A DATA-CENTRIC UNSUPERVISED 3D MESH SEGMENTATION METHOD

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

TALYA TÜMER-SİVRİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF MODELLING AND SIMULATION

DECEMBER 2022

A DATA-CENTRIC UNSUPERVISED 3D MESH SEGMENTATION METHOD

submitted by **TALYA TÜMER-SİVRİ** in partial fulfillment of the requirements for the degree of **Master of Science in Modelling and Simulation Department, Middle East Technical University** by,

Prof. Dr. Banu Günel Kılıç
Dean, **Graduate School of Informatics, METU**

Assoc. Prof. Dr. Elif Sürer
Head of Department, **Modelling and Simulation, METU**

Assoc. Prof. Dr. Yusuf Sahillioğlu
Supervisor, **Computer Engineering, METU**

Examining Committee Members:

Assoc. Prof. Dr. Elif Sürer
Modelling and Simulation, METU

Assoc. Prof. Dr. Yusuf Sahillioğlu
Computer Engineering, METU

Assist. Prof. Dr. Cemil Zalluhoğlu
Computer Engineering, Hacettepe University

Date: 02.12.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Talya TÜMER-SİVRİ

Signature :

ABSTRACT

A DATA-CENTRIC UNSUPERVISED 3D MESH SEGMENTATION METHOD

TÜMER-SİVRİ, Talya

M.S., Department of Modelling and Simulation

Supervisor: Assoc. Prof. Dr. Yusuf Sahillioğlu

December 2022, 52 pages

Modeling, texture mapping, shape compression, simplification, and skeleton extracting are popular and essential topics in mesh segmentation applications. As it serves various purposes in computer science, the mesh segmentation problem is an active and prominent research area. With the help of growing machine learning, deep learning algorithms, and computation power, different methods have been applied to solve the 3D mesh segmentation problem more efficiently. In this thesis, we solve the 3D mesh segmentation problem from a different perspective. We present a novel data-centric AI approach for the segmentation of 3D meshes. We used node2vec, a semi-supervised learning algorithm, to train vector embedding representation for each node in a 3D mesh graph. This method makes the mesh data easier to process and more compact. In addition, we make dimension reduction with this method, which is very important for reducing computation costs and eliminating the curse of dimensionality. In other words, we learn information from nodes and edge connections between nodes. Then, the unsupervised learning algorithm K-Means was used to cluster each node according to node embedding information and two different initialization method was performed. Moreover, our data-centric approach is much lower in computational cost than complex models such as CNN and RNN. Instead of using complex and computationally expensive models, we apply data-centric methods to improve the raw data representation. The main contribution of this study is developing a data-centric AI framework by utilizing a node2vec embedding algorithm, machine learning, and deep learning techniques. Additionally, we adapt the cosine similarity method to compare and evaluate the node embedding vectors trained with different hyperparameters. Also, we developed a new algorithm for choosing the optimal cluster number, calculated with geodesic distance on the 3D mesh. Thus, we provide competitive results compared to the state-of-the-art mesh segmentation methods.

Keywords: 3D mesh segmentation, unsupervised learning, embedding, node2vec, k-means, geodesic distance

ÖZ

VERİ MERKEZLİ DENETİMSİZ 3B NESNE SEGMENTASYON METOTU

TÜMER-SİVRİ, Talya

Yüksek Lisans, Modelleme ve Simülasyon Bölümü

Tez Yöneticisi: Doç. Dr. Yusuf Sahillioğlu

Aralık 2022, 52 sayfa

Bu tez, 3B nesnelerin bölütlemesini veri merkezli bir YZ yaklaşımı olarak sunar. Modelleme, doku eşleme, şekil sıkıştırma, basitleştirme ve iskelet çıkarma uygulamalarında bölütleme popüler ve önemli bir alandır. Bilgisayar bilimlerinde çeşitli amaçlara hizmet ettiği için nesne bölütleme problemi aktif ve öne çıkan bir araştırma alanıdır. Büyüyen makine öğrenmesi, derin öğrenme algoritmaları ve hesaplama gücü yardımıyla 3B nesne bölütlemesi problemini daha verimli bir şekilde çözmek için farklı bir yöntem uygulanmıştır. Bu tezde, 3B nesne bölütlemesi problemini veri merkezli bir YZ yaklaşımını merkeze alarak farklı bir perspektiften çözüyoruz. Yarı denetimli bir öğrenme algoritması olan node2vec'i, bir 3B ağ grafiğindeki her düğüm için vektör gömme temsilini eğitmek için kullandık. Bu yöntem, ağ verilerinin işlenmesini daha kolay ve daha kompakt hale getirmektedir. Ayrıca hesaplama maliyetlerini azaltmak ve boyutsallık lanetini ortadan kaldırmak için çok önemli olan bu yöntemle boyut küçültme yapıyoruz. Başka bir deyişle, bilgiyi düğümlerden ve düğümler arasındaki uç bağlantılardan öğreniyoruz. Daha sonra, denetimsiz öğrenme algoritması K-Means'i iki farklı ilklendirme metotu ile, her düğümü düğüm yerleştirme bilgisine göre kümelemek için kullanıldı. Ayrıca, veri merkezli yaklaşımımız, hesaplama maliyeti açısından CNN ve RNN gibi karmaşık modellerden çok daha düşüktür. Karmaşık ve hesaplama açısından pahalı modeller kullanmak yerine, ham veri sunumunu iyileştirmek için veri merkezli yöntemler uyguluyoruz. Bu çalışmanın ana katkısı, bir node2vec gömme algoritması, makine öğrenimi ve derin öğrenme teknikleri kullanarak veri merkezli bir yapay zeka çerçevesi geliştirmektir. Ek olarak, farklı hiperparametrelerle eğitilmiş düğüm yerleştirme vektörlerini karşılaştırmak ve değerlendirmek için kosinüs benzerliği yöntemini uyarlanmıştır. Ayrıca, 3B nesne üzerinde jeodezik mesafe ile hesaplanan en uygun küme sayısını seçmek için yeni bir algoritma geliştirdik. Böylece, son teknoloji mesh bölütleme yöntemlerine kıyasla rekabetçi sonuçlar sağlıyoruz.

Anahtar Kelimeler: 3B nesne bölütlemesi, denetimsiz öğrenme, gömme, node2vec, kmeans, jeodezik uzunluk

To my family

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my thesis advisor Assoc. Prof. Dr. Yusuf Sahillioglu for his advice.

I want to thank my family for gifting me opportunities that they never had, allowing me to achieve every single success of mine. Every day, I'm grateful for the sacrifices you make. I could not have undertaken this journey without my family's deepest supports.

I would like to thank my warm-hearted, dedicated, and helpful husband Ali Haydar Sivri for his emotional and physical support with an endless tolerance and love while sharing the home and life with me throughout my thesis process.

I would like to thank my father Kenan Tayfun Tümer, my mother Dilek Tümer and my little brother Ata Deniz Tümer for their endless emotional support with my whole life without hesitating.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
YZ	Yapay Zeka
1D	1 Dimentional
2D	2 Dimentional
3D	3 Dimentional
ML	Machine Learning
DL	Deep Learning
LLE	Locally Linear Embedding
HOPE	High-Order Proximity Preserved Embedding
SPE	Structure Preserving Embedding
GF	Graph Factorization
SDNE	Structural Deep Network Embedding
DNGR	Deep Neural Networks for Learning Graph Representations
BFS	Breadth Fast Sampling
DFS	Depth First Sampling
VGAE	Variational Graph Auto-Encoders
VAE	Variational Auto-Encoder
HARP	Hierarchical Representation Learning for Networks
RNN	Recurrent Neural Network
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
OPTICS	Ordering Points to Identify the Clustering Structure
DBSCAN	Density Based Spatial Clustering of Applications with Noise

PAM	Partitioning Around Medoids
CLARA	Clustering Large Applications
CLARANS	Clustering Large Applications based on Randomized Search
AP	Affinity Propagation
CURE	Clustering Using Representatives
ROCK	Robust Clustering using Links
FCM	Fuzzy C-means Clustering
FCS	Fuzzy C-Shells Clustering
MM	Mountain Method
DBCLASD	Distribution Based Clustering of Large Spatial Databases
GMM	Gaussian Mixture Method
CLICK	Cluster Identification via Connectivity Kernels
MST	Minimum Spanning Tree
STING	Statistical Information Grid
CLIQUE	Clustering in Quest
FC	Fuzzy Clustering
COBWEB	Hierarchical Conceptual Clustering
GMM	Gaussian Mixture Models
SOM	Self-Organizing Maps
ART	Adaptive Resonance Theory
ECD	Exact Convex Decomposition
ACD	Approximate Convex Decomposition
VM	Volumetric Meshes
SP	Space Partitioning
RG	Region Growing
WS	Watershed Segmentation

IC	Iterative Clustering
HC	Hierarchical Clustering
BS	Boundary Segmentation
MA	Medial Axis
RG	Reeb Graph
MC	Mesh Contraction
VC	Volumetric Contraction
SGD	Stochastic Gradient Descent
PYG	PyTorch Geometric
WCSS	Within-Cluster Sum of Square

CHAPTER 1

INTRODUCTION

Segmentation is one of computer science's most attractive research areas in computer science, such as animation, object recognition, robotics, and texture application. It aims to separate objects in a meaningful subpart. Meshes, which are representations of 3-dimensional objects, were used in this thesis. Mesh segmentation is crucial for the graphics industry since it is vital in computer vision and many more areas.

Artificial intelligence researchers are developing various responses for solving the mesh segmentation problem. However, they are mainly focused on labeled data. With the rise of data-centric AI solutions to the problem in many areas, such as object detection, there is an open door for proposing solutions in data-centric rather than model-centric AI. In this thesis, unlabeled 3D triangular meshes are our focus data to perform segmentation. We will consider both a data-centric approach and fully unlabeled experiments. We perform our proposed algorithm on the 3D human body and some objects such as a goblet, guitar, alien, chair, and vase 3D mesh data. As well as a new approach to the 3D mesh segmentation algorithm, we applied and developed a new inertia evaluation method on 3D mesh data, which is geodesic inertia, and an embedding quality evaluation method for finding better vector representation of 3D mesh data. Moreover, our proposed solution works well with the 3D human body and object meshes.

1.1 Problem Definition and Motivation

Model-centric AI generally aims to improve models by making them complex. These complex models have many hyperparameters and finding the right combination of hyperparameters is challenging in terms of human effort and computation cost. For this reason, researchs on data-centric AI are increasing day by day. Furthermore, there is a negative effect on training complex models from an energy consumption point of view. A recent study [10] showed that dimensionality reduction could lower model training's energy requirements up to 76% while maintaining model performance. When the number of data points is decreased, the improvement in energy efficiency becomes more striking. This thesis has taken a data-centric approach, emphasizing data preparation rather than complex models.

Most of the data used in machine learning and deep learning are labeled, but in real life, most data is found unlabeled. LeCun et al., who are AI pioneers, predict that unsupervised learning will be much more critical in the long run. They emphasize that humans and animals generally learn unsupervised and continue to give importance to unsupervised techniques by underlying that discover the world by observing, not by naming each object [11]. In the light of what we explained above, our motivation is

that compared to the supervised approach that uses ground truth labels, we developed an unsupervised algorithm that works best for train time thanks to the data-centric approach.

1.2 Proposed Method Overview and Contributions

In this thesis, unsupervised learning has been studied. Data-centric and unsupervised learning approach combined. 3D mesh data has been transformed into a meaningful and compact form by applying the node2vec embedding algorithm implemented on data nodes to eliminate the curse of dimensionality. In this way, computation cost has been significantly reduced, and results can compete with models such as meshWalker [12] and meshCNN [8] that need serious GPU and computation power.

The contributions which we provide in the thesis are:

- Most of the artificial intelligence-based mesh segmentation models are performed with model-centric techniques. We propose a data-centric approach.
- By making the data clear and compact with embedding and dimension reduction techniques, the computation cost has significantly decreased via a non-complex model. In this way, we achieved similar results with models with high computation costs.
- Cosine similarity metric is used for evaluating and choosing the best hyperparameters for each 3D mesh data object.
- To our best knowledge, we are the first who used the node2vec algorithm to represent 3D mesh data for dimensionality reduction and embedding vector representation.
- Two different initialization methods for clustering algorithm is applied and compared.
- Developing an alternative method for choosing optimal cluster number calculating geodesic distance between cluster centroids and cluster elements.

1.3 The Outline of the Thesis

The thesis is divided into four main chapters, the introduction chapter consideration. These are:

In Chapter 2, the literature review and background of graph embedding techniques, unsupervised learning techniques, and segmentation techniques are covered in detail.

In Chapter 3, the proposed 3D mesh segmentation method is described in detail in two parts: the node embedding method and the unsupervised clustering method. Additionally, geodesic inertia calculation algorithm and cosine similarity evaluation algorithm are explained and illustrated.

In Chapter 4, we explained the dataset which we used in experiments at the beginning of the chapter. Experimental results on training the node embeddings and how to choose the hyperparameters both in embedding vector representation and finding optimum cluster number are shown. Different initialization methods are illustrated and explained. The new algorithm for choosing the optimal cluster

number has experimented. Furthermore, the clustering results for the segmentation are presented and compared with the classical inertia calculation method.

In Chapter 5, the conclusion of the method used in the thesis, limitations and future work are outlined.

CHAPTER 2

BACKGROUND AND RELATED WORK

Before explaining our contributions in the thesis in detail, briefly explaining the underlying topics is important for our study to be more coherent. Under this scope, graph embedding methods in section 2.1, unsupervised learning techniques in section 2.2, and mesh segmentation methods in section 2.3 will be clearly explained.

2.1 Graph Embedding Techniques

A graph is a mathematical structure that models pairwise relationships between objects. It consists of nodes, V (also named as vertices or points) and edges, and E (also named as links or lines) that connect nodes with a semantic link. A graph is notated as $G = (V, E)$, where V is a set of vertices (nodes) and E is a set of edges (faces). There are several graph types, such as directed, undirected, and so forth. It is essential to know edge information that holds which node is input and which is output in a directed graph while we do not need to capture that information in undirected graphs. Graphs appear in many real-world applications more than we can imagine, like friendship prediction in social networks and function prediction for protein interactions. In addition to all real-world applications, graphs are heavily used in computer graphics. In computer graphics, a mesh, a particular graph structure, is used for segmentation, animation, solid modeling, etc. Mesh can be a form of any polygonal. We used a triangular mesh with three nodes and three edges on every face in our work. 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. [1] divided graph embedding techniques into four categories which are Factorization, Random Walk, and Deep Learning, see Fig. 1.

Factorization-based Methods. Factorization-based techniques factorize a matrix representing the connections between nodes to generate the embedding. Node adjacency, Laplacian, node transition probability, and Katz similarity matrices are some of the matrices used to express the connections. Locally linear embedding, Laplacian eigenmaps, graph factorization, graph representation, and high order proximity preserved embedding are different kinds of factorization-based techniques. LLE [13] is a method for unsupervised learning that computes neighborhood-preserving, low-dimensional embeddings of high-dimensional inputs. Belkin and Niyogi et al. [14] present a geometrically driven approach to create a representation for samples derived from a low-dimensional manifold embedded in a higher-dimensional space. Sundaresan and Chellappa [15] suggest a primary method for segmenting 3D voxel data of humans into various articulated chains using Laplacian Eigenmaps and a graphic

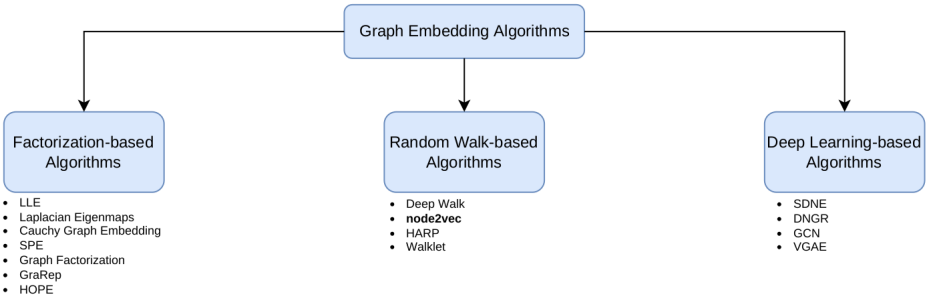


Figure 1: Categorization of graph embedding algorithms (Figure source [1]).

representation of the human body. The local topology can rarely be preserved by the Laplacian embedding as effectively as we would like. Luo et al. [16] provide a novel Cauchy graph embedding that preserves the similarity relationships of the original data in the embedded space by using a new objective that focuses on the preservation of dissimilarity between nodes rather than similarity for improving the local topology preservation property of graph embedding. The focus of graph embedding research is now on developing scalable graph embedding methods that take advantage of the sparsity of real-world networks. For instance, Graph Factorization's embedding approximates the adjacency matrix factorization [17]. Cao et al. [18] propose a model for learning weighted graph vertex representations.

