

UC Davis

UC Davis Electronic Theses and Dissertations

Title

Multimedia Signal Processing based on Advanced Graph Approaches

Permalink

<https://escholarship.org/uc/item/35t9b081>

Author

Deng, Qinwen

Publication Date

2024

Peer reviewed|Thesis/dissertation

Multimedia Signal Processing based on Advanced Graph Approaches

By

QINWEN DENG

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Zhi Ding, Chair

Lifeng Lai

Khaled Abdel-Ghaffar

Committee in Charge

2024

Abstract

When dealing with data residing on irregular and complex structures, graph signal processing (GSP) offers the ability to model and process them directly. While most existing GSP methods use graphs and edges to model pairwise relationships of such data, practical data may reside on more complex structures with multilateral interactions, which motivates processing methods with high-dimensional graphs like hypergraphs and multilayer graphs (MLG).

Although both hypergraph signal processing (HGSP) and MLG have enjoyed additional notable successes in segmentation, clustering, and classification, it is not clear whether they can provide an advantage in specific applications such as point cloud resampling and motion segmentation. Also, it is shown that high-dimensional graph processing algorithms with high-dimensional graphs constructed for the whole data are computationally costly, especially when the number of nodes is large.

Efficient processing and feature extraction of large-scale datasets are important in related computer vision and cyber-physical systems. In this dissertation, we investigate two applications for advanced graph approaches. The first application is point cloud resampling based on hypergraph signal processing to better explore the underlying relationship among different points in the point cloud and to extract contour-enhanced features. Specifically, we design hypergraph spectral filters to capture multi-lateral interactions among the signal nodes of point clouds and to better preserve their surface outlines. Without the need and the computation to first construct the underlying hypergraph, our low complexity approach directly estimates the hypergraph spectrum of point clouds by leveraging hypergraph stationary processes from the observed 3D coordinates. The second application is multilayer graph signal processing (M-GSP) based approaches to human motion segmentation. Specifically, our approach involves modeling spatial-temporal relationships of the human motion capture data via MLG and incorporating M-GSP spectrum analysis for feature extraction.

Furthermore, this dissertation optimizes the computational complexity of key algorithms in

graph based spectral analysis and processing methodologies. We propose an incremental EVD algorithm for low rank symmetric matrices (IEVD-LR) to update the top k eigen-pairs of the graph representation matrix. This algorithm has a novel error correction branch whenever the approximation error exceeds the defined tolerance.

Equipped with the techniques presented in this dissertation, we extend the current research on advanced graph based processing to a variety of new directions. Based on our success in applying advanced graph-based processing algorithms to static point clouds and time-varying motion capture data, our exploration can extend to further applications of high-dimensional graph processing techniques on diverse multimedia datasets with varying structures, such as dynamic point clouds. We will also extend the low-complexity eigen-updating algorithm to general dynamic systems, where the number of data points may decrease over time. Another direction is to consider the singular space updating problem for the representation tensor of high-dimensional graphs, initialized by orthogonal CP decomposition or higher-order singular value decomposition result.

Acknowledgement

I would like to extend my deepest gratitude to my advisor, Prof. Zhi Ding, for his unwavering guidance, invaluable insights, and continual support throughout the entirety of my Ph.D. journey. I could not finish my research projects without all his help in selecting research topics, discussing technical solutions, revising manuscripts, and providing financial support. His mentorship and dedication have been instrumental in shaping the direction of my research and fostering my academic growth.

Besides my advisor, I am thankful to the rest members of my dissertation committee, Prof. Lifeng Lai and Prof. Khaled Abdel-Ghaffar, for their constructive feedback and expert evaluation of my work. Their collective expertise has enriched the quality of my research.

A special acknowledgment goes to Prof. Songyang Zhang, whose profound guidance and mentorship have been a beacon of inspiration. Prof. Songyang Zhang has generously shared his wisdom, experience, and insights, playing a pivotal role in steering the course of our research team. His mentorship has been a source of motivation for me and has significantly contributed to the success of this dissertation.

I am indebted to all my friends and colleagues at University of California, Davis, Xiaochuan Ma, Carlos Feres, Achintha Wijesinghe, Suchinthaka Wanninayaka, Zhelun Zhang, Lahiru D. Chamain, Shusen Jing, Weiwei Wang, Chih-ho Hsu, Mason Del Rosario, Yibo Ma, Vincent Cong Vinh Huynh, Yu-Chien Lin, Siyu Qi and Taha Bouchoucha for their camaraderie, insightful discussions, and mutual support. The collaborative spirit and shared dedication to excellence within the department have enhanced the overall research experience on my academic journey. I would like to express my gratitude to all my colleagues and fellow researchers, Yao Ge at Nanyang Technological University, Prof. Yangwen Zhang and Prof. Mo Li at the University of Louisiana at Lafayette for their continuous suggestions and help in my research. Each one of you has played a role in making this journey memorable.

I am deeply thankful to my family for their unwavering encouragement, understanding, and

love. Thank my parents, Hong Liu and Zhihua Deng, for their endless love and steadfast support. Their encouragement has been my pillar of strength throughout this academic endeavor.

This dissertation is a testament to the collective efforts, encouragement, and support from everyone who has been a part of this transformative experience.

Contents

Abstract	ii
Acknowledgement	iv
List of Symbols	xv
1 Introduction	1
1.1 Background	1
1.2 Overview of Advanced Graph Approaches	3
1.3 Motivation and Challenges	4
1.4 Objective of the Dissertation	5
1.5 Significance and Contributions	6
1.6 Structure of the Dissertation	7
2 Introduction to advanced graph signal processing	9
2.1 Graph Signal Processing	9
2.2 Hypergraph Signal Processing	13
2.3 Multilayer Graph Signal Processing	15
2.4 Conclusion	18
3 An Efficient Hypergraph Approach to Robust Point Cloud Resampling	19
3.1 Introduction	20
3.2 HGSP Point Cloud Resampling	23

3.2.1	Hypergraph Kernel Convolution (HKC) based Method	24
3.2.2	Hypergraph Kernel Filtering (HKF) Resampling	29
3.2.3	Local Hypergraph Filtering (LHF) Algorithm	30
3.2.4	Discussion	34
3.3	Experimental results	35
3.3.1	Edge Preservation of Simple Synthetic Point Clouds	35
3.3.2	Edge Preservation Results on Real-Life Point Clouds	38
3.3.3	Point Cloud Recovery from Resampling	39
3.3.4	Runtime of all methods	47
3.3.5	Parameter selection	49
3.4	Conclusion	50
4	Body Motion Segmentation via Multilayer Graph Processing	51
4.1	Introduction	51
4.2	Related Works	54
4.2.1	Unsupervised motion clustering	55
4.2.2	Supervised Motion Recognition	56
4.3	Problem Description and Modeling	58
4.3.1	Problem Description	58
4.3.2	Multilayer Graph Construction	59
4.4	M-GSP Body Motion Segmentation	61
4.4.1	Spectrum-based MLG Motion Segmentation	62
4.4.2	Vertex-based MLG Segmentation	65
4.5	Experiments	67
4.5.1	Dataset	67
4.5.2	Unsupervised Motion Segmentation	69
4.5.3	Supervised Motion Recognition	72
4.5.4	Ablation study on window size	73

4.5.5	Robustness	75
4.6	Conclusions	77
5	Efficient Eigenvalue Decomposition for Low-Rank Symmetric Matrices in Graph Sig- nal Processing	78
5.1	Introduction	78
5.2	Related Works	81
5.2.1	Fast Matrix Decomposition	82
5.3	Method and Analysis	83
5.3.1	Basic Incremental eigenvalue decomposition	83
5.3.2	Fast Incremental eigenvalue decomposition	88
5.4	Experiments	93
5.4.1	Synthetic dataset	94
5.4.2	Real Multimedia Datasets	97
5.5	Conclusion	108
6	Summary and Future Direction	109
6.1	Summary of Key Findings	109
6.2	Future Directions	110

List of Figures

1.1	Example of multilayer graph built on the motion capture data.	1
1.2	Example of graph and hypergraph for 3D point cloud representation: (a) original point cloud with nodes at the joint of body, (b) graph built on the nodes, (c) hypergraph built on the nodes.	2
1.3	Diagram of the relationship between main chapters in this dissertation.	5
2.1	Example of a graph constructed on point cloud.	10
2.2	Example of a hypergraph constructed on point cloud.	14
2.3	Example of hypergraph constructed on motion sequence.	16
3.1	Example of Contour-Enhanced Resampling: (a) original point cloud with 272705 points, and (b) resampled building based on the proposed local hypergraph filtering with 20% samples.	22
3.2	Example of Convolution Kernels.	24
3.3	An example of local signal $s_{\mathcal{L},i}$: Points in the middle layer are marked by blue circles and points in the back layer are marked by red squares. The voxels of slicing block are delineated by the dashed black lines. There is at most one point in each voxel in this example, such that $s_{\mathcal{L},i}(n) = 1$ or 0, respectively, depending on whether the n -th voxel contains a point.	25
3.4	Resampled Results of Dragon Using Local Hypergraph Filtering based Method. . .	31
3.5	Synthetic Point Clouds with Labeled Edge.	37

3.6	Examples of Edge Detection for Realistic Practical Point Clouds.	38
3.7	Examples of Edge Detection for Boxer Point Cloud in 8iVSLF Dataset	39
3.8	Resampled results using ScanLAB Projects: Bi-plane point cloud data set	39
3.9	Example of original point cloud, resampled point cloud and the recovered point cloud for HKC method.	40
3.10	Example of resampled results of EA with different α	43
3.11	Example of resampled results of PCA-AC with different α	43
3.12	Plots of recovered accuracy against resampling ratio α of all methods in ShapeNet Dataset	46
3.12	Continue of the Plots of recovered accuracy against resampling ratio α of all methods in ShapeNet Dataset	47
3.13	Resampling result of PCA-AC method using resampled Boxer Point cloud.	47
3.14	Runtime of all methods	48
4.1	Illustration of Human Motion: (a) Example of Motion Segmentation; (b) Spatial-Temporal Relationships Modeled By Multilayer Graph.	52
4.2	Example of Skeleton-based Human Motion Dataset in CMU graphics lab motion capture database.	57
4.3	Example of MLG model for one motion sequence.	58
4.4	Example of window cuts in motion sequence and the multilayer network construction.	61
4.5	Block Diagram of Spectrum-based MLG Motion Segmentation. The diagram is broken into two parts for clearer view, where the overlapping elements in two parts are labeled in green.	62
4.6	Block Diagram of Vertex-based MLG Motion Segmentation. The diagram is broken into two parts for clearer view, where the overlapping elements in two parts are labeled in green.	67
4.7	Example of the location of inertial measurement units in HuGaDB.	68

4.8	Segmentation result on CMU trial 86.	70
4.9	Example of the similarity matrix for different window sizes on HuGaDB dataset with GYRO data.	73
4.10	Segmentation result on HuGaDB trial 01 data 00.	74
4.11	Average accuracy for different window sizes.	75
4.12	Average accuracy on CMU trial 86 with different noisy frame ratio. (a) $\sigma = 0.1$; (b) $\sigma = 0.2$	76
5.1	Example of the dynamic graph at time t and time $t + 1$. The additional vertices and edges at time $t + 1$ are labeled in red and red dash lines, respectively.	79
5.2	Example of iterative update of our IEVD-LR algorithm. \mathbf{D} is the original low rank symmetric matrix whose EVD matrices are known. In each iteration of the IEVD-LR algorithm, one new pair of α_i and d_i are used as the inputs of the algorithm to update the EVD result.	84
5.3	Estimation errors of the EVD algorithm on synthetic dataset. (a) Average estimation errors of eigenvector matrix \mathbf{Q} , measured by Frobenius norm. (b) Average estimation errors of eigenvector matrix Σ , measured by L_2 norm.	95
5.4	Average runtime of the EVD algorithm on synthetic dataset.	96
5.5	Example of dynamic point cloud with additional nodes in successive frames. (a) Example of dynamic point cloud, the points on the right foot does not captured by the sensors in the first frame. (b) Example of the graphs built on the dynamic point cloud, the additional points in the second frame resulting in the addition of nodes and edges in the new graph, corresponding to the additional rows and columns in the new adjacency matrix.	99

5.6	Example of spectral clustering result in our experiments. (a) Clustering result of the original point cloud using the ground truth EVD result, updated points are labeled in red. (b) Clustering result of the updated point cloud using ground truth. (c) Clustering result of the updated point cloud using estimated EVD results from IEVD-LR-FAST algorithm, clustering errors are marked in pink.	101
5.7	Boxer point cloud in 8i Voxelized Surface Light Field (8iVSLF) Dataset.	102
5.8	Peak memory usage of the EVD algorithms on Boxer point cloud.	103
5.9	Example of motion capture data.	104
5.10	Example of the hyperspectral image with RGB colors.	106

List of Tables

3.1	Numerical results of methods using point clouds of all shapes.	35
3.2	Example of original and resampled point clouds with resampling ratio $\alpha = 0.2$. . .	41
3.3	Example of original and recovered point clouds with resampling ratio $\alpha = 0.2$. . .	42
3.4	Mean distance between the best recovered point cloud and the original point cloud for resampling ratio $\alpha = 0.2$ using ShapeNet dataset.	44
3.5	Average number of points within d_θ between the best recovered point cloud and the original point cloud for resampling ratio $\alpha = 0.2$ using ShapeNet dataset. . . .	44
3.6	Mean distance between the best recovered point cloud and the original point cloud for different resampling ratios using downsampled Boxer point cloud in 8iVSLF dataset.	44
3.7	Average of distance and dual distance between the best recovered point cloud and the original point cloud for different resampling ratios using downsampled Boxer point cloud in 8iVSLF dataset.	45
3.8	Comparison of average running time (in seconds) for point clouds in different datasets.	48
3.9	Mean Distance of Different Methods.	49
3.10	Robustness over Different Parameters: Edge Detection for Synthetic Dataset and Recovery for Realistic Dataset.	49
4.1	Accuracy of motion segmentation on CMU 86 dataset	69
4.2	Average accuracy of motion segmentation on HuGaDB dataset	72

4.3	Average accuracy on HuGaDB dataset with both accelerometer and gyroscope data.	72
5.1	Average clustering errors and low-pass filter errors on dynamic point clouds	100
5.2	Average clustering errors and low-pass filter errors on motion capture data	106
5.3	Average clustering errors and low-pass filter errors on hyperspectral images	108

List of Symbols

\mathcal{G}	graph
\mathcal{V}	set of vertices of the graph
v_i	i -th vertex of the graph
\mathcal{E}	set of edges of the graph
$\mathbf{F}_{\mathcal{G}}$	representation matrix for graph
$\mathbf{A}_{\mathcal{G}}$	adjacency matrix for graph
$\mathbf{L}_{\mathcal{G}}$	Laplacian matrix for graph
$\mathbf{L}_{\mathcal{G},norm}$	normalized Laplacian matrix for graph
$\mathbf{s}_{\mathcal{G}}$	graph signal
$\hat{\mathbf{s}}_{\mathcal{G}}$	spectral domain graph signal
$\tilde{\mathbf{s}}_{\mathcal{G}}$	filtered spatial domain graph signal
$s_{\mathcal{G},i}$	i -th element of graph signal $\mathbf{s}_{\mathcal{G}}$
ϵ	distance threshold to determine the neighborhood for ϵ -neighborhood graph
d_r	intrinsic resolution of dataset
$A_{\mathcal{G},ij}$	element at the i -th row and j -th column of matrix $\mathbf{A}_{\mathcal{G}}$
\mathbf{a}_i	vector of data attributes for the i -th vertex
\mathbf{Q}	orthonormal matrix whose columns are the eigenvectors
\mathbf{Q}^{-1}	inverse matrix of \mathbf{Q}
\mathbf{Q}^{\top}	transpose of \mathbf{Q}
\mathbf{Q}^H	Hermitian transpose of \mathbf{Q}

Σ	diagonal matrix whose diagonal elements are the eigenvalues
$\lambda_{\mathcal{G},i}$	i -th eigenvalue of representation matrix of graph
\mathbf{I}	identity matrix
$h(\mathbf{A}_{\mathcal{G}})$	shift-invariant graph filter
$\mathbf{A}_{\mathcal{H}}$	representation tensor for hypergraph
$A_{\mathcal{H},i_1 i_2 \dots i_M}$	entry located at the indices i_1, i_2, \dots, i_M in the M -th order tensor $\mathbf{A}_{\mathcal{H}}$
$\lambda_{\mathcal{H},i}$	i -th spectrum coefficient corresponding to the i -th basis for hypergraph \mathcal{H}
$\mathbf{f}_{\mathcal{H},i}$	i -th orthonormal basis for hypergraph representation tensor
$\mathbf{V}_{\mathcal{H}}$	hypergraph spectrum basis matrix
$\mathbf{s}_{\mathcal{H}}$	hypergraph signal
$s_{\mathcal{H},i}$	i -th element of hypergraph signal $\mathbf{s}_{\mathcal{H}}$
$\mathcal{F}_{\mathcal{H}}$	hypergraph Fourier transform
$\mathcal{F}_{\mathcal{H}}^{-1}$	inverse hypergraph Fourier transform
\circ	tensor outer product
$\mathbf{A}_{\mathcal{M}}$	representation tensor for multilayer graph
u_i	i -th entity in multilayer graph
ℓ_{α}	α -th layer in multilayer graph
$\mathbf{f}_{\mathcal{M},\alpha}$	α -th orthonormal basis for multilayer graph representation tensor characterizing the properties of layers
$\mathbf{e}_{\mathcal{M},i}$	i -th orthonormal basis for multilayer graph representation tensor characterizing the properties of entities
$\lambda_{\mathcal{M},\alpha i}$	spectrum coefficient corresponding to the basis of α -th layer and the i -th entity
\times_n	n -mode product
\mathbf{S}	core tensor of higher-order singular value decomposition
$\mathbf{U}^{(n)}$	n -th unitary matrix containing bases out of higher-order singular value decomposition
\mathbf{W}_f	orthonormal matrix containing all basis for multilayer graph representation tensor characterizing the properties of layers

\mathbf{W}_e	orthonormal matrix containing all basis for multilayer graph representation tensor characterizing the properties of entities
$\mathbf{s}_{\mathcal{M}}$	multilayer graph signal
$s_{\mathcal{M},\alpha i}$	element corresponds to α -th layer and i -th entity in multilayer graph signal
$\check{\mathbf{s}}_{\mathcal{M},L}$	layer-wise multilayer graph singular transformed (M-GST) signal
$\check{\mathbf{s}}_{\mathcal{M},N}$	entity-wise multilayer graph singular transformed (M-GST) signal
$\check{\mathbf{s}}_{\mathcal{M}}$	joint multilayer graph singular transformed (M-GST) signal
\mathbf{P}	location matrix containing 3D coordinates of points in a point cloud
\mathbf{p}_i	i -th point's coordinates in point cloud
\mathbf{X}_i	i -th coordinates of all data points in point cloud
N_k	number of voxels in the slicing block
$\mathbf{s}_{\mathcal{L},i}$	local signal around i -th point in the point cloud
\mathbf{G}	convolution kernel
d_c	distance between the centers of two nearby voxels in the kernel
\mathbf{P}_c	coordinates of voxel centers in the kernel
β_i	smoothness for the i -th point for hypergraph kernel convolution (HKC) algorithm
σ_i	smoothness for the i -th point for hypergraph kernel filtering (HKF) algorithm
γ_i	sharpness for the i -th point for local kernel filtering (LHF) algorithm
\mathcal{W}	weight in calculating γ_i for local kernel filtering (LHF) algorithm
α	resampling ratio
σ	standard deviation in Gaussian kernel
V_{Haar}	Haar-like highpass filter in multilayer graph
\mathcal{M}	sparse self similarity matrix (SSSM)
$\mathbf{z}^{(m)}$	feature vector for the i -th segment

Chapter 1

Introduction

1.1 Background

Conventional graph approaches, including graph signal processing (GSP) [1] and graph neural networks (GNN) [2, 3], have played a pivotal role in advancing computational methods. These approaches, rooted in graph theory, have demonstrated significant success across diverse applications in multimedia, such as the Internet, social networks, financial data, sensor networks, traffic patterns, and biological systems [1, 4, 5]. All the network-structured data in these domains can be naturally modeled by graphs. For example, as shown in Fig. 1.1, the motion data captured

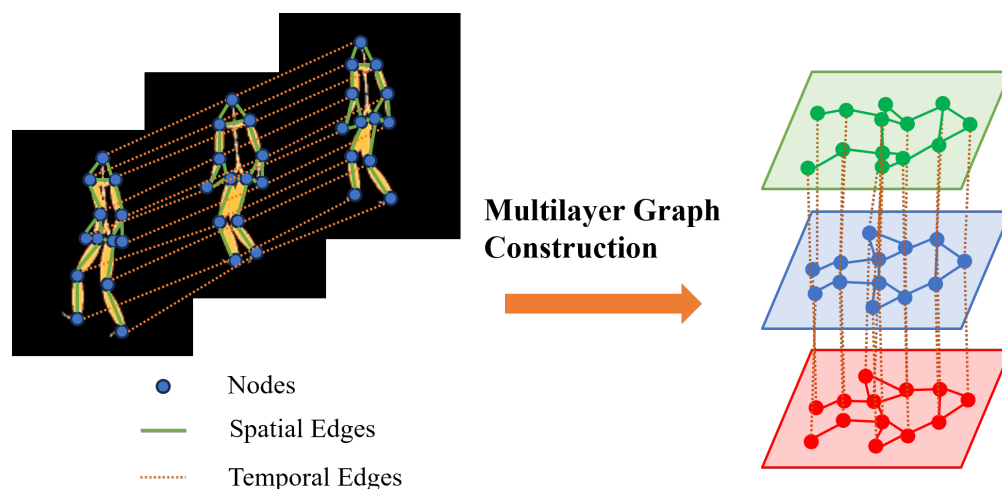


Figure 1.1: Example of multilayer graph built on the motion capture data.

by a group of sensors on the human body over sequential times is aptly modeled by a multilayer graph. Each layer of this multilayer graph aligns with a distinct data frame. This versatile property highlights the significance of employing graph-based methods for signal analysis and processing.

