use three classes of COSEG [40] dataset, which are guitar, vase, goblet, chair, and alien. We use Plotly, which is a Python library for data visualization (Figs. 4 and 5).

4.2 Experiments and evaluations on node embedding

The first part of the experiments consists of obtaining embedding for 3D mesh data. PyTorch Geometric [10] library, which leans on Pytorch [30], was used for training graph-based learning models and processing 3D graph data such as mesh and point clouds. We created all of our embeddings from scratch for our experiments. We used NVIDIA GeForce GTX 1650 Ti GPU while training the node embeddings for a single object.

4.2.1 Hyperparameter selection

Hyperparameter tuning is a process used in machine learning and deep learning to improve the performance of a model by adjusting the settings of its hyperparameters. In this case, a linear neural network with one hidden layer was trained to compute the weights of the neural network for node embedding representation vectors, using biased random walks and a determined context size. The node embeddings were trained using early stopping, which monitors the change in the loss value between iterations and stops the training if the change falls below a certain threshold. In this case, the threshold was set at 1% and the "patience" parameter, which determines how many times the training is allowed to continue after the threshold is reached, was set at 4. Sparse Adam optimizer [19] is used for all the training. Context size, embedding dimension, walk length, walks per node, learning rate, p, q, and batch size are shown in detail in Eq. (3), where CS is context size, ED is embedding dimension, WL is walk length, $W_{\mathrm{p}}N$

Table 1 Embedding representation quality results according to cosine similarity values' minimum and maximum values for all 3D mesh data types

3D object	Min	Max
FAUST	0.36	0.98
SCAPE	0.68	0.99
ALIEN	0.32	0.98
VASE	0.07	0.94
GUITAR	0.13	0.96
CHAIR	0.13	0.98
GOBLET	0.06	0.97

Table 3 Optimal cluster values according to elbow method for each 3D mesh object

3D object	*k*
FAUST	14
SCAPE	11
ALIEN	11
VASE	5
GUITAR	7
CHAIR	11
GOBLET	3

Table 2 Optimal hyperparameters according to cosine similarity quality measurement experimented with chosen hyperparameter value sets

3D object	CS	ED	WL	WpN	LR	*P*	*Q*
FAUST	40	32	100	10	0.01	1	1
SCAPE	20	10	100	30	0.005	0.3	0.4
ALIEN	20	16	25	15	0.005	0.7	0.9
VASE	20	16	30	25	0.005	1.2	0.1
GUITAR	18	12	30	25	0.005	0.2	0.3
CHAIR	24	16	30	25	0.005	1.5	0.9
GOBLET	20	12	25	10	0.005	0.4	0.2

CS context size, *ED* embedding dimension, *WL* walk length, *WpN* walks per node, *LR* learning rate

is walks per node, LR is learning rate, P is return parameter, Q is the in–out parameter, and BS is the batch size.

$$\begin{aligned}
\mathrm{CS} &= \{8, 10, 12, 15, 18, 20, 24, 40\} \\
\mathrm{ED} &= \{10, 12, 16, 32, 64, 72, 96, 128, 192, 256\} \\
\mathrm{WL} &= \{15, 20, 25, 30, 40, 60, 80, 100, 120, 150, 200\} \\
W_{\mathrm{p}}N &= \{2, 5, 10, 15, 20, 25, 30, 40\} \\
\mathrm{LR} &= \{0.0005, 0.005, 0.001, 0.005, 0.002, 0.01, 0.02\} \\
P &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5\} \\
Q &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5\} \\
\mathrm{BS} &= \{8, 16, 32\}
\end{aligned} \tag{3}$$

In experiments, we randomly selected hyperparameters to generate more than 70 embedding vectors for each 3D mesh data type. Using the cosine similarity algorithm, the results were impressive; see Table 1, with values ranging from 0.36 to 0.98 for the FAUST dataset, etc. These results indicate that the node embedding vector representations are qualified. The following experiments were continued with the parameters (Table 2) that gave the optimal results for each dataset and found that different datasets required different hyperparameters. The batch size was determined to be 8 based on the results of the cosine similarity score.

4.2.2 Experiments and evaluations on node clustering

In the second part of our 3D mesh segmentation algorithm, we used node clustering to group similar nodes together. This involved using the node embedding representations, which we had previously trained and evaluated. We tested different cluster numbers using the K-Means algorithm with both random and K-Means++ initialization. The optimal number of clusters for each dataset was determined using the elbow method; see Table 3.