Factorization-based approaches cannot learn every function, such as one that explains network connectivity. Thus, they cannot learn structural equivalence unless it is explicitly incorporated into their goal function. Graph factorization has few hyperparameters and is scalable, but other methods look more promising, like node2vec. In this thesis, we do not want to use factorization-based methods since our research question is based on data-centric approach with unsupervised learning. Since node2vec algorithm is based on word2vec-skip gram architecture, proven itself in natural language processing algorithms, adapting to obtain vector representation of 3D mesh data is an interesting research topic.

Random Walk-based Methods. The process through which randomly moving objects depart from their starting point is known as a random walk, see Fig. 2. Node centrality [19] and similarity [20] 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 [12]. DeepWalk, node2vec, HARP, and walklet are algorithms for obtaining node representations in a graph. Truncated random walks are used by DeepWalk [21] to learn latent representations. This method is scalable and preserves high-order proximity between nodes. node2vec [2] 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, see Fig. 3. [22] implemented a random walk strategy to segment 3D meshes.

Using better weight initialization, HARP [23] presents a method to enhance the solution and prevent local optima. For this reason, HARP aggregates nodes from the previous layer of hierarchy via graph coarsening to form a hierarchy of nodes. Then, it creates an embedding of the coarsest graph and uses the learned embedding to initialize the node embeddings of the refined graph (one level up in the hierarchy). This concept of explicit modeling and random walks are combined in Walklets [24]. The model adjusts the DeepWalk algorithm's random walk method by excluding some network nodes. We prefer

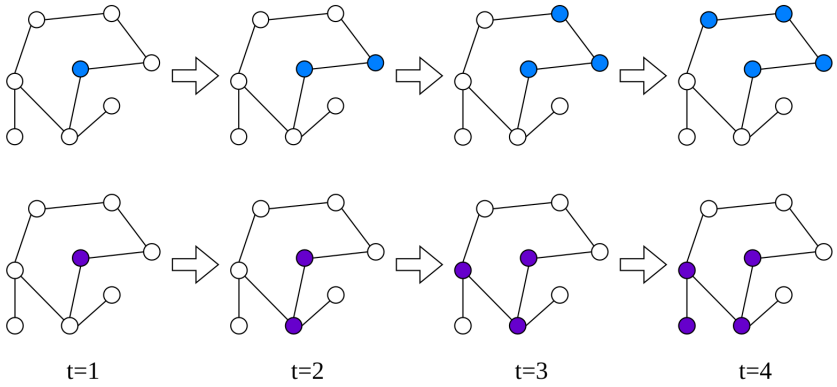


Figure 2: An example of two random walks on a graph while between t=1 and 4.

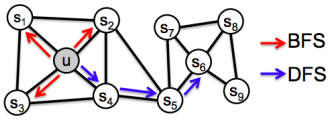


Figure 3: An example of BFS and DFS behavior (Figure source [2]).

to work with the node2vec algorithm since there is no implementation on 3D meshes for segmentation problems, according to our best knowledge. Moreover, experimenting with the segmentation problem with the power of biased random walk led us to perform a data-centric approach to reducing the curse of dimensionality.

Deep Learning-based Methods. Deep learning has been applied in many areas such as computer vision, natural language processing, etc. Deep learning algorithms are being developed for graph embeddings as a growing research area. SDNE, DNGR, GCN, and VGAE are DL-based methods. SDNE [25] is a semi-supervised model that has layers with non-linear functions for capturing non-linearity while preserving the local and global structures. Cao et al. [3] developed a model for learning graph representations that creates a low-dimensional vector representation for each vertex by encoding the graph's structural information. A random surfing model was adopted to capture structural graph information directly. See Fig. 4 for the DNGR model. GCN [4] 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, see Fig. 5, which solves the problems in other models. For example, Sever et al. [26] used GCN for a 3D mesh segmentation problem using sparse face labels of a 3D object. Additionally, Perek et al. [27] build a segmentation model that takes the full structure of the item as input and outputs its segmentation, treating a mesh object as a graph that needs to be labeled. VGAE [28] is an unsupervised learning technique in which a variational autoencoder is applied to the graph. In [28], 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.

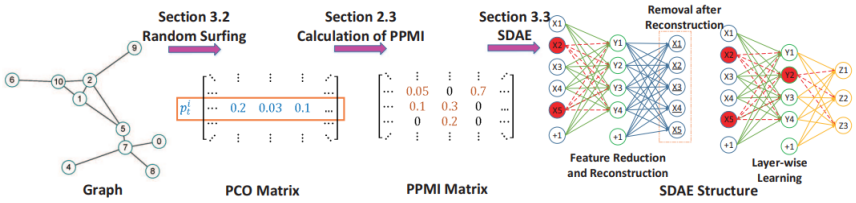


Figure 4: Main components of DNGR model (Figure source [3]).

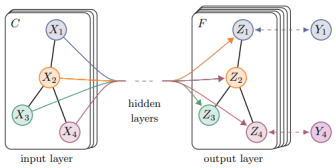


Figure 5: multilayer GCN network ex. C:input channels, F:feature maps (Figure source [4]).

2.2 Unsupervised Learning Techniques

Unsupervised learning is a machine learning technique that learns hidden patterns, distributions, and probabilities by not using supervised target outputs. These algorithms can be used to analyze and cluster unlabeled datasets and dimensionality reduction [29]. In [29], the author emphasizes that unsupervised learning can be thought of as learning a probabilistic model of the data. More specifically, the probability distribution for a new input x_t, $P(x_t \mid x_1, ..., x_{t-1})$, is represented using previous inputs $x_1, ..., x_{t-1}$. More simply, one can say that a simple unsupervised algorithm can build a model of the relevant data, where data points are independently and identically chosen from some distribution $P(x)^2$. In the field of mesh segmentation, unsupervised learning is one of the methods used. In [9], the authors applied K-Means and Fuzzy C-Means algorithms to output direct clustering results. There are several types of clustering methods. In [5], the authors divided traditional cluster algorithms into nine categories. These are partition-based, hierarchical-based, fuzzy theory-based, distribution-based, density-based, graph theory-based, grid-based, fractal theory-based, and model-based methods. The most used clustering algorithms are shown in Fig. 6. Details of clustering algorithms will be explained in related sections.

Partition-based Methods. Considering the center of the data points as the center of the corresponding cluster is the fundamental idea of this kind of clustering algorithm. Since clustering depends on the center of data points, partition-based methods are called centroid-based methods. Figure 7a is an illustration of a partition-based method example that has 4 clusters. K-Means [30], K-Medoids [31], PAM [32], and CLARA [33] are some of the famous clustering algorithms. In K-Means, updating the center of each cluster is the base idea using iterative process and computation until the convergence is met with the given criteria. K-Medoids are similar to K-Means, but they can deal with discrete data. K-Means is the most used since its computation time is the lowest [5]. Compared with other clustering methods, partition-based ones have high computing efficiency and low time complexity. However, they are unsuitable for non-convex data and easily manipulated by outliers. We prefer to use K-Means since our research question considered low time complexity. Also, we wondered how the embeddings, calculated using the connection between vertex and nodes, would fit into cluster centers. Our proposed

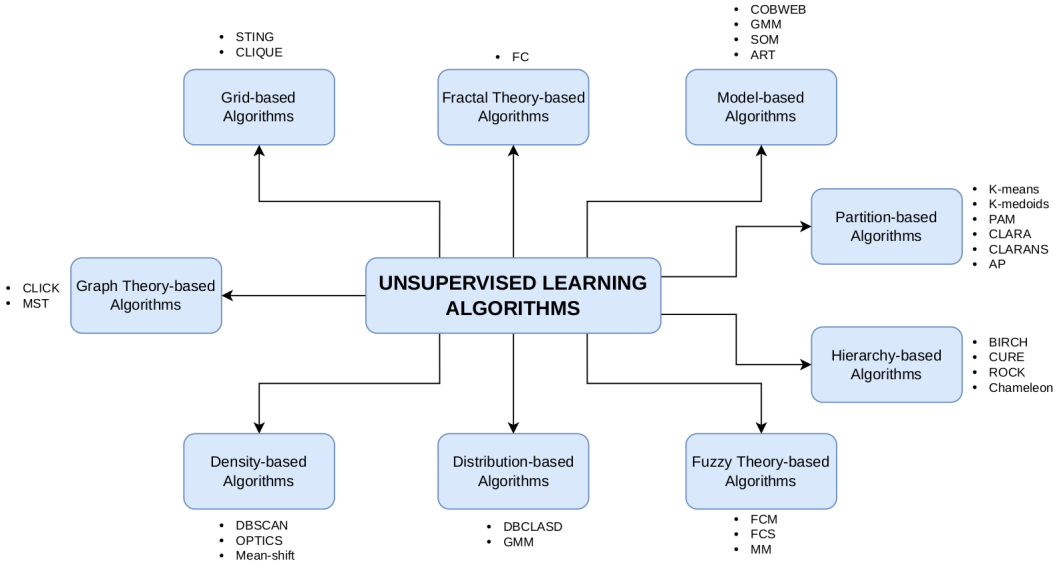


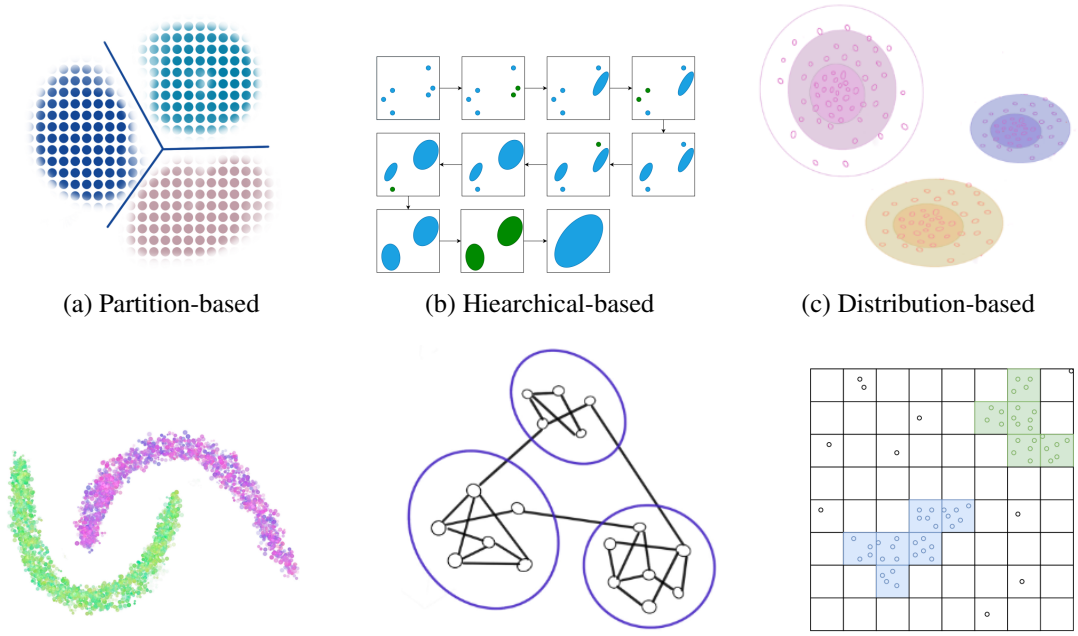
Figure 6: Categorization of unsupervised learning algorithm (Figure source [5]).

algorithm focuses on clustering 3D data nodes. Unlike the others, we performed K-Means with the node2vec algorithm.

Hierarchical-based Methods. Constructing the hierarchical relationship in the data is the core idea of hierarchical-based clustering methods [34]. Considering each data point as a separate cluster is the first step. After that, the algorithm repeatedly executes that identifying the two of them are closest enough. This process is continued until all clusters are gathered as one, see Fig. 7b. BIRCH [35], and CURE [36] are typical algorithms of the hierarchical-based clustering method. BIRCH performs the clustering algorithm by constructing the feature tree, updated when the new data point comes, of the clustering where a node represents a subset. CURE can perform on large-scale datasets and clusters using random sampling in many experiments. Hierarchical-based methods can be easily detected and have relatively high scalability. However, they have a high time complexity [5]. Since our research question considered low time complexity, we did not prefer hierarchical-based methods.

Fuzzy Theory-based Methods. The idea behind the fuzzy theory-based methods is that take the discrete value of labels and converting into a continuous interval for describing the relationship between data objects. FCM [37], and FCS [38] are typical algorithms. Calculating the membership of each data point to all clusters is the key idea of FCM by optimizing the objective function. FCS differs from traditional ones. It works on multidimensional hyperspace by prototyping each cluster and evaluating the distance function. Nevertheless, they can easily get stuck in local minima and are sensitive to initial parameters.

Distribution-based Methods. Distribution-based methods operate on data distributions. They decide on clusters by looking at the distribution variety, i.e., the number of distribution varieties of clusters. DBCLASD [39] and GMM [40] are typical algorithms. DBCLASD is that a data point is a part of a cluster if it satisfies the distribution of related cluster properties and the distribution of expected distance. GMM considers that data points must have the same independent Gaussian distribution. In Figure 7c, you can see an example of this type of clustering method idea. Distribution-based methods



(a) Partition-based (b) Hiearchical-based (c) Distribution-based

(d) Density-based (e) Graph theory-based (f) Grid-based

Figure 7: Illustrations of different unsupervised methods for clustering.

support well-developed statistical algorithms because they have relatively high scalability by changing the distribution. Nevertheless, they have a high time complexity relatively, and parameters can affect the results easily [5].

Density-based Methods. The core idea of density-based methods is that the data considered to belong to the same cluster are those in the area of the data space with the highest density. DBSCAN [41], and OPTICS [42] are commonly used algorithms. DBSCAN is an algorithm that reflects the core idea of density-based methods. For example, see Fig. 7d. On the other hand, the OPTICS algorithm is sensitive to the radius of the neighborhood and the minimum number of points in a neighborhood, according to DBSCAN. In general, density-based methods show high efficiency in clustering and can be used for data with arbitrary shapes. However, if the density of data space is not even and hyperparameters are not chosen regarding the data, clustering results will have low quality. Also, the data volume affects the computation memory size. We did not prefer density-based methods since it is not a perfect match for our segmentation problem.

Graph Theory-based Methods. Graph structure consists of vertices which are data points in our context, and edges which are a structure that connects vertices. Graph theory-based methods use edge and vertex relationship information. See Fig. 7e for an example illustration. CLICK [43] and MST-based clustering [44] are typical algorithms. The fundamental principle of CLICK is to divide the graph into clusters by applying iterative minimum weight division. The crucial step for the MST-based clustering algorithm is generating the minimal spanning tree from graph. These types of algorithms give high efficiency and high accuracy. However, time complexity increases when the graph complexity increases.