Graph frequency analysis was applied to many applications in physical networks. The ideal low-pass and high-pass graph Fourier filters were used in [6] in the anomaly detection algorithms for wireless sensor networks. The authors in [7] used graph frequency analysis to identify anatomy-aligned function signals in the brain and uncover an integrated structure-function relation of human behavior. More potentials of graph frequency analysis and graph Fourier filtering in the structure-informed study of functional brain dynamics were revealed in [8]. A local graph-based filter was used for image denoising in [9]. Another example is the spectral clustering methods based on low-frequency eigenvectors of the Laplacian matrix [10], which was widely used as a benchmark clustering method.

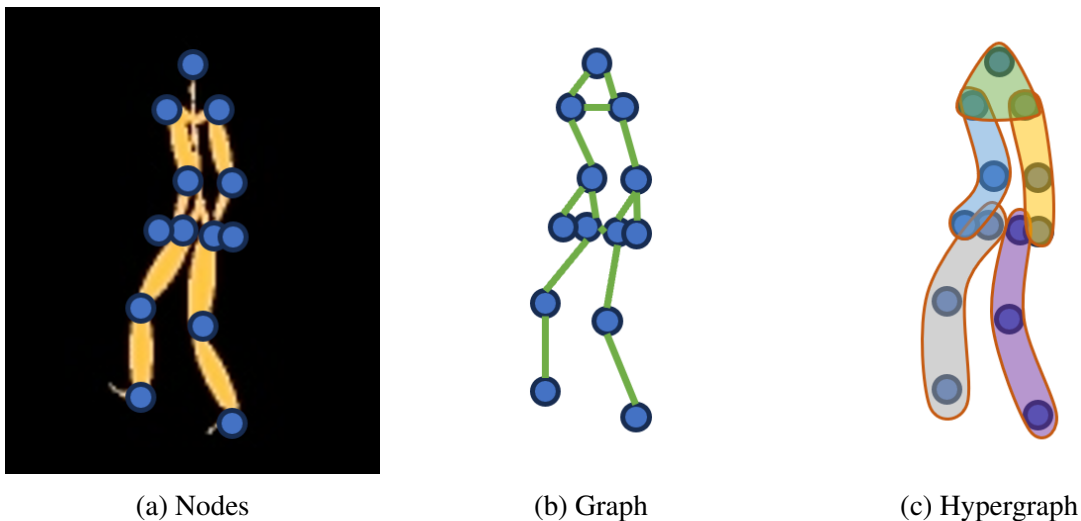


Figure 1.2: Example of graph and hypergraph for 3D point cloud representation: (a) original point cloud with nodes at the joint of body, (b) graph built on the nodes, (c) hypergraph built on the nodes.

Despite conventional graph-based methods having clearly achieved success in many fields, some limitations remain. Conventional graph-based methods tend to focus on pairwise relationships between different points since each graph edge only connects two signal nodes. However, it is clear that multilateral interactions of data points could model the more informative

characteristics of 3D point clouds. As shown in Fig. 1.2(b), bilateral graph node connections are insufficient to directly describe multilateral relationships among points sharing the same property [11]. Conversely, the hyperedges in hypergraph can connect more than two nodes, and it can capture the multilateral relationships among points, such as whether they belong to the same arm or leg, as shown in Fig. 1.2(c). Furthermore, in graph-based methods, the efficient construction of a suitable graph to represent an arbitrary point cloud poses another challenge.

1.2 Overview of Advanced Graph Approaches

More recently, high-dimensional graphs, such as hypergraphs and multilayer graphs (MLG), have been successfully applied in representing and characterizing the underlying multilateral interactions among multimedia data points [12]. A hypergraph extends the basic graph concept into higher dimensions, in which each hyperedge can connect more than two nodes [13]. Therefore, a hypergraph provides a more general representation to characterize multilateral relationships for points on object surfaces such that one hyperedge can cover multiple nodes on the same surface. Furthermore, by generalizing graph signal processing [1], hypergraph signal processing (HGSP) [14][15] provides a theoretical foundation for spectral analysis in hypergraph-based point cloud processing. Specifically, stationarity-based hypergraph estimation, in conjunction with hypergraph-based filters, has demonstrated notable successes in processing point clouds for various tasks including segmentation, sampling, and denoising [11, 16].

On the other hand, MLG structure is able to capture the underlying geometric structures on multiple dimensions, which is useful for modeling multi-dimensional multimedia data, such as dynamic point clouds. Within the context of GSP, a multilayer graph signal processing (M-GSP) framework has been introduced for MLG based on tensor representation [17]. M-GSP allows different spatial layers to represent heterogeneous geometric structures and defines a joint MLG spectral space for data analysis. M-GSP has shown great potential in spectrum analysis, image compression, clustering, hyperspectral image segmentation, and classification [17–19].

1.3 Motivation and Challenges

While high-dimensional graphs have great potential for analyzing and processing multimedia data, the practical utility of these methods faces a significant challenge — the escalating demand for additional memory space to store and process the representation tensors. For example, to represent a hypergraph constructed from a dataset with M data points, where each hyperedge contains a maximum of N data points, a M -th-order N -dimensional representation tensor $\mathbf{A}_{\mathcal{H}} \in \mathbb{R}^{N^M}$ is required. The size of this representation tensor grows exponentially with the maximum length of hyperedge. Furthermore, when working on multimedia datasets like point clouds, which can have over a million data points, the memory storage challenge emerges as a significant bottleneck for algorithms based on high-dimensional graphs, obstructing the full exploitation of high-dimensional graph processing capabilities. In response to this dilemma, researchers have explored two distinct approaches to mitigate the impact of storage constraints and enhance the feasibility of these algorithms.

The first approach adopts a ‘divide and conquer’ strategy, partitioning the high-dimensional graph and processing smaller subsets of data iteratively. This modular processing not only aids in circumventing storage limitations but also facilitates parallelization, enhancing the overall efficiency of the algorithm. By breaking down the problem into manageable components, the divide-and-conquer approach enables the algorithm to scale efficiently, and allows researchers to navigate the intricate relationship of high-dimensional data without succumbing to resource constraints.

Complementary to the partitioning strategy, the second approach delves into algorithmic optimizations aimed at reducing computational complexity. By optimizing the underlying algorithms, researchers aim to minimize the computational resources required for high-dimensional graph processing. This optimization not only addresses storage challenges but also contributes to the development of more efficient and scalable algorithms. Investigating these algorithmic enhancements is a crucial aspect of our dissertation, as we seek to unravel the potential applications of high-dimensional graph processing algorithms on various multimedia data types.

1.4 Objective of the Dissertation

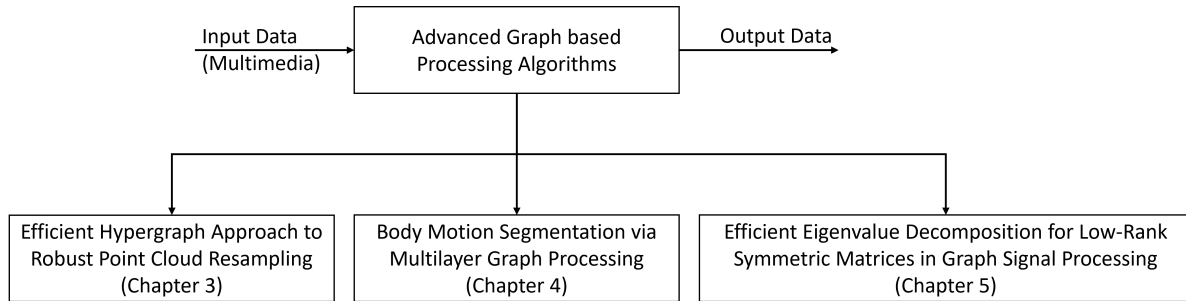


Figure 1.3: Diagram of the relationship between main chapters in this dissertation.

In this dissertation, our primary aim is to explore the diverse applications of high-dimensional graph processing algorithms on multimedia data. As shown in Fig. 1.3, the three main advances in this dissertation are all related to the exploration of high-dimensional graph processing algorithms on multimedia data. This exploration is driven by the recognition of the burgeoning challenges posed by storage limitations and computational complexity in the analysis of intricate, high-dimensional datasets. Our goals can be summarized as follows:

- To Investigate 'Divide and Conquer' Strategies: One major objective is to delve into the practical implementation and efficacy of 'divide and conquer' strategies for processing high-dimensional graphs on various multimedia data types. By partitioning and processing smaller subsets of data, we aim to not only overcome storage challenges but also enhance the scalability and efficiency of our algorithms.
- To Optimize Computational Complexity of key algorithm: Another key focus is the optimization of algorithmic complexity in high-dimensional graph processing. Through the refinement of algorithms with the conditions given by practical applications, we aim to reduce computational demands, making the processing of multimedia datasets, such as point clouds and motion capture data, more feasible with acceptable compromising accuracy on the result.
- To Apply Advanced Graph based Algorithms to Multimedia Data: The dissertation aims

to extend the theoretical groundwork to practical applications by applying high-dimensional graph processing algorithms to specific multimedia data types. Examples include point cloud data and motion capture data, both of which present unique challenges and opportunities for algorithmic exploration.

By explicitly outlining these objectives, this dissertation endeavors to provide a structured and comprehensive investigation into the applications of high-dimensional graph processing algorithms, thereby contributing to the advancement of knowledge in this field.

1.5 Significance and Contributions

This dissertation contributes to the field through the exploration and practical implementation of innovative strategies for high-dimensional graph processing. This dissertation comprises three main chapters:

- In Chapter 3, we introduce hypergraph spectral filters designed to capture multilateral interactions among the signal nodes in point clouds, aiming to better preserve their surface outlines. Unlike conventional approaches that involve constructing the underlying hypergraph, our proposed low complexity approach directly estimates the hypergraph spectrum of point clouds by leveraging hypergraph stationary processes from the observed 3D coordinates. To assess the model-preserving capabilities of our proposed algorithms on complex point cloud models, we employ a straightforward point cloud recovery method for the resampled results, and use the difference between the recovered point cloud and the original ones as the metrics. This point cloud recovery method, based on alpha complex and Poisson sampling, does not require additional information such as the model structure, and is applicable to all resampling algorithms. Through the evaluation of our proposed resampling methods using various metrics, our test results validate the effectiveness of hypergraph characterization of point clouds and highlight the robustness of hypergraph-based resampling under noisy observations. This work has been published as [20, 21].

- In Chapter 4, we delve into the emerging field of multilayer graph signal processing (M-GSP) and propose innovative M-GSP based approaches to human motion segmentation. Specifically, our approach involves modeling spatial-temporal relationships of the human motion capture data via multilayer graphs (MLG) and incorporating M-GSP spectrum analysis for feature extraction. Subsequently, we present two distinct unsupervised segmentation algorithms in M-GSP, operating within the MLG spectrum and vertex domains, respectively. Our experimental results on various datasets underscore the effectiveness of the proposed methods and the potential of M-GSP in motion analysis. Additionally, our results showcase the robustness of the proposed segmentation methods in the presence of noisy observations. This work is in preparation as [22].
- In Chapter 5, we propose an incremental eigenvalue decomposition (EVD) algorithm for low rank symmetric matrices (IEVD-LR) to update the top k eigen-pairs of the graph representation matrix. This EVD algorithm is one of the fundamental algorithms in graph based spectral analysis and processing methodologies. To accommodate the incremental growth of the graph size, we design an iterative eigen-updating algorithm, incorporating a novel error correction branch whenever the approximation error exceeds the defined tolerance. Then, we conduct an extensive analysis of the computational complexity and memory usage of our proposed IEVD-LR algorithm to showcase its accuracy and efficiency. Subsequently, we compare our approach with the existing method on both synthetic and real-world datasets in the context of spectral clustering and graph filtering. Our experimental results corroborate the accuracy and efficiency of the proposed IEVD-LR algorithm. This work is in preparation as [23].

1.6 Structure of the Dissertation

The following of the dissertation is structured as follows: we start by providing an overview of graph signal processing, followed by the discussion of hypergraph signal processing and multilayer

graph signal processing frameworks in Chapter 2. Building upon these foundational concepts, we continue to explore the specific applications and methodologies in the subsequent chapters. In Chapter 3, we present our first work on hypergraph spectral filters applied to point clouds, aiming to better preserve their surface outlines. In Chapter 4, we propose two innovative M-GSP based approaches to human motion segmentation. In Chapter 5, we propose an incremental EVD algorithm for low rank symmetric matrices (IEVD-LR) to update the top k eigen-pairs of the graph representation matrix. Our findings are consolidated and summarized in Chapter 6. Built upon these conclusions, we outline the future directions of our research within the same chapter.

Chapter 2

Introduction to Advanced Graph Signal Processing

Graph signal processing serves as the cornerstone for understanding and analyzing complex relational structures inherent in various domains, ranging from social networks to biological systems. This chapter provides a comprehensive introduction to the fundamentals of graph signal processing, including the representation of graphs, graph Fourier transform as well as graph filter designs. Building upon this foundation, we delve into the extensions of traditional graph signal processing, introducing the intricacies of hypergraph signal processing and multilayer graph signal processing. By navigating through these advanced frameworks, we aim to unravel the relationships embedded in data structures, as further explored in Chapter 3 and Chapter 4.

2.1 Graph Signal Processing

Graph signal processing (GSP) has recently emerged as an important tool for structural data analysis due to its power in capturing underlying data correlations [1]. An example of a graph constructed on point cloud is shown in Fig. 2.1. Consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N vertices, where \mathcal{V} denotes the set of vertices, i.e., $\mathcal{V} = \{v_1, \dots, v_N\}$, and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ represents the edges. In the undirected graph, for every edge (v_i, v_j) that exists, there is also an

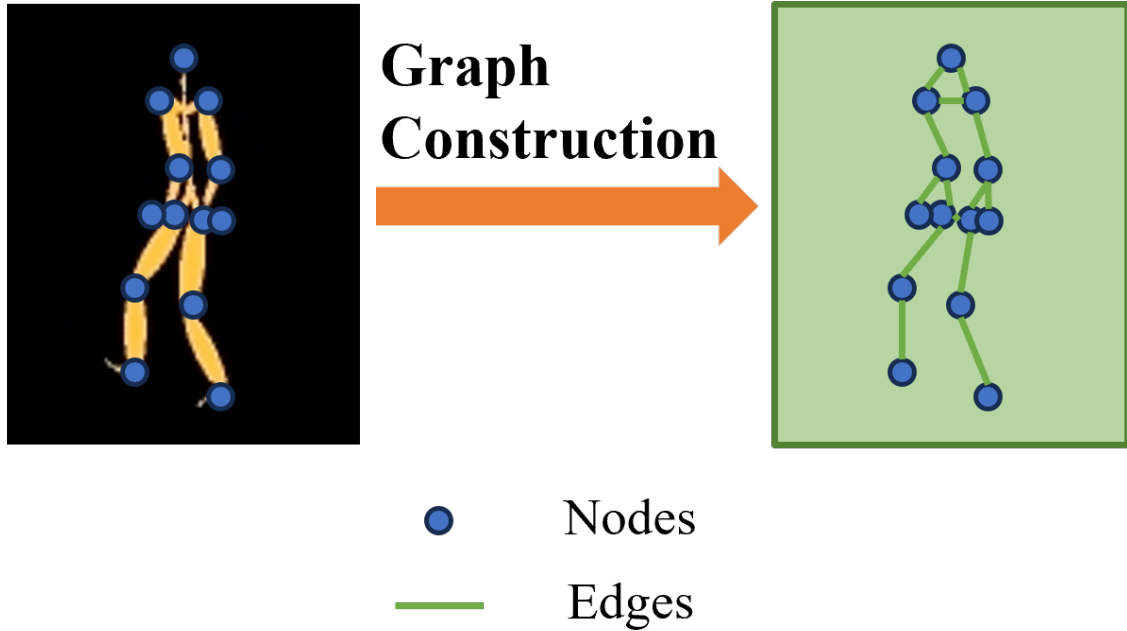


Figure 2.1: Example of a graph constructed on point cloud.

edge (v_j, v_i) . A representation matrix $\mathbf{F}_{\mathcal{G}} \in \mathbb{R}^{N \times N}$ can be used to describe the geometric structure of the graph \mathcal{G} . Commonly used representation matrices are adjacency matrix $\mathbf{A}_{\mathcal{G}}$ or Laplacian matrix $\mathbf{L}_{\mathcal{G}}$. Graph signals are the vector of attributes of vertices, which can be written as vector $\mathbf{s}_{\mathcal{G}} = [s_{\mathcal{G},1}, s_{\mathcal{G},2}, \dots, s_{\mathcal{G},N}]^T \in \mathbb{R}^N$.

Real-world datasets, such as point clouds, sensor data and images, can be naturally and efficiently represented by graphs. Given the data points, the construction of the representation matrices of graphs plays a pivotal role in capturing the underlying topology of the dataset[4]. Among the various methods for graph construction, a notable approach is model-based graph construction. This technique leverages domain knowledge to construct graphs using models, such as ϵ -neighborhood graph, tailored to the dataset at hand. In an ϵ -neighborhood graph, two vertices are connected by an edge if their Euclidean distance is smaller than a given threshold. In our experiment, we build the ϵ -neighborhood graph based on the intrinsic resolution d_r of the dataset

and the Gaussian kernel. The edge weight between the i -th and j -th vertex is

$$A_{\mathcal{G},ij} = \begin{cases} \exp\left(-\frac{\|\mathbf{a}_i - \mathbf{a}_j\|^2}{d_r^2}\right) & \|\mathbf{a}_i - \mathbf{a}_j\|^2 \leq \epsilon \\ 0 & \text{otherwise} \end{cases}, \quad (2.1)$$

where \mathbf{a}_i is the vector of data attributes for the i -th vertex. The intrinsic resolution d_r of the dataset is defined as the mean of the smallest distance from each vertex to all other vertices in the dataset, i.e., for a dataset with N vertices, $d_r = \frac{1}{N} \sum_{i=1, i \neq j}^N \min_j \|\mathbf{a}_i - \mathbf{a}_j\|$.

The graph spectral space, also referred to as the graph Fourier space, is defined based on the eigenspace of the representing matrix $\mathbf{F}_{\mathcal{G}}$ such that $\mathbf{F}_{\mathcal{G}} = \mathbf{Q}\mathbf{\Sigma}\mathbf{Q}^{-1}$, where $\mathbf{Q} \in \mathbb{R}^{N \times N}$ denotes the orthonormal matrix whose columns are the eigenvectors of $\mathbf{F}_{\mathcal{G}}$ and $\mathbf{\Sigma} \in \mathbb{R}^{N \times N}$ is a diagonal matrix with the eigenvalues of $\mathbf{F}_{\mathcal{G}}$ on the diagonal, i.e., $\mathbf{\Sigma} = \text{diag}(\lambda_{\mathcal{G},1}, \dots, \lambda_{\mathcal{G},N})$. Then the graph Fourier transform (GFT) is defined as

$$\hat{\mathbf{s}}_{\mathcal{G}} = \mathbf{Q}^{-1} \mathbf{s}_{\mathcal{G}}, \quad (2.2)$$

whereas the inverse GFT is given by $\mathbf{s}_{\mathcal{G}} = \mathbf{Q} \hat{\mathbf{s}}_{\mathcal{G}}$.

From the definitions of GFT, the concept of graph spectral analysis was developed as an important tool for signal processing and data analysis. Graph frequency analysis and graph Fourier filter design were introduced in [24, 25]. The basic non-trivial filter defined on the graph \mathcal{G} is called the graph shift, which is defined as

$$\tilde{\mathbf{s}}_{\mathcal{G}} = \mathbf{A}_{\mathcal{G}} \mathbf{s}_{\mathcal{G}}. \quad (2.3)$$

Additionally, the linear, shift-invariant graph filters can be expressed as the polynomials of the adjacency matrix $\mathbf{A}_{\mathcal{G}}$, which can be written as

$$h(\mathbf{A}_{\mathcal{G}}) = h_0 \mathbf{I} + h_1 \mathbf{A}_{\mathcal{G}} + \dots + h_L \mathbf{A}_{\mathcal{G}}^L, \quad (2.4)$$

whose output signal can be expressed as

$$\tilde{\mathbf{s}}_{\mathcal{G}} = \mathbf{H}(\mathbf{s}_{\mathcal{G}}) = h(\mathbf{A}_{\mathcal{G}})\mathbf{s}_{\mathcal{G}}. \quad (2.5)$$

By using GFT, the Fourier transform coefficients of the filtered signal $\tilde{\mathbf{s}}_{\mathcal{G}}$ can be written as

$$\begin{aligned} \mathbf{Q}^{-1}\tilde{\mathbf{s}}_{\mathcal{G}} &= \mathbf{Q}^{-1}h(\mathbf{A}_{\mathcal{G}})\mathbf{Q}\hat{\mathbf{s}}_{\mathcal{G}} \\ &= \begin{bmatrix} h(\lambda_{\mathcal{G},1}) & & \\ & \ddots & \\ & & h(\lambda_{\mathcal{G},N}) \end{bmatrix} \hat{\mathbf{s}}_{\mathcal{G}}, \end{aligned} \quad (2.6)$$

which indicates that we can design the graph filter directly on the graph frequency domain.

There are two kinds of commonly used graph filters: ideal graph filter and Haar-like graph filter. The expression of $h_{ideal}(\mathbf{A}_{\mathcal{G}})$ for ideal graph filter is

$$h_{ideal}(\mathbf{A}_{\mathcal{G}}) = \text{diag}(1, \dots, 1, 0, \dots, 0). \quad (2.7)$$

The number of ones for ideal graph filter can be determined by the threshold of the eigenvalues.

On the other hand, the $h_{Haar}(\mathbf{A}_{\mathcal{G}})$ for Haar-like graph filter can be written as

$$h_{Haar}(\mathbf{A}_{\mathcal{G}}) = \mathbf{I} - \mathbf{\Lambda}_{\mathcal{G},norm}, \quad (2.8)$$

where \mathbf{I} represents the identity matrix of size N by N and $\mathbf{\Lambda}_{\mathcal{G},norm}$ is the normalized eigenvalue matrix by normalizing the largest eigenvalue of $\mathbf{A}_{\mathcal{G}}$ to 1.