It is described that a data-centric 3D mesh segmentation algorithm that uses K-Means++ initialization and cosine similarity and the elbow method to find the optimal number of clusters. The results of the algorithm show that it can successfully segment distinct parts of objects, as illustrated in Fig. 6. The algorithm was applied to two human-shaped datasets, FAUST and SCAPE, and achieved good results in terms of symmetry and meaningfully for the FAUST dataset, but not as good results for the SCAPE dataset. Additionally, it is adaptable to apply obtained clusters for other members of the SCAPE and FAUST datasets since they have the same node and face properties. We experimented with 5 objects: a goblet, a vase, a guitar, a chair, and an alien. It was observed that the model was able to segment the objects into their component parts well, except for some small issues in specific areas; see Fig. 6. In general, the model was able to segment objects with different structures, like the guitar and the chair, and was also able to distinguish texture differences. The alien, which has many extremities and unusual eyes, was also well segmented by the model. To sum up, our data-centric 3D mesh segmentation model works well for different kinds of objects where their structures are different.

The segmentation model experimented with randomly initialized centroids to see how well it performed. We then compared its results to those of a model using the K-Means++ initialization method. We evaluated the performance of the models using two numeric results: the Euclidean distance between the initial and final centroid locations; see Table 4, and the inertia values calculated using the elbow method; see Table 5. It was found that the K-Means++ method generally produced better results than the random initialization method,

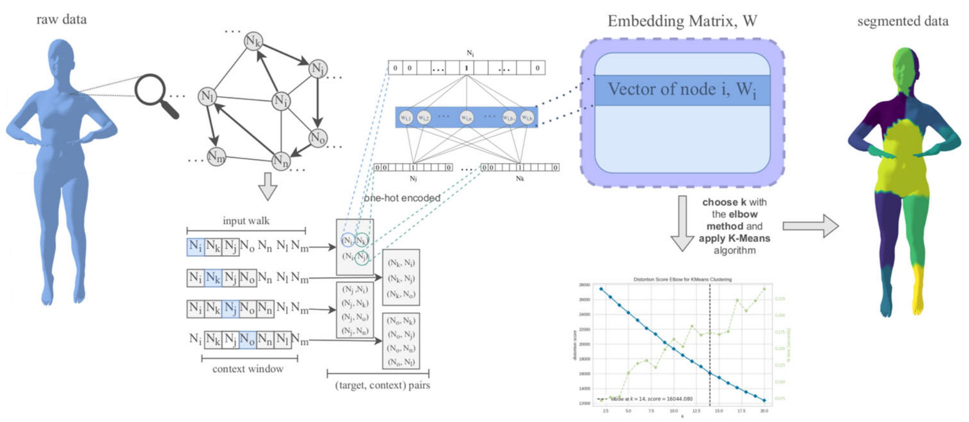
raw data
Embedding Matrix, W
segmented data
Vector of node i, W_i
one-hot encoded
input walk
context window
(target, context) pairs
choose k with the elbow method and apply K-Means algorithm
Distortion Score Elbow for KMeans Clustering
elbow at k = 14, score = 16044.080