Grid-based Methods. The fundamental concept behind these clustering methods is to transform the original data space into a grid structure with a predetermined size for clustering, see Fig 7f. STING [45] which divides the data space into several rectangular units by creating a hierarchical structure and then cluster the data according to the structural levels and CLIQUE [46] are typical algorithms. CLIQUE uses the advantages of grid-based and density-based algorithms. On the other hand, it can easily be affected by mesh size.

Fractal Theory-based Methods. The core principle of fractal theory-based methods is self-similarity in the data contained can be calculated using fractal dimension. FC [47] is the typical algorithm. The fundamental idea of FC is that the fractal dimension's inherent properties are unaffected by any changes to a cluster's inner data. It provides high efficiency and high scalability, is not affected by outliers, and is suitable for data with arbitrary shapes and high dimensions. However, clustering results are highly sensitive to hyperparameters.

Model-based Methods. The primary idea is to choose a specific model for each cluster and determine which model fits each cluster the best. Model-based clustering algorithms can be broadly divided into two categories: those based on statistical learning, which is COBWEB [48] and GMM [40], and those based on neural network learning, which is SOM [49] and ART [50]. The main goal of COBWEB is to implement hierarchical clustering by creating a tree based on some criteria under the hypothesis that each attribute's probability distribution is independent. The primary notion of SOM is to construct a mapping from the input space of high dimension to the output space of low dimension. The fundamental principle of ART, an additive algorithm, is to dynamically generate a new neuron to find a new pair pattern when the present neurons are insufficient to form a cluster.

2.3 Mesh Segmentation Methods

Mesh segmentation has been a popular research field for decades in computer graphics since it can be used in application areas such as modeling, rigging, texturing, shape-retrieval, deformation, etc. Mesh segmentation can decompose a mesh into smaller and substantial sub-meshes [51]. A vast collection of proposed mesh segmentation methods have implementations of different approaches. Surveys [6, 51, 52, 53] which include diverse perspectives in terms of their categories, solution methods, performances, and comparison in between, are written about this active research area. In [6], the authors separated mesh segmentation into four distinct categories, volume-based segmentation, surface-based segmentation, skeleton-based segmentation, and multiple shape-based segmentation (see Fig. 8, input and output data properties are exemplified).

Volume-based Segmentation. Volume-based segmentation methods use 3D volume information as an input and output. Exact Convex Decomposition, Approximate Convex Decomposition, Volumetric Meshes, and Space Partitioning are algorithms for volume-based decomposition. In exact convex decomposition, the main idea is to decompose 3D mesh objects into precise convex sub-volumes. For example, Chzelle et al. [54] use exact decomposition, and this study can be considered as an origin. In approximate convex decomposition, decomposition yields volumetric convex pieces instead of producing precise convex hulls. Kreavoy et al. [55] and Liu et al. [56] used approximate convex decomposition. In volumetric meshes, decomposition results in a collection of basic volumetric data types like voxels, tetrahedra, hexahedra, and so on. Typical examples of volumetric meshes are Attene et al. [57] and Xian et al. [58]. In space partitioning, using the characteristics of the mesh objects,

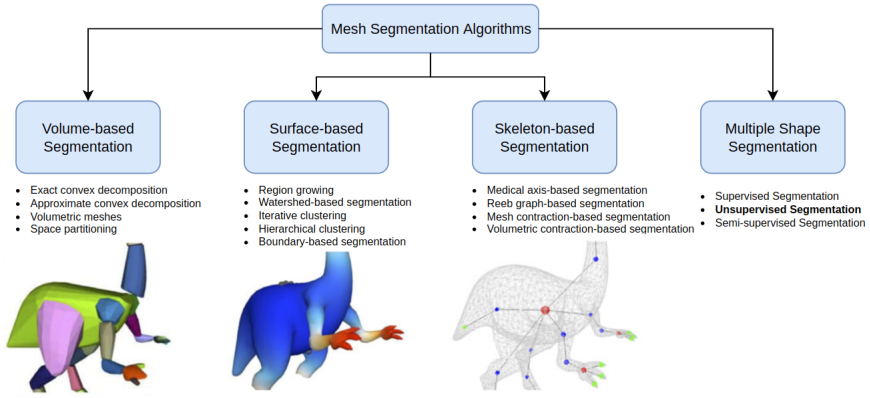


Figure 8: Categorization of 3D mesh segmentation algorithms. Figures are taken from [6].

the space is divided into regions to do the decomposition. Simari et al. [59] have developed a multi-objective space partitioning optimization problem based on Voronoi and K-Means clustering. We did not prefer volume-based segmentation methods since we proposed a data-centric approach using node and edge information.

Surface-based Segmentation. The Surface-Based Segmentation type of algorithms produce 2D output regions by using their 3D surfaces, and they use physical facts like angles and curves in the surfaces. According to [6], surface-based segmentation methods can be divided into five categories. These are Region Growing, Watershed Segmentation, Iterative Clustering, Hierarchical Clustering, and Boundary Segmentation. In region growing, Lavoue and Wolf [60] have developed an algorithm that uses Markov Random Fields for approximating a globally optimal solution using only local features of the mesh structure. In watershed segmentation, Mangan and Whitaker [61] introduced an algorithm to implement watershed image segmentation using created ground pieces in surfaces corresponding to the curvatures of the surfaces. Convexity, curvature, and proximity parameters were used by Shlafman et al. [62] in the concept of iterative clustering. Using geodesic distance and convexity was a strategy for decomposing object meshes into segmented parts developed by Katz and Tal [7], see an example in Fig. 9. Lastly, Lin et al. [63] applied the part salience' technique which is used in the cognitive psychology mainly, for the mesh segmentation as a boundary segmentation method.

Skeleton-based Segmentation. Skeleton-based segmentation methods are primarily used in animation, virtual navigation, and segmentation. Typically, given 3D mesh input to algorithms processed as an output-shaped 1D skeleton structure. Medial axis, reeb graph, and geometric contraction are skeleton-based segmentation methods. Authors of [64] have developed to solve robot grasping problem with a Reeb graph-based segmentation. Mortera et al. [65] uses medial axis method to extract a model structure for human-shaped bodies. Li et al. [66] use Geometric contraction.

Multiple Shape Segmentation Multiple Shape Segmentation methods use shape collections that can be fully labeled, partially labeled, or non-labeled objects. Mesh segmentation studies are becoming popular in these areas, as the computational power is high, and it facilitates the study of 3D data. These kinds of methods have three different categories which are supervised segmentation, semi-supervised segmentation, and unsupervised segmentation.

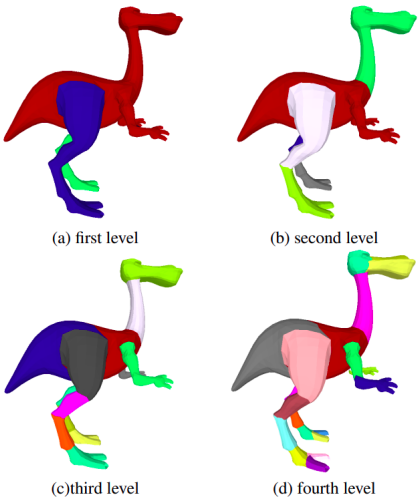


Figure 9: Hierarchical clustering using k-means and graph cut introduced in [7] (Figure Source: [7]).

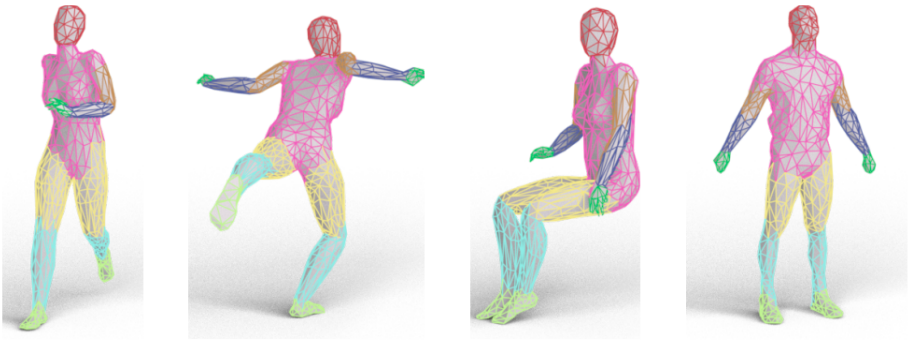


Figure 10: Result of the supervised mesh segmentation algorithm introduced in [8] (Figure Source: [8]).

- **Supervised Segmentation** uses fully labeled data in order to train the segmentation model. In [8], the authors developed a new mesh convolution layer. They used it in the segmentation problem, see Fig. 10. Lahav and Tal et al. [12] 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 supervised segmentation model. Kalogerakis et al. [67] developed a conditional random field model for the segmentation and optimized it with the JointBoost classifier [68]. We did not prefer to work on supervised methods for instance [8] and [12]. Since our purpose is solving the segmentation problem using a data-centric approach, methods mentioned as supervised learning are model-centric approaches. In other words, our goal is to solve a complex problem with high-quality data extracted using the node2vec algorithm using a relatively simple model which does not require high computation power.

- **Semi-Supervised Segmentation** essentially requires using labeled data on the object to do segmentation. In [69], the labeling is done by adding the energy term of unlabeled data to the conditional random fields. Shu et al. [70] developed scribble-based segmentation with weakly labeled

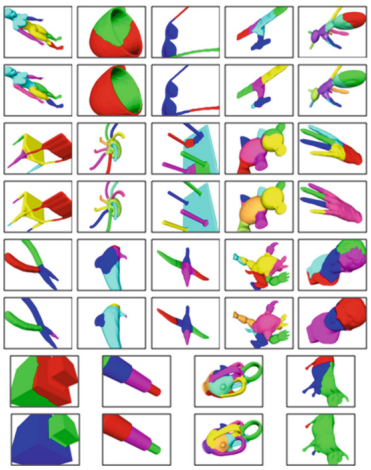


Figure 11: Result of the unsupervised mesh segmentation algorithm introduced in [9]. The first row of each model shows the results of the K-means-based clustering while the second row shows the results of the FCM-based clustering technique(Figure Source: [9]).

objects. Additionally, Shu et al. [71] 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. We did not prefer semi-supervised learning model to see the performance on fully unlabeled problem.

- **Unsupervised Segmentation** models based on unlabeled data. The algorithms use information/feature extraction techniques to segment 3D data. Also, the same classes of 3D objects approximately have the same segmentation models. As mentioned in [9], the authors applied unsupervised segmentation on the face part in 3D data using K-Means and FCM, see Fig. 11. Sidi et al. [72] performed a descriptor space study, and using the results, they produced a spectral clustering for segmentation.

In our work, we propose an algorithm that can be divided into two main unsupervised parts. Firstly, our algorithm uses a graph embedding technique that outputs high-quality data for extracting the information among nodes using connections between them. Next, we perform clustering algorithms using node embedding.

CHAPTER 3

SEGMENTATION MODEL

In the deep learning era, developing algorithms for data representation techniques are growing cumulatively. Embedding vectors, one of the data representation techniques, have significantly improved in the last decade. Embedding is a numerical representation of the data that helps to reduce the size of data into computable amounts to prevent the curse of dimensionality. In other words, it extracts information from high-dimensional data to create low-dimensional representation with capturing enough data to solve the problem. For example, we use the embedding representation to reduce the vector size from $n \times l$, where n is the number of nodes, to $n \times m$, where m is the embedding size, $m < l$, with evaluating neighborhood and node connection information, in our case. Additionally, algorithms that map the raw data to an embedding vector enable obtaining uniquely identified representations using its features.

It is essential to develop data-centric and unsupervised algorithms since, in the real world, we do not always have a chance to train complex models in terms of computation time and power and to label the data. 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. Our proposed algorithm is novel since we are the first to use node2vec and K-Means together for 3D mesh segmentation, according to our best knowledge.

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. In short, 3D mesh data is mapped to an embedding representation to be segmented into meaningful subparts. In the following sections, we describe the algorithm in detail.

3.1 Node Embedding Representation

Embedding representation is a way of running mathematical operations numerically feasible on powerful computation tools. It is widely used in many deep learning and machine learning application areas, such as natural language processing techniques, recommendation algorithms, language translation techniques, 3D deep learning, etc. Moreover, it is a powerful concept in terms of extracting similar information for objects or data features. For instance, political opinions, user behavior, recordings, weather cycles, and 3D mesh objects. As can be seen, embedding vector representations help to improve the technology advancements in real-life. Since 3D mesh data is a part of our world, to use

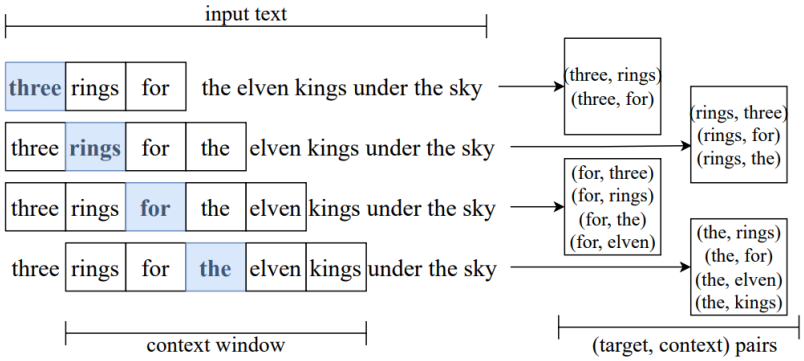


Figure 12: An example of how to pair target and context data for skip-gram architecture. The passage is taken from the Lord of the Rings, the Fellowship of the Ring by J. R. R. Tolkien.

embedding vector representation with them is inevitable. As we mentioned earlier in the related work section 2, a considerable amount of work uses embedding vector representation to map 3D mesh data.

Our 3D mesh segmentation algorithm's first step is mapping 3D mesh data to embedding vectors. 3D mesh data structure forms an indissoluble relationship between positions in the 3D space and the connection between them. We use node2vec [2] algorithm to reflect the neighborhood information since it uses the node connection with a biased random walk. node2vec, a semi-supervised algorithmic framework, is developed to learn continuous feature vector representations for nodes in graphs (as in our case, 3D meshes). The algorithm discovers a node mapping to features' low-dimensional space, which maximizes the likelihood of conserving node network neighborhoods. A random walk with bias methodology effectively explores various neighborhoods and develops a flexible concept of a node's connected neighborhood. The methodology proposes that the extra flexibility in exploring neighborhoods is the key to learning broader representations since it can capture the more comprehensive attributes of a node. Moreover, it characterizes prior findings based on rigid conceptions of network neighborhoods.

node2vec algorithm was developed based on the idea behind the word2vec/skip-gram [73] algorithm which is related with word embeddings. To explain briefly, word2vec/skip-gram is a fully connected deep learning architecture with one hidden layer and one output layer; see Fig. 14. The embedding vector size determines the hidden layer size. Input and output vectors are one-hot encoded vectors, which are the target and context pairs, respectively. These pairs are obtained by sliding context window, which is an important hyperparameter to capture the detail, where the target is in the middle of the context window, and the context is the other words in the context window, see Fig. 12. Additionally, this algorithm uses a negative sampling strategy to optimize the training process. Negative sampling is developed for updating only a small subset of weights per training sample since training all contexts for a single sample is expensive computation. Also, it provides a data-centric approach since it enables capturing data's relevant context. [73] showed that sampling more frequently, words are pruned to selected as negative according to unigram distribution, used for selecting negative samples from the whole. Positive samples, context-target pairs, are still trained fully. The training process is held to obtain finalized weights. These weights are considered as word embedding vectors for each word in the vocabulary. In this thesis, nodes are used instead of words to obtain the embedding vector for 3D mesh data, and biased random walks are used as input text via the node2vec algorithm.