Graph frequency analysis was applied to many applications on physical networks. The ideal low-pass and high-pass graph Fourier filters were used in [6] in the anomaly detection algorithms for wireless sensor networks. The authors in [7] used graph frequency analysis to identify anatomy-aligned function signals in the brain and uncover an integrated structure-function relation of human behavior. More potentials of graph frequency analysis and graph Fourier filtering in

structure-informed study of functional brain dynamics were revealed in [8]. A local graph-based filter was used for image denoising in [9]. Another example is the spectral clustering methods based on low-frequency eigenvectors of the Laplacian matrix [10], which was widely used as a benchmark clustering method.

Within the scope of updating the spectral clustering results for dynamic graphs, existing works can be divided into two categories. The first approach is to iteratively update the eigenvalue decomposition (EVD) result, then update the clustering result based on the approximated eigenvectors, which will be reviewed in Section 5.2.1. The second approach is to estimate the clustering result based on the representative sets, such as the method proposed in [26]. This method instantly assigns cluster labels to newly added nodes based on the representative reliability of every node in each cluster. After that, the eigenvalues are also updated to estimate the number of clusters. However, when the estimated number of clusters changes, this method needs to re-initialize the algorithm by clustering based on the EVD of the new graph, which increases the computational complexity.

2.2 Hypergraph Signal Processing

Hypergraph signal processing (HGSP) is an analytic framework that uses hypergraph and tensor representation to model high-order signal interactions [14]. An example of a hypergraph constructed on point cloud data is shown in Fig. 2.2. Within this framework, a M -th-order N -dimensional representation tensor $\mathbf{A}_{\mathcal{H}} = (A_{\mathcal{H},i_1 i_2 \dots i_M}) \in \mathbb{R}^{N^M}$ models a hypergraph with N vertices in each hyperedge, which is capable of connecting maximum of M nodes. We may call the number of nodes connected by a hyperedge as its length. Weights of hyperedges with lengths less than M are normalized according to combinations and permutations [27, 28].

Orthogonal CANDECOMP/PARAFAC (CP) decomposition [29]-[31] enables the (approximate)

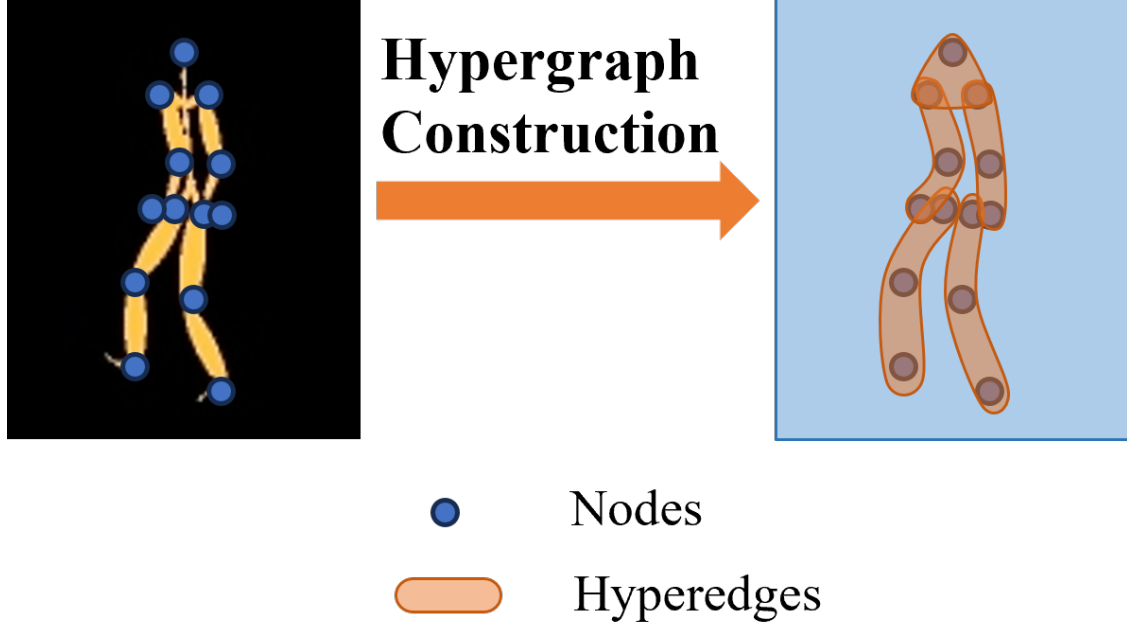


Figure 2.2: Example of a hypergraph constructed on point cloud.

decomposition of a representing tensor into

$$\mathbf{A}_{\mathcal{H}} \approx \sum_{r=1}^N \lambda_{\mathcal{H},r} \cdot \underbrace{\mathbf{f}_{\mathcal{H},r} \circ \dots \circ \mathbf{f}_{\mathcal{H},r}}_{M \text{ times}}, \quad (2.9)$$

where \circ denotes the tensor outer product, $\{\mathbf{f}_{\mathcal{H},1}, \dots, \mathbf{f}_{\mathcal{H},N}\}$ are orthonormal basis to represent spectrum components, and $\lambda_{\mathcal{H},r}$ is the r -th spectrum coefficient corresponding to the r -th basis. Spectrum components $\{\mathbf{f}_{\mathcal{H},1}, \dots, \mathbf{f}_{\mathcal{H},N}\}$ form the full hypergraph spectral space.

Similar to GSP, hypergraph signals are attributes of nodes. Intuitively, a signal is defined as $\mathbf{s}_{\mathcal{H}} = [s_{\mathcal{H},1} \ s_{\mathcal{H},2} \ \dots \ s_{\mathcal{H},N}]^T \in \mathbb{R}^N$. Since the adjacency tensor $\mathbf{A}_{\mathcal{H}}$ describes high-dimensional interactions of signals, we define a special form of the hypergraph signal to work with the representing tensor, i.e.,

$$\mathbf{s}_{\mathcal{H}}^{[M-1]} = \underbrace{\mathbf{s}_{\mathcal{H}} \circ \dots \circ \mathbf{s}_{\mathcal{H}}}_{M-1 \text{ times}}. \quad (2.10)$$

Given the definitions of hypergraph spectrum and hypergraph signals, hypergraph Fourier

transform (HGFT) is given by

$$\hat{\mathbf{s}}_{\mathcal{H}} = \mathcal{F}_{\mathcal{H}}(\mathbf{s}) = [(\mathbf{f}_1^{\top} \mathbf{s})^{M-1} \cdots (\mathbf{f}_N^{\top} \mathbf{s})^{M-1}]^{\top}. \quad (2.11)$$

From the graph specific HGFT, hypergraph spectral convolution can be generalized [12] as

$$\mathbf{x} \diamond \mathbf{y} = \mathcal{F}_{\mathcal{H}}^{-1}(\mathcal{F}_{\mathcal{H}}(\mathbf{x}) \odot \mathcal{F}_{\mathcal{H}}(\mathbf{y})), \quad (2.12)$$

where $\mathcal{F}_{\mathcal{H}}$ is the HGFT, $\mathcal{F}_{\mathcal{H}}^{-1}$ denotes inverse HGFT (iHGFT), and \odot denotes Hadamard product [14]. This definition applies the basic relationship between convolution and spectrum product, and generalizes convolution in the vertex domain into the product in the hypergraph spectrum domain.

To be concise, we refrain from a full review of HGSP here. Instead, we refer readers to [14] and related works for a more extensive introduction to HGSP concepts, such as filtering, hypergraph Fourier transform, and sampling theory, among others. Equally important are the concepts and properties of hypergraph stationary processes which can be found in, e.g., [11].

2.3 Multilayer Graph Signal Processing

M-GSP is a tensor-based framework for MLG analysis [17]. An example of MLG constructed on motion sequence data is shown in Fig. 2.3. A multilayer graph with M layers and N nodes in each layer can be viewed as projecting N virtual entities $\{u_1, \dots, u_N\}$ into M layers $\{\ell_1, \dots, \ell_M\}$, such as spectrum band frames for hyperspectral images and color frames for RGB images. In skeleton-based human motion dataset, each joint can be viewed as an entity u_i and each temporal frame serves layer ℓ_{α} . Then, a motion sequence as shown in Fig. 2.3 can be modeled as an MLG with the same number of nodes in each layer. In M-GSP, such an MLG structure can be represented by a 4-th order adjacency tensor defined as follows:

$$\mathbf{A}_{\mathcal{M}} = (A_{\mathcal{M}, \alpha i \beta j}) \in \mathbb{R}^{M \times N \times M \times N}, \quad (2.13)$$

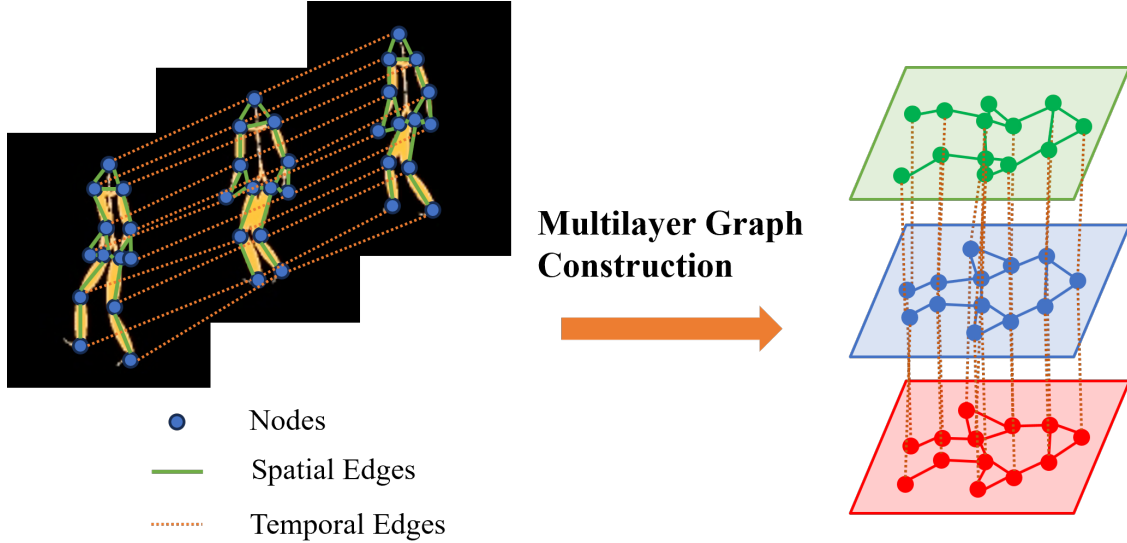


Figure 2.3: Example of hypergraph constructed on motion sequence.

where $1 \leq \alpha, \beta \leq M, 1 \leq i, j \leq N$, and \mathcal{M} denotes the symbols for MLG. Here, each entry $A_{\mathcal{M}, \alpha i \beta j}$ of adjacency tensor $\mathbf{A}_{\mathcal{M}}$ indicates the edge strength between entity j 's projected node in layer β and entity i 's corresponding node in layer α . Note that tensor representation requires each layer to have the same number of nodes. We can generate such an MLG by:

- Adding isolated nodes to layers with fewer nodes to reach N nodes and setting the interpolated signals as zeros; Since the isolated node does not connect to any other nodes, they would not affect the message passing in MLG. This method is suitable for physical networks, such as smart grid and cyber-physical systems [32].
- Aggregating similar nodes as supernodes to reduce the node number to N : This can be an intuitive method for image processing, where several pixels can be grouped into superpixels.

Similar to traditional GSP, we define the MLG Fourier space via tensor decomposition. In an undirected MLG, the adjacency $\mathbf{A}_{\mathcal{M}}$ is partially symmetric between orders one and three, and between orders two and four, respectively. Then, it can be approximated via orthogonal

CANDECOMP/PARAFAC (CP) decomposition [17] as

$$\mathbf{A}_{\mathcal{M}} \approx \sum_{\alpha=1}^M \sum_{i=1}^N \lambda_{\mathcal{M},\alpha i} \cdot \mathbf{f}_{\mathcal{M},\alpha} \circ \mathbf{e}_{\mathcal{M},i} \circ \mathbf{f}_{\mathcal{M},\alpha} \circ \mathbf{e}_{\mathcal{M},i}, \quad (2.14)$$

where \circ is the tensor outer product [17], $\mathbf{f}_{\mathcal{M},\alpha} \in \mathbb{R}^M$ and $\mathbf{e}_{\mathcal{M},i} \in \mathbb{R}^N$ are orthonormal bases characterizing the properties of layers and entities, respectively. $\lambda_{\mathcal{M},\alpha i}$ is the spectrum coefficient corresponding to $\mathbf{f}_{\mathcal{M},\alpha}$ and $\mathbf{e}_{\mathcal{M},i}$.

Besides MLG Fourier space, the singular space is defined from higher-order singular value decomposition (HOSVD) [33] as an alternative subspace of MLG, i.e.,

$$\mathbf{A}_{\mathcal{M}} = \mathbf{S} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)} \times_4 \mathbf{U}^{(4)}. \quad (2.15)$$

Here, \times_n denotes the n -mode product introduced in [17], which can be used to modify the dimension of the n -th order. $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times I_n}$ is a unitary matrix with $I_1 = I_3 = M$ and $I_2 = I_4 = N$ [19]. Similar to MLG Fourier space, there are two modes of singular spectra, i.e., $(\gamma_\alpha, \mathbf{f}_\alpha)$ for mode 1, 3, and (σ_i, \mathbf{e}_i) for mode 2, 4. More specifically, $\mathbf{U}^{(1)} = \mathbf{U}^{(3)} = (\mathbf{f}_\alpha)$ and $\mathbf{U}^{(2)} = \mathbf{U}^{(4)} = (\mathbf{e}_i)$. Both singular tensor analysis and spectral analysis are efficient tools for image processing depending on specific tasks. In this work, we explore MLG singular analysis in human motion.

We now introduce the M-GSP singular transform (M-GST). Suppose that the singular vectors form $\mathbf{W}_f = [\mathbf{f}_1 \cdots \mathbf{f}_M] \in \mathbb{R}^{M \times M}$ and $\mathbf{W}_e = [\mathbf{e}_1 \cdots \mathbf{e}_N] \in \mathbb{R}^{N \times N}$. Given an MLG signal $\mathbf{s}_{\mathcal{M}} = (s_{\mathcal{M},\alpha i}) \in \mathbb{R}^{M \times N}$, the layer-wise M-GST can be defined as

$$\check{\mathbf{s}}_{\mathcal{M},L} = \mathbf{W}_f^\top \mathbf{s}_{\mathcal{M}} \in \mathbb{R}^{M \times N}, \quad (2.16)$$

and the entity-wise M-GST can be defined as

$$\check{\mathbf{s}}_{\mathcal{M},N} = \mathbf{s}_{\mathcal{M}} \mathbf{W}_e \in \mathbb{R}^{M \times N}. \quad (2.17)$$

The joint M-GST can be calculated by

$$\check{\mathbf{s}}_{\mathcal{M}} = \mathbf{W}_f^{\top} \mathbf{s}_{\mathcal{M}} \mathbf{W}_e \in \mathbb{R}^{M \times N}. \quad (2.18)$$

For brevity, here we only present the basic concepts of M-GSP. Interested readers could refer to [17] for more details, including M-GSP spectral transform, filter design and spectral analysis.

2.4 Conclusion

In conclusion, this chapter has provided an overview of graph signal processing, delving into the fundamentals that underpin the analysis and processing of signals on graphs. The extensions to hypergraph signal processing and multilayer graph signal processing have expanded our understanding of complex relational structures within diverse datasets. As we navigate through these advanced frameworks, we gain valuable insights into the relationships embedded in data structures. In the following chapters, we will build upon these foundations, exploring specific applications and methodologies in the context of hypergraph signal processing (Chapter 3) and multilayer graph signal processing (Chapter 4). Through this journey, we aim to contribute to a deeper comprehension of signal processing on multimedia data in interconnected systems.

Chapter 3

An Efficient Hypergraph Approach to Robust Point Cloud Resampling

A point cloud is a collection of points on the surface of a 3D target object. Each point consists of its 3D coordinates and may contain further features, such as colors and normals [34]. In this work, we focus on the coordinates of data points and point cloud resampling. A point cloud can be represented by the coordinates of N data points written as an $N \times 3$ real-valued location matrix

$$\mathbf{P} = [\mathbf{X}_1 \quad \mathbf{X}_2 \quad \mathbf{X}_3] = \begin{bmatrix} \mathbf{p}_1^\top \\ \mathbf{p}_2^\top \\ \vdots \\ \mathbf{p}_N^\top \end{bmatrix} \in \mathbb{R}^{N \times 3}, \quad (3.1)$$

where \mathbf{X}_i denotes a vector of the i -th coordinates of all N data points whereas 3×1 vector \mathbf{p}_i indicates the i -th point's coordinates. In this chapter, we explore the application of the hypergraph signal processing method on point cloud resampling.¹

¹Part of this chapter is reprinted, with permission, from [Q. Deng, S. Zhang and Z. Ding, "An Efficient Hypergraph Approach to Robust Point Cloud Resampling," *IEEE Transactions on Image Processing*, vol. 31, pp. 1924-1937, 2022.]

3.1 Introduction

3D perception plays an important role in the high growth fields of robotics and cyber-physical systems and continues to drive much progress made in advanced point cloud processing. 3D point clouds provide efficient exterior representation for complex objects and their surroundings. Point clouds have seen broad applications in many areas, such as computer vision, autonomous driving and robotics. Notable examples of point cloud processing include surface reconstruction [35], rendering [36], feature extraction [37], shape classification [38], and object detection/tracking [39]. When constructing a point cloud of a target object, however, modern laser scan systems can generate millions of data points [40]. To achieve better storage efficiency and lower point cloud processing complexity, point cloud resampling aims to reduce the number of points in a cloud to achieve data compression while preserving the vital 3D structural and surface features. Point cloud resampling represents an important tool in various applications such as point cloud segmentation, object classification and efficient data representation. An example of point cloud resampling proposed in [41] suggested a graph-based filter to downsample point clouds and capture the original object surface contour.

The literature already contains a variety of works on different aspects of point cloud resampling. For instance, a centroidal Voronoi tessellation method in [42] can progressively generate high-quality resampling results with isotropic or anisotropic distributions from a given point cloud to form compact representations of the underlying cloud surface. Another 3D filtering and downsampling technique [43] relies on a growing neural gas network, to deal with noise and outliers within data provided by Kinect sensors. Of particular interest is a graph-based resampling approach which has exhibited desirable capability to capture the underlying structures of point clouds [44]. The graph-based method of [45] applies embedded binary trees to compress the dynamic point cloud data. Another interesting work [41] proposes several graph-based filters to capture the distribution of point data to achieve computationally efficient resampling. In addition, a contour-enhanced resampling method introduced in [46] utilizes graph-based highpass filters.

In addition to the aforementioned class of graph-based methods, a competing class of

feature-based approaches via edge detection and feature extraction has also been popular. In [47], the authors presented a sharp feature detector via Gaussian map clustering on local neighborhoods. Bazazian *et al.* [48] extended this principle by leveraging principal component analysis (PCA) to develop a new agglomerative clustering method. The efficiency and accuracy of this work can further benefit from spectral analysis of the covariance matrix defined by k -nearest neighbors. Another typical approach represented by [13] processes a noisy and possibly outlier-ridden point set in an edge-aware manner.

Both graph-based and feature-based methods have clearly achieved success in point cloud resampling. However, some limitations remain. Graph-based methods tend to focus on pairwise relationships between different points, since each graph edge only connects two signal nodes. However, it is clear that multilateral interactions of data points could model the more informative characteristics of 3D point clouds. Bilateral graph node connections cannot even describe the multilateral relationship among points on the same surface (e.g. 3 points of a triangle) directly [11]. Furthermore, in graph-based methods, the efficient construction of a suitable graph to represent an arbitrary point cloud poses another challenge. Among feature-based methods, performance would vary with respect to the feature selection and filter designs. The open issues are the adequate and robust selection of features and filter parameters for practical point cloud processing.

More recently, hypergraphs have been successfully applied in representing and characterizing the underlying multilateral interactions among multimedia data points [12]. A hypergraph extends the basic graph concept into higher dimensions, in which each hyperedge can connect more than two nodes [13]. Therefore for point clouds, hypergraph provides a more general representation to characterize the multilateral relationship for points on object surfaces such that one hyperedge can cover multiple nodes on the same surface. Furthermore, by generalizing graph signal processing [1], hypergraph signal processing (HGSP) [14][15] provides a theoretical foundation for spectral analysis in hypergraph-based point cloud processing. Specifically, stationarity-based hypergraph estimation, in conjunction with hypergraph-based filters, has demonstrated notable successes in processing point clouds for various tasks including segmentation, sampling, and denoising [11, 16].

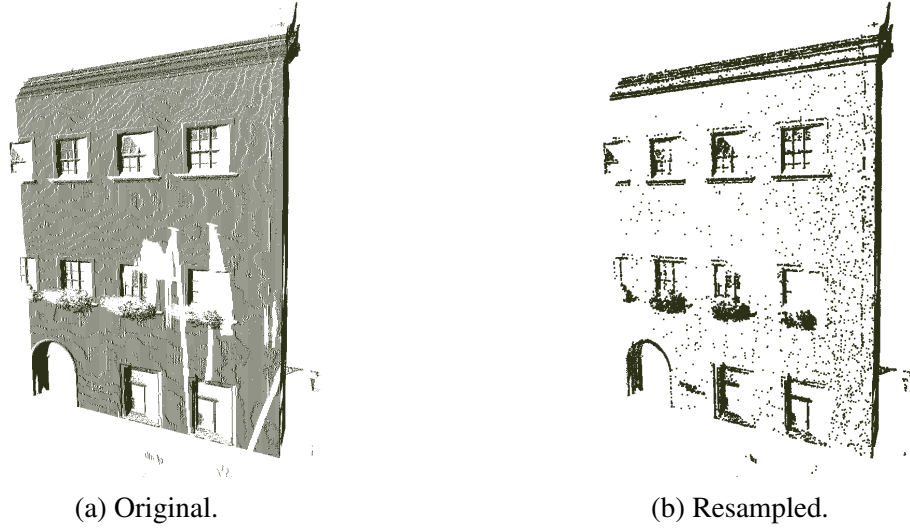


Figure 3.1: Example of Contour-Enhanced Resampling: (a) original point cloud with 272705 points, and (b) resampled building based on the proposed local hypergraph filtering with 20% samples.