Fig. 4 Our 3D mesh segmentation algorithm illustration

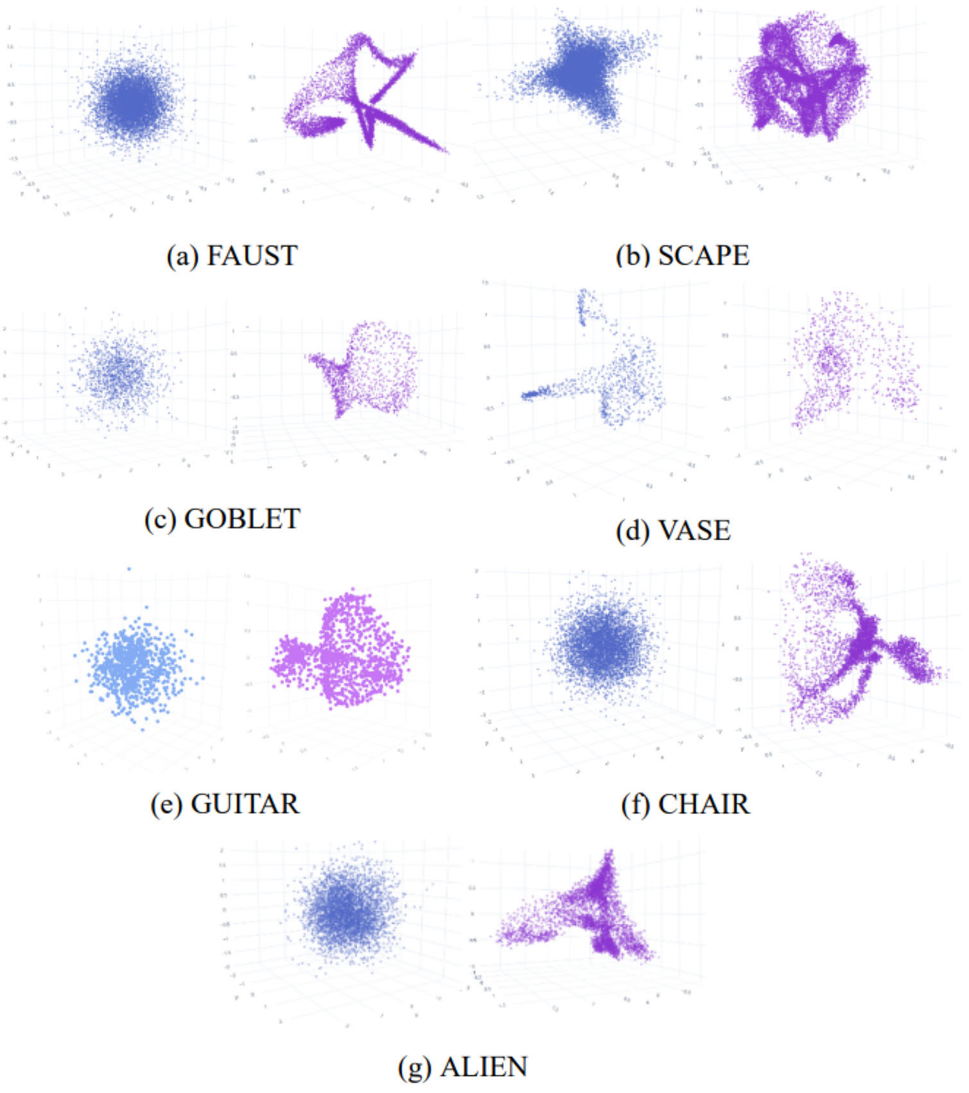

Fig. 5 Examples of ill- and well-trained 3D mesh data node embedding vector representations on 3D scatter plot. Blue (left side) ones show ill-represented examples, and purple (right side) ones show well-represented examples

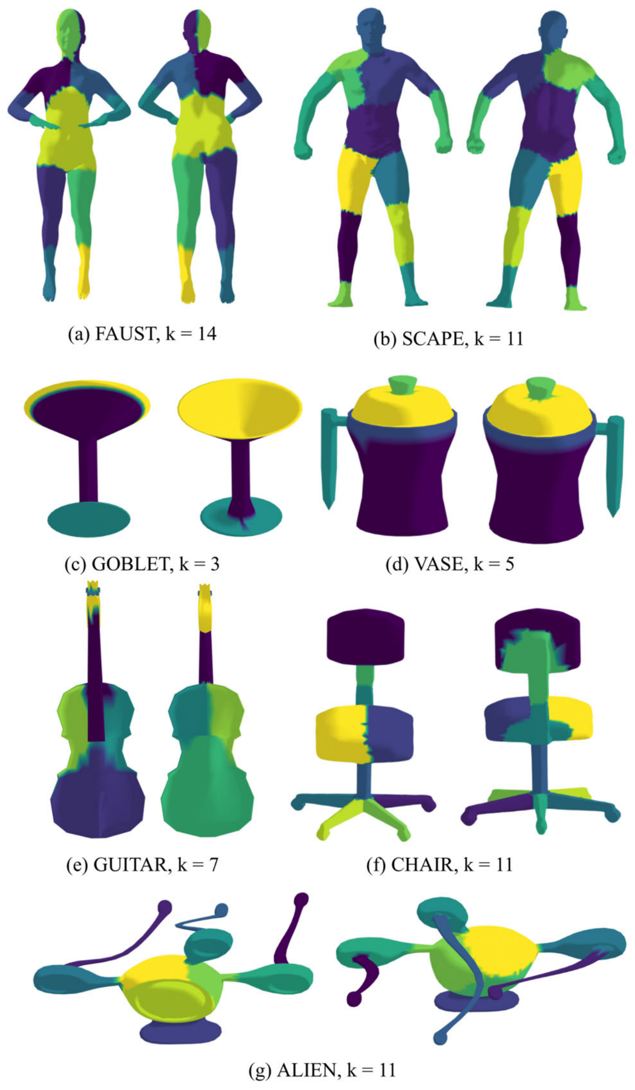

Fig. 6 Visual results with *k* values suggested by elbow method using Eucldiean distance

with smaller distances and inertia values. These calculations were performed on the vector representations of 3D mesh data. The results of using the K-Means++ and random initialization algorithms for K-Means clustering are presented in Fig. 7. It is clear that the way in which the initial centroids are placed plays a crucial role in the quality of the resulting clusters, as well as the number of clusters and the quality of the data. In all of the 3D mesh data objects, K-Means++ produces better results in terms of symmetry and the ability to separate distinct parts of the objects, such as the handle of a vase or the feet of a chair. The problems with the random initialization can be caused by the stucking the local optima. Overall, K-Means++ initialization produces better results than random initialization because the initial centroids are distributed more evenly and are less likely to get stuck in local optima.