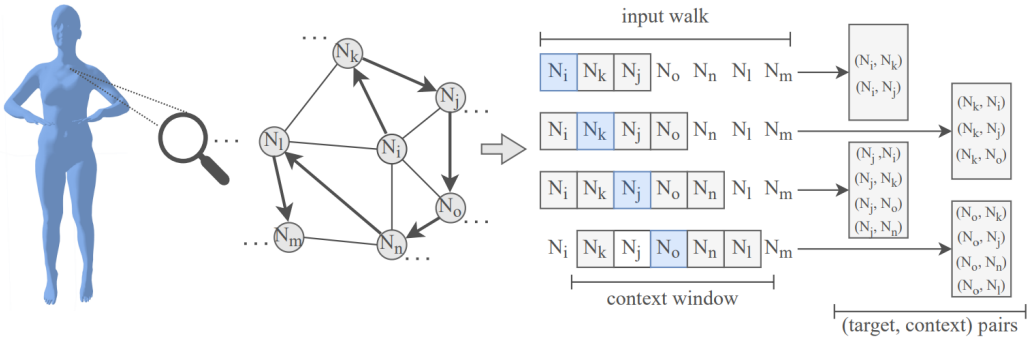


Figure 13: A diagram of how skip-gram architecture is applied on 3D mesh data.

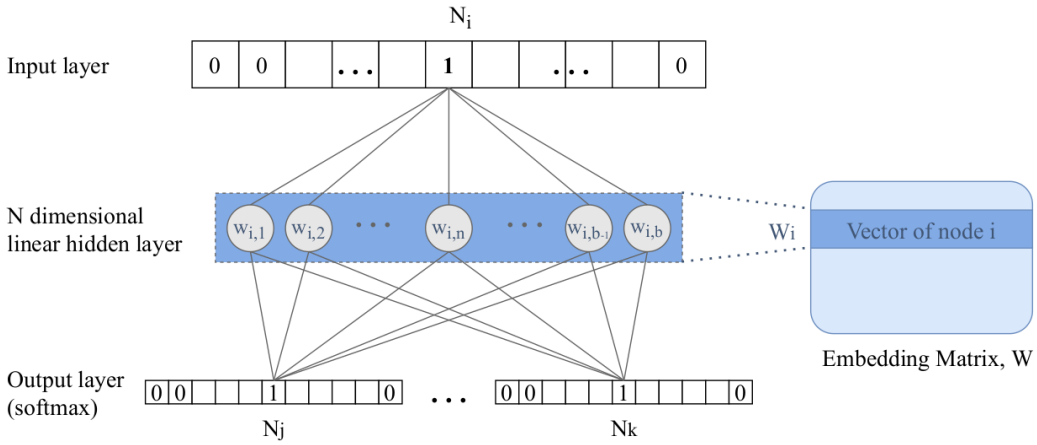


Figure 14: Skipgram architecture. N_i is the target node and N_j, N_k are two example samples from context node paired with the target.

Random walks are used in the node2vec algorithm, as are sentences in the word2vec algorithm. Since node2vec input and output vectors are built using random walks, it is good to mention random walks as well. Random walks can be explained as sequenced nodes that are randomly chosen from connected nodes. For example, consider a sample graph taken from 3D mesh data; see Fig. 13. In the figure, starting point, sequence first element, is chosen as N_i. Next, N_k is randomly chosen. This random process follows six times. Finally, the sequence is obtained in an order as $N_i, N_k, N_j, N_o, N_n, N_l, N_m$. To explain mode scientifically, equation 1, which decides the c_i's in the walk, is used to calculate the probability of walking to node x from the node v [2]. In the walk, $c_0 = u$ denotes the initial node while c_i denotes the ith node. In the equation 1, $\frac{\pi_{vx}}{Z}$ is the normalized transition probability, where Z is the normalization constant, between x and v. As in the case of node2vec, bias is added to walk between nodes according to BFS and DFS strategies; see Fig. 3. BFS search only nodes that are directly adjacent to the source are allowed in the neighborhood. On the other hand, DFS seeks that the neighborhood is formed of nodes that were sampled in increasing order of proximity to the source node. Consequently, the probability of unnormalized transition is calculated using $\pi_{vx} = \alpha_{pq}(t, x).w_{vx}$, where w_{vx} is 1 in unweighted graphs, α_{pq} as in the Eq. 2, taken from [2] and d_{tx} is the shortest path between t and x which are nodes.

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{2}$$

The training process comes after exploring the 3D mesh data with biased random walks. As it can be shown in Fig. 14, training is held between one-hot encoded target and context node pairs, $(N_i, N_j), (N_i, N_k)$, where $N_i, N_j, N_k \in N$ (see eq. 3) coming from biased random walks. N is the set of nodes of 3D mesh data. For target input nodes N_i, $i \in \{1, 2, \cdots a\}$, two layers, shallow, deep learning is trained with all their context nodes. The hidden layer is linear, and the output layer has a softmax function that calculates probabilities for pairs. Context-paired nodes are also called positive samples. On the other hand, as we mentioned when explaining the word2vec algorithm, the negative sampling method is also used for the optimized training process. Negative samples are selecting a similar strategy, unigram distribution. In other words, the most repeated nodes are not taken into account in training, i.e., their weights are not updated. At the end of the training of the neural network with positive and negative sampling, we obtain weight matrix, also called embedding matrix, W, where each W_i belongs N_i, $i \in \{1, 2, \cdots a\}$. W's dimension is $a \times b$, where b is the embedding size hyperparameter.

$$N = \{N_1, N_2, \cdots, N_a\}, \text{where } a \text{ is number of nodes.} \tag{3}$$

$$W = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_a \end{bmatrix}, \text{where } a \text{ is the number of nodes} \tag{4}$$

There are important hyperparameters that we need to emphasize. The first one is that the embedding size hyperparameter determines the size of the output vector (embedding representation vector), which is $a \times b$, where a is the number of nodes and b is the embedding size parameter. Secondly, walk length is the length of each biased random walk for each node, and context size is related to the efficient sampling rate by reprocessing samples from various source nodes. Walks per node specify how many walks should be sampled for each node.

There are several types of random walks, such as first-order and second-order random walks. In the first-order random walk, the probability of walking the next node is equal since the current state is important, and the previous node's information is not kept. In the second-order random walk, called also biased random walk, the previous and the current node information is essential for the next choice. Probabilities or weights for the next node are selected according to important hyperparameters in node2vec. These are p, the return parameter, and q, the in-out parameter. The likelihood of rapidly returning to a walk node is controlled by the parameter p. Sampling a node that has previously been visited is less likely if it sets to a high value. q controls the search to distinguish between "outward" and "inward" nodes. Finally, the learning rate controls the speed of training embeddings and convergence to

the optimal solution with the SGD optimization algorithm. It is essential to choose the hyperparameters to represent the raw data according to each problem.

3.2 Measuring Embedding Vector Quality

Embedding vector is an important way of representing the data numerically. Since the models and evaluations are made on embedding vectors, it is highly important that data features are as representative as the original data. There are numerous implementations of word embeddings for evaluating the quality and also the similarity. For example, inferring semantic similarity between two texts demonstrates the quality of the representations used [74]. Cosine similarity is used to evaluate word embedding quality in this paper. The inspiration comes from the examples we mention to apply the cosine similarity. While the word embeddings compare the similarity between the meaning of the text or word, we compare the neighbor nodes and how they are similar due to the biased random walk.

In our implementation, we implemented and developed a strategy that can measure both biased random walk and neural network training by using the similarity metric within the neighbor nodes. Since the neighbor nodes share common features, corresponding node embedding vectors also must. According to this idea, we used the cosine similarity score to evaluate how the node embedding vectors are trained and represented. To explain in detail, cosine similarity evaluates the similarity score between two vectors by finding the cosine in the n-dimensional space; see equation 5. The result can be in $[-1, 1]$. When the result is close to -1, there is no similarity, and when the result is close to 1, there is a strong similarity between the node embedding vectors. If it is around 0, it means that they are orthogonal to each other. To sum up, we evaluated the well-trained node embedding vectors by similarity measure metric between the neighbors.

$$\text{cosine similarity} = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|}, \text{ where A, B} \in \mathbb{R}^n \tag{5}$$

In order to find the similarity score, some important steps need to follow; see algorithm 1. Firstly, neighboring nodes of corresponding 3D mesh data are detected and paired uniquely since we are only interested in the relationship between neighboring nodes. Secondly, we calculate the cosine similarity 5 for detected pairs and append them to a list. Finally, to decide whether the embedding vector is qualified enough, we evaluate the descriptive statistics outputs, which is the mean value. Mean is an important value since it gives the general behavior and tendency of a series in the cosine similarity results. If the mode is closer to 1, we claim and accept that the embedding vector of 3D mesh data is well-represented. It is ill-represented if it is close to 0 or -1.

3.3 Node Clustering

The second part of our 3D mesh segmentation algorithm applies a clustering algorithm to the node embedding. Clustering is an unsupervised machine learning problem where we do not have ground truth labels. In our case, we assume that we do not have and do not calculate ground truth labels since we want to remove labeling costs. K-Means [30] is a clustering algorithm that is partition or centroid-based. K-Means relies on the assumption that objects in data are similar to one another. It is

Algorithm 1 Embedding vector quality measurement.

.

Definitions :
F : Faces of 3D mesh data
N : Face number
E : Embedding vector

$F = \begin{bmatrix} f_1 & f_2 & f_3 \end{bmatrix} \in \mathbb{R}^{N \times 3}$
$E = \begin{bmatrix} e_1 \\ \dots \\ e_n \end{bmatrix} \in \mathbb{R}^{n \times d}$, where n is the node number, d is the embedding dimension
Initialize pairs
for $n \leq N$ **do**
  if $(f_{i_k}, f_{j_m}) \notin$ pairs **then**
    append (f_i, f_j) to pairs, where $i, j \in \{1, 2, 3\} \wedge i \neq j \wedge k, m \in \{1, 2, \cdots, N\}$
  end if
end for
p: length of the pairs
for $k \leq p$ **do**
  $c_k =$ cosine similarity(e_i, e_j) 5
end for
result $= \frac{\sum_{k=1}^{p} c_k}{N}$
return result

---

an iterative algorithm that attempts to partition the dataset into k unique, non-overlapping subgroups (clusters). While keeping the clusters as far apart as possible, it aims to keep the data points made of intra-cluster as comparable as possible. It distributes data points to clusters in a way that minimizes the sum of the squared distances between the centroid of the cluster and data points, which is the average value of all the data points in the cluster. The similarity of the data points within a cluster increases as the amount of variance within the cluster decreases. The iteration process for updating the cluster centroids continues until they stay still.

Selecting the centroid of the initial clusters is crucial for the iterations' convergence in the K-Means algorithm. There are several initialization techniques. Two of them are random initialization, and K-Means++ [75] initialization. Firstly, random initialization is basically running the clustering algorithm with different random centroids and selecting the best solutions between them. Secondly, the K-Means++ algorithm was released as a more innovative technique for initialization. The algorithm reduces the risk of the suboptimal converged solution by choosing centroid points that are far from each other. In other words, the contributions to the total inertia of data points' empirical probability distribution are used as a sampling for initial clusters. Thus, convergence speed is increased, and it is $O(logk)$ optimally, which is proved theoretically. Authors of [75] mentioned that computation cost is higher in K-Means++ according to the classic method. However, it is still more advantageous because it reaches the converge point more quickly; that is, the number of iterations is reduced.

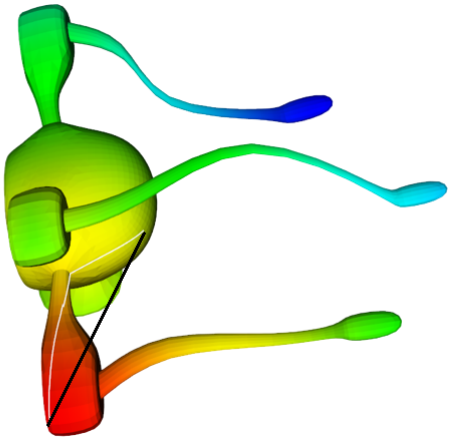


Figure 15: Example of Euclidean distance (black) and geodesic distance (white) between two nodes on alien object. Coloring is based on distance contour from source point, in the red area.

3.4 Choosing the Cluster Number

Unsupervised learning models are important for real-world applications since we do not have a chance to access labeled data easily. However, it has challenges, such as deciding the cluster number. Following that, evaluating whether the chosen cluster number value is suitable for the problem is another issue to be considered. There are several ways to find the optimal cluster number, such as silhouette score, Calinski-Harabasz index, Davies-Bouldin index, etc., where the labels do not exist. In our work, we implemented the elbow method for the optimal cluster number. The elbow method is independent of the relationship between clusters. 3D mesh data's segmented parts' border lines are joint with each other since independence is important for our problem. Thus we choose the elbow method, i.e., the within-cluster sum of squares evaluation metric.

K-Means algorithm evaluates the inertia value, also known as WCSS (equation 6), when the iterations for the optimum centroid location are converged. WCSS evaluates the sum of the squared distance for each sample in the same cluster to its centroid. The highest result is obtained when the cluster number k is 1. As the number of clusters increases, the inertia value begins to decrease. In other words, the distance between data points and related cluster centers decreases because as centroid centers increase. The preferable cluster number is when the inertia value is lower. However, it is not correct to choose the lowest value at k since there will be a lower one at $k + 1$. There is an important way to choose the optimal k, which is the rate of decrease of inertia value. After some cluster number's inertia value decreases, the rate begins to slow. It can be determined using the plot of inertia results. The rapid change point forms a shape of an elbow, which is the reason for naming the method. Hence, the elbow point is the optimal cluster number for the data.

$$inertia = \sum_{k=1}^{k_c} \sum_{i=1}^{N} (x_i - C_k)^2 \tag{6}$$

3.5 Geodesic Distance Inertia Method

Various applications in digital geometry, scientific computing, computer graphics, and computer vision require numerical computation of shortest paths on curved domains as well as the associated geodesic distance, which is the equivalent of a straight line length on a curved surface. Due to the influence of curvature on shortest path behavior and because the domain may not be exactly represented, these tasks are more challenging than Euclidean distance computations. It is a fundamental operation in computational science to be able to calculate the shortest paths and/or distances between pairs or sets of points. Compared with computing distances on Euclidean domains, this problem is complicated by curvature and the fact that the domain itself may not be precisely known.

Geodesic distance is calculated by passing the shortest path on the dataset, as opposed to Euclidean distance, which ignores the shape of the dataset. For example, in Fig. 15, a difference between Euclidean and geodesic distance is illustrated between two nodes where the black line shows the Euclidean and the white line shows the geodesic distance. As can be seen, geodesic distance is calculated on the shortest path between two nodes where the 3D mesh data's shape is important, unlike the Euclidean distance. Using geodesic distance on raw 3D data, we can evaluate the number of clusters we found with embeddings using Euclidean distance. Geodesic distance on raw 3D data can also be used to evaluate whether our embedding work has yielded successful results. Also, we can compare the two results, which are raw data and embeddings, for an optimal cluster number.

We define how we evaluate the geodesic inertia in detail in the algorithm 2. Firstly, we determine the centroid nodes for each cluster. We return the cluster centroids on node embedding vectors to do so since we experiment with clustering on node embeddings. Since the corresponding nodes' indexes are the same as the corresponding embedding index, we can easily find the cluster centroid node by applying the minimum distance on 3D axes. After obtaining the node centroids for clusters, corresponding nodes are separated according to their cluster. Next, the geodesic distances are evaluated according to the shortest path between the cluster member and the centroid node. Since the inertia is the sum of squared distance, we take the square of the evaluated geodesic distance and sum the squared distances of all cluster members. After processing this step for all clusters, we finalize the inertia score for summing all cluster scores. The final score gives the geodesic inertia value for the k cluster.