In this chapter, we investigate point cloud resampling based on hypergraph spectral analysis. Instead of the traditional uniform resampling, we investigate contour-enhanced resampling to select a subset of points in the point cloud and to extract distinct surface features. A heuristic example is illustrated in Fig. 3.1 showing a point cloud successfully resampled with only 20% samples for the building. To briefly highlight the novelty of our proposed approaches, we estimate the hypergraph spectrum basis for point clouds under study by leveraging the hypergraph stationary process. We propose three novel 3D point cloud resampling methods:

- 1) Hypergraph kernel convolution method (HKC);
- 2) Hypergraph kernel filtering method (HKF);
- 3) Local hypergraph filtering method (LHF).

The kernel convolution method defines a local smoothness among signals based on an operator and hypergraph convolution. The kernel filtering method defines the local smoothness with respect to highpass filtering in spectrum domain. The local hypergraph filtering method utilizes a local sharpness definition with respect to highpass filtering in spectrum domain. In order to test the

model preserving property on complex point cloud models, we apply a simple method for point cloud recovery based on alpha complex and Poisson sampling. We then test the proposed methods under several metrics to demonstrate the compression efficiency and robustness of our proposed resampling methods with respect to the general feature preservation of point clouds under study.

We summarize the major contributions of this chapter:

- We propose three novel hypergraph-based resampling methods to preserve distinct and sharp point cloud features;
- We provide novel definitions of hypergraph-based indicators to evaluate the smoothness or sharpness over point clouds;
- We apply different metrics to demonstrate the effectiveness of our proposed methods.

We organize the rest of the chapter as follows. We develop the foundation of HGSP based point cloud resampling and derive three new resampling methods in Section 3.2. We provide the test results of the proposed resampling methods in Section 3.3, before formalizing our conclusions in Section 3.4.

3.2 HGSP Point Cloud Resampling

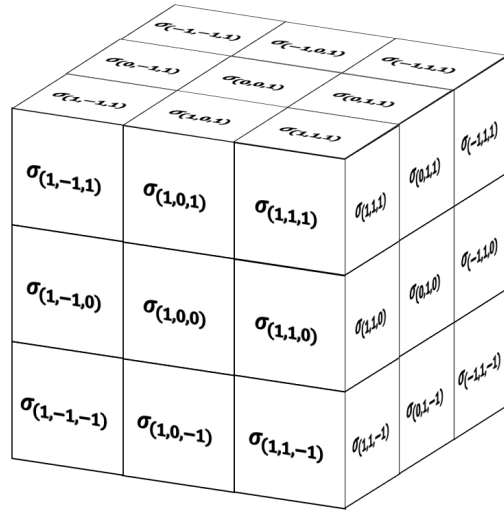
Within the HGSP framework, we now propose three novel edge-preserving methods for point cloud resampling. First, we develop a hypergraph kernel convolution (HKC) method inspired by convolution-based edge detection in image processing. Next, we propose a hypergraph kernel filtering (HKF) algorithm targeting local smoothness to reduce the computational complexity of HKC. Finally, we design a local hypergraph filter (LHF) to target point clouds with non-uniformly distributed points over the surface of an object.

3.2.1 Hypergraph Kernel Convolution (HKC) based Method

In traditional image processing, kernel convolution methods such as Sobel and Prewitt [49] have achieved notable successes in edge detection. Inspired by these 2D kernel convolution methods in 2D image processing, e.g. Fig. 3.2(a), we define a square $k \times k \times k$ 3D cube as the slicing block to define a local signal $\mathbf{s}_{\mathcal{L},i} \in \mathbb{R}^{N_k}$ and a convolution kernel $\mathbf{G} \in \mathbb{R}^{N_k}$ with $N_k = k^3$ aimed at extracting sharp outliers of the point cloud under study. An example of 3^3 cubic 3D convolution kernel is shown in Fig. 3.2(b). Note that the hypergraph convolution kernel can assume different shapes and sizes depending on the datasets.

1	0	-1
2	0	-2
1	0	-1

(a) 2D Sobel Kernel.



(b) 3D Hypergraph Kernel.

Figure 3.2: Example of Convolution Kernels.

For the i -th point in an original point cloud, its corresponding local signal $\mathbf{s}_{\mathcal{L},i} \in \mathbb{R}^{N_k}$ is defined according to the number of points in the voxel of kernel centered at i -th point. An example of the local signal is shown in Fig. 3.3. Although the idea behind the use of, e.g., 3D Sobel operator is straightforward, technical obstacles arise mainly due to two reasons: (1) nodes in 3D point cloud are not always on the grid; (2) two signals in graph/hypergraph based convolution must have the same length. Thus, we let N_k be equal to the number of voxels in the kernel. Let d_c be the distance between the centers of two nearby voxels in the kernel. A proper selection of d_c should allow $\mathbf{s}_{\mathcal{L},i}$ to capture the local geometric information. If d_c is too small, only a few neighbors of i -th point are

included in the $s_{\mathcal{L},i}$ and it is sensitive to measurement noise. If d_c is too large, a large number of neighbors are included in each voxel, which may lead to the blurring of detailed local geometric information. Since intrinsic resolution describes the point cloud density and can be estimated by averaging all distances between each point and its nearest neighbor, we set d_c as the intrinsic resolution of a point cloud.

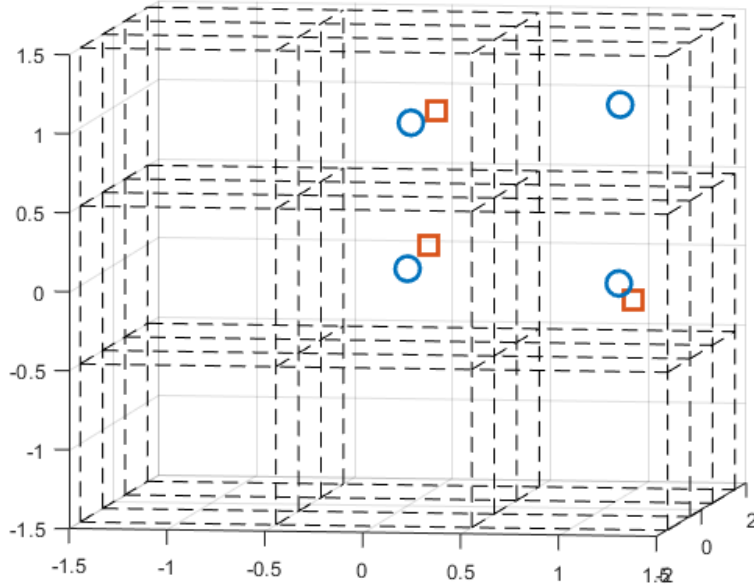


Figure 3.3: An example of local signal $s_{\mathcal{L},i}$: Points in the middle layer are marked by blue circles and points in the back layer are marked by red squares. The voxels of slicing block are delineated by the dashed black lines. There is at most one point in each voxel in this example, such that $s_{\mathcal{L},i}(n) = 1$ or 0 , respectively, depending on whether the n -th voxel contains a point.

Given the definition of signals, we now propose a new hypergraph convolution based method. There are two main steps in our kernel convolution based method: 1) estimation of hypergraph spectrum; and 2) implementation of kernel convolution.

We first estimate the hypergraph spectrum based on hypergraph stationary process. Introduced in [11], a stochastic signal $s_{\mathcal{H}}$ is weak-sense hypergraph stationary if and only if it has zero-mean and its covariance matrix has the same eigenvectors as the hypergraph spectrum basis, i.e.,

$$\mathbb{E}[s_{\mathcal{H}}] = \mathbf{0}, \quad (3.2)$$

and

$$\mathbb{E}[\mathbf{s}_{\mathcal{H}}\mathbf{s}_{\mathcal{H}}^H] = \mathbf{V}_{\mathcal{H}}\Sigma_{\mathbf{s}_{\mathcal{H}}}\mathbf{V}_{\mathcal{H}}^H, \quad (3.3)$$

where $\mathbf{V}_{\mathcal{H}}$ is the hypergraph spectrum basis and $\Sigma_{\mathbf{s}_{\mathcal{H}}}$ denotes the diagonal matrix whose diagonal elements are the eigenvalues. Since the 3D coordinates can be viewed as three observations of the point cloud from different projection directions, we can estimate the hypergraph spectrum from the covariance matrix of three coordinates based on the assumption of signal stationarity. Note that, since GSP is a special case of HGSP when the number of edges is reduced to two, HGSP-based stationary process is also compatible with GSP-based stationarity. Here, we use the same spectrum estimation strategy as [11], and consider the adjacency tensor as a third-order tensor, which is the minimum number of nodes to form a surface. Interested readers can refer to [11, 16] for more detailed discussions on the spectrum estimation based on HGSP-based stationary process.

HKC Algorithm

Let $\mathbf{P}_c = [\mathbf{X}_1 \ \mathbf{X}_2 \ \mathbf{X}_3] \in \mathbb{R}^{N_k \times 3}$ be the coordinates of the total N_k voxel centers in the kernel. For our $3 \times 3 \times 3$ kernel example, $N_k = 27$. The distance d_c is set to equal the intrinsic resolution of the point cloud, which can be estimated by the average distances between each point and its nearest neighbor in the point cloud. We then normalize the coordinates \mathbf{P}_c to obtain a zero mean signal \mathbf{P}'_c . By calculating the eigenvectors $\{\mathbf{f}_1, \dots, \mathbf{f}_{N_k}\}$ for $R_{\mathbf{P}'_c} = \mathbf{P}'_c\mathbf{P}'_c{}^\top$, we can estimate the hypergraph spectrum basis $\mathbf{V}_{\mathcal{H}} = [\mathbf{f}_1, \dots, \mathbf{f}_{N_k}]$.

Next, we implement convolution between the local signal $\mathbf{s}_{\mathcal{L},i}$ and the kernel \mathbf{G} . Since we consider the third-order tensor and are only interested in signal energies in the spectrum domain, we can utilize the hypergraph spectrum to calculate a simplified-form of HGFT $\hat{\mathbf{s}}_{\mathcal{L},i}$ and a corresponding inverse HGFT (iHGFT) of the original signals $\mathbf{s}_{\mathcal{L},i}$, instead of the HGFT and iHGFT of the hypergraph signal $\mathbf{s}_{\mathcal{L},i}^{[M-1]}$ in (2.11).

Algorithm 3.1 Hypergraph Kernel Convolution (HKC)

- Input:** A point cloud of N nodes with coordinates $\mathbf{P} = [\mathbf{p}_1^\top \cdots \mathbf{p}_N^\top]^\top$, resampling ratio α .
1. Calculate the intrinsic resolution of point cloud;
 2. Use intrinsic resolution d_c to find coordinates $\mathbf{P}_c \in \mathbb{R}^{N_k \times 3}$ of voxel centers in the kernel;
 3. Use the coordinates \mathbf{P}_c of voxel centers in the kernel to estimate the hypergraph spectrum basis $\mathbf{V}_\mathcal{H} = [\mathbf{f}_1, \cdots, \mathbf{f}_{N_k}]$ and corresponding eigenvalues $\lambda_\mathcal{H}$;
- for** $i = 1, 2, \cdots, N$ **do**
4. Use hypergraph spectrum basis $\mathbf{V}_\mathcal{H}$ to calculate the Fourier transform $\hat{\mathbf{s}}_{\mathcal{L},i}$ in (3.4);
 5. Calculate the Hadamard product $\hat{\mathbf{s}}_{oi} = \hat{\mathbf{s}}_{\mathcal{L},i} \odot \hat{\mathbf{G}}$;
 6. Calculate inverse Fourier transform \mathbf{s}_{oi} using (3.5);
 7. Calculate local smoothness β_i in (3.7) ;
- end for**
8. Sort the local smoothness β_i in descending order and select the bottom $N_r = \alpha N$ nodes as the resampled point cloud.
-

Recall from [14] that simplified HGFT and iHGFT of original signal are, respectively,

$$\mathcal{F}(\mathbf{s}_{\mathcal{L},i}) = \hat{\mathbf{s}}_{\mathcal{L},i} = \mathbf{V}_\mathcal{H}^H \mathbf{s}_{\mathcal{L},i}, \quad (3.4)$$

$$\mathcal{F}^{-1}(\hat{\mathbf{s}}_{\mathcal{L},i}) = \mathbf{s}_{\mathcal{L},i} = \mathbf{V}_\mathcal{H} \hat{\mathbf{s}}_{\mathcal{L},i}. \quad (3.5)$$

Note that the hypergraph signal is a $(M - 1)$ -fold tensor outer product of the original signal in vertex domain, corresponding to $(M - 1)$ -fold Hadamard product in spectrum domain, where they share the same bandwidth.

Directly designing convolution kernel \mathbf{G} in vertex domain is challenging for two main reasons. First, since hypergraph spectrum basis $\mathbf{V}_\mathcal{H}$ would vary for different kernels, finding a general \mathbf{G} in vertex domain that performs equally well for various different bases would be difficult. Second, since the convolution result \mathbf{s}_{oi} between the signal $\mathbf{s}_{\mathcal{L},i}$ and the kernel \mathbf{G} can be expressed as

$$\begin{aligned} \mathbf{s}_{oi} &= \mathbf{V}_\mathcal{H}(\hat{\mathbf{G}} \odot \hat{\mathbf{s}}_{\mathcal{L},i}) \\ &= \mathbf{V}_\mathcal{H}(\text{diag}(\hat{\mathbf{G}})\hat{\mathbf{s}}_{\mathcal{L},i}) \\ &= \mathbf{V}_\mathcal{H}\text{diag}(\hat{\mathbf{G}})\mathbf{V}_\mathcal{H}^H \mathbf{s}_{\mathcal{L},i}, \end{aligned} \quad (3.6)$$

it is convenient to design spectrum domain $\hat{\mathbf{G}}$ directly.

In order to preserve edges in the resampled point cloud, we use a highpass filter defined in spectrum domain. Since sharp features of point clouds correspond to high frequency elements in the hypergraph spectral space [11], a highpass filter is designed to preserve edges in resampled point clouds. More specifically, in our experiments, we use a simple and well-known Haar-like highpass design [41], i.e., $\hat{\mathbf{G}} = \mathbf{1} - \boldsymbol{\lambda}_{\mathcal{H}}$, where $\boldsymbol{\lambda}_{\mathcal{H}} = [\lambda_1 \ \lambda_2 \ \cdots \ \lambda_{N_k}]^\top$ are eigenvalues corresponding to eigenvectors $\{\mathbf{f}_1, \cdots, \mathbf{f}_{N_k}\}$. Since larger λ corresponds to lower frequency, Haar-like design could highlight high frequency components in the signal [1]. Note that, here we provide the Haar-like highpass filter as an example of convolution kernels. Other elegant highpass filters can also be used as the convolution kernel for the edge preserving purpose. We use the ratio between the norm of the convolution output \mathbf{s}_{oi} and the norm of $\mathbf{s}_{\mathcal{L},i} - \mathbf{s}_{oi}$ to measure smoothness β_i for the i -th point, i.e.,

$$\beta_i = \frac{\|\mathbf{s}_{oi}\|}{\|\mathbf{s}_{\mathcal{L},i} - \mathbf{s}_{oi}\|}. \quad (3.7)$$

In resampling, we would like to extract distinct and sharp features by selecting points that exhibit lower β_i value from the resampling output. Our algorithm is summarized as Algorithm 3.1, also known as the HKC resampling algorithm.

HKC Algorithm Complexity

In an unorganized point cloud, the computational complexity of HKC amounts to $O(N^2 + N \log N + N_k(N_k + 1)N + N_k^3)$, while in an organized point cloud, the complexity order is $O(N \log N + (N_k^2 + N_k + 1)N + N_k^3)$. Here are the details. First, for generating $\mathbf{s}_{\mathcal{L},i}$ of all points in an unorganized point cloud, the computation order is $O(N^2)$ because of the need to search the point cloud to find actual neighbors for each data point. In an organized point cloud, such computation can be reduced to $O(N)$. Next, to estimate the hypergraph spectrum basis $\mathbf{V}_{\mathcal{H}}$ and $\mathbf{V}_{\mathcal{H}}^H$, we need to find eigenvectors of $R_{\mathbf{P}_c}$, requiring computation order of $O(N_k^3)$. Because the same kernel is used for every point in the point cloud, we only need to estimate hypergraph spectrum basis once.

In subsequent steps, computing all HGFT and iHGFT pairs amounts to $O(NN_k^2)$. Finally, it takes $O(NN_k)$ to calculate all β_i 's in (3.7), and additional $O(N \log N)$ to sort them.

3.2.2 Hypergraph Kernel Filtering (HKF) Resampling

To present method, we still use the $3 \times 3 \times 3$ cube as the example kernel, and the same local signal $s_{\mathcal{L},i}$ in the Kernel convolution based method.

HKF Algorithm

Consider convolution via Hadamard product and inverse Fourier transform in the HKC resampling algorithm, we need to transform the local signal $s_{\mathcal{L},i}$ from vertex domain to spectrum domain. A simpler alternative is to compute local smoothness directly for signals in spectrum domain. The computational complexity of the algorithm is reduced by eliminating the inverse transform.

For edge-preserving, we wish to separate the high frequency coefficients from the low frequency coefficients in spectrum domain. Recall that the spectrum bases corresponding to smaller λ represent higher frequency components [14]. Because $R_{s'}$ is a real symmetric matrix, its eigenvalues must be real. Thus, we can sort the eigenvalues of $R_{s'}$ as $0 \leq \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_{N_k}$ with corresponding eigenvectors $\{f_1, \dots, f_{N_k}\}$. We can devise a threshold θ to separate the high frequency components from the low frequency components according to a sharp rise of successive eigenvalues.

Given a threshold selection of θ , we could further define a local smoothness σ_i to select the resampled points:

$$\sigma_i = \frac{\sum_{k \in \{1, 2, \dots, \theta\}} |\hat{s}_{\mathcal{L},i}(k)|}{\sum_{k \in \{1, 2, \dots, N_k\}} |\hat{s}_{\mathcal{L},i}(k)|}, \quad (3.8)$$

which is the fraction of high frequency energy within total signal energy. Finally, we resample the point cloud by selecting the points with smaller σ_i , which correspond to points containing larger amount of higher-frequency components in the hypergraph. We summarize our algorithm as

Algorithm 3.2 Hypergraph Kernel Filtering (HKF)

- Input:** A point cloud with N nodes characterized by $\mathbf{P} = [\mathbf{p}_1^\top \cdots \mathbf{p}_N^\top]^\top$, resampling ratio α .
1. Calculate the intrinsic resolution of point cloud;
 2. Use the intrinsic resolution as d_c to get the coordinates $\mathbf{P}_c \in \mathbb{R}^{N_k \times 3}$ of voxel centers in the kernel;
 3. Use the coordinates \mathbf{P}_c of the voxel centers in the kernel to estimate the hypergraph spectrum basis $\mathbf{V}_{\mathcal{H}} = [\mathbf{f}_1, \cdots, \mathbf{f}_{N_k}]$;
for $i = 1, 2, \cdots, N$ **do**
 4. Use hypergraph spectrum basis $\mathbf{V}_{\mathcal{H}}$ to calculate the Fourier transform $\hat{s}_{\mathcal{L},i}$ in (3.4);
 5. Calculate the local smoothness σ_i in (3.8);**end for**
 6. Sort the local smoothness σ_i and select the bottom $N_r = \alpha N$ points as the resampled point cloud.
-

Algorithm 3.2, also known as HKF resampling algorithm. The general steps of HKF are similar to those of HKC, while the major differences lie in the definition of smoothness as shown in (3.8). Since the resampled point clouds favor high-frequency points, they tend to contain more sharp features and are less smooth.

HKF Algorithm Complexity

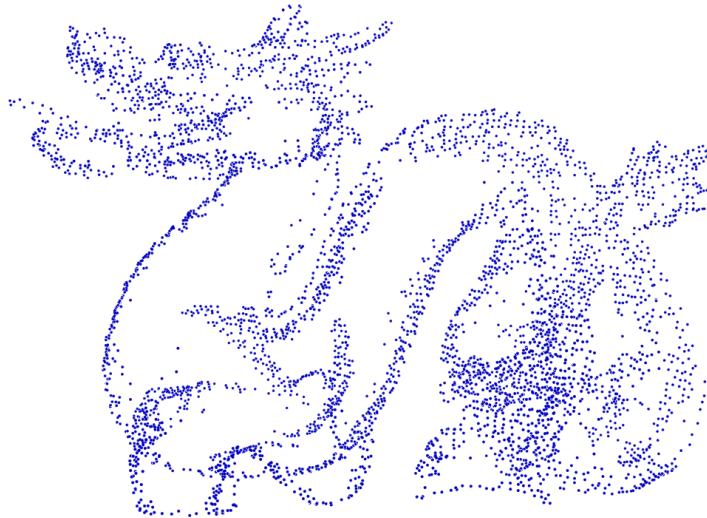
Algorithm 3.2 (HKF) and HKC have a similar order of computational complexity, while the runtime of HKF would be shorter than HKC by eliminating the iHGFT. Similar to HKC resampling, in an unorganized point cloud, the complexity for generating $s_{\mathcal{L},i}$ for all points is $O(N^2)$, whereas this complexity is reduced to $O(N)$ for an organized point cloud. The complexity of estimating the hypergraph spectrum basis $\mathbf{V}_{\mathcal{H}}$ and $\mathbf{V}_{\mathcal{H}}^H$ is $O(N_k^3)$. The cost of requisite Fourier transforms amounts to $O(NN_k^2)$. The total complexity for computing β_i in (3.8) is $O(NN_k)$. It further requires $O(N \log N)$ to sort the σ_i .

3.2.3 Local Hypergraph Filtering (LHF) Algorithm

In the aforementioned HKC and HKF algorithms, we use the same kernel to define all the local signals for points in the point cloud. If the distribution of the points are highly non-uniform on the surface of an object, it is difficult to find a value of d_c suitable for every local signal. The distance



(a) Resampled results of “dragon” using local hypergraph filtering based method with signal length $N_i = 3$. Details in the body part are kept.



(b) Resampled results of “dragon” using local hypergraph filtering based method with signal length $N_i = 6$. Details in the body part are ignored.