Table 4 Euclidean distance averages between initial centroids and final centroids for random and K-Means++ initialization methods

3D object	Random init.	K-Means++ init.
FAUST	2.80	2.78
SCAPE	2.09	1.75
ALIEN	2.13	2.15
CHAIR	2.36	2.15
GUITAR	2.29	2.46
VASE	2.23	2.50
GOBLET	2.48	2.32

Table 5 Inertia values at *k* which is decided by elbow method for random and K-Means++ initialization methods

3D object	Random init.	K-Means++ init.
FAUST	16,798	16,044
SCAPE	11,324	10,906
ALIEN	6041	5975
CHAIR	7598	7391
GUITAR	2020	1968
VASE	1939	1924
GOBLET	3231	3109

Our data-centric AI approach has proven its success in performance by demonstrating the difference in computational time. For instance, our 3D mesh segmentation algorithm performs at high speed, unlike the supervised learning model for 3D mesh segmentation [22]. The timing of the segmentation algorithm including hyperparameter selection is outlined in seconds in Table 6. In [22], which uses random walk and RNN with labeled data, training time last approximately 12 h for the segmentation of the human body with a GTX 1080 TI graphics card. The model uses SHREC11 dataset and COSEG dataset for training the model. SHREC11 dataset's women and men classes have 252 nodes and 500 faces. Although the dataset has much less node and face data according to SCAPE which has 12,500 nodes and 24,998 faces and FAUST which has 6890 nodes and 13,776 faces, the training time of MeshWalker model's is higher than ours. Because they reproduced the data using random walks and they feed their deep learning model data which is a model-centric approach. Unlike the Meshwalker model, we extracted the information from a 3D mesh data which helps to lower the dimension with node2vec algorithm and trained a single-layer deep learning model.

4.2.3 Geodesic inertia

The inertia value is calculated to determine the best number of clusters where the experiments were made on 3D mesh data

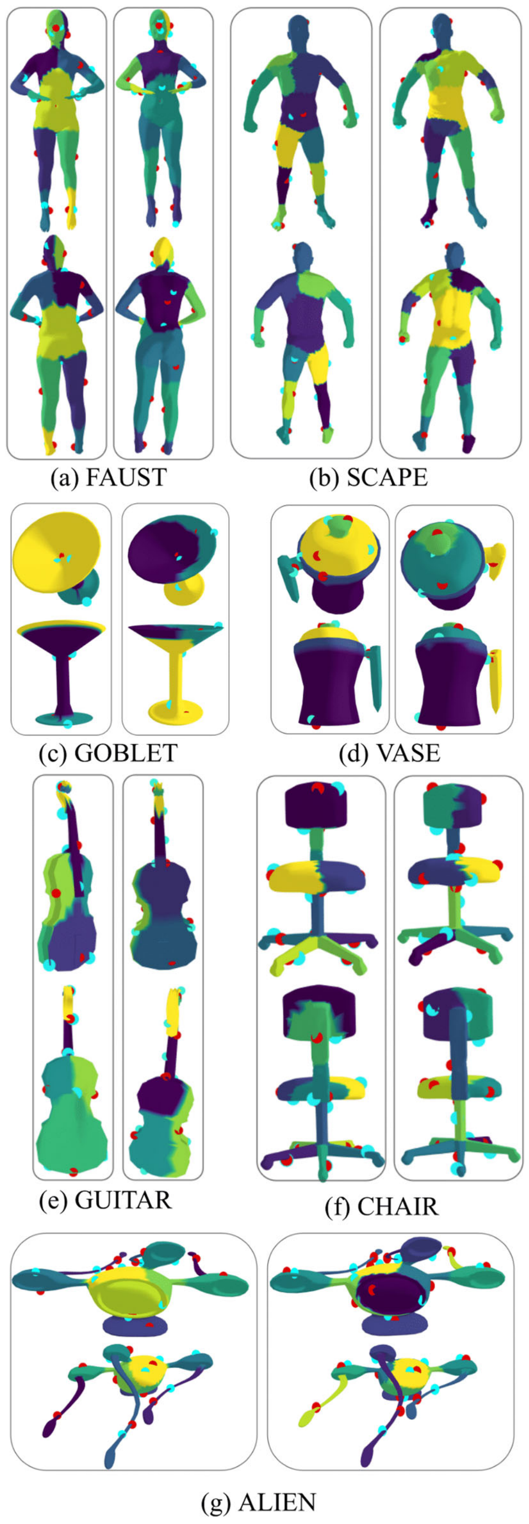

Fig. 7 Visual results with k values suggested by elbow method and initial and final centroid positions. The left part of the each examples illustrates the results that produced by the K-Means++ initialization and the right part illustrates the random initialization. Blue dots denotes the initial centroid positions and red dots denotes final centroids positions

Table 6 3D mesh segmentation train and clustering times in seconds

3D object	NT	NC	Total	Node num.
FAUST	91.26	0.19	91.45	6890
SCAPE	97.64	0.16	97.80	12,500
ALIEN	24.20	0.08	24.28	4060
CHAIR	21.37	0.10	21.47	5000
GUITAR	13.60	0.04	13.67	1152
VASE	8.65	0.02	13.62	874
GOBLET	7.23	0.03	13.63	1202