3D mesh segmentation is a hard and challenging problem since the computation cost is high because of the data size. However, we propose a novel data-centric 3D segmentation algorithm. In the above, our algorithm's details are explained. In algorithm 3, the technical details are explained in short. Firstly, the training embedding vectors is done several times, depending on the user. The hyperparameter space is defined by the user and randomly chosen by the algorithm. After training the embedding vectors for the 3D mesh data, the cosine similarity measurement algorithm 1 is applied to find the best embedding. According to the chosen embedding vector, the K-Means algorithm trained for cluster values between 2 and 20. The elbow method is used for deciding the optimal cluster number. Finally, the last labels for the 3D mesh segmentation algorithm are obtained by the chosen node vector embedding and the elbow method result.

Algorithm 2 Inertia evaluation using geodesic distance.

Definitions :
G : 3D mesh data
C : Centroid position vector
E : Embedding vector
L : Labels

$E = \begin{bmatrix} e_1 \\ \dots \\ e_n \end{bmatrix} \in \mathbb{R}^{n \times d}$, where n is the node number, d is the embedding dimension
$C = \{c_1, c_2, \cdots, c_k\}$, where k is the cluster number and c_k's are cluster centroids
$N = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \in \mathbb{R}^{n \times 3}$, where v_i's are position vectors for each node
$d_i m = \|e_i\|\|c_m\|$, where $i \in \{1, 2, ..., n\}$ and $m \in \{1, 2, ..., k\}$ k is cluster number and n is the node number
$L = \{l_1, l_2, ..., l_k\}$, where l_k's are the subsets of each cluster that contains node indexes, k is the cluster number
$v_{i_k} \leftarrow \min_i d_i m$, where v_{i_k} is the centroid node and k is the corresponding cluster
$C = \{v_{i_k} \mid i$ is the corresponding centroid node index and$k$ is the corresponding cluster$\}$
Initialize intra cluster I_{c_k}
for $k \leftarrow 2$ **to** 20 **do**
  for v_i, where i is the centroid node index for the cluster k **do**
    $d_g =$ geodesic distance(v_{i_k}, l_k)
    $I_{c_k} += d_g^2$
  end for
end for
$inertia = \sum_{k=2}^{K} I_{c_k}$, where $K \in \{2, ..., 20\}$
return $inetia$

Algorithm 3 Our proposed 3D mesh segmentation algorithm.

Definitions :
G : Graph
H : Hyperparameter space
h_n: User defined embedding number for hyperparameter tuning

---

for k $\leftarrow 1$ **to** h_n **do**
  $embedding =$ **node2vec**(G, H) [2]
  Hyperparameters of node2vec are randomly set according to H
  Select the higher embedding score with node embedding quality (algoritma 1)
end for
Decide k_c with the elbow method
$labels =$ **K-Means**$(embedding, k_c)$ [30]
return $labels$

CHAPTER 4

EXPERIMENTS AND RESULTS

4.1 Dataset

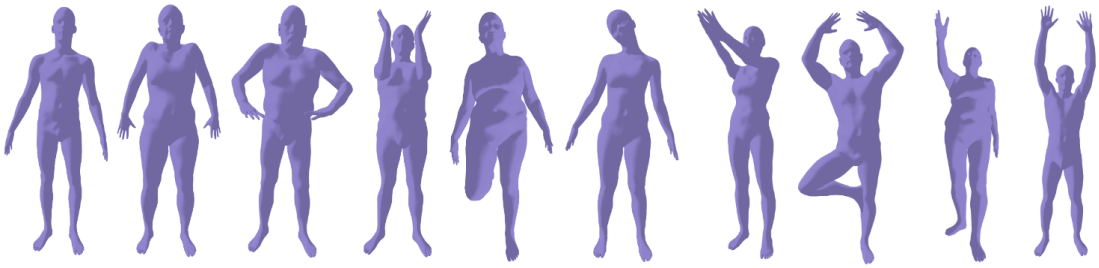
In 3D mesh segmentation, there are widely used datasets with many different shapes, such as human and animal categories, statues, vehicles, types of furniture, etc. In this thesis, we experimented with and tested our new segmentation algorithm on different datasets. The first one is FAUST [76] dataset. The dataset has 100 scans of 10 distinct humans with 10 different poses, which are high-resolution triangulated meshes; see Fig. 16a. 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 [77], which has 71 different poses and high-resolution mesh objects; see Fig. 16b. Another dataset is that we use three classes of COSEG [78] dataset, which are guitar, vase, goblet, chair, and alien, see Fig. 17. We use Plotly, which is a Python library for data visualization.

4.2 Experiments and Evaluations

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.

4.2.1 Experiments and Evaluations on Node Embedding

The first part of the experiments consists of obtaining embedding for 3D mesh data. PyTorch Geometric [79] library, which leans on Pytorch [80], 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.



(a) FAUST dataset

(b) SCAPE dataset

Figure 16: FAUST 16a and SCAPE 16b datasets visualized in different humans with poses.

(a) guitar (b) chair (c) goblet (d) vase (e) alien

Figure 17: 3D mesh data objects of COSEG dataset, which we used.

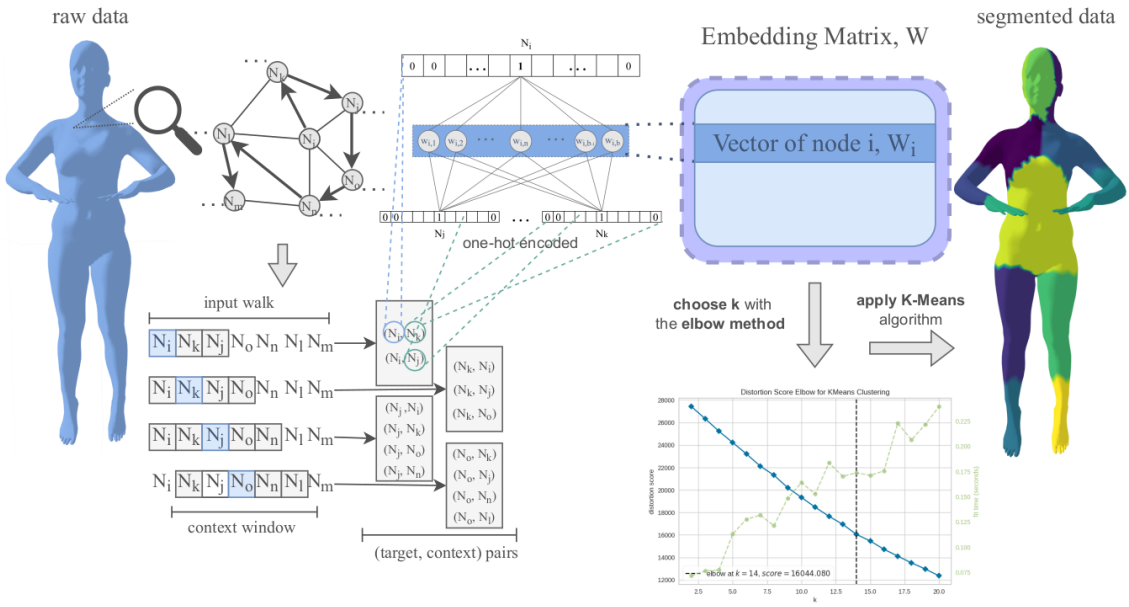


Figure 18: Our 3D mesh segmentation algorithm illustration.

4.2.1.1 Hyperparameter Selection

Hyperparameter tuning is a common technique used in machine learning and deep learning to adjust the settings of hyperparameters or parameters that affect the performance of the model. One common model example includes adjusting the learning rate of the neural network so that you get the best performance possible without overfitting the model. Many machine learning model algorithms contain many hyperparameters that need to be adjusted before you can use them in practice. Hyperparameter tuning is the process of changing the values of the hyperparameters of a model in order to improve its performance or accuracy. Plenty of hyperparameters can affect the model's performance, including optimizers, regularization methods, activation functions, and many more. Adjusting those hyperparameters will give you more control of your model and make training more efficient.

Different hyperparameters will work best for different datasets. Because of this reason, there is no silver bullet in the hyperparameters tunning issue, and it can be difficult and time-consuming to find the optimal values for all of these hyperparameters. There is some software to automate this process and make it much easier for you to find an optimal set of hyperparameters for your model. In supervised learning problems, hyperparameter optimization is done by comparing the most suitable performance metric between real targets and model predictions, where the model is trained on the train set, validated on the validation set, and tested on the test set. However, in unsupervised and semi-supervised learning models, we do not have to chance to compare results with respect to truth values, and no meaningful information comes from splitting the dataset into a train, validation, and test sets. Hence, we evaluated the results and did hyperparameter optimization according to the algorithm which we developed and adapted to 3D mesh data by evaluating the cosine similarity of neighboring nodes.

In our case, we trained one linear neural network with one hidden layer to compute the weights of the neural network for node embedding representation vector obtained with biased random walks under

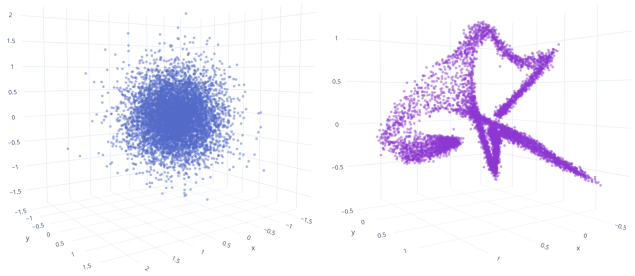
the determined context size. Moreover, we trained the node embeddings using an early stopping which we write for this case. Early stopping works as a self-controller, i.e., it evaluates the change percentage between previous and current loss values. Also, we define the patience parameter, which decides how many times the early stopping allows the training by looking at the change percentage. In our experiments, we set the change percentage as 1% and patience as 4. Since it has deep learning architecture as a semi-supervised learning based on word2vec algorithm, it has important hyperparameters which can be both categorical and numeric. The categorical one is optimizer which is Sparse Adam Optimizer [81]. Sparse Adam optimizer is used for all the training. Numeric ones are context size, embedding dimension, walk length, walks per node, learning rate, p, q, and batch size. The numeric hyperparameter values' sets are shown in detail in 7, where CS is context size, ED is embedding dimension, WL is walk length, W_pN is walks per node, LR is learning rate, P is return parameter, Q is the in-out parameter, and BS is the batch size.

$$\begin{aligned} CS &= \{8, 10, 12, 15, 18, 20, 24, 40\} \\ ED &= \{10, 12, 16, 32, 64, 72, 96, 128, 192, 256\} \\ WL &= \{15, 20, 25, 30, 40, 60, 80, 100, 120, 150, 200\} \\ W_pN &= \{2, 5, 10, 15, 20, 25, 30, 40\} \\ LR &= \{0.0005, 0.005, 0.001, 0.005, 0.002, 0.01, 0.02\} \\ P &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5\} \\ Q &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5\} \\ BS &= \{8, 16, 32\} \end{aligned} \tag{7}$$

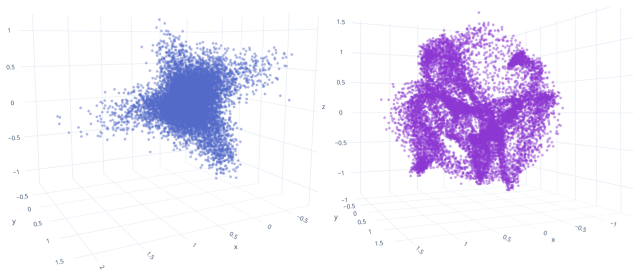
According to experiments, we obtained more than 70 embedding vectors for each 3D mesh data type by choosing the hyperparameters randomly from the sets 7. Applying the cosine similarity quality measurement algorithm, we observed that results are impressive in Table 1, minimum and maximum values that come from cosine similarity results between neighboring algorithms. The results' range are between 0.36 and 0.98 for the FAUST dataset, 0.68 and 0.99 for the SCAPE dataset, 0.32 and 0.98 for an alien object, 0.07 and 0.94 for a vase object, 0.13 and 0.96 for a guitar object, 0.13 and 0.98 for a chair object and 0.06 and 0.97 for a goblet object. As we mentioned in the previous section, the closer the value is to 1, the greater the similarity. Our results for all 3D mesh data objects are highly close to 1. We can conclude that node embedding vector representations are qualified enough. For the remaining part of the segmentation algorithm, we continued by using the parameters (Table 2) of the node embedding representation that gave the optimal results. When we look at our experimental results, we see that different hyperparameters give the optimal results for different datasets. At this point, each dataset has different properties and therefore gives the optimal results with different hyperparameters.

4.2.2 Experiments and Evaluations on Node Clustering

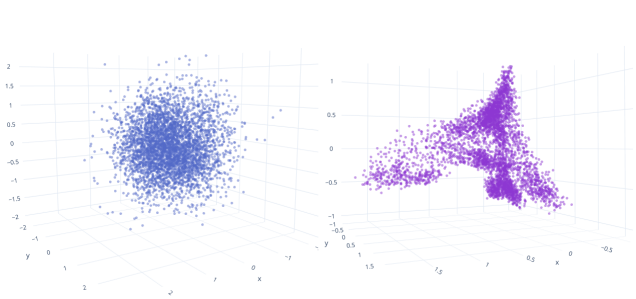
Node clustering is the second major part of our 3D mesh segmentation algorithm. In this part of the algorithm, we used the node embedding representations, which explained how the node embedding is evaluated and trained in the previous section for clustering. We did our experiments using the K-Means algorithm implemented on both random initialization and K-Means++ initialization.



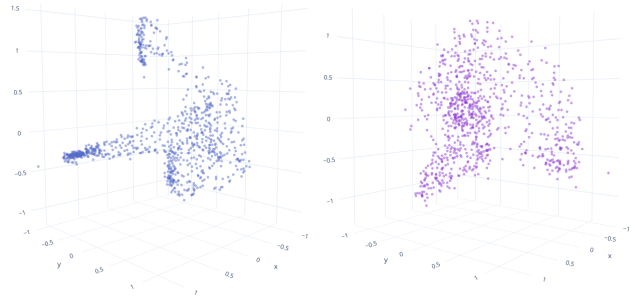
(a) FAUST



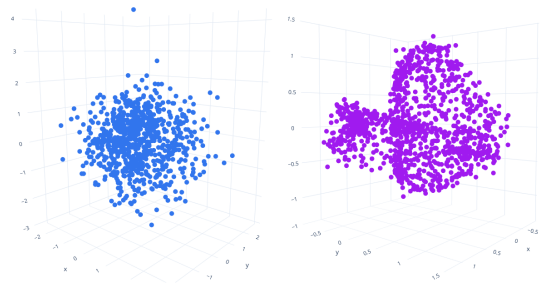
(b) SCAPE



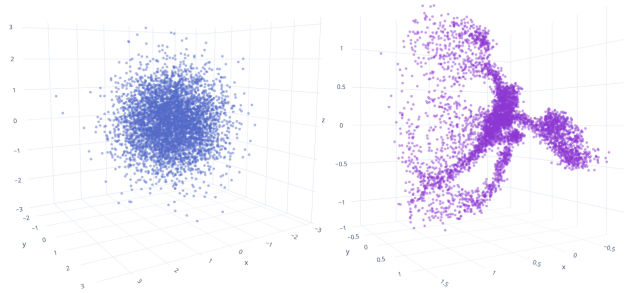
(c) ALIEN



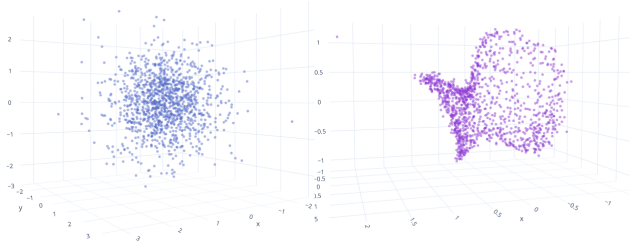
(d) VASE



(e) GUITAR



(f) CHAIR



(g) GOBLET

Figure 19: 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.