Figure 3.4: Resampled Results of Dragon Using Local Hypergraph Filtering based Method.

parameter may either be too small for the low density part or too large for the high density part. As a result, the performance of HKC and HKF algorithms may be erratic for such highly uneven point distribution. To mitigate this problem, we further propose another HGSP approach to model the local signal by incorporating the vector between the i -th point and each of its $(N_i - 1)$ closest

neighbors. In particular, we define a local signal for the i -th node of order N_i as

$$\mathbf{s}_{\mathcal{L},p,i} = [\mathbf{0}^\top (\mathbf{p}_{n_1} - \mathbf{p}_i)^\top \cdots (\mathbf{p}_{n_{N_i-1}} - \mathbf{p}_i)^\top]^\top \in \mathbb{R}^{N_i \times 3}, \quad (3.9)$$

where n_1, \dots, n_{N_i-1} are the indices of its $(N_i - 1)$ nearest neighbors.

We then build hypergraphs over these points and use hyperedge to connect point i and its (N_i-1) nearest neighbors. Similar to the HKF algorithm, we also devise a filter defined in spectrum domain to process the local signal. Although, strictly speaking, we could define a unique N_i for each of the i -th point, we find it more convenient to consider some fixed selections for all points to avoid comparing hyperedges of different lengths.

Fig. 3.4 provides an example to show the effect of N_i selection. When N_i is small, $\mathbf{s}_{\mathcal{L},p,i}$ describes the local geometric information in a smaller region. Consequently, the filtered results tend to vary more and capture sharp features of the point cloud in Fig. 3.4(a). When N_i is large, $\mathbf{s}_{\mathcal{L},p,i}$ characterizes the local geometry of a larger region around the i -th point. As a result, its local information is blurred with other local information from its neighbors such that the filtered results tend to be smoother and tend to highlight the contour of the point cloud as seen from Fig. 3.4(b).

Local Hypergraph Filtering (LHF) Algorithm

Because we would like to preserve both the sharp features and the surface contour of the point cloud to achieve consistently good performance across different point clouds, we propose to apply several values of N_i for all points. In particular, we consider two different lengths N_a, N_b to construct two different sets of local signals. We would then integrate the filtered results.

Our local hypergraph filtering (LHF) based resampling consists of two main steps: i) hypergraph spectrum construction, ii) spectrum domain filtering. We first estimate hypergraph spectrum by applying the same process used in HKC and HKF algorithms. In this new LHF method, each point has its own (small-scale) hypergraph. We should estimate the hypergraph spectrum for each point using two different local signals $\mathbf{s}_{\mathcal{L},p,i,a} \in \mathbb{R}^{N_a \times 3}$ and $\mathbf{s}_{\mathcal{L},p,i,b} \in \mathbb{R}^{N_b \times 3}$.

Once the estimation of the corresponding hypergraph spectrum bases $\mathbf{V}_{\mathcal{H},i,a}$ and $\mathbf{V}_{\mathcal{H},i,b}$ is completed, we apply (3.4) to derive the Fourier transform $\hat{\mathbf{s}}_{\mathcal{L},p,i,a}$ and $\hat{\mathbf{s}}_{\mathcal{L},p,i,b}$, respectively.

Similar to spectrum filter in the HKF method, we define two thresholds $\theta_{i,a}$ and $\theta_{i,b}$, respectively, for $\hat{\mathbf{s}}_{\mathcal{L},p,i,a}$ and $\hat{\mathbf{s}}_{\mathcal{L},p,i,b}$. Two local sharpness metrics are further defined as

$$\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,a}) = \frac{\sum_{j \in \{1,2,\dots,\theta_{i,a}\}} \sum_{k=1}^3 |\hat{\mathbf{s}}_{\mathcal{L},p,i,a}(j, k)|}{\sum_{j \in \{1,2,\dots,N_i\}} \sum_{k=1}^3 |\hat{\mathbf{s}}_{\mathcal{L},p,i,a}(j, k)|}, \quad (3.10a)$$

$$\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,b}) = \frac{\sum_{j \in \{1,2,\dots,\theta_{i,b}\}} \sum_{k=1}^3 |\hat{\mathbf{s}}_{\mathcal{L},p,i,b}(j, k)|}{\sum_{j \in \{1,2,\dots,N_i\}} \sum_{k=1}^3 |\hat{\mathbf{s}}_{\mathcal{L},p,i,b}(j, k)|}, \quad (3.10b)$$

where $\theta_{i,a}$ and $\theta_{i,b}$ correspond to the thresholds for $\hat{\mathbf{s}}_{\mathcal{L},p,i,a}$ and $\hat{\mathbf{s}}_{\mathcal{L},p,i,b}$, respectively, with N_a and N_b as the respective corresponding lengths.

Upon completion of sharpness evaluation, for each signal point, we apply a weighted average of $\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,a})$ and $\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,b})$ to form a combined sharpness result

$$\gamma_i = \mathcal{W}\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,a}) + (1 - \mathcal{W})\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,b}), \quad (3.11)$$

where \mathcal{W} denotes the weight.

To balance the effect of two local sharpness metrics, we sort both $\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,a})$ and $\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,b})$ and design the weight \mathcal{W} according to the top α fraction of $\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,a})$ and $\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,b})$, denoted by Γ_a and Γ_b , respectively. We can define node-specific weights

$$\mathcal{W} = \frac{\Gamma_b}{\Gamma_a + \Gamma_b}. \quad (3.12)$$

Finally, we sort the γ_i of (3.11) and select the top $N_\alpha = \alpha N$ points as the resampled point cloud. The whole algorithm is summarized as Algorithm 3.3, also known as the LHF algorithm. In Algorithm 3, steps 1-3 correspond to the hypergraph spectrum estimation, while steps 4-6 describe the process of sharpness-based filtering as (3.11).

LHF Algorithm Complexity

The computational complexity of LHF is $O(N^2 + N \log N + N_k(N_k^2 + N_k + 1)N)$ for an unorganized point cloud, and $O(N \log N + (N_k^3 + N_k^2 + N_k + 1)N)$ for an organized point cloud. First, the complexity for generating $\mathbf{s}_{\mathcal{L},p,i}$ for all points is $O(N^2)$ and $O(N)$ for an unorganized point cloud and organized point cloud, respectively. Second, unlike HKC and HKF, the hypergraph spectrum bases would differ for each point and its corresponding local signals. Thus, one needs to estimate hypergraph spectrum basis $\mathbf{V}_{\mathcal{H},i,a}$ s and $\mathbf{V}_{\mathcal{H},i,b}$ s for each of point. Consequently, the total computational complexity is of $O(NN_k^3)$. Next, Fourier transforms and inverse Fourier transforms further require computation of $O(NN_k^2)$. Finally, the total complexity of computing γ_i is $O(NN_k)$, plus $O(N \log N)$ for sort the resulting γ_i .

Algorithm 3.3 Local Hypergraph Filtering (LHF)

Input: A point cloud with N nodes characterized by $\mathbf{P} = [\mathbf{p}_1^\top \cdots \mathbf{p}_N^\top]^\top$, resampling ratio α , local lengths N_a, N_b .

for $i = 1, 2, \dots, N$ **do**

1. Find the nearest $(N_a - 1)$ and $(N_b - 1)$ neighbors of point i ;
2. Use coordinates of point i and its $(N_a - 1)$ and $(N_b - 1)$ neighbors to estimate the hypergraph spectrum bases $\mathbf{V}_{\mathcal{H},i,a}$, $\mathbf{V}_{\mathcal{H},i,b}$, respectively;
3. Use hypergraph spectrum basis $\mathbf{V}_{\mathcal{H},i,a}$ and $\mathbf{V}_{\mathcal{H},i,b}$ to calculate the Fourier transform $\hat{\mathbf{s}}_{\mathcal{L},p,i,a}$ and $\hat{\mathbf{s}}_{\mathcal{L},p,i,b}$, respectively;
4. Calculate the local sharpness $\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,a})$ and $\gamma(\hat{\mathbf{s}}_{\mathcal{L},p,i,b})$ in (3.10a) and (3.10b);

end for

5. Calculate the weighted average of $\gamma_i(\hat{\mathbf{s}}_{\mathcal{L},p,i,a})$ and $\gamma_i(\hat{\mathbf{s}}_{\mathcal{L},p,i,b})$ using (3.11) and (3.12).

6. Sort the local sharpness γ_i and select the top $N_r = \alpha N$ points as the resampled point cloud.
-

3.2.4 Discussion

To conclude, HKC and HKF are more suitable for evenly distributed point clouds, while LHF exhibits more robust performance over unevenly distributed point clouds. All three algorithms have comparable complexity. The HKC algorithm has the complexity of $O(N^2 + N \log N + N_k(N_k + 1)N + N_k^3)$ for an unorganized point cloud, while in an organized point cloud, the computational complexity is only $O(N \log N + (N_k^2 + N_k + 1)N + N_k^3)$. The runtime of HKF is shorter than

HKC by eliminating the iHGFT. Such difference is more significant when applying a large N_k in the convolution kernel. The runtime of LHF is higher than both HKC and HKF because of the need to estimate hypergraph spectrum basis for every local signal.

3.3 Experimental results

Table 3.1: Numerical results of methods using point clouds of all shapes.

	HKC				HKF			
Noise Level	Precision	Recall	F1-Score	Mean Distance	Precision	Recall	F1-Score	Mean Distance
No Noise	0.3701	0.8558	0.4824	1.3731	0.3470	0.8581	0.4734	1.5441
10%	0.3502	0.8217	0.4579	1.5003	0.3265	0.7818	0.4308	1.5773
15%	0.3344	0.7722	0.4337	1.6691	0.3211	0.7480	0.4178	1.7409
20%	0.2686	0.6035	0.3436	1.9078	0.2452	0.5539	0.3116	2.0004
25%	0.2105	0.4577	0.2656	2.3482	0.2153	0.4615	0.2695	2.4392
30%	0.1774	0.3635	0.2181	2.8434	0.1795	0.3623	0.2190	2.9141
	LHF				GFR			
Noise Level	Precision	Recall	F1-Score	Mean Distance	Precision	Recall	F1-Score	Mean Distance
No Noise	0.3265	0.8069	0.4345	1.6662	0.4254	0.9168	0.5324	1.0931
10%	0.2017	0.5079	0.2670	1.9891	0.3917	0.8903	0.5015	2.7102
15%	0.1719	0.4252	0.2262	2.2813	0.3306	0.7198	0.4214	3.1772
20%	0.1492	0.3514	0.1915	2.5181	0.2407	0.5411	0.3068	3.6121
25%	0.1431	0.3339	0.1838	2.7349	0.1959	0.4182	0.2450	3.9423
30%	0.1317	0.2990	0.1677	2.9003	0.1624	0.3333	0.1994	4.1074
	EA				PCA-AC			
Noise Level	Precision	Recall	F1-Score	Mean Distance	Precision	Recall	F1-Score	Mean Distance
No Noise	0.3269	0.8417	0.4471	2.0033	0.3417	0.8605	0.4594	2.2471
10%	0.3264	0.8534	0.4487	1.7571	0.3451	0.8517	0.4592	1.8261
15%	0.3211	0.8429	0.4418	1.8173	0.3364	0.8379	0.4498	1.7489
20%	0.3099	0.8097	0.4251	1.9690	0.3226	0.8182	0.4349	1.7061
25%	0.2770	0.7245	0.3798	2.3033	0.3126	0.7966	0.4223	1.6027
30%	0.2377	0.6156	0.3241	2.6570	0.2945	0.7447	0.3966	1.8527

We now describe our experiment setup and present test results of the three proposed new resampling algorithms.

3.3.1 Edge Preservation of Simple Synthetic Point Clouds

As we described in Section III, one important resampling objective is to preserve sharp features in a point cloud such as edges and corners. In this part, we study the edge preserving capability of our proposed algorithms by testing over several simple synthetic point clouds. The reason for selecting

synthetic point clouds in this test is to take advantage of the known ground truth regarding edges and our ability to label them. We generate these synthetic point clouds by uniformly sampling the external surface of models constructed from combinations of cubes, cylinders and pyramids of various sizes. Examples of synthetic point clouds are shown in Fig. 3.5, where the points on edges are marked in red while the remaining points are in blue.

To measure the accuracy of the preserved edges, we evaluate the F_1 score, defined by

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \quad (3.13)$$

where precision denotes the fraction of edge points correctly preserved among all (false or correct) edge points for a resampling algorithm while the recall is the ratio of correctly preserved edge points versus all ground truth edge points. We also calculate the mean distance to their closest ground truth edge point respectively to show the ability of the algorithms in capturing the model surface.

We compare the three proposed algorithms with graph-based resampling and traditional edge detection methods. For graph-based resampling, we use the graph-based fast resampling (GFR) method with the Haar-like highpass graph filter introduced in [41] for comparison to show the strength of hypergraph in capturing multilateral features by generalizing traditional GSP and the advantage of applying hypergraph analysis in point cloud resampling over graph based methods. In addition, we also consider an edge detection method based on eigenvalues analysis (EA) and another edge detection scheme using Principal Component Analysis (PCA) and agglomerative clustering (PCA-AC) [48]. The parameters of GFR method are set to the typical values suggested by [41]. For EA and PCA-AC, we retain the points with higher cluster numbers or larger surface variation in the resampled point cloud to yield the same resampling ratio. Here, we set the resampling ratio $\alpha = 0.2$ for all point clouds as an example. Additional results with different resampling ratios will be further presented in Section IV-C. In order to study the robustness of algorithms, we also add 10% to 30% of Gaussian measurement noises, i.e., noises of

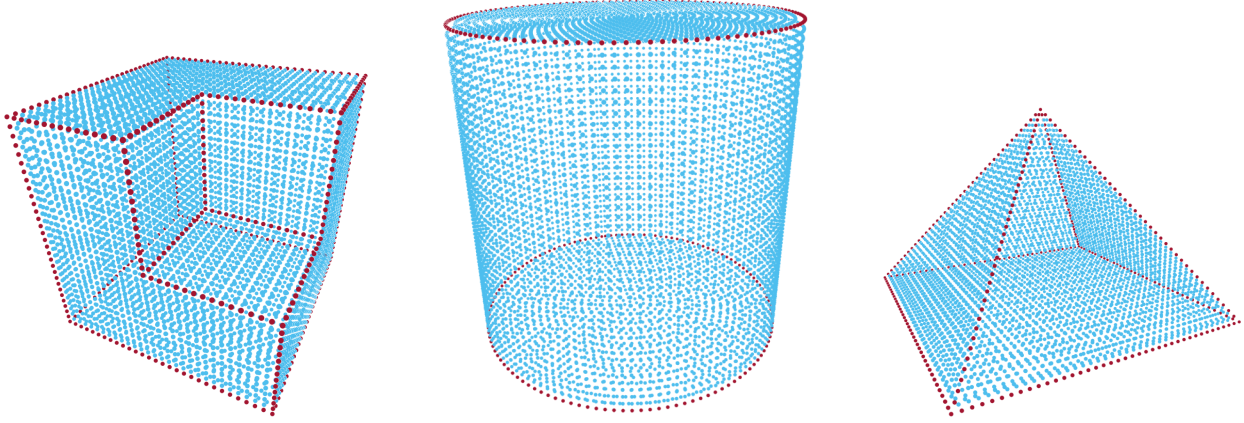


Figure 3.5: Synthetic Point Clouds with Labeled Edge.

$\mathcal{N}(0, (0.1d^{(i)})^2)$ to $\mathcal{N}(0, (0.3d^{(i)})^2)$, to point coordinates of the cloud. Here $d_c^{(i)}$ denotes the intrinsic resolution of point clouds. Table. 3.1 summarizes our test results. To better illustrate test results, we mark the best results among HGSP and GSP methods in bold font and underlined the best results of all methods. We also mark results with different colors from warm to cold to demonstrate poor to good performance (i.e. red marks poor performance and green marks good performance).

Compared with the GSP based GFR method, the newly proposed HKC algorithm performs robustly for point clouds under larger measurement noises. Since the local signals in HKC are defined by the number of points in the voxel of kernel, weaker noises on a single point with perturbation below $d_c/2$ would not affect local signals. In addition, the hypergraph-based methods exhibit a smaller mean distance than GFR in all the shapes, which indicates that our proposed methods generate more robust edges than the graph-based method.

On the other hand, LHF algorithm does not perform well against noisy point clouds because sizable noises directly distort the local hypergraph and neighbors. Overall, our proposed HGSP-based methods demonstrate stronger robustness than the traditional graph-based GFR algorithm for noisy data. Using a generic signal processing approach they also deliver competitive performance against non-graph based EA and PCA-AC methods that were designed specifically for edge detection.

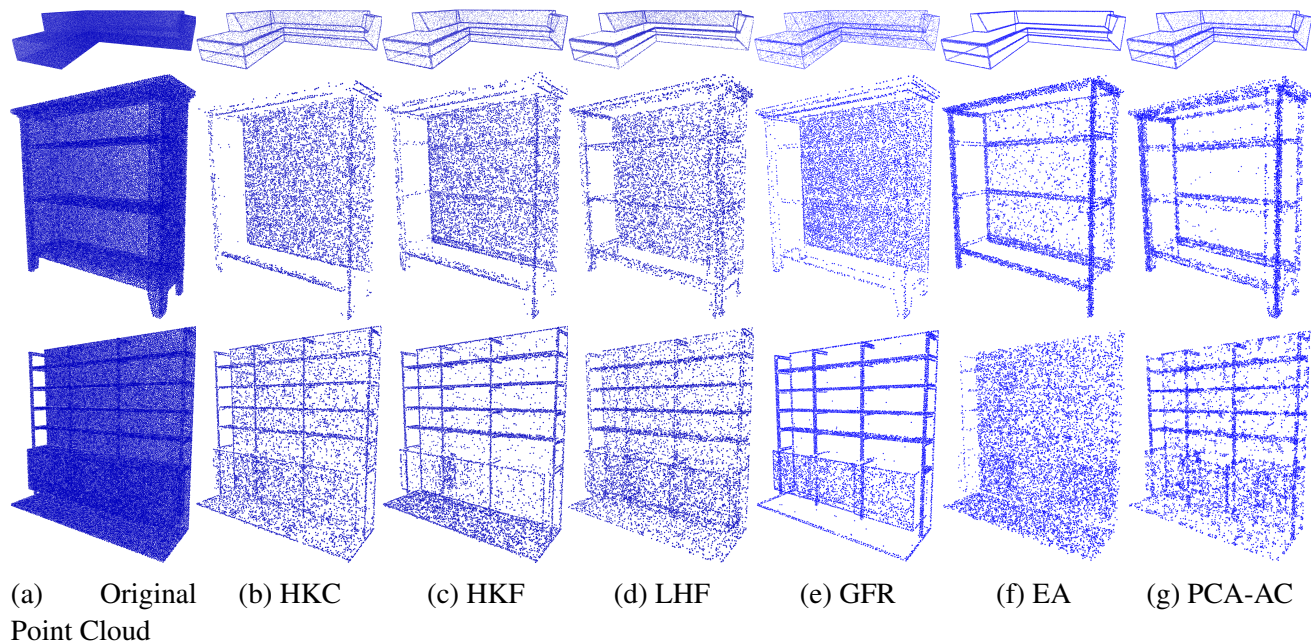


Figure 3.6: Examples of Edge Detection for Realistic Practical Point Clouds.

3.3.2 Edge Preservation Results on Real-Life Point Clouds

To test our proposed algorithm in a more general setting, we also implement edge detection based on our resampled data in more complex and practical point clouds. For these datasets, there are no explicit ground truth edges to provide quantitative results. Therefore, we present these results as visible point cloud pictures to illustrate the test performance in Fig. 3.6, where the left column shows the original point clouds and the right columns are the resampled point clouds for our proposed methods and methods under comparison, respectively. From Fig. 3.6, all methods can detect the edges of the model. Hypergraph and graph based methods tend to contain some points on a surface, while EA and PCA-AC methods tends to emphasize points closer to edges for point clouds of sofa and bookshelf in the first and second rows. We also test the algorithms on both the Boxer point cloud in 8i Voxelized Surface Light Field (8iVSLF) Dataset [50] and the Bi-plane point cloud in ScanLAB Projects [51]. We considered resampling ratio α of 0.001 and 0.005, respectively. The results are shown in Fig. 3.7 and Fig. 3.8. As shown in Fig. 3.7, the HKC and HKF methods can detect continuous edges and textures on the clothes of Boxer point cloud, similar to results from EA and PCA-AC. However, GFR method fails to detect continuous textures. It tends

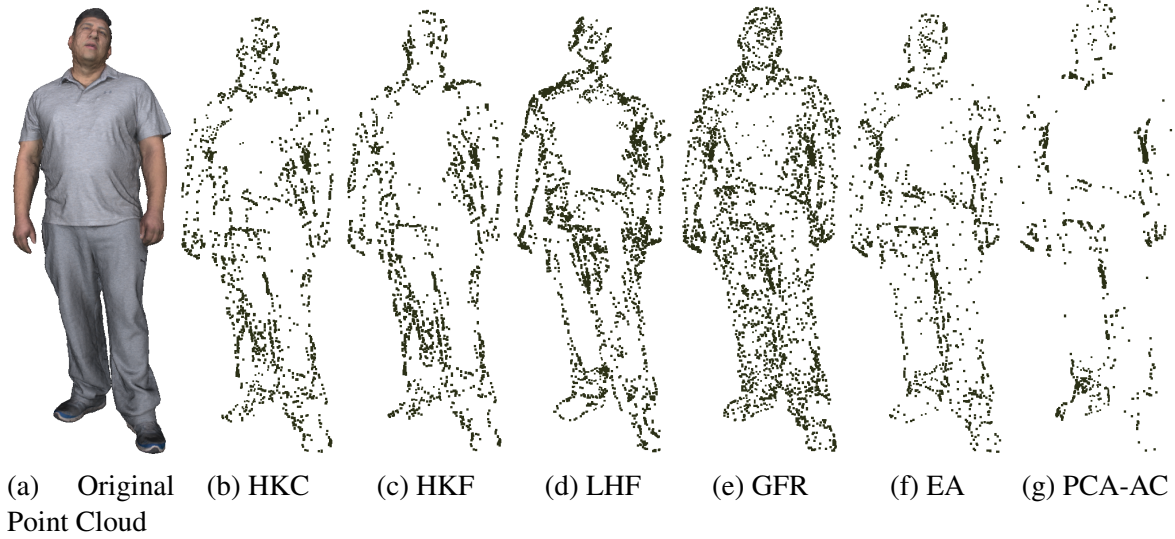


Figure 3.7: Examples of Edge Detection for Boxer Point Cloud in 8iVSLF Dataset

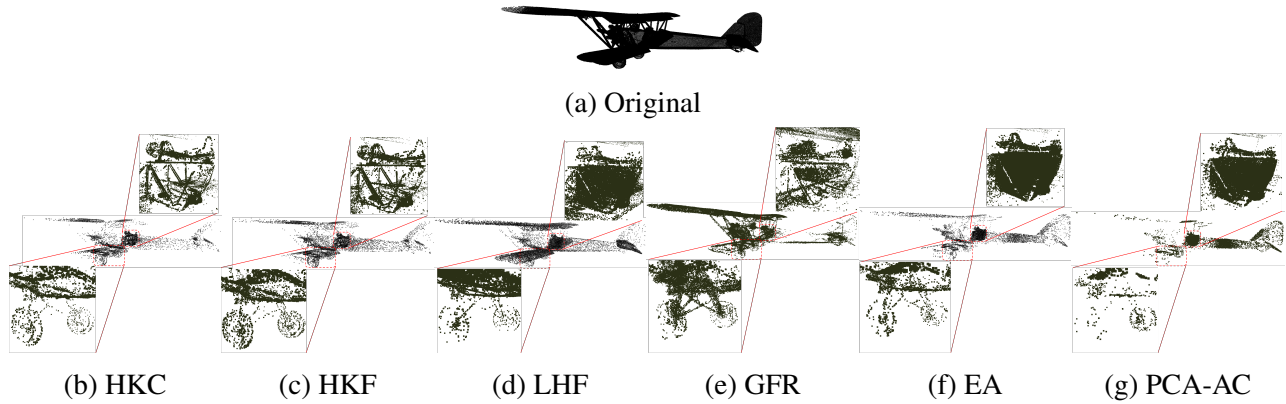


Figure 3.8: Resampled results using ScanLAB Projects: Bi-plane point cloud data set

to keep more points on the entire surface. Furthermore, as shown in Fig. 3.8, both HKC and HKF methods are able to capture clear edges and preserve the shape of the cabin on the biplane better than other methods. We also note that the LHF method captures more clear features of the wheels. These results show that our resampling methods effectively detect 3D object contours (outlines) in real scenarios.