NT node training, *NC* node clustering

Table 7 Optimal cluster numbers according to metrics, Euclidean and geodesic distance

3D object	Euclidean, k	Geodesic, k
FAUST	14	7
SCAPE	11	7
ALIEN	11	9
CHAIR	11	4
GUITAR	7	4
VASE	5	5
GOBLET	3	3

between 2 and 20 clusters. Instead of using the Euclidean distance to calculate the inertia value, a new method called geodesic inertia is used, which takes into account the original structure of the data. The geodesic inertia method was tested on different datasets and was found to return different results than the Euclidean inertia method; see Table 7. While making the experiments, we used the geodesic library [15]. In general, the geodesic inertia method was found to produce more general and symmetrical segmentation results. It was also easier to identify the optimal number of clusters using the geodesic inertia method with the elbow method (Figs. 8, 9 and 10).

4.3 Comparisons

Our proposed model has been compared with other state-of-the-art studies using the COSEG dataset by choosing k according to the ground truth labels. Evaluations have been conducted under three big categories of COSEG: vase, alien, and chair datasets. For comparing our proposed model, no specific training or tuning has been applied based on the ground truth labels. The only adjustment in our algorithm made was choosing the number of clusters in accordance with the ground truth labels for comparison with the other models. Tables 8 and 9, respectively, show training time and accuracy results based on the ground truth labels. It has been observed that our algorithm is the fastest among the exist-

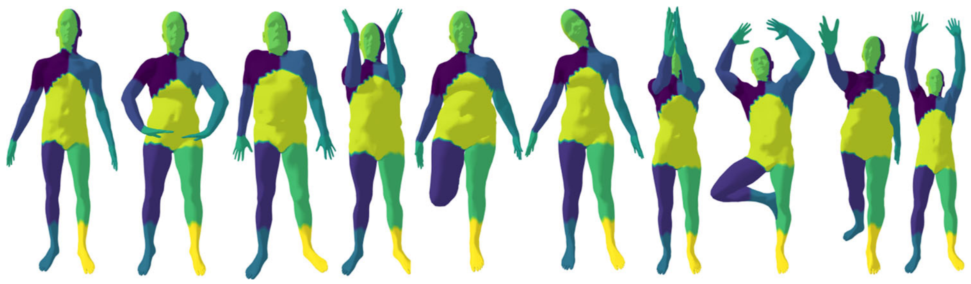
Fig. 8 Our 3D mesh segmentation results on FAUST dataset for different people and poses, $k = 14$

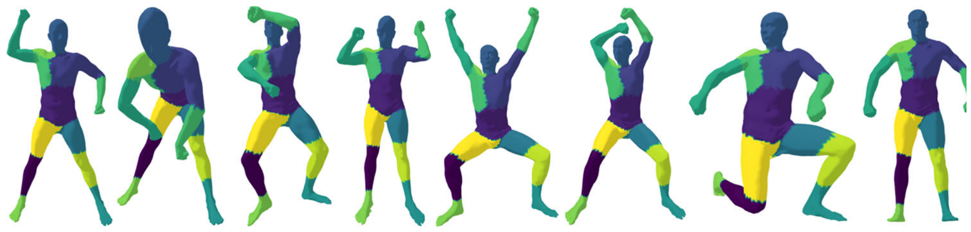
Fig. 9 Our 3D mesh segmentation results on SCAPE dataset showed in 8 unique poses with $k = 11$

ing algorithms; see Table 8. In MeshWalker [22], random walk and RNN are used with labeled data. Though it utilizes with fewer nodes and faces, its training time is higher due to data reproduction via random walks and a model-centric approach. In contrast, our model employs node2vec to extract information from 3D mesh data, reducing dimensions and training a single-layer deep learning model more efficiently. Moreover, our algorithm is faster than SCMS-Net [16] due to differences in architectural structure and processing of a single object at a time. These results show training time.

When evaluating the accuracies, it was noted that applying the hyperparameters generated for specific objects to the entire dataset using ground truth labels did not yield results as successful as those of other models. Furthermore, as the difference between the optimal cluster count identified by our algorithm and the ground truth count increased, a decline in accuracy was observed. This situation is particularly noticeable for the alien dataset. Our algorithm finds the optimal cluster number values (k) to be 9 and 11 depending on the distance metric used, while the ground truth value is 4. Additionally, our algorithm learns the 3D mesh structure using biased random walks, resulting in discrepancies between the learned structure and ground truth labels. For example, in the vase object depicted in Fig. 11, our algorithm segments the vase into interior and exterior parts, whereas the other model distinguishes it based on the base and top sections. Considering all these factors, it has been observed that the segmentation accuracy results, which rely on the ground truth, are lower for our algorithm compared to other models; see Table 9. We also compared qualitative the results with the SCMS-Net. It was observed that the intersection of segmented parts was more accurate in our case for the examples in Fig. 11.