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

3D Object Type	**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 2: The optimal hyperparameters according to cosine similarity quality measurement experimented with chosen hyperparameter value sets. CS: context size, ED: embedding dimension, WL: walk length, WpN: walks per node, LR: learning rate, BS: batch size

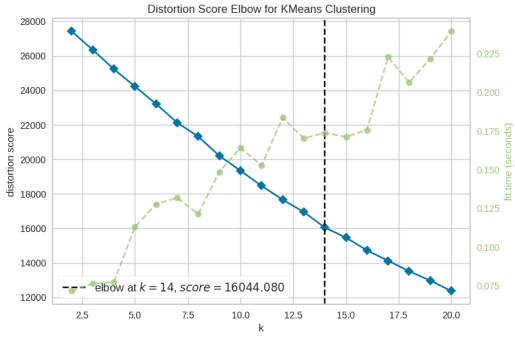
3D Object Type	**CS**	**ED**	**WL**	**WpN**	**LR**	**P**	**Q**	**BS**
FAUST	40	32	100	10	0.01	1	1	8
SCAPE	20	10	100	30	0.005	0.3	0.4	8
ALIEN	20	16	25	15	0.005	0.7	0.9	8
VASE	20	16	30	25	0.005	1.2	0.1	8
GUITAR	18	12	30	25	0.005	0.2	0.3	8
CHAIR	24	16	30	25	0.005	1.5	0.9	8
GOBLET	20	12	25	10	0.005	0.4	0.2	8

Deciding how many clusters we need to use is a crucial step and a hard question to answer for clustering, especially for 3D mesh data since it is fully unsupervised, i.e., there are no ground truth labels. However, there are many ways of determining the number of clusters with the elbow method, Calinski-Harabasz index, and Davies-Bouldin index. Firstly, we also did experiments using Calinski-Harabasz and David-Bouldin indexes. They produced unreliable cluster number results. To be more specific, Calinski-Harabasz and David-Bouldin suggested the following cluster number respectively, for FAUST 2 and 20, for SCAPE 5 and 17, for alien object 2 and 18, for vase object 4 and 20, for guitar object 2 and 17, for chair object 2 and 20 and for goblet object 2 and 19. As can be seen from the results, they have no common and close to each other for the same 3D mesh data since the indexes work better in well-separated in terms of distance between clusters and compact clusters. In other words, the separation distance is important. In our case, since the 3D mesh (graph) structure maps to node representation vectors, there is no clear distinction between clusters obtained from embedding vectors. Thus, these indexes are not applicable to this problem. However, the elbow method calculates the WCSS, which computes the cluster nodes and the cluster centroids only, not looking at the relationship between clusters. The optimal cluster numbers for each 3D mesh data are shown in Table 3. The elbow method suggests that FAUST's cluster number is 14, SCAPE, alien, and chair objects' cluster numbers are 11, the vase's is 4, the guitar's is 5, and the goblet's is 3. To find the optimal cluster number, we used the elbow method by looking at the output graphics from the Yellowbrick machine learning visualization library. The results are shown in Fig 20.

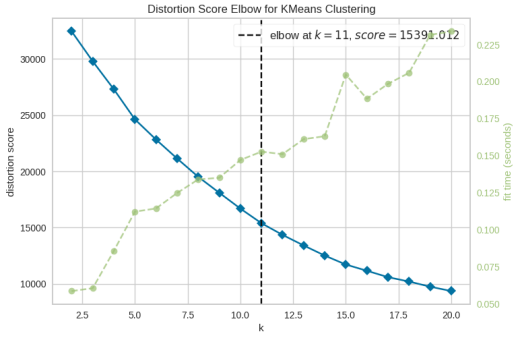
Table 3: Optimal cluster values according to elbow method for each 3D mesh object

3D Object Type	FAUST	SCAPE	ALIEN	VASE	GUITAR	CHAIR	GOBLET
k	14	11	11	5	7	11	3

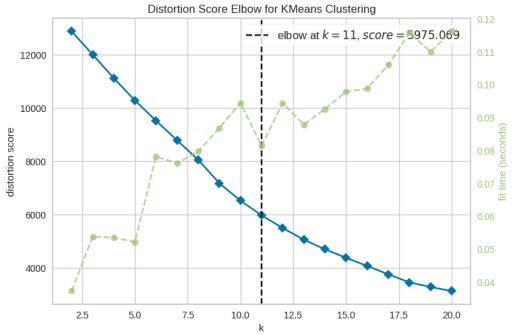
We perform clustering firstly, using K-Means++ initialization. Results provide that our data-centric 3D mesh segmentation algorithm, which includes hyperparameter choice with cosine similarity and elbow method, shows remarkable results, see Fig. 26. It seems that the results are symmetric and successful for segmenting distinct parts of the objects. For the FAUST dataset, segmented the 3D mesh into 14 symmetric parts, see Fig. 26a. Firstly, our model segmented the head part of the body into 3 three which are the face, symmetrically left side, and right side of the head. Also, the right-left hand, and arm, upper leg are symmetrically segmented. Moreover, the body part of the object is segmented into three parts. The surface extending from the chest to the legs and the part that connects the shoulders with the neck and separates them from the chest to the chest are symmetrically segmented. The second human-shaped object is a member SCAPE dataset. It is segmented 11 parts; see Fig. 26b. The lower body where below the chest, is symmetrically segmented. Also, the head part is segmented as one piece. However, there is small confusion about the segmentation of the arms and hands area. They are segmented but not successful as the FAUST dataset. We also observe that there is a missegmented part around the right shoulder. 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.



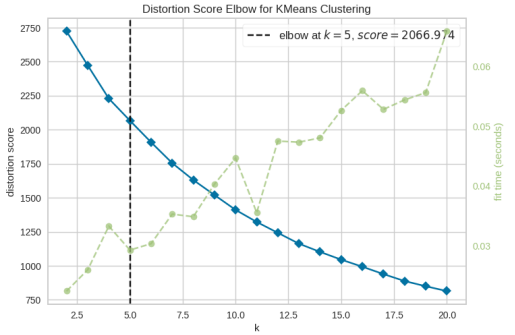
(a) FAUST



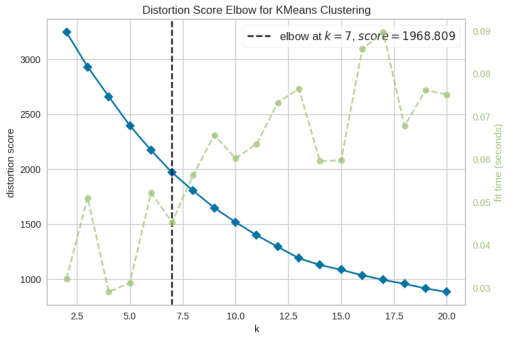
(b) SCAPE



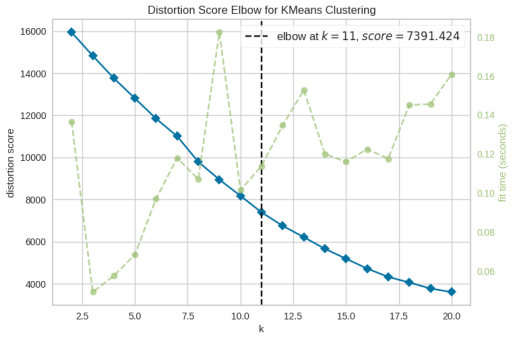
(c) ALIEN



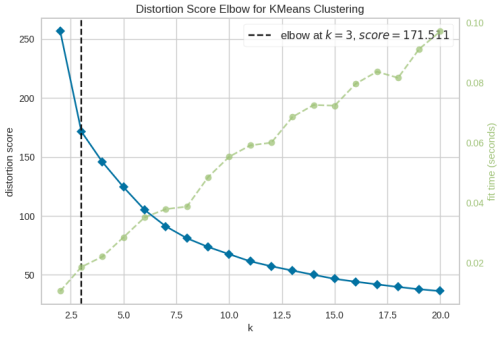
(d) VASE



(e) GUITAR



(f) CHAIR



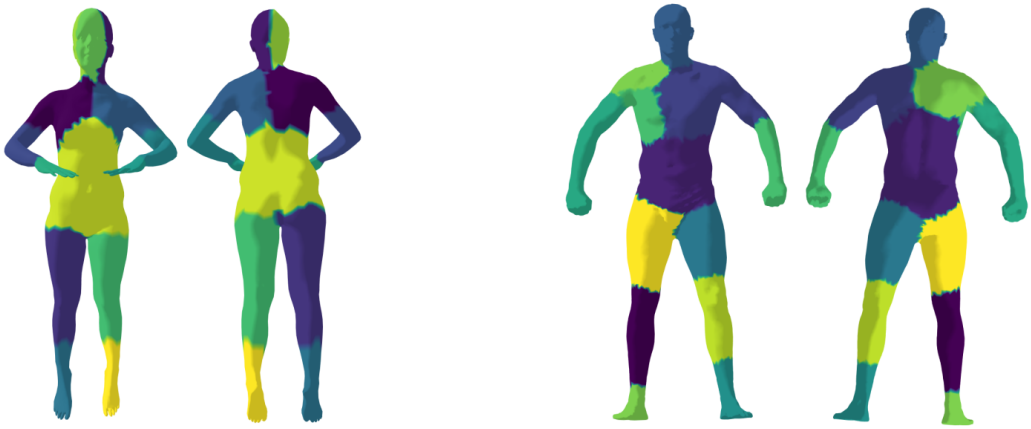
(g) GOBLET

Figure 20: Suggested optimal cluster numbers graphics for each 3D mesh data according to the elbow method

For the nonhuman-shaped objects, we experimented on 5 objects: a goblet, a vase, a guitar, a chair, and an alien. Firstly, the goblet object is segmented into 3 parts which are the inner part of the bowl of the goblet, the outer part of the bowl of the goblet joined with the stem, and the foot, see Fig. 26c. We observed that it is segmented well except for the small noise at the little part of the foot. The outer part of the bowl of the goblet and stem are segmented together because of the mesh structure. Secondly, the vase object was segmented; see Fig. 26d. Each part, like the handle of the cover and the main handle, is smoothly segmented except for the little place where the blue and purple part intersects at the right of the main handle. The other segmented 3D mesh object is a guitar; see Fig. 26e. We observed that it is segmented as symmetrically as well. For example, the upper bound, where the neck and body intersect, is symmetrically segmented at the front and back view. However, in the part where the neck and body touch each other close to the boundary, the neck part protrudes a little towards the upper body. Also, we observe that segmented parts can distinguish the texture difference on the guitar beautifully. Additionally, the model separated the headstock from the neck and found the tuners in the headstock important to separate. The next object is a chair which has a shape like an office chair; see Fig. 26f. The model segmented unique foot parts well. The seat was separated into two symmetrical parts. However, there is a flow from the part where the back and the piece between the seat and the back to the back part. The last non-human-shaped 3D mesh object is an alien. Alien is described as one that has many extremities and unusual eyes; see Fig. 26g. Our model segmented extremities as unique parts. Also, an eye is segmented separately from the eyeball, which is segmented into three parts, and the body is segmented as one. To sum up, our data-centric 3D mesh segmentation model works well for different kinds of objects where their structures are different.

So far, the 3D mesh data objects were segmented by applying the K-Means clustering algorithm with the K-Means++ initialization method to embedding vector representation obtained with the node2vec algorithm. We also experimented segmentation model with random initialization to see the performance. We evaluated two numeric results to compare the models with different initialization methods. The first one is evaluating the Euclidean distance between the initial centroid and final centroid locations. The purpose is to detect which model has an optimum displacement. The smallest distance is an indicator of successful initialization since after initializing the first centroid, the probability distribution is likely to place the new centroid for another cluster by regarding each point's squared distances, preventing the K-Means++ from stuck into local optima. In other words, K-Means++ considers the distance between initial centroids, unlike random initialization. We showed the numerical results in Table 4. We observed that results are different for different 3D mesh data objects. While FAUST, SCAPE, chair, and goblet types' initial and final centroid's distance is smaller for the K-Means++ initialization, alien, guitar, and vase produce opposite results. The second one is inertia results at k, which is determined by the elbow method for each data type. Inertia value can help to evaluate the results since the value is calculated by summing the squared distances between cluster centroids and cluster points. If the cluster centroids are positioned better, clustering can be more successful. In Table 5, inertia results for each data type for random and K-Means++ initialization are calculated. We observed that K-Means++ initialization's inertia values return smaller than random initialization. Thus, according to inertia results, the K-Means++ initialization method is more precise than random initialization for all data types. All the calculations are made on embedding vector representations of 3D mesh data.

The results of the random and K-Means++ initialization are shown as well as numeric ones visually; see Fig. 22. The left part of each example illustrates the results produced by the K-Means++ initialization, and the right part illustrates the random initialization. Blue dots denote the initial centroid

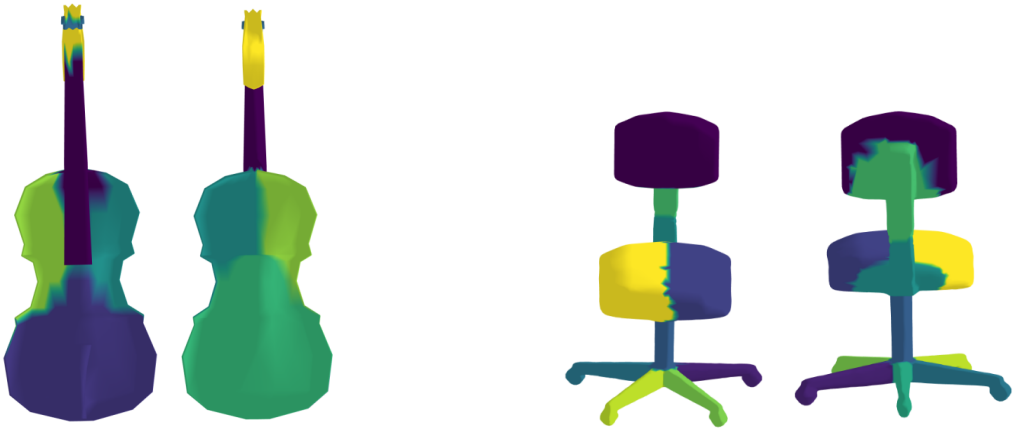


(a) FAUST, k=14

(b) SCAPE, k=11

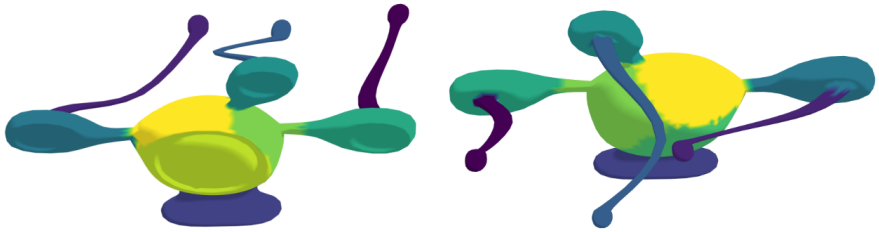
(c) GOBLET, k=3

(d) VASE, k=5



(e) GUITAR, k=7

(f) CHAIR, k=11



(g) ALIEN, k=11

Figure 21: Visual results with k values suggested by elbow method.

positions, and red dots denote the final centroid positions. It can be observed that centroid initialization plays a crucial role in K-Means clustering as well as cluster number and data quality. For all 3D mesh data objects, K-Means++ produces better results in terms of symmetry and distinct parts of the objects like the handle of the vase, the chair's foot, alien's extremities. For FAUST data, random initialization seems to have caused the symmetry to be lost and stuck to the local optima. As it can be seen from Fig. 22a, random initialization centroid positions are close to each other such as centroids on hands or faces. As a result of this initialization, clusters are not correctly segmented. For SCAPE data, similar problems with FAUST data can be observed; see 22b. For guitar data, we observe that only the headstock part seems to be segmented in success; other segmented parts are not meaningful, see Fig. 22c. For the vase data, the handle is segmented well in both initialization methods; however, other parts do not draw a meaningful segmentation result in the random initialization method. The next object is the goblet which is segmented wrongly in the random initialization method; see Fig. 22e. For the chair object, it can be observed that foot pieces can not be segmented into distinct parts in the random initialization method; see Fig. 22f. Also, the back and seat parts of the chair are segmented into not symmetric two pieces. However, the part that connects the back of the chair to the seat is more successfully separated in random initialization than the K-Means++ initialization method. The last object is an alien. Both initialization methods segmented distinct parts of the object; however, random initialization lost the smooth borderlines, such as from eye to body, see Fig. 22g. To sum up, the K-Means++ initialization algorithm is better than random initialization since the distance between initial centroids is distributed more equally than random initialization and protected from local minimums.