3.3.3 Point Cloud Recovery from Resampling

In the next test, we investigate the new algorithms' ability to preserve high degree of point cloud information after resampling. In particular, we shall attempt to recover the dense point cloud after

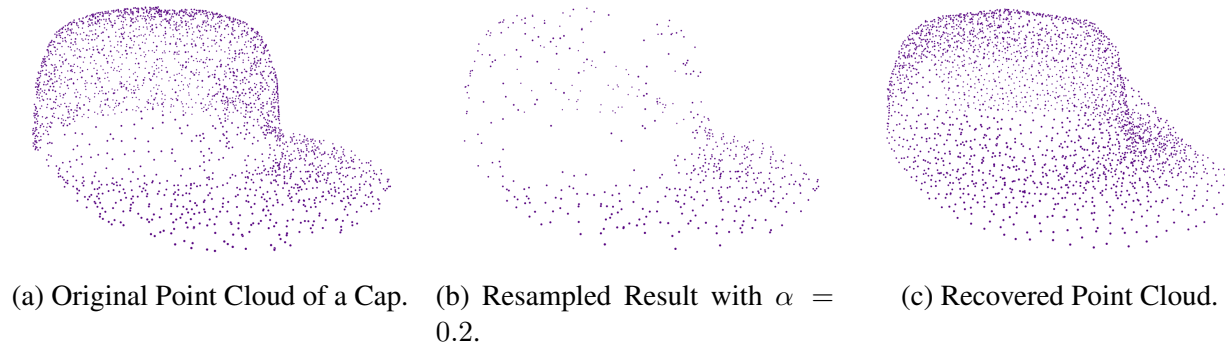


Figure 3.9: Example of original point cloud, resampled point cloud and the recovered point cloud for HKC method.















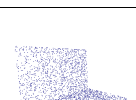
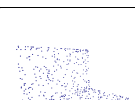


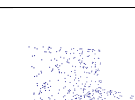
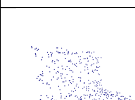
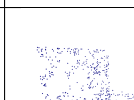
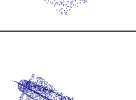
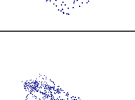
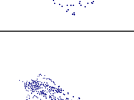
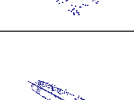
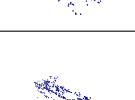
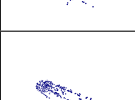
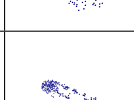
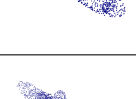


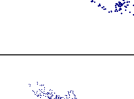
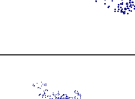

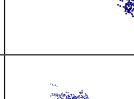
resampling and assess the similarity between the original point cloud and the recovered point cloud from resampling.

Dense Point Cloud Recovery

A typical method for dense point cloud recovery consists of two steps: a) reconstructing the surface of object from the resampled point cloud; and b) sampling the reconstructed object surface to generate a recovered point cloud. Since points of edge preserving resampled point clouds tend to concentrate near areas of high local variations, e.g., edges/corners, points of these resampled point clouds are not uniformly distributed, as shown in Fig. 3.9(b). For this reason, some generic surface reconstruction methods such as Poisson reconstruction [52] may perform poorly on such sparse point clouds. We must pay special attention to surface reconstruction methods chosen for such type of resampled point cloud data.

In order to reconstruct surfaces from edge preserving and sparsely resampled point clouds, we propose to first construct the alpha complex [53] from the resampled point cloud, since this approach is well-known and widely-used method for surface reconstruction based on 3D coordinates of points, and is also quite robust when handling unevenly distributed point clouds. To mitigate the potentially degrading impact of imperfect reconstruction, we decide to reconstruct six different surface models for each resampled point cloud by applying different parameters. We then apply Poisson-disk resampling to sample the alpha complex to form a recovered point cloud.

Table 3.2: Example of original and resampled point clouds with resampling ratio $\alpha = 0.2$.


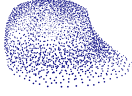
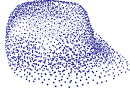

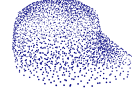
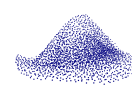
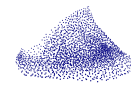


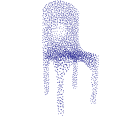






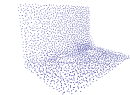
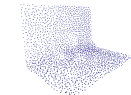
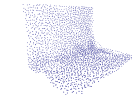
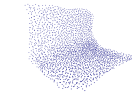

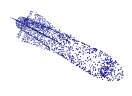
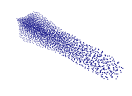
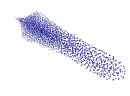
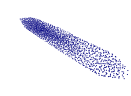
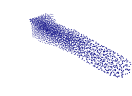
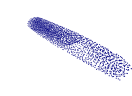
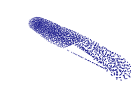
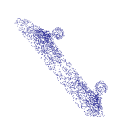
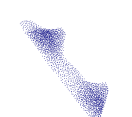
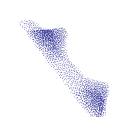


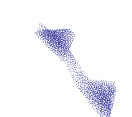

Original	HKC	HKF	LHF	GFR	EA	PCA-AC
						
						
						
						
						

To further mitigate the effect due to the possible construction of extraneous surfaces absent from the original object, we select a threshold distance d_θ three times the intrinsic resolution of the original point cloud. Using the threshold distance, we only retain the best recovered point cloud which contains the largest number of points that are within the threshold distance d_θ from the original point cloud.

Distance Between Point Clouds

To assess the quality of point cloud recovery, we need to define distances between the original and the recovered point clouds. Let p_i denote a point in the original and $p_{c,j}$ denote a point in the recovered point cloud. When computing our distance between two point clouds, we neglect any distances between point p_i in the original point cloud and $p_{c,j}$ in the recovered point cloud such that the minimum distances $\min_j \|p_i - p_{c,j}\|$ and $\min_i \|p_i - p_{c,j}\|$ are greater than d_θ .

Table 3.3: Example of original and recovered point clouds with resampling ratio $\alpha = 0.2$.

Original	HKC	HKF	LHF	GFR	EA	PCA-AC
						
						
						
						
						

We define a distance and a dual distance between the original and the recovered point cloud as

$$D_0 = \frac{1}{N_1} \sum_{i=1}^{N_1} \min_{j: \|p_i - p_{c,j}\| < d_\theta} \|p_i - p_{c,j}\|, \quad (3.14)$$

$$\check{D}_0 = \frac{1}{N_2} \sum_{j=1}^{N_2} \min_{i: \|p_i - p_{c,j}\| < d_\theta} \|p_i - p_{c,j}\|, \quad (3.15)$$

where N_1 is the number of points in the original point cloud that satisfy $\|p_i - p_{c,j}\| < d_\theta$ for some $p_{c,j}$ and N_2 is the number of points in the recovered point cloud that satisfy $\|p_i - p_{c,j}\| < d_\theta$ for some p_i . In other words, D_0 is the average distance for points that are in the original point cloud within d_θ from the closest points in the best recovered point cloud. The dual distance \check{D}_0 is the average distance for points in the best recovered point cloud that are within d_θ from their closest points in the original point cloud.

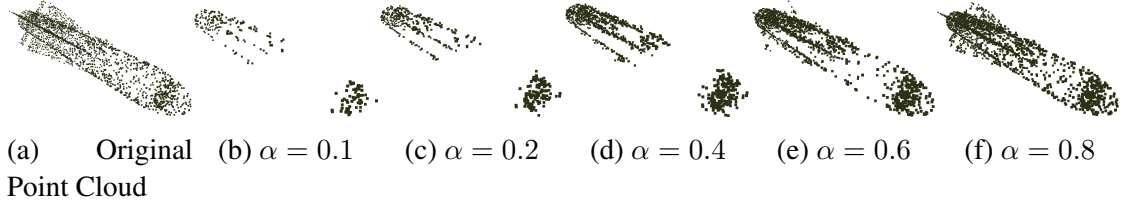


Figure 3.10: Example of resampled results of EA with different α .

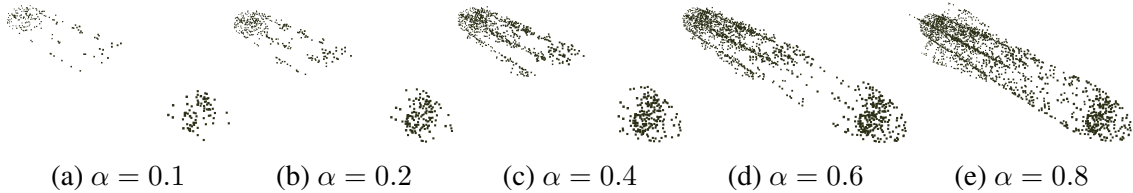


Figure 3.11: Example of resampled results of PCA-AC with different α .

Visual and Numerical Results

We use six different categories of point clouds from ShapeNet [54] in our experiments. Similar to experiments discussed earlier, we test our HKF method together with the GSP-based GFR method in [41] plus the EA and PCA-AC methods from [48]. We also test each method using downsampled Boxer point cloud in 8iVSLF dataset [50]. We first uniformly downsample the Boxer point cloud with 10% points remaining in the output point cloud, before sending the resulting point cloud as the input for each method. We use the downsampled point cloud for memory and transport efficiency because the original Boxer point cloud has nearly 3,500,000 points. For fairness, we use the same resampling ratio for all the methods.

Our experiments follow the following steps. First, we apply resampling and edge detection methods to calculate the resampled point clouds with resampling ratio α . Next, we apply our proposed recovery method to generate recovered point clouds. Based on the resampled point clouds, we compute the numerical performance metrics for different algorithms in comparison. We measure the performance under three metrics: 1) distance defined in (3.14); 2) average of distance and dual distance as defined in (3.14) and (3.15), respectively; and 3) average number N_1 of points within the threshold d_θ between the original and recovered point cloud. Smaller distance and larger number of points within the threshold indicate better performance.

Table 3.4: Mean distance between the best recovered point cloud and the original point cloud for resampling ratio $\alpha = 0.2$ using ShapeNet dataset.

Categories	HKC	HKF	LHF	GFR	EA	PCA-AC
Cap	0.0111	0.0101	0.0117	0.0102	0.0087	0.0115
Chair	0.0111	0.0113	0.0116	0.0118	0.0125	0.0126
Laptop	0.0106	0.0106	0.0103	0.0105	0.0110	0.0110
Mug	0.0134	0.0134	0.0150	0.0141	0.0150	0.0147
Rocket	0.0069	0.0069	0.0078	0.0070	0.0070	0.0073
Skateboard	0.0079	0.0080	0.0080	0.0079	0.0082	0.0083
Average	0.0102	0.0101	0.0107	0.0103	0.0104	0.0109

Table 3.5: Average number of points within d_θ between the best recovered point cloud and the original point cloud for resampling ratio $\alpha = 0.2$ using ShapeNet dataset.

Categories	HKC	HKF	LHF	GFR	EA	PCA-AC
Cap	2628.3	2632.4	2315.7	2635	1427.5	2160.4
Chair	2656.1	2657.9	2658	2653.9	2458.2	2469.9
Laptop	2754.4	2784.4	2785.6	2774.3	2626.4	2584.6
Mug	2818.6	2819.9	2716.6	2810.7	2633.7	2418.2
Rocket	2364.4	2361	2317.4	2360.6	1904.4	2004.5
Skateboard	2559.8	2562	2568.2	2563.1	2409.9	2381.2
Average	2630.3	2636.3	2560.25	2632.93	2243.35	2336.47

Table 3.6: Mean distance between the best recovered point cloud and the original point cloud for different resampling ratios using downsampled Boxer point cloud in 8iVSLF dataset.

α	HKC	HKF	LHF	GFR	EA	PCA-AC
0.1	2.7889	2.7942	2.6646	2.7705	2.7656	2.3536
0.2	2.4446	2.1716	2.8015	2.4362	2.7639	2.2698
0.4	1.9837	1.9841	2.4528	1.9747	2.7877	2.3825
0.6	1.8525	1.8522	1.9854	1.8522	2.8042	2.5412
0.8	1.7802	1.7790	1.8574	1.8598	2.8128	2.6750
Average	2.1700	2.1162	2.3523	2.1787	2.7868	2.4444

To start, Table. 3.2 and Table. 3.3 provides a set of the resampled point clouds and their corresponding recovered ones from different methods of ShapeNet dataset. Visual inspection shows that our proposed HKC, HKF, and LHF algorithms generally deliver consistently strong results in resampling and recovery of point clouds, regardless of the dataset under study.

To quantitatively illustrate the performance comparison, Table. 3.4 and Table. 3.5 present the numerical results for $\alpha = 0.2$. From the test results, we observe that our HKF algorithm exhibits

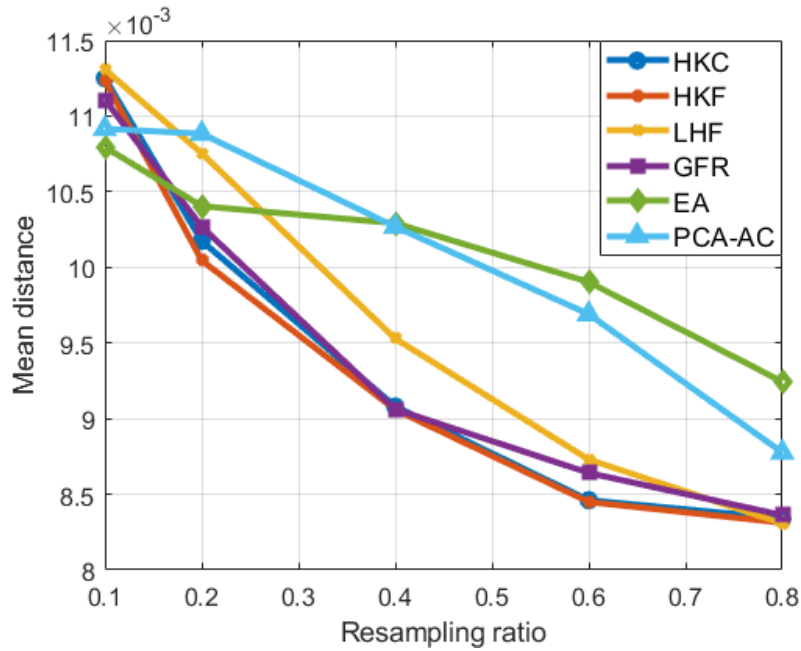
Table 3.7: Average of distance and dual distance between the best recovered point cloud and the original point cloud for different resampling ratios using downsampled Boxer point cloud in 8iVSLF dataset.

α	HKC	HKF	LHF	GFR	EA	PCA-AC
0.1	2.5368	2.5450	2.4381	2.4859	2.5660	4.9831
0.2	2.2974	2.1641	2.4492	2.3440	2.5225	2.3647
0.4	2.1475	2.0771	2.3687	2.0614	2.5381	5.2107
0.6	2.0890	2.0892	2.0752	2.0856	2.5576	5.2772
0.8	1.9825	1.9816	2.0204	2.0196	2.5703	2.5087
Average	2.2160	2.1714	2.2703	2.1993	2.5509	4.0689

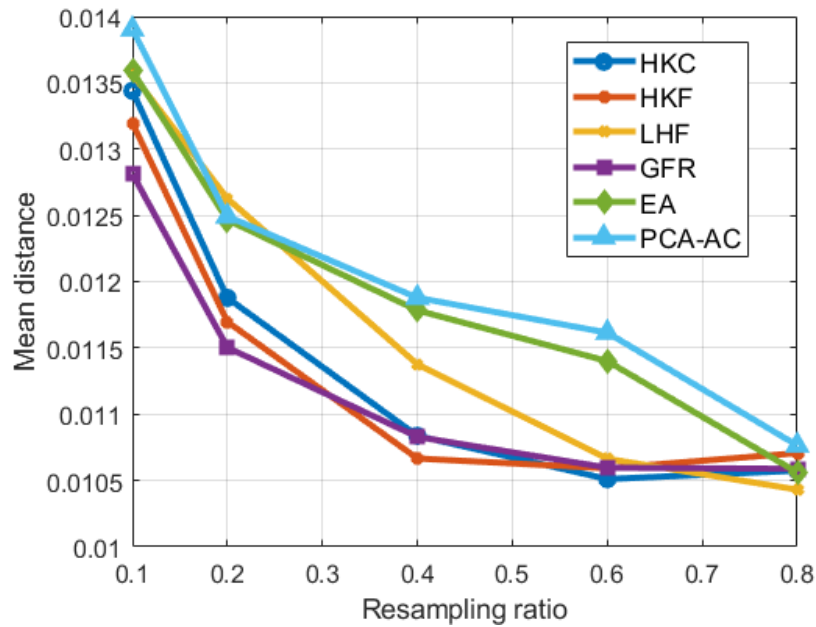
the best performance in terms of both mean distance and number of matched points versus the traditional GFR graph method and two edge detection methods. Compared with the edge detection methods, our proposed algorithms consistently retain larger numbers of points within the threshold d_θ in most point cloud categories.

The comparison also demonstrates a potential issue of the specialized edge detection based methods. In particular, resampled point cloud of edge detection methods may over-emphasize only part of the original point cloud, as shows in Fig. 3.10 and Fig. 3.11. The points in the middle to the top of the rocket are kept only for α larger than or equal to 0.6. As a result, substantial number of points may not be retained by the generic edge detection methods during resampling.

We also examine the effect of sampling ratio α and graphically illustrate the variation of mean distance, average of distance and dual distance, and average number within threshold against different sampling ratios in Fig. 3.12, Table. 3.6 and Table. 3.7. Here we use the same set of point clouds in the numerical results of ShapeNet dataset for $\alpha = 0.2$. Note that the behavior of PCA-AC is more erratic because it could over-emphasize certain parts of the original point cloud, as shown in Fig. 3.13. It is clear and intuitive that higher resampling ratio leads to better performance of all methods under study. It is important to note that our proposed methods exhibit performances superior to the traditional EA and PCA-AC methods in terms of mean distance for various resampling ratios. This result indicates that our proposed hypergraph-based methods tend to preserve the geometric information more efficiently in resampling. The hypergraph-based methods also exhibit better results with respect to the number of corresponding nodes between



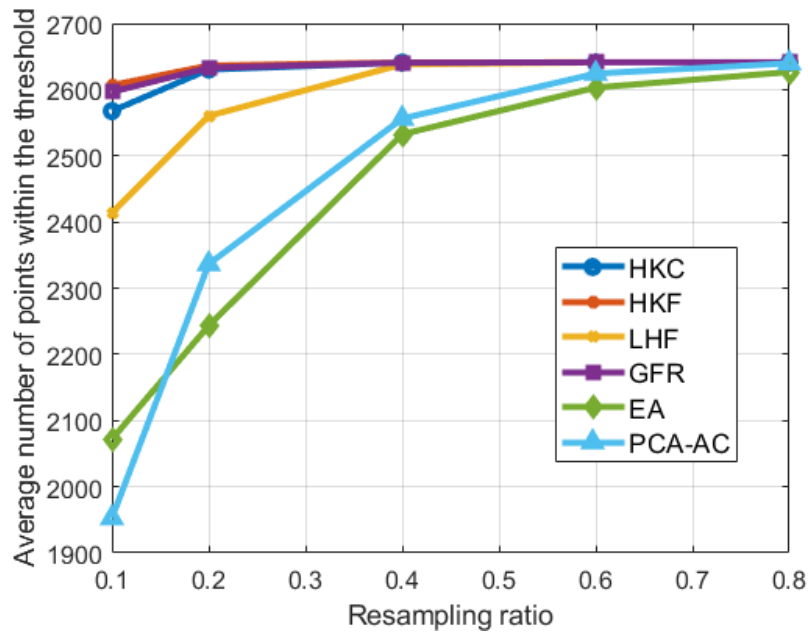
(a) Mean distance against α .