Upon examining the accuracy results in greater detail, we observed that objects similar to the ones for which we tuned the hyperparameters to find the optimal embeddings yielded better outcomes; see Table 9. While determining the optimal embeddings, we performed hyperparameter tuning on a single object and applied the resulting parameters to all objects. At this point, it has been observed that objects with similar structures exhibit better segmentation accuracies, as shown in Table 9. For instance, the hyperparameters that yield the optimal embeddings have been selected based on the chair structure featuring a single connection between the legs and the seating area. These results have been obtained by selecting a subset of 25–45 similar objects, depending on the size of the dataset.

5 Conclusions and future work

In this work, a novel unsupervised method for 3D mesh segmentation is presented. The method maps 3D mesh data to vector representations using the node2vec algorithm, which allows for control over the discovery of mesh structure through the use of biased random walks. This data-centric approach does not require ground truth labels and is computationally efficient, making it suitable for use on large datasets. The method was tested on different datasets, including human-shaped and non-human-shaped objects, and was found to produce similarity scores between 0.94 and 0.99. The K-Means clustering algorithm was applied to the resulting vector representations, and a new method for determining the optimal number of clusters was developed using geodesic distance. This method was found to focus on more general structures than classical methods and was easier to interpret. Overall, this work presents a simple and effective approach to 3D mesh segmentation that is applicable to a wide range of datasets. At this point, when we look at computation cost, today's GPUs consume much power and contradict the Green

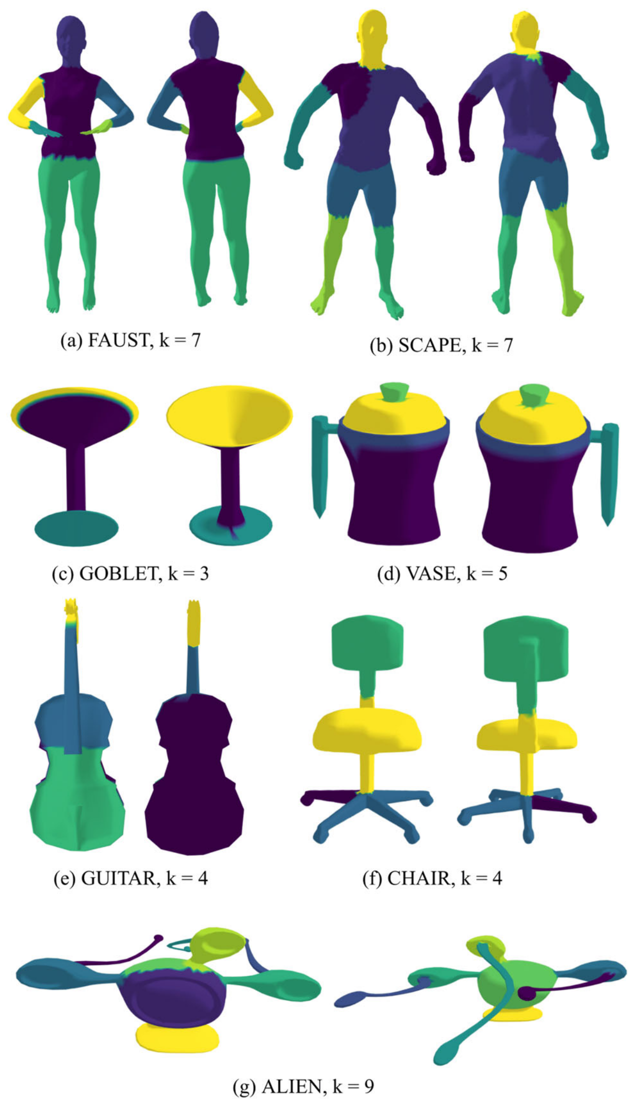

Fig. 10 Visual results with *k* values suggested by elbow method using geodesic distance