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

3D Object type	**Random initialization**	**K-Means++ initialization**
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

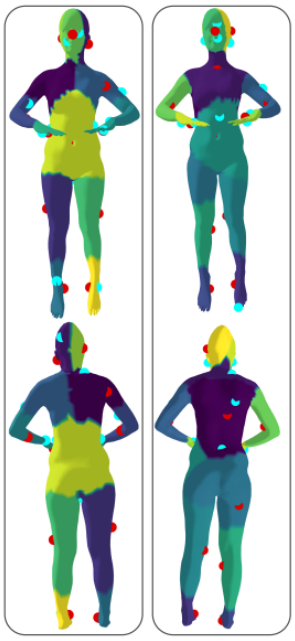
Our purpose is to segment 3D objects into smaller parts. While developing the algorithm and performing the experiments, reducing the train time has been one of our main focuses besides the success of the segmentation. To sum up, our data-centric AI approach has proven its success in performance by demonstrating 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 [12]. The timing of the segmentation algorithm is outlined in seconds in Table 6. In [12], which uses random walk and RNN with labeled data, training time last approximately 12 hours 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 12500 nodes and 24998 faces and FAUST which has 6890 nodes and 13776 faces, the training time of MeshWalker model's is higher than ours. Because they reproduced the data using random walks and they

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

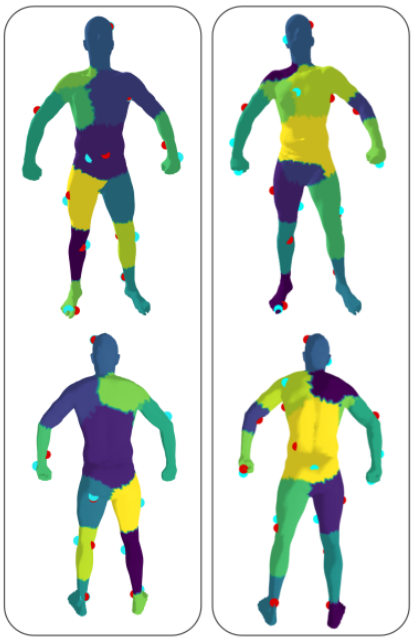
3D Object type	Random initialization	K-Means++ initialization
FAUST	16798	16044
SCAPE	11324	10906
ALIEN	6041	5975
CHAIR	7598	7391
GUITAR	2020	1968
VASE	1939	1924
GOBLET	3231	3109

Table 6: 3D mesh segmentation train and clustering times in seconds. NT: node training, NC: node clustering

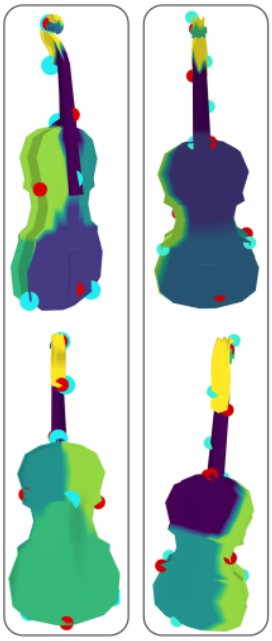
3D Object type	Node training	Node clustering	Total	Node num.
FAUST	91.26	0.19	91.45	6890
SCAPE	97.64	0.16	97.80	12500
ALIEN	24.20	0.08	24.28	4060
CHAIR	21.37	0.10	21.47	5000
GUITAR	13.60	0.04	13.67	1152
VASE	8.65	0.02	13.62	874
GOBLET	7.23	0.03	13.63	1202



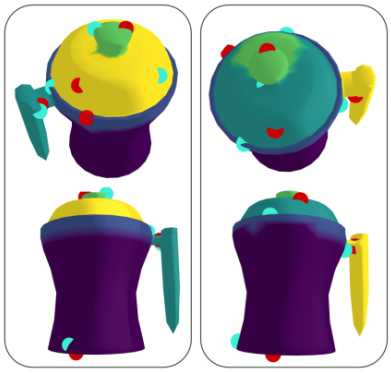
(a) FAUST, k=14



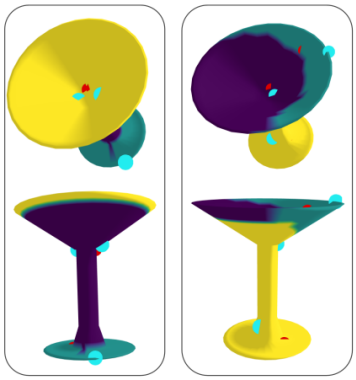
(b) SCAPE, k=11



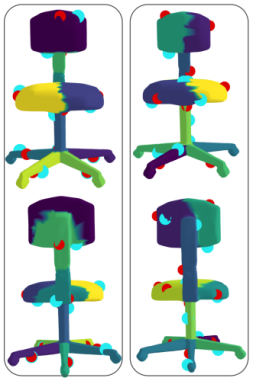
(c) GUITAR, k=7



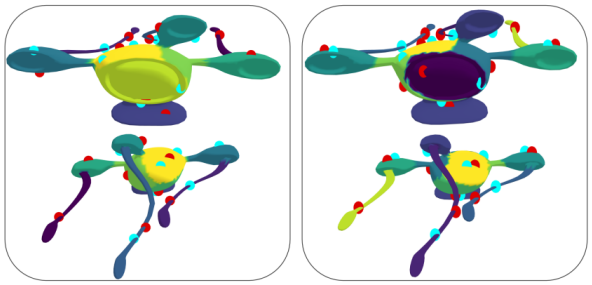
(d) VASE, k=5



(e) GOBLET, k=3



(f) CHAIR, k=11



(g) ALIEN, k=11

Figure 22: Visual results with k values suggested by elbow method and initial and final centroid positions. Left part of the each examples illustrates the results that produced by the K-Means++ initialization and right part illustrates the random initialization. Blue dots denotes the initial centroid positions and red dots denotes final centroids positions.

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.

Furthermore, we perform a segmentation problem focusing on node features, unlike the other implementations [8, 12, 22] perform their algorithms on faces. The authors developed an algorithm for convolution operators that enables performing operations on 3D mesh data in [8]. They used face information to perform segmentation training opposite from us. The authors of [12] designed their model based on random walks, and they feed the RNN model opposite from us in terms of a data-centric approach. The walk sampling in node2vec is based on two hyperparameters, p, and q, that introduce bias into the walk sampling. We implement the node2vec algorithm as an improvement of the random walk used in [22]. [22] uses fine-scaling to empower segmentation results. However, we did not use an additional scaling algorithm to improve the results. Our algorithm purely depends on embedding quality which is an important issue for a data-centric AI approach. Also, we did not define user-defined ground-truth labels. We developed an algorithm that perform labeling according to embedding vector feature properties and depending on the Euclidean and geodesic distance metrics.

4.2.2.1 Geodesic Inertia

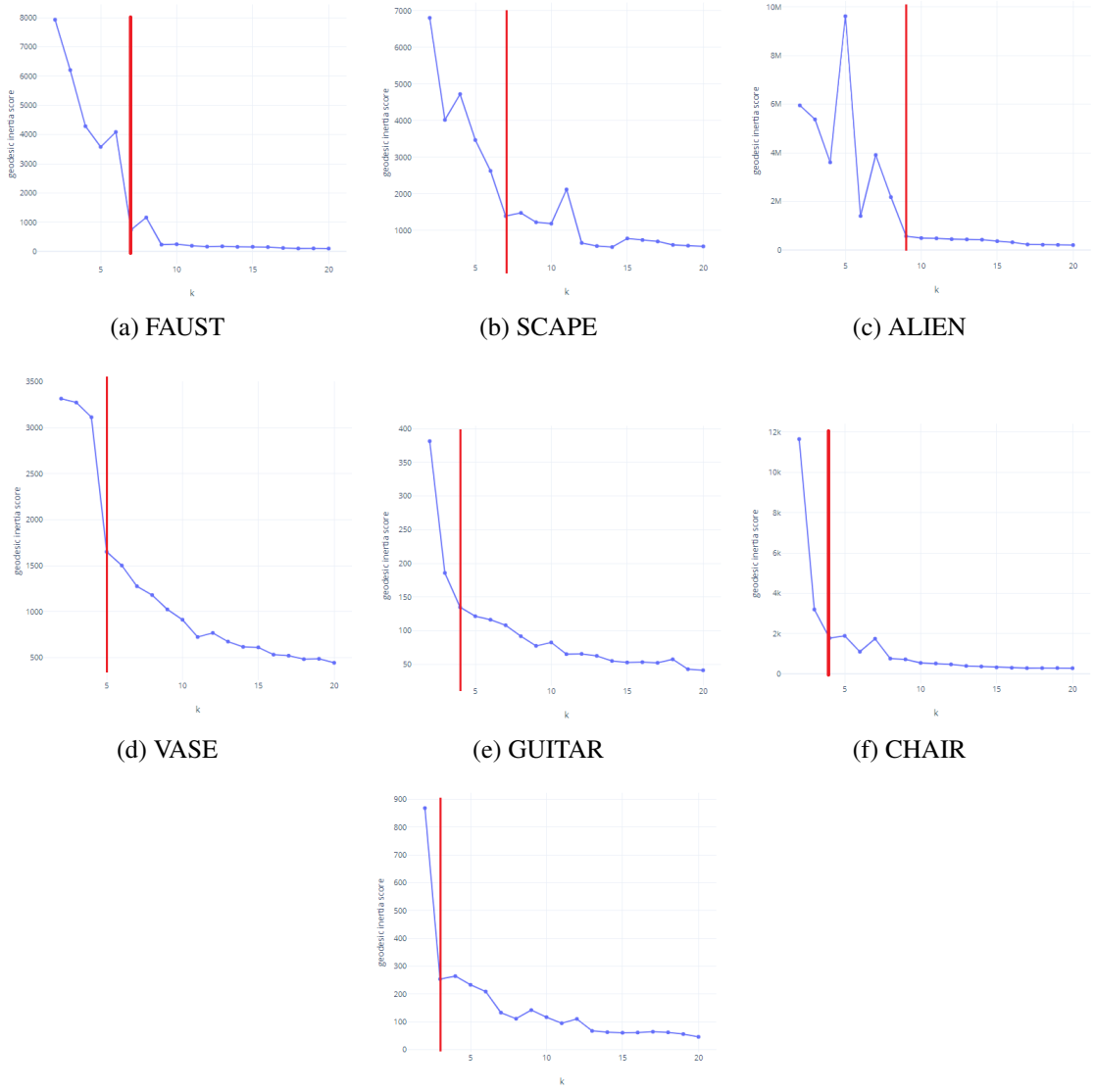
Labeling data is a long and challenging task. Many problems, such as human error, can be encountered during data labeling. At this point, our approach to the mesh segmentation problem is unsupervised. This means that our data does not contain any labeled data or clusters. The user can determine the label or number of clusters according to the need or usage scenario. On the other hand, there are algorithms that try to determine the best number of clusters in unsupervised problems automatically. In our approach, clusters are created with the K-means algorithm between 2 and 20 on the embedding vector representations trained with 3D mesh data. Then the inertia value is checked to choose the best number of clusters. At this point, instead of the euclidean distance, the geodesic distance method is used when calculating the inertia value. Geodesic inertia is a new method without ignoring the original structure that we developed to evaluate the optimum cluster number according to geodesic distance. While making the experiments, we used the geodesic library [82], which is developed for performing functions related to the 3D mesh data. According to the results that we obtained with the geodesic inertia, it returns different elbow results than the Euclidean distance; see Table 7. For FAUST and SCAPE datasets, the optimal cluster number is 7. Also, it returns 9 for the alien, 4 for the chair and guitar, 5 for the vase, and 3 for the goblet. Although the results are different for most types, the vase and goblet's results are the same with Euclidean inertia, which is evaluated on the node embedding vectors. Moreover, it can be observable that geodesic inertia returns more general segmentation results than the inertia results of Euclidean distance. For example, it evaluates the legs as one segment and the head as one. Also, for the guitar object, it segmented the most base parts. Additionally, symmetrical and meaningful results are obtained in both methods; it can be said that the embedding vector representation we obtained using node2vec reflects the raw data well. Lastly, it is easy to distinguish the elbow point on the geodesic inertia plot than on the classical inertia plot.

Figure 23: Our 3D mesh segmentation results on FAUST dataset for different people and poses, k=14.

Table 7: The optimal cluster numbers according to metrics, Euclidean and geodesic distance

3D Object type	**Euclidean inertia, k**	**Geodesic inertia, k**
FAUST	14	7
SCAPE	11	7
ALIEN	11	9
CHAIR	11	4
GUITAR	7	4
VASE	5	5
GOBLET	3	3

Figure 24: Our 3D mesh segmentation results on SCAPE dataset showed in 8 unique poses with k=11.



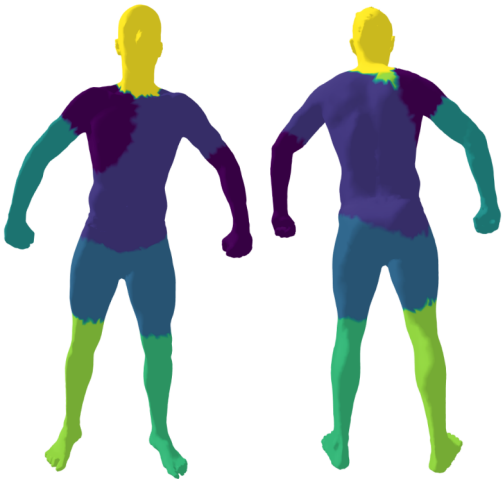
(a) FAUST (b) SCAPE (c) ALIEN

(d) VASE (e) GUITAR (f) CHAIR

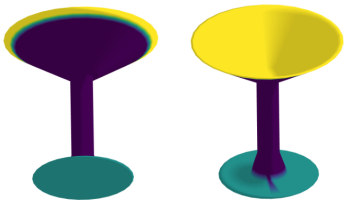
(g) GOBLET

Figure 25: Suggested optimal cluster numbers graphics for each 3D mesh data according to the elbow method applied on geodesic distance

(a) FAUST, k=7

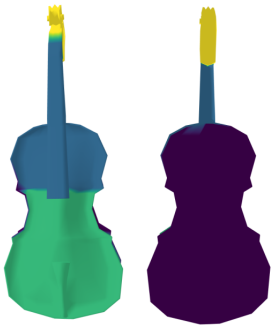


(b) SCAPE, k=7

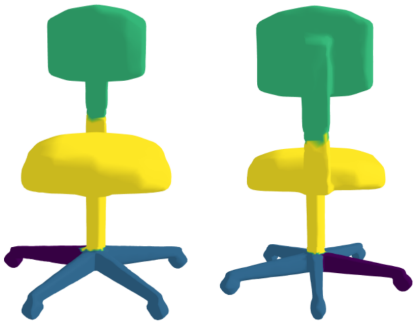


(c) GOBLET, k=3

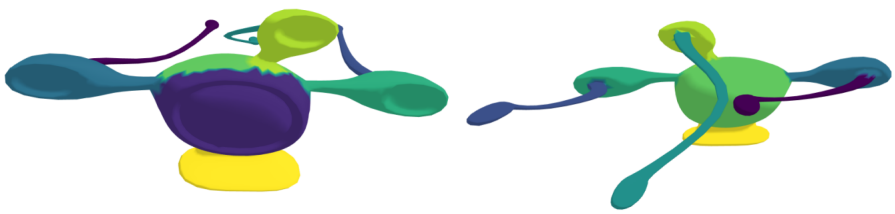
(d) VASE, k=5



(e) GUITAR, k=4



(f) CHAIR, k=4



(g) ALIEN, k=9

Figure 26: Visual results with k values suggested by elbow method.