(b) Average of distance and dual distance against α .

Figure 3.12: Plots of recovered accuracy against resampling ratio α of all methods in ShapeNet Dataset

reconstructed and original point clouds.



(c) Average number of points in recovered point cloud d_θ against α .

Figure 3.12: Continue of the Plots of recovered accuracy against resampling ratio α of all methods in ShapeNet Dataset



Figure 3.13: Resampling result of PCA-AC method using resampled Boxer Point cloud.

3.3.4 Runtime of all methods

To compare the complexity of the proposed methods with traditional graph-based and PCA-based methods, we record the runtime of all methods using the downsampled Boxer point clouds in

8iVSLF dataset [50] with different number of points. The resampling ratio α is 0.2 for all methods and the result is as Fig. 3.14. We also record the average runtime of biplane point cloud with the resampling ratio of 0.005. The result is summarized in Table 3.8.

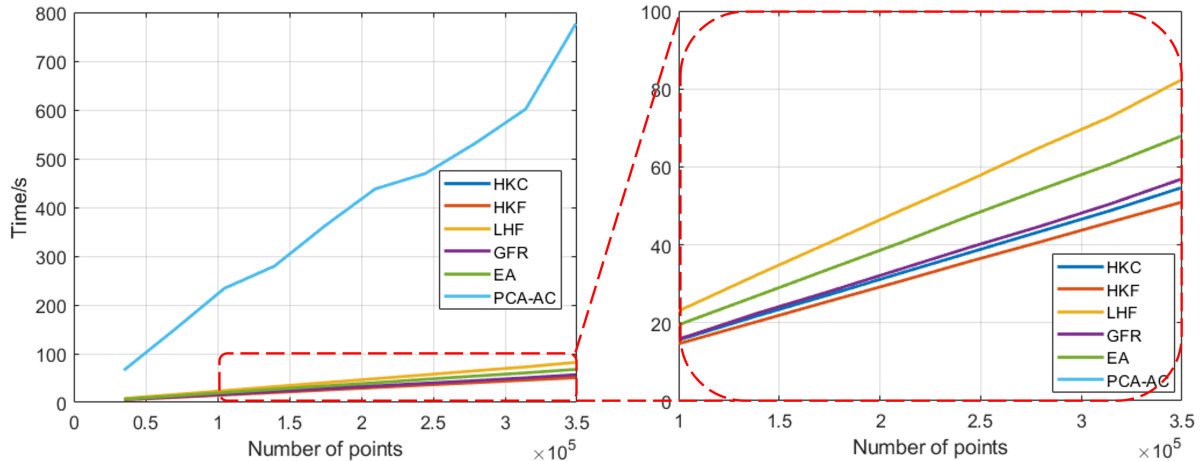


Figure 3.14: Runtime of all methods

Table 3.8: Comparison of average running time (in seconds) for point clouds in different datasets.

	Boxer (s)	Bi-plane (s)
HKC	54.603	1501.288
HKF	50.880	1445.801
LHF	82.243	2225.681
GFR	56.815	6277.067
EA	67.840	1937.953
PCA-AC	776.976	4793.139

The runtime result shows that our HKC and HKF algorithms require a shorter runtime than GFR method. The runtime of the LHF method is longer than GFR method because we used two different local signal lengths N_a and N_b such that the entire progress is calculated twice, unlike for HKC and HKF.

Note that the computational time of GFR increases drastically for the biplane point cloud. This happens because it is nearly impossible to store the entire adjacency matrix in memory and implement the GFR processing for such a large point cloud. To make the GFR method feasible, we modified GFR by calculating the optimal resampling distribution for each point directly without

forming the adjacency matrix. Unlike GFR, our proposed HKC and HKF algorithms are more efficient and do not require the formation of hypergraphs for the whole point cloud.

In summary, our test results demonstrate the efficiency of the proposed resampling while preserving the underlying structural features and geometric information among point cloud data. They further demonstrate that hypergraph presents a promising alternative beyond regular graph for modeling point clouds in some point cloud related applications.

3.3.5 Parameter selection

Table 3.9: Mean Distance of Different Methods.

Noise Level	No Noise	10%	15%	20%	25%	30%
HKC						
Basic (d_c)	1.915	2.176	3.412	4.027	4.291	4.374
Noise-dependent	↑	2.071	2.399	2.574	2.679	2.843
Optimized	1.373	1.500	1.669	1.908	2.348	2.843
HKF						
Basic (d_c)	1.910	2.370	3.533	4.088	4.308	4.376
Noise-dependent	↑	2.090	2.498	2.671	2.757	2.914
Optimized	1.492	1.577	1.741	2.000	2.439	2.914
LHF						
Basic (6)	3.090	3.808	4.050	4.209	4.281	4.331
Noise-dependent	↑	2.807	3.043	3.119	3.147	3.147
Optimized	1.666	1.989	2.281	2.518	2.735	2.900
GFR						
Optimized	1.093	2.710	3.177	3.612	3.942	4.107

Table 3.10: Robustness over Different Parameters: Edge Detection for Synthetic Dataset and Recovery for Realistic Dataset.

	HKC	HKF	LHF	GFR
Variation of Mean Distance (Synthetic Dataset)	1.2321	1.1906	0.5416	1.4154
Variation of Average Distance (ShapeNet Dataset)	3.04E-07	2.25E-07	6.29E-06	6.97E-07

In this part, we provide guidelines for the parameter selection. Given a point cloud without prior knowledge of overall measurement error (noise), we suggest setting the kernel size as the

resolution d_c and $N_i = 6$ as baseline values. When the measurement accuracy is known (i.e., the level of noise is given in practice), we recommend increasing 30% of the kernel size each time the noise grows by 10% for HKC and HKF. For LHF, we increase N_i by 4 for every 10% noise increase. We compare the mean distance of edge detection with GFR in the synthetic datasets under different guidelines as Table 3.9. From the results, we can see that our proposed methods deliver better performance under some simple guidelines without exhaustive tuning or hindsight.

3.4 Conclusion

This chapter investigates new ways for efficient and feature preserving resampling of 3D point cloud based on hypergraph signal processing (HGSP). We have established HGSP as an efficient tool to model multilateral point relationship and to extract features in point cloud applications. We have proposed three new methods based on HGSP kernel convolution and spectrum filtering. Although typical HGSP tools tend to require high computational complexity, our proposed algorithms bypass certain steps for hypergraph construction and only require modest complexity to implement. Our experimental results have demonstrated that the proposed hypergraph resampling algorithms can outperform traditional graph-based methods in terms of feature preservation and robustness to measurement noise.

Chapter 4

Body Motion Segmentation via Multilayer Graph Processing

4.1 Introduction

Human motion analysis has recently emerged as an active research field, stemming from its broad applications in many areas, ranging from human-robot interaction to autonomous driving [55–57]. Among a variety of tasks, human motion segmentation serves as an important preprocessing step, benefiting a wide range of motion/action-related tasks, such as gesture recognition, human activity recognition, and human gait analysis [58]. Generally, human motion segmentation aims to divide a long sequence of motion frames into several short, non-overlapping temporal sections [59], each of which has its distinct physical meaning, as shown in Fig. 4.1(a). Specifically, we aim to process motion skeleton data extracted from video or synthesized from multiple sensors, instead of the raw video footage. Thus, the question of how to cluster the motion video/sequence into meaningful clips plays an important role in human motion analysis.

Despite many works focused on temporal segmenting videos about motions [60–62], the existing motion segmentation methods within the scope of processing motion sequences can be categorized into either unsupervised segmentation or supervised classification. Unsupervised

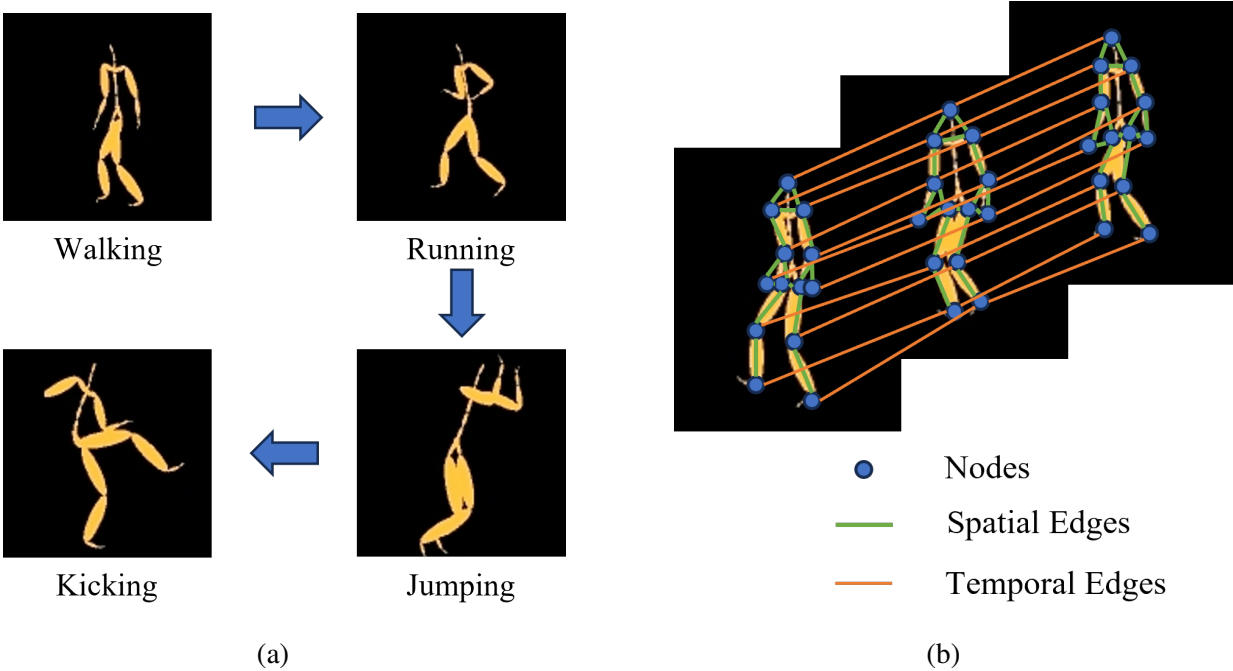


Figure 4.1: Illustration of Human Motion: (a) Example of Motion Segmentation; (b) Spatial-Temporal Relationships Modeled By Multilayer Graph.

motion segmentation usually utilizes the temporal dependency in the video sequence, including that in accelerometer and gyroscope data, to segment different motions. Typical clustering methods include low-rank transfer clustering [59], hierarchical aligned cluster analysis (HACA) [63], and subspace clustering [64]. However, these clustering-based approaches usually focus on the temporal information and require some additional information, such as the exact number of different actions, which may not be contained in the datasets. On the other hand, supervised approaches usually assume prior-labeled datasets to train the deep learning networks. For example, in [65], a deep neural network is proposed for human action classification. Other methods also include long short-term memory networks (LSTM) [66] and few-shot learning [71]. Despite some notable successes, most learning based methods need training sets that are often labeled by human supervisors, which may be inaccurate and unavailable in real applications, limiting the practicability of the supervised learning methods. The development of a more efficient method for motion segmentation remains an open direction of endeavor.

Recent development of geometric approaches, including graph signal processing (GSP) [1] and

graph neural networks (GNN) [2, 3], has provided another promising alternative to solve human motion segmentation, for both supervised and unsupervised scenarios. In [72], a spatial temporal graph convolutional networks (ST-GCN) is introduced for skeleton-based action recognition. Extended from ST-GCN, the authors of [73] proposed a multi-stage spatial-temporal graph convolutional neural network (MS-GCN). In addition, [79] introduced spatio-temporal graph cuts for event-based motion segmentation. However, most existing works assume that human motion relies on a homogeneous spatial graph structure whereas, in fact, an alternative multilayer heterogeneous structure could be more informative. As shown in Fig. 4.1b, joints in each temporal framework might have different underlying geometric structures due to the motion dynamics, suitable for a multilayer graph (MLG) structure. Moreover, limited by the homogeneous spatial structure, existing works are inefficient in processing the inter-layer (temporal) and intra-layer (spatial) correlations jointly, but they separate the spatial and temporal analysis. How to jointly extract spatial-temporal geometric features remains a challenge. Fortunately, within the context of GSP, a multilayer graph signal processing (M-GSP) framework has been introduced for MLG based on tensor representation [17]. Different from traditional multiway GSP (MWGSP) [81], M-GSP allows different spatial layers to represent heterogeneous geometric structures, and defines a joint MLG spectral space for data analysis. M-GSP has shown great potentials in spectrum analysis, image compression, clustering, hyperspectral image segmentation and classification [18, 19].

To capture the heterogeneous underlying geometry and address the spatial-temporal relationships jointly, we apply M-GSP and propose two novel MLG-based methods for unsupervised human motion segmentation. More specifically, we first introduce the MLG modeling for human motion datasets and define a MLG singular space for motion analysis. We then investigate the M-GSP spectral properties and design a M-GSP Haar-like highpass filter for feature extraction, based on which a spectral segmentation is implemented. To reduce the complexity and enhance the efficiency, we present another M-GSP based approach according to the tensor representation of MLG. Our experimental results demonstrate the power of M-GSP in extracting

spatial-temporal features, as well as the efficiency of the proposed method. We summarize our contributions as follows:

- To characterize the spatial-temporal geometric correlations in human motion sequences, we introduce an MLG model, together with its tensor representation, for motion segmentation. To our best knowledge, we are the first to apply M-GSP/GSP in human motion analysis.
- To derive the geometric features of human motions, we propose an M-GSP spectral method for unsupervised motion segmentation, and to investigate the properties in the MLG singular space.
- Beyond spectral analysis, we also introduce an M-GSP based motion segmentation in the vertex domain by exploring the tensorial and structural features of MLG.
- Based on guidelines for parameter selection, our experimental results demonstrate the efficacy of the proposed methods in both unsupervised and supervised testing setups.

We organize the rest of the chapter as follows. In Section 4.2, we first review related works on motion segmentation of motion capture data. Next, we present the MLG models for human motion datasets in Section 4.3, with which we propose two novel unsupervised motion segmentation algorithms in the spectrum domain and the vertex domain, respectively, in Section 4.4. We present the experimental results of the proposed methods in both supervised and unsupervised setup in Section 4.5, before summarizing our work in Section 4.6.

4.2 Related Works

In this section, we first briefly review the existing works on motion segmentation. Generally, the existing motion segmentation can be categorized into either unsupervised motion clustering or supervised motion recognition.

4.2.1 Unsupervised motion clustering

The unsupervised motion clustering usually exploits the global information of a motion sequence, and divides the sequence into several meaningful sections [84]. For example, conventional clustering algorithms, such as K-means clustering [82] and spectral clustering [83], can be applied for human motion segmentation. However, these traditional clustering algorithms are often inefficient to capture geometric information in human motions. For example, the efficacy of some conventional clustering algorithms, such as K-means clustering, are constrained by the fact that they are only optimal for spherical clusters, making them unsuitable for capturing distinctive distances of sections in motion sequences [63]. Furthermore, even for the same motion, the lengths of segments in human motion vary due to inconsistency in movement speeds, leading to difficulties for these basic clustering algorithms. Extending traditional clustering, an aligned cluster analysis (ACA) together with its extension hierarchical ACA (HACA) is introduced in [63] as a generalization of kernel k-means (KKM) and spectrum clustering (SC) for time series clustering and embedding. The ACA algorithm combines dynamic time alignment kernel (DTAK) with KKM and SC to better capture features of segments with different lengths. Leveraging ACA, the HACA provides a hierarchical structure at different temporal scales to refine temporal segmentation results while reducing computational complexity. Other typical algorithms also include low-rank transfer clustering [59], transfer subspace clustering, kernel subspace clustering [64] and auto-encoder [85].

In addition, most existing clustering algorithms focus on temporal dynamics while ignoring the joint spatial-temporal information, which can be more informative in realistic scenarios. These algorithms simply treat all data within the same time frame as a vector of features. However, in realistic scenarios, changes of spatial connections within a single frame, along with the consideration of joint spatial-temporal connections can provide more informative insights. As shown in Fig. 4.1, a multilayer graph (MLG) built on the motion sequence can naturally capture spatial-temporal connections. Also, the M-GSP framework proposed in [17] has shown its ability to capture and process joint spatial-temporal information. In this chapter, we investigate the MLG-based clustering algorithm to characterize both spatial and temporal correlations, which

is capable of efficient joint spatial-temporal processing.

4.2.2 Supervised Motion Recognition

Supervised motion recognition usually assumes a given prior-labeled dataset to train the neural networks. For example, a deep neural network called SE3-NETS was proposed in [65] to segment point clouds into distinct objects and jointly predict their rigid body motion. Another typical type of learning framework is temporal convolutional networks (TCN). Specifically, a multi-stage temporal convolutional networks (MS-TCN) was proposed in [94]. By stacking multiple stages sequentially, MS-TCN can process on all temporal resolutions of videos to achieve better results. However, the predictions of each stage in MS-TCN tend to have over-segmentation errors. To address this issue, MS-TCN++ was proposed in [67] by introducing a dual dilated layer, which combines both large and small receptive fields. Additionally, the authors in [68] proposed a new cascading paradigm and a smoothing operation to enhance the adaptability and improve prediction confidence of the model for ambiguous frames. Another approach called efficient two-step network (ETSN) was introduced in [69] by using local burr suppression (LBS) to significantly reduce the over-segmentation errors. To further improve the performance, [70] presented a hierarchical action segmentation refiner (HASR), which can be plugged into MS-TCN model to refine the segment labels by referring to the entire video.

Recently, graph neural networks have attracted significant attention in motion segmentation. A spatial temporal graph convolutional network (ST-GCN) has been introduced in [72] for skeleton-based action recognition. Later, the authors in [74] extended ST-GCN by introducing the stacked hourglass architecture to improve the accuracy. Meanwhile, a decoupling GCN model was proposed in [75]. Similar to the decoupling aggregation mechanism in CNNs, this decoupling GCN model can improve the graph modeling ability without additional cost. Another graph convolutional network called central difference graph convolution (CDGC) was proposed in [76] by considering aggregating both node and gradient information in the learning model. Despite the successes, their graph modeling are normally limited by physical adjacency of the

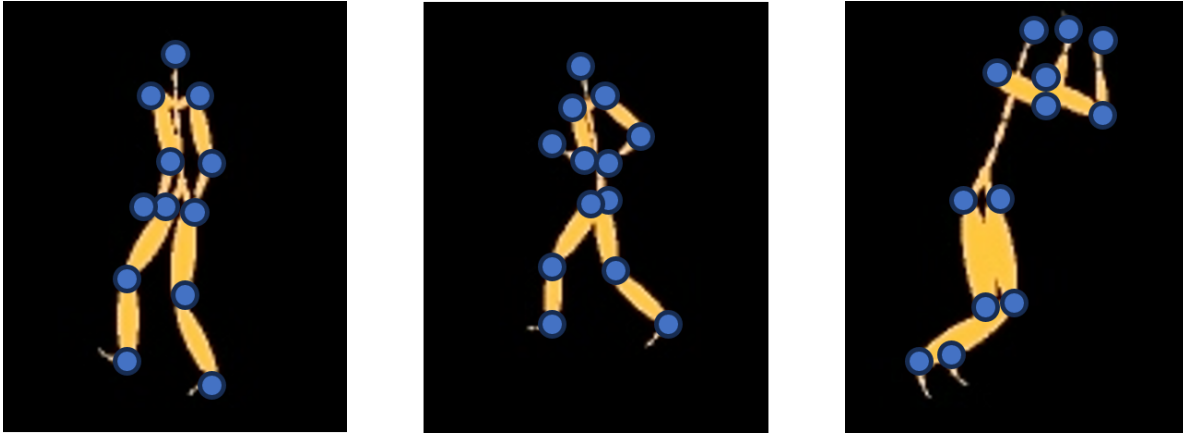


Figure 4.2: Example of Skeleton-based Human Motion Dataset in CMU graphics lab motion capture database.

elements. To address this issue, the authors in [77] introduced two separate GCN models for spatial and temporal information modeling. To further improve the performance, the authors of [73] combined temporal convolutional neural network (TCN) with ST-GCN blocks to build a multi-stage spatial-temporal graph convolutional neural network (MS-GCN), which can lead to better segmentation. In addition, the authors of [71] added the connectionist temporal classification (CTC) into MS-GCN to improve temporal alignment between network predictions and ground truth. In [92], a local self-expression subspace learning network was proposed, where local self-expression layers maintain the representation relations between temporally adjacent motion frames. Besides, an end-to-end involving distinguished temporal graph convolutional networks called IDT-GCN was introduced in [78], where an involving distinction graph convolutional model and temporal segment regression module could enhance the spatial and temporal modeling capacity, respectively. However, most learning-based methods require training datasets that are labeled by human supervisors, which are usually unavailable and inaccurate in many practical applications.

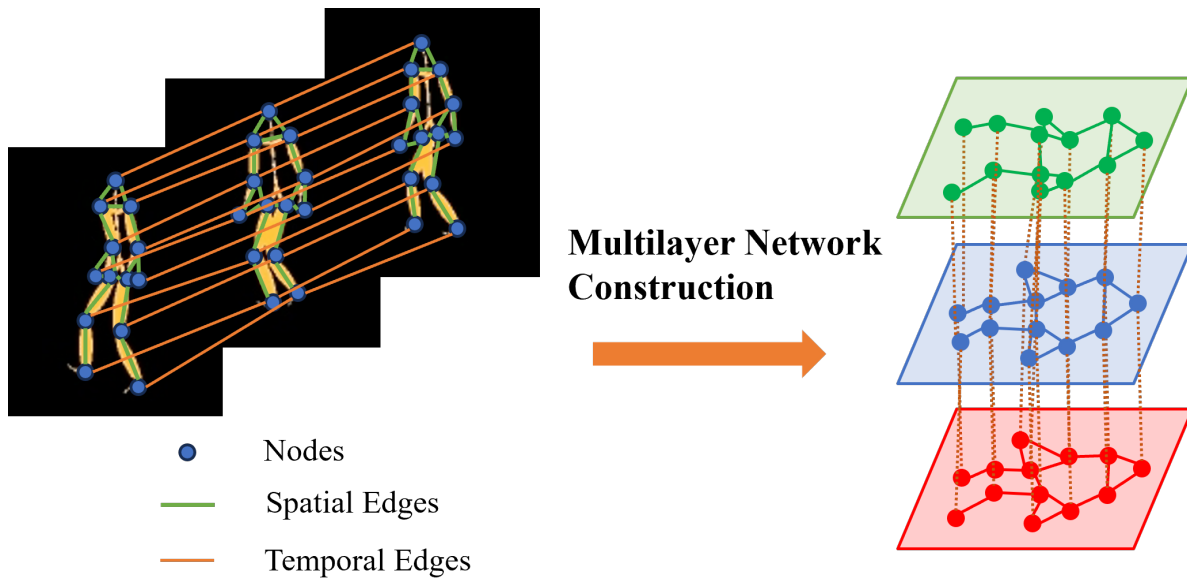


Figure 4.3: Example of MLG model for one motion sequence.