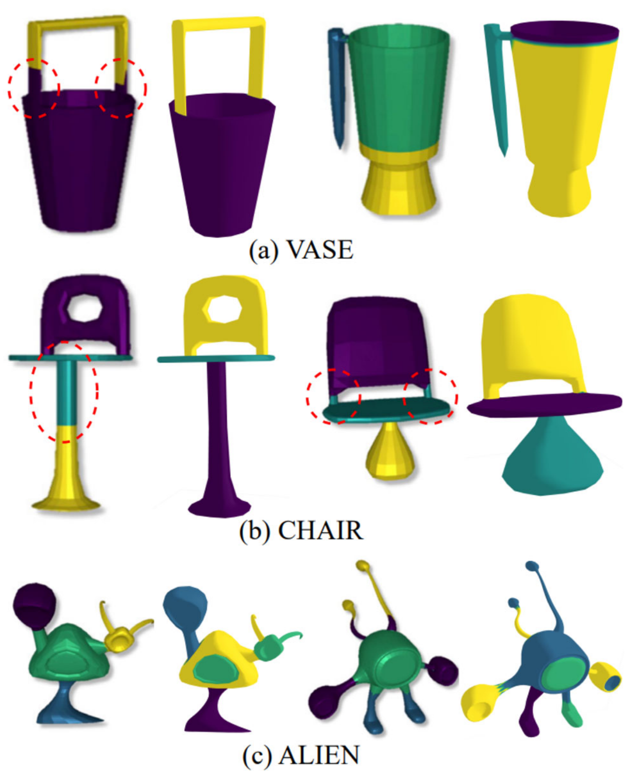

Fig. 11 Qualitative results between ours (right ones) and SCMS-Net [16] (left ones) results

Table 8 Average training time comparison table

Model	Time	Type	GPU
Ours	38 s	U	GTX 1650 TI
SCMS-Net [16]	145 s	U	RTX 2080 TI
MeshWalker [22]	12 h	S	GTX 1080 TI

U Unsupervised, *S* Supervised

AI concept. We benefited from the data-centric approach's computation cost and simple model outputs, which AI pioneers also emphasized.

The need for more computational power and the limitations of current hardware mean that efforts to optimize the training time of AI models will continue to be important. In particular, the data-centric approach to AI will likely become increasingly critical in the future. To improve the performance of these models, researchers can work on optimizing the node2vec algorithm or develop new embedding algorithms. Additionally, using preprocessing techniques on the data can help make the distinctions between clusters sharper, and using autoencoders to transfer the data to a latent space can help reduce the dimensionality of the data while still preserving important information. Furthermore, classifying objects with similar structures in the dataset and determining the hyperparameters that provide optimal embeddings for each class, followed by applying them to their respective classes, could potentially improve the overall results. These approaches can be applied to a variety of datasets, including point clouds and different classes of data.

Table 9 Comparison between our results tested with ground truth labels and other models' including unsupervised and supervised

Model	Vase	Chair	Alien	Type
Ours (subset)	0.769	0.901	0.625	U
Ours	0.723	0.748	0.562	U
SCMS-Net [16]	0.874	0.918	0.821	U
Sidi et al. [37]	0.69	0.80	–	U
Wu et al. [41]	0.87	0.90	0.75	U
SCMS-Net [16]	0.934	0.967	0.961	S
MeshWalker [22]	0.987	0.996	0.991	S
MeshCNN [14]	0.923	0.929	0.962	S
Laplacian2Mesh [9]	0.945	0.965	0.950	S

Subset refers to a set with a similar structure with a single object for which hyperparameter tuning has been performed to obtain the optimal embedding

U Unsupervised, *S* Supervised

Acknowledgements This work was supported by TUBITAK under the project EEEAG-119E572.