4.3 Limitations

Our proposed algorithm has produced promising results. However, it has some limitations, like all other machine learning and deep learning models. For example, there are some overflows at the part which connects the chair's seat and back to the chair's back, see Fig. 26f. Also, it missegmented the SCAPE data's left shoulder, see Fig. 26b and led to the broken symmetry for the shoulders.

CHAPTER 5

CONCLUSIONS AND FUTURE WORKS

In this thesis, we worked on segmenting 3D objects using a data-centric and unsupervised approach. Since 3D mesh objects are not ready to use for the model, it is an essential step to map the data to vector representations using mesh processing algorithms. In this work, we applied a unique way of mapping the 3D mesh data to a vector representation which is node2vec. To our best knowledge, it is the first study using the node2vec algorithm. The benefit of this algorithm is using biased random walks. Thus, the discovery of mesh structure is controllable by return and in-out parameters and collecting the local or global information. Moreover, it does not require any label or ground truth information since it is semi-supervised learning based on the word2vec algorithm, which has proven its success on words. Additionally, the embedding vectors are trained using a simple deep-learning structure in which the embedding vector size hyperparameter determines a single hidden layer. Hence, the computation cost is very low according to more complex models. It provides a data-centric approach which helps to reduce energy requirements either.

While training the node embedding vector, we do not have to compare the results according to the ground truth since we study an unsupervised learning model. Thus, we adapt the cosine similarity algorithm to find the similarity value between the neighboring nodes. The results are a helper to find how well the hyperparameters of biased random walk and the skip-gram architecture represent the 3D mesh data. According to our experiments, we obtained similarity score between 0.94 and 0.99 for different data objects. Also, we illustrated the embedding vector representations, which give the best and worst results.

After selecting the embedding vectors for each data object according to cosine similarity values, the clustering algorithm, K-Means applied, and the optimal k values are found for each. Since there are many metrics or indexes to find the optimal cluster number, the elbow method is used. The other metrics or indexes produce results that are based on the relationships between intra cluster and outer clusters. However, 3D mesh datasets have a structure that borders are really close to each other. After deciding the optimal k values for each data object with the elbow method, the results are obtained and illustrated. While experimenting with the clustering, we perform two different initialization methods: K-Means++ and random initialization. According to the results we obtained, K-Means++ initialization results are better than the random initialization method since K-Means++ initialization uses the trick that the initial centroids are located fairly equidistant from each other.

Since the elbow methods produce good and meaningful results evaluated on embedding vectors for the 3D mesh segmentation method, we developed a new method for deciding the optimal cluster number, which is evaluated on 3D mesh data using geodesic distance. We adapt to the sum of the squared

distance between cluster centroid and cluster members, using geodesic distance. We experimented with the method, the cluster number from 2 to 20. According to the results, we concluded that the new method focuses on more general structures than the classical method. For example, while the leg is segmented into 3 parts in the classical method, the geodesic method segmented a leg as one part. Additionally, detecting the elbow point is easier than the classical method. It can be helpful when the elbow is difficult to see.

To sum up, we developed and applied a novel unsupervised method for the 3D mesh segmentation problem. It is a data-centric model which focuses on data, not the model power, and helps to reduce the curse of the dimensionality using the node2vec algorithm. For instance, we reduce the dimension for FAUST from 6890 to 40, for SCAPE from 12500 to 20, for the alien from 4060 to 20, for the vase from 874 to 20, for the guitar from 1152 to 18, for the chair from 5000 to 25, and for the goblet from 1202 to 20. This approach benefits in many areas, such as lower energy consumption, easiness of computation, and easy-to-perform hyperparameter tuning. We experimented with the algorithm on different datasets, which are human-shaped (SCAPE, FAUST) and nonhuman shape datasets (COSEG). In real life, most data exist unlabeled, and it is crucial to be able to use this unlabeled data. Can be eliminated the labeling cost or effort can be using our algorithm. This novel data-centric approach reduced the need for high-power GPU and computation power. In addition, complex models have too many hyperparameters, and finding the right combination is a painstaking task. At this point, using a simpler model like K-Means with high interpretability brings the data-centric approach one step forward in generalization and easier tuning. The data-centric approach is becoming essential and valuable day by day. At this point, when we look at computation cost, today's GPUs consume much power and contradict the Green AI concept. We benefited from the data-centric approach's computation cost and simple model outputs, which AI pioneers also emphasized.

5.1 Future Work

Since we need more computation power and the hardware is insufficient at some points, and we always need more computation power, efforts to optimize the models' training time will continue. At this point, the data-centric AI approach seems to become even more critical in the future. Efforts can be made to improve the node2vec embedding algorithm or work on new embedding algorithms. In addition, special transformation or preprocess techniques can be applied to the data in the border region to make the distinctions at the intersections of the cluster's sharper. Moreover, the created embeddings can be transferred to the latent space with the autoencoder architecture, and the modeling phase can be started over the latent space. In this way, we can store core information contained in embeddings and reduce the dimension further. Additionally, we can implement different datasets, including point clouds and classes.

REFERENCES

[1] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, 2018.

[2] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 855–864, Association for Computing Machinery, 2016.

[3] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Feb. 2016.

[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[5] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, 2015.

[6] R. S. Rodrigues, J. F. Morgado, and A. J. Gomes, "Part-based mesh segmentation: A survey," *Computer Graphics Forum*, vol. 37, 2018.

[7] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," vol. 22, p. 954–961, 2003.

[8] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "Meshcnn: A network with an edge," *ACM Transactions on Graphics*, vol. 38, pp. 1–12, 2019.

[9] D. Khattab, H. M. Ebeid, A. S. Hussein, and M. F. Tolba, "3d mesh segmentation based on unsupervised clustering," *Advances in Intelligent Systems and Computing*, vol. 533, 2017.

[10] R. Verdecchia, L. Cruz, J. Sallou, M. Lin, J. Wickenden, and E. Hotellier, "Data-centric green ai an exploratory empirical study," in *2022 International Conference on ICT for Sustainability (ICT4S)*, pp. 35–45, 2022.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436–444, 2015.

[12] A. Lahav and A. Tal, "Meshwalker: Deep mesh understanding by random walks," *ACM Transactions on Graphics*, vol. 39, pp. 1–13, 2020.

[13] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[14] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in Neural Information Processing Systems* (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), vol. 14, MIT Press, 2001.

[15] A. Sundaresan and R. Chellappa, "Model driven segmentation of articulating humans in laplacian eigenspace," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 10, pp. 1771–1785, 2008.

[16] D. Luo, C. Ding, F. Nie, and H. Huang, "Cauchy graph embedding," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pp. 553–560, 2011. Cited By :87.

[17] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, (New York, NY, USA), p. 37–48, Association for Computing Machinery, 2013.

[18] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," CIKM '15, (New York, NY, USA), p. 891–900, Association for Computing Machinery, 2015.

[19] M. J. Newman, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, no. 1, pp. 39–54, 2005.

[20] F. Fouss, A. Pirotte, J.-m. Renders, and M. Saerens, "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 3, pp. 355–369, 2007.

[21] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, (New York, NY, USA), p. 701–710, Association for Computing Machinery, 2014.

[22] Y. K. Lai, S. M. Hu, R. R. Martin, and P. L. Rosin, "Fast mesh segmentation using random walks," 2008.

[23] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "HARP: hierarchical representation learning for networks," *CoRR*, vol. abs/1706.07845, 2017.

[24] B. Perozzi, V. Kulkarni, and S. Skiena, "Walklets: Multiscale graph embeddings for interpretable network classification," *CoRR*, vol. abs/1605.02115, 2016.

[25] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 1225–1234, Association for Computing Machinery, 2016.

[26] O. I. Sever, "Mesh segmentation from sparse face labels using graph convolutional neural networks.," Master's thesis, Middle East Technical University, 2020.

[27] E. K. Perek, "Supervised mesh segmentation for 3d objects with graph convolutional neural networks," Master's thesis, Middle East Technical University, 2019.

[28] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016.

[29] Z. Ghahramani, "Unsupervised learning," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3176, pp. 72–112, 2004.

[30] J. B. MacQueen, "Kmeans and analysis of multivariate observations," *5th Berkeley Symposium on Mathematical Statistics and Probability 1967*, vol. 1, p. 281–297, 1967.

[31] H. S. Park and C. H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert Systems with Applications*, vol. 36, p. 3336–3341, 2009.

[32] L. Kaufman and P. J. Rousseeuw, "Partitioning around medoids (program pam).finding groups in data: An introduction to cluster analysis (wiley series in probability and statistics)," *Eepe.Ethz.Ch*, vol. 66, 1990.

[33] L. Kaufman and P. J. Rousseeuw, "Finding groups in data: An introduction to cluster analysis (wiley series in probability and statistics)," *Eepe.Ethz.Ch*, vol. 344, 2008.

[34] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, p. 241–254, 1967.

[35] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 25, p. 103–104, 1996.

[36] S. Guha, R. Rastogi, and K. Shim, "Cure: An efficient clustering algorithm for large databases," *SIGMOD Record*, vol. 27, p. 73–84, 1998.

[37] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. 1981.

[38] R. N. Dave and K. Bhaswan, "Adaptive fuzzy c-shells clustering and detection of ellipses," *IEEE Transactions on Neural Networks*, vol. 3, p. 643–662, 1992.

[39] X. Xu, M. Ester, H. P. Kriegel, and J. Sander, "Distribution-based clustering algorithm for mining in large spatial databases," *Proceedings - International Conference on Data Engineering*, pp. 324–331, 1998.

[40] C. E. Rasmussen, "The infinite gaussian mixture model," *Advances in Neural Information Processing Systems*, p. 554–560, 2000.

[41] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," p. 226–231, 1996.

[42] M. Ankerst, M. M. Breunig, H. P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 28, p. 49–60, 1999.

[43] R. Sharan and R. Shamir, "Click: a clustering algorithm with applications to gene expression analysis.," *Proceedings / ... International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology*, vol. 8, pp. 307–316, 2000.

[44] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," vol. 31, p. 264–323, 1999.

[45] W. Wang, J. Yang, and R. Muntz, "Sting : A statistical information grid approach to spatial data mining," p. 186–195, 1997.

[46] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," *SIGMOD Record*, vol. 27, pp. 94–105, 1998.

[47] D. Barbará and P. Chen, "Using the fractal dimension to cluster datasets," p. 260–264, 2000.

[48] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine Learning*, vol. 2, p. 139–172, 1987.

[49] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, p. 1464–1480, 1998.

[50] G. A. Carpenter and S. Grossberg, "The art of adaptive pattern recognition by a self-organizing neural network," *Computer*, vol. 21, pp. 77–88, 1988.

[51] A. Shamir, "A survey on mesh segmentation techniques," *Computer Graphics Forum*, vol. 27, p. 1539–1556, 2008.

[52] P. Theologou, I. Pratikakis, and T. Theoharis, "A comprehensive overview of methodologies and performance evaluation frameworks in 3d mesh segmentation," *Computer Vision and Image Understanding*, vol. 135, p. 49–82, 2015.

[53] K. Wu, "A review on mesh segmentation techniques," *Certified International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 9001, 2008.

[54] B. Chazelle, "Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm.," *SIAM Journal on Computing*, vol. 13, p. 488–507, 1984.

[55] V. Kreavoy, A. Sheffer, and D. J, "Shuffler: Modeling with interchangeable parts," *Proceedings - IEEE International Conference on Shape Modeling and Applications 2006, SMI 2006*, vol. 2007, 2007.

[56] H. Liu, W. Liu, and L. J. Latecki, "Convex shape decomposition," p. 97–104, 2010.

[57] M. Attene, M. Mortara, M. Spagnuolo, and B. Falcidieno, "Hierarchical convex approximation of 3d shapes for fast region selection," *Computer Graphics Forum*, vol. 27, p. 1323–1332, 2008.

[58] C. Xian, S. Gao, and T. Zhang, "An approach to automated decomposition of volumetric mesh," vol. 35, p. 461–470, 2011.

[59] P. Simari, D. Nowrouzezahrai, E. Kalogerakis, and K. Singh, "Multi-objective shape segmentation and labeling," *Computer Graphics Forum*, vol. 28, 2009.

[60] G. Lavoué and C. Wolf, "Markov random fields for improving 3d mesh analysis and segmentation," p. 25–32, 2008.

[61] A. P. Mangan and R. T. Whitaker, "Partitioning 3d surface meshes using watershed segmentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, pp. 308,321, 1999.

[62] R. Liu and H. Zhang, "Segmentation of 3d meshes through spectral clustering," p. 298–305, 2004.

[63] H. Y. S. Lin, H. Y. M. Liao, and J. C. Lin, "Visual salience-guided mesh decomposition," 2007.

[64] J. Aleotti and S. Caselli, "A 3d shape segmentation approach for robot grasping by parts," *Robotics and Autonomous Systems*, vol. 60, p. 358–366, 2012.

[65] M. Mortara, G. Patané, and M. Spagnuolo, "From geometric to semantic human body models," *Computers and Graphics (Pergamon)*, vol. 30, pp. 185,196, 2006.

[66] X. Li, T. W. Woon, T. S. Tan, and Z. Huang, "Decomposing polygon meshes for interactive applications," pp. 35–42, 2001.

[67] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri, "3d shape segmentation with projective convolutional networks," vol. 2017-January, p. 3779–3788, 2017.

[68] A. Torralba, K. P. Murphy, and W. T. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, p. 854–869, 2007.

[69] J. Lv, X. Chen, J. Huangy, and H. Bao, "Semi-supervised mesh segmentation and labeling," vol. 31, p. 2241–2248, 2012.

[70] Z. Shu, X. Shen, S. Xin, Q. Chang, J. Feng, L. Kavan, and L. Liu, "Scribble-based 3d shape segmentation via weakly-supervised learning," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, 2020.

[71] Z. Shu, S. Yang, H. Wu, S. Xin, C. Pang, L. Kavan, and L. Liu, "3d shape segmentation using soft density peak clustering and semi-supervised learning," *Computer-Aided Design*, vol. 145, p. 103181, 4 2022.

[72] O. Sidi, Y. Kleiman, D. Cohen-Or, O. van Kaick, and H. Zhang, "Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering," *ACM Transactions on Graphics*, vol. 30, 2011.

[73] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[74] M. A. Iqbal, O. Sharif, M. M. Hoque, and I. H. Sarker, "Word embedding based textual semantic similarity measure in bengali," *Procedia Computer Science*, vol. 193, pp. 92–101, 2021. 10th International Young Scientists Conference in Computational Science, YSC2021, 28 June – 2 July, 2021.

[75] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," vol. 07-09-January-2007, 2007.

[76] F. Bogo, J. Romero, M. Loper, and M. J. Black, "Faust: Dataset and evaluation for 3d mesh registration," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[77] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "Scape: Shape completion and animation of people," vol. 24, 2005.

[78] Y. Wang, S. Asafi, O. V. Kaick, H. Zhang, D. Cohen-Or, and B. Chen, "Active co-analysis of a set of shapes," vol. 31, 2012.

[79] M. Fey, J. You, R. Ying, G. Li, J. Sunil, J. E. Lenssen, I. Bahtchevanov, and J. Leskovec, "Pyg." https://www.pyg.org/.

[80] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS 2017 Workshop on Autodiff*, 2017.

[81] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.

[82] M. Hogg, “Pygeodesic.,” May 2021. https://pypi.org/project/pygeodesic/.