Data availability Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

1. Abbasi, A., Kalkan, S., Sahillioğlu, Y.: Deep 3D semantic scene extrapolation. Vis. Comput. **35**, 271–279 (2019)
2. Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V., Smola, A. J.: Distributed large-scale natural graph factorization. In: Proceedings of the 22nd International Conference on World Wide Web, WWW '13, pp. 37–48, New York, NY, USA. Association for Computing Machinery (2013)
3. Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., Davis, J.: Scape: shape completion and animation of people, vol. 24 (2005)
4. Arthur, D., Vassilvitskii, S.: K-means++: the advantages of careful seeding. vol. 07, 09-January 2007 (2007)
5. Bogo, F., Romero, J., Loper, M., Black, M. J.: FAUST: dataset and evaluation for 3D mesh registration. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2014)
6. Cao, S., Lu, W., Xu, Q.: Grarep: learning graph representations with global structural information. CIKM '15, pp. 891–900, New York, NY, USA. Association for Computing Machinery (2015)
7. Cao, S., Lu, W., Xu, Q.: Deep neural networks for learning graph representations. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, issue 1 (Feb. 2016)
8. Chen, H., Perozzi, B., Hu, Y., Skiena, S.: HARP: hierarchical representation learning for networks. CoRR, arXiv:1706.07845 (2017)
9. Dong, Q., Wang, Z., Gao, J., Chen, S., Shu, Z., Xin, S.: Laplacian2mesh: Laplacian-based mesh understanding. IEEE Trans. Vis. Comput. Gr. (2022). https://doi.org/10.1109/TVCG.2023.3259044
10. Fey, M., You, J., Ying, R., Li, G., Sunil, J., Lenssen, J. E., Bahtchevanov, I., Leskovec, J.: Pyg. https://www.pyg.org/
11. Fouss, F., Pirotte, A., Renders, J.-M., Saerens, M.: Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. IEEE Trans. Knowl. Data Eng. **19**(3), 355–369 (2007)
12. Goyal, P., Ferrara, E.: Graph embedding techniques, applications, and performance: a survey. Knowl. Based Syst. **151**, 78 (2018)
13. Grover, A., Leskovec, J.: Node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pp. 855–864, New York, NY, USA. Association for Computing Machinery (2016)
14. Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., Cohen-Or, D.: Meshcnn: a network with an edge. ACM Trans. Gr. **38**, 1–12 (2019)
15. Hogg, M.: Pygeodesic. https://pypi.org/project/pygeodesic/ (May 2021)
16. Jiao, X., Chen, Y., Yang, X.: SCMS-Net: self-supervised clustering-based 3D meshes segmentation network. Comput. Aided Des. **160**, 103512 (2023)
17. Katz, S., Tal, A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. ACM Trans. Gr. (TOG) **22**, 954–961 (2003)
18. Khattab, D., Ebeid, H. M., Hussein, A. S., Tolba, M. F. 3d mesh segmentation based on unsupervised clustering. Adv. Intell. Syst. Comput. 533, 598–607 (2017)
19. Kingma, D. P., Ba, J.: Adam: a method for stochastic optimization (2014)
20. Kipf, T. N., Welling, M.: Semi-supervised classification with graph convolutional networks. CoRR, arXiv:1609.02907 (2016)
21. Kipf, T. N., Welling, M.: Variational graph auto-encoders, (2016)
22. Lahav, A., Tal, A.: Meshwalker: deep mesh understanding by random walks. ACM Trans. Gr. **39**, 1–13 (2020)
23. Lai, Y. K., Hu, S. M., Martin, R. R., Rosin, P. L.: Fast mesh segmentation using random walks. (2008)
24. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
25. Luo, D., Ding, C., Nie, F., Huang, H.: Cauchy graph embedding. In: Proceedings of the 28th International Conference on Machine Learning, ICML 2011, pp. 553–560. Cited by: 87 (2011)
26. Lv, J., Chen, X., Huangy, J., Bao, H.: Semi-supervised mesh segmentation and labeling. vol. 31, pp. 2241–2248 (2012)
27. MacQueen, J. B.: K-means and analysis of multivariate observations. In: 5th Berkeley Symposium on Mathematical Statistics and Probability 1967, vol. 1, pp. 281–297 (1967)
28. Mikolov,T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013)
29. Newman, M.J.: A measure of betweenness centrality based on random walks. Soc. Netw. **27**(1), 39–54 (2005)
30. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In :NIPS 2017 Workshop on Autodiff (2017)
31. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, page 701–710, New York, NY, USA. Association for Computing Machinery (2014)
32. Perozzi, B., Kulkarni, V., Skiena, S.: Walklets: multiscale graph embeddings for interpretable network classification. CoRR, arXiv:1605.02115 (2016)
33. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science **290**(5500), 2323–2326 (2000)

34. Sever, O. I.: Mesh segmentation from sparse face labels using graph convolutional neural networks. Master's thesis, Middle East Technical University (2020)
35. Shu, Z., Shen, X., Xin, S., Chang, Q., Feng, J., Kavan, L., Liu, L.: Scribble-based 3D shape segmentation via weakly-supervised learning. IEEE Trans. Vis. Comput. Gr. **26**, 2671 (2020)
36. 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. Comput. Aided Des. **145**, 103181 (2022)
37. Sidi, O., Kleiman, Y., Cohen-Or, D., van Kaick, O., Zhang, H.: Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. ACM Trans. Gr. **30** (2011)
38. Verdecchia, R., Cruz, L., Sallou, J., Lin, M., Wickenden, J., Hotellier, E.: Data-centric green ai an exploratory empirical study. In: 2022 International Conference on ICT for Sustainability (ICT4S), pp. 35–45 (2022)
39. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pp. 1225–1234, New York, NY, USA. Association for Computing Machinery (2016)
40. Wang, Y., Asafi, S., Kaick, O. V., Zhang, H., Cohen-Or, D., Chen, B.: Active co-analysis of a set of shapes. vol. 31 (2012)
41. Wu, Z., Wang, Y., Shou, R., Chen, B., Liu, X.: Unsupervised co-segmentation of 3D shapes via affinity aggregation spectral clustering. Comput. Gr. **37**(6), 628–637 (2013)

Talya Tümer Sivri received her BS degree from the Computer Engineering Department of Middle East Technical University, Turkey, and her MS degree from the Multimedia Informatics Department of Middle East Technical University, Turkey. She is now working as a senior engineer in TUSAS, Turkey.

Yusuf Sahillioğlu is an associate professor in the Department of Computer Engineering at Middle East Technical University. He worked as a postdoctoral research in University of Pennsylvania. His research interests include digital geometry processing and computer graphics. He has a PhD in computer science from Koç University, Turkey. Contact him at ys@ceng.metu.edu.tr or visit http://www.ceng.metu.edu.tr/ys/.