4.3 Problem Description and Modeling

We now introduce our problem description, together with the MLG models of the human motion sequence.

4.3.1 Problem Description

Similar to [72], we focus on the skeleton-based human motion segmentation for wearable sensors. Such skeleton-based motion dataset can be collected by body sensors or reconstructed from videos [91]. As shown in Fig. 4.2, a human body in motion within the skeleton dataset is abstracted into N joints, which can be annotated with additional information, including three-dimensional (3D) coordinates, accelerometer data and gyroscope data. Suppose that the human motion sequence contains M frames. Then, the annotated signals of the joint i in layer α is defined by a feature vector $\mathbf{x}_{\alpha i}$. Our goal in this work is to segment the N temporal frames into several successive sections, which could capture the features of human behavior and match the realistic human motion. Instead of training on prior-labeled data samples, we focus on the unsupervised human motion segmentation.

4.3.2 Multilayer Graph Construction

Next, we introduce the MLG construction for human motion sequence. For most skeleton-based motion data captured by sensors, the number of data points within one frame is constant for one motion sequence. Thus, we can apply an MLG with the same number of nodes on each layer to model one such motion sequence shown as Fig. 4.3. Suppose that the motion sequence contain M temporal frames and N joints in each frame. Intuitively, such multilayer spatial-temporal structure can be viewed as projecting N entities into M layers, where the entity is defined by the spatial joints and the layer is defined by temporal frames. Note that, if the motion data sequence contains unequal numbers of joints across layers, we can add some dummy nodes in the MLG to keep the same number of nodes across different layers. These dummy nodes are isolated to all other nodes, and would not change the topological structure of the original multilayer architecture.

With such definition, any skeleton-based motion sequence can be intuitively modeled by an MLG, which can be represented by the forth-order adjacency tensor

$$\mathbf{A}_{\mathcal{M}} = (A_{\mathcal{M},\alpha i \beta j}) \in \mathbb{R}^{M \times N \times M \times N}, \quad (4.1)$$

where $1 \leq \alpha, \beta \leq M, 1 \leq i, j \leq N$.

Here, $\alpha, \beta \in [1, M]$ are the indices of layer in MLG, while $i, j \in [1, N]$ are the indices of joints in each layer. Each entry in the adjacency tensor, i.e., $A_{\mathcal{M},\alpha i \beta j}$, represents the relationship between the i -th joint in α -th temporal layer and the j -th joint in β -th temporal layer. Now, we need to define the weights of $A_{\mathcal{M},\alpha i \beta j}$ to capture the geometric similarity among different joints. One common choice is to set the value of the element $A_{\mathcal{M},\alpha i \beta j}$ by Gaussian kernel [1], i.e.,

$$A_{\mathcal{M},\alpha i \beta j} = \exp\left(-\frac{\|\mathbf{x}_{\alpha i} - \mathbf{x}_{\beta j}\|^2}{\sigma^2}\right), \quad (4.2)$$

where $\mathbf{x}_{\alpha i}$ and $\mathbf{x}_{\beta j}$ represent data vectors, such as the coordinates, of the i -th joint in α -th temporal layer and the j -th node in β -th temporal layer, respectively. Also, the standard deviation σ controls

the support of the kernel function.

Considering the different natures in the interlayer (inter-temporal) and intralayer (spatial) connections, we apply different σ . i.e., $\sigma = \sigma_s$ when calculating the (spatial) similarity between two nodes within the same layer with $\alpha = \beta$, while using $\sigma = \sigma_t$ when calculating the (temporal) similarity between two nodes in different layers with $\alpha \neq \beta$. More specially, the value of σ_s and σ_t should be related to the statistics of all spatial and temporal distances. Thus, we apply the average of all distances as the value in this work, i.e.,

$$\sigma_s = \frac{1}{N_s} \sum_{i,j \in [1,N], \alpha=\beta} \|\mathbf{x}_{\alpha i} - \mathbf{x}_{\beta j}\|, \quad (4.3)$$

and

$$\sigma_t = \frac{1}{N_t} \sum_{\alpha \in [1, M-1], \beta=\alpha+1, i=j} \|\mathbf{x}_{\alpha i} - \mathbf{x}_{\beta j}\|, \quad (4.4)$$

where N_s is the total number of point pairs that are on the same frame, N_t is the total number of point pairs that are on the successive frames with the same index.

To highlight the interlayer correlations from the same joint, we utilize the multiplex structure in which each node only connects to its counterparts in its successive layers with $i = j$ and $|\alpha - \beta| = 1$. This structure further simplifies the inter-layer geometric models. Then, the final weight of each entry $A_{\mathcal{M}, \alpha i \beta j}$ shall be calculated as

$$A_{\mathcal{M}, \alpha i \beta j} = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_{\alpha i} - \mathbf{x}_{\beta j}\|^2}{\sigma_s^2}\right) & \text{if } \alpha = \beta \\ \exp\left(-\frac{\|\mathbf{x}_{\alpha i} - \mathbf{x}_{\beta j}\|^2}{\sigma_t^2}\right) & \text{if } i = j \text{ and } |\alpha - \beta| = 1 \\ 0 & \text{otherwise} \end{cases} \quad (4.5)$$

With the calculated adjacency tensor $\mathbf{A}_{\mathcal{M}}$, tensor decomposition can be applied via (2.14) or (2.15) to obtain the MLG Fourier space or singular space for data analysis. Since the HOSVD is faster and more robust in comparison with CP decomposition, we calculate the singular space of the undirected multilayer graph via HOSVD for spectral feature extraction. More details regarding

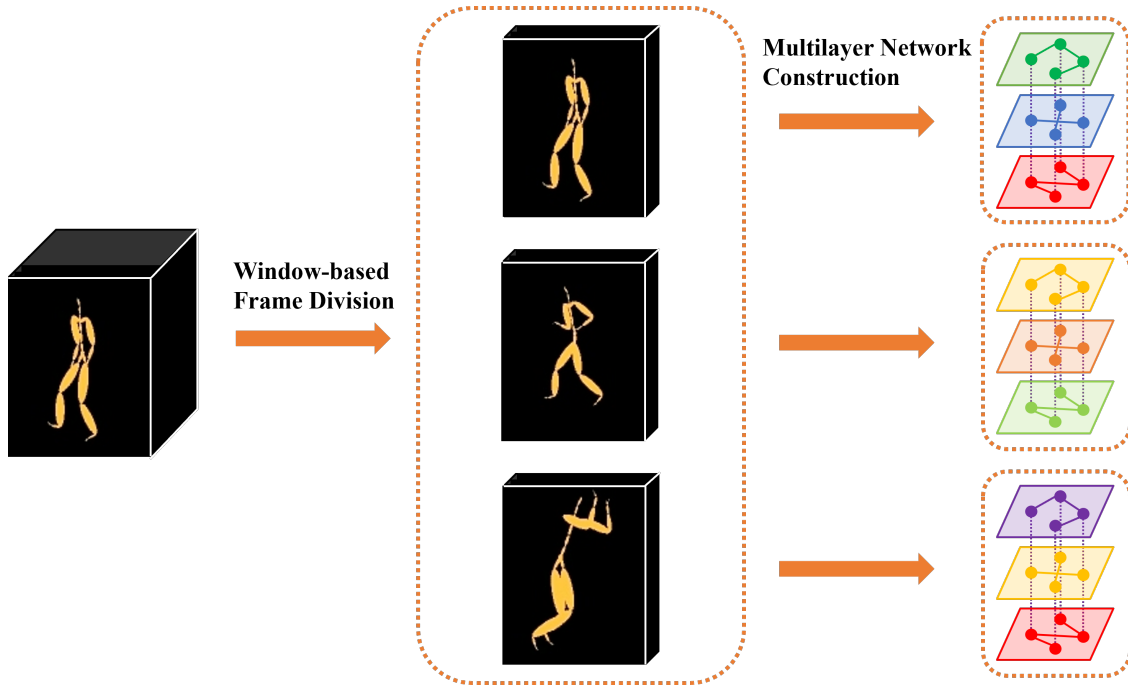


Figure 4.4: Example of window cuts in motion sequence and the multilayer network construction.

M-GSP singular analysis shall be discussed later in Section 4.4.1.

4.4 M-GSP Body Motion Segmentation

We now introduce our MLG-based body motion segmentation. Since many body motion sequences in most dataset have thousands of frames with dozens of data points in each frame, it is unpractical, in terms of memory and computational time, to build the MLG and decompose the MLG for the entire sequence. To solve this problem, we focus on a short-time processing method. As shown in Fig. 4.4, we first cut the entire motion sequence into N_{seg} shorter segments with window length W_d for temporal frames. Successive segments may overlap with one other to give a smooth representation of the current motion. For each segment, we build an MLG using the 3D coordinates or other annotated information by using the model introduced in Section 4.3.2. We then extract features from the MLG as representation of the corresponding motion. In this work, we have two different MLG segmentation approaches: 1) spectrum-based MLG motion segmentation based on spectral signals; and 2) vertex-based MLG motion segmentation based on structure signals.

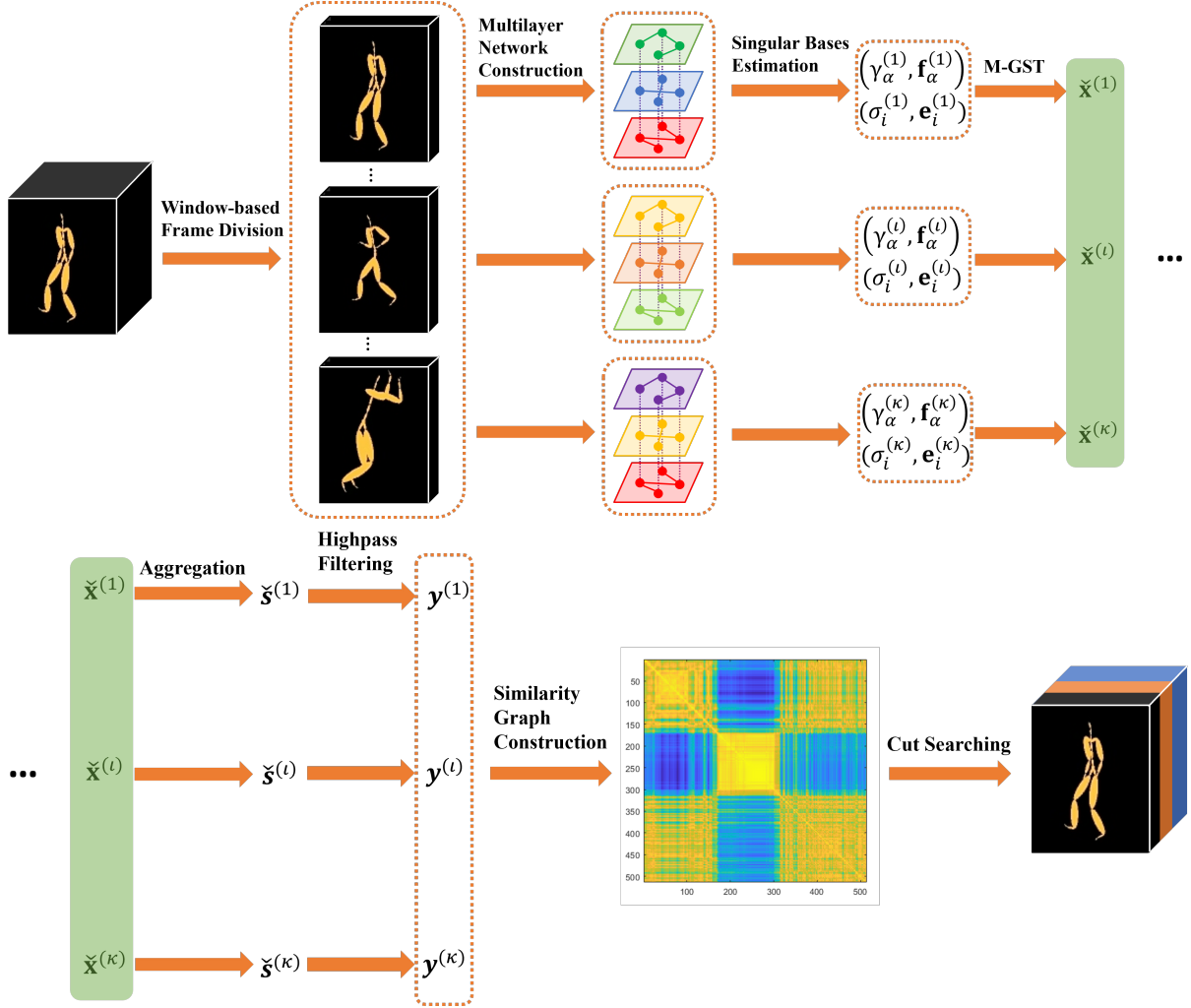


Figure 4.5: Block Diagram of Spectrum-based MLG Motion Segmentation. The diagram is broken into two parts for clearer view, where the overlapping elements in two parts are labeled in green.

4.4.1 Spectrum-based MLG Motion Segmentation

As shown in Fig. 4.1(a), the body motion sequence consists of some major movements, which can be captured by the low-frequency components, and detailed joint actions, which can be captured by the high-frequency components. To highlight the detailed behavior of body motions, such as leg movement and hand waving, we apply an M-GSP filter for feature extraction. To locate high-frequency components in M-GSP singular domain, we first estimate the singular space of each segment using HOSVD in (2.15). We then transform the signal into the singular space using these spectrum basis. Given the features of motion sequences, we denote the features of the joint i

in layer α as $\mathbf{x}_{\alpha i} \in \mathbb{R}^K$, where K is the feature dimension of each joint in a given temporal frame, including angles, coordinates and other available features. Suppose that $\mathbf{x}_{\alpha i}[a]$ is the a -th annotated feature of the joint i in layer α . The whole signals for the a -th feature is represented by

$$\mathbf{x}[a] = (\mathbf{x}_{\alpha i}[a]) \in \mathbb{R}^{M \times N} \quad (4.6)$$

According to (2.18), the joint M-GST of the a -th feature signal can be calculated by

$$\check{\mathbf{x}}[a] = \mathbf{W}_f^T \mathbf{x}[a] \mathbf{W}_e \in \mathbb{R}^{M \times N}. \quad (4.7)$$

Next, we aggregate all features into one signal $\check{\mathbf{s}} = (\check{s}_{\alpha i}) \in \mathbb{R}^{M \times N}$, where each entry is calculated as

$$\check{s}_{\alpha i} = \|\check{\mathbf{x}}_{\alpha i}[1], \dots, \check{\mathbf{x}}_{\alpha i}[K]\|_2^2, \quad (4.8)$$

where K is the dimension of features.

For the aggregated signals, an M-GSP highpass filter can be designed to extract the details in body motions. In this work, we combine two different kinds of highpass filters: ideal highpass filter and Haar-like highpass filter. Given the singular domain signal $\check{\mathbf{s}} \in \mathbb{R}^{M \times N}$, we first flatten it into a vector, and keep top k elements in the high frequency part, i.e., $\check{\mathbf{s}}' = [\check{s}_1, \dots, \check{s}_k, 0, \dots, 0] \in \mathbb{R}^{MN}$.

Thereafter, we use a Haar-like highpass filter V_{Haar} to process $\check{\mathbf{s}}'$. Suppose that $\lambda_{\mathcal{M}, \alpha i}$ denotes the spectrum coefficient corresponding to the basis of α -th layer and the i -th entity. The filter V_{Haar} is defined as

$$V_{Haar} = \mathbf{I} - \text{diag}(\boldsymbol{\lambda}_{\mathcal{M}}) \quad (4.9)$$

$$= \begin{bmatrix} 1 - \lambda_{\mathcal{M},11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 - \lambda_{\mathcal{M},MN} \end{bmatrix}. \quad (4.10)$$

The filtered signal is calculated as

$$\check{\mathbf{y}} = V_{Haar}\check{\mathbf{s}}' \in \mathbb{R}^{MN}. \quad (4.11)$$

Finally, we reshape $\check{\mathbf{y}}$ into $\check{\mathbf{s}}'' \in \mathbb{R}^{M \times N}$ and implement the inverse M-GST to get the vertex domain signal \mathbf{y} , which can be expressed as

$$\mathbf{y} = \mathbf{W}_f \check{\mathbf{s}}'' \mathbf{W}_e^T \in \mathbb{R}^{M \times N}. \quad (4.12)$$

Upon obtaining the vertex domain signal for all N_{seg} segments in one body motion sequence, we calculate a similarity graph $\mathbf{A}_{\mathcal{M},sim} \in \mathbb{R}^{N_{seg} \times N_{seg}}$, whose elements are given by

$$\mathbf{A}_{\mathcal{M},sim}[m, n] = e^{-\|\mathbf{y}^{(m)} - \mathbf{y}^{(n)}\|_2^2}, \quad (4.13)$$

where $\mathbf{y}^{(m)}$ and $\mathbf{y}^{(n)}$ are the vertex domain signal for m -th and n -th segment in the motion sequence, respectively. We then convert the similarity graph $\mathbf{A}_{\mathcal{M},sim}$ into a sparse self similarity matrix (SSSM) $\mathcal{M} \in \mathbb{R}^{N_{seg} \times N_{seg}}$ by finding the peak values in $\mathbf{A}_{\mathcal{M},sim}$ in each row. The threshold for finding the peaks is set to top 3% largest value among all elements in $\mathbf{A}_{\mathcal{M},sim}$. To keep the \mathcal{M} symmetric, once the element $\mathbf{A}_{\mathcal{M},sim}[m, n]$ is selected in each row as the peak values, the element $\mathbf{A}_{\mathcal{M},sim}[n, m]$ will also be set to the same value. All these peaks and its symmetrical elements are considered as the nearest neighbors.

To find the cutting frame based on the SSSM \mathcal{M} , we use a similar region growing search technique introduced in [84]. The search technique contains two steps: forward step and backward step. The forward step starts from the upper left corner of \mathcal{M} and attempts to extend the connected region to the next row, while the backward step starts from the lower right corner of \mathcal{M} and tries to extend the connected region to the previous row. In the forward step, a connected region starts as a seed denoted by $\mathcal{M}[1, 1]$. The region is extended to the next rows as long as the nearest neighbors in the updated region increases. Similarly, in the backward step, a connected region starts as a

Algorithm 4.1 Spectrum-based MLG Segmentation (SMLGS)

Input: Features of motion sequences with M temporal frames and N joints in each frame, where each joint in a given frame is annotated by K -dimensional features.

1. Calculate the spatial and temporal intrinsic resolutions, σ_s and σ_t , of motion sequences in (4.3) and (4.4);

2. Cut the motion sequences into overlapping segments of length W_d in frames;

for each segment do

3. Construct the adjacency tensor based on (4.5);

4. Estimate the singular bases using HOSVD in (2.15);

5. Transform the signals/features to the singular space using these spectrum bases;

6. Use a Haar-like highpass filter to extract features as (4.9) and (4.11);

end for

7. Calculate the similarity graph using the extracted features as (4.13);

8. Find the cutting frame based on the similarity graph using the region growing search technique in [84].

seed i.e., $\mathcal{M}[N_{seg}, N_{seg}]$, and the region is extended to the previous rows as long as the nearest neighbors in the updated region increases. If no new neighbors are found between segment i and $i + \omega$ in the larger region, except for the neighbors from the main diagonal of \mathcal{M} , then the current region search is considered complete. The parameter ω is set to 8 in our experiments. The major steps of spectrum-based MLG Motion Segmentation (SMLGS) is presented in Algorithm 4.1.

4.4.2 Vertex-based MLG Segmentation

To reduce complexity, another set of features to consider is the multilayer graph structure signal.

To extract structural features in the vertex domain, we first reshape the adjacency tensor $\mathbf{A}_{\mathcal{M}} \in \mathbb{R}^{M \times N \times M \times N}$ into a feature vector after constructing the MLG for each overlapping shorter segment with length W_d . We denote the feature vector for the m -th segment as $\mathbf{z}^{(m)} \in \mathbb{R}^{M^2 N^2}$. Once we get $\mathbf{z}^{(m)}$ for all N_{seg} segments, we concatenate all $\mathbf{z}^{(m)}$ into one matrix

$$\mathbf{z} = [\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots, \mathbf{z}^{(N_{seg})}] \in \mathbb{R}^{M^2 N^2 \times N_{seg}}. \quad (4.14)$$

We subsequently reduce the size of \mathbf{z} by keeping top k elements with highest variance across all segments as follows:

Algorithm 4.2 Vertex-based MLG Segmentation (VMLGS)

Input: Features of motion sequences with M temporal frames and N joints in each frame, where each joint in a given frame is annotated by K -dimensional features.

1. Calculate the spatial and temporal intrinsic resolutions, σ_s and σ_t , of motion sequences in (4.3) and (4.4);

2. Cut the motion sequences into overlapping windows of length W_d in frames;

for each segment do

3. Construct the adjacency tensor based on (4.5) and reshape it into a vector $\mathbf{z}^{(m)}$;

end for

4. Concatenate all $\mathbf{z}^{(m)}$ into feature matrix \mathbf{z} and calculate the variance for each row of \mathbf{z} ;

5. Select the rows in \mathbf{z} with k highest variance to form the sampled feature matrix \mathbf{z}' ;

6. Calculate the similarity graph using the extracted features as (4.15);

7. Find the cutting frame based on the similarity graph using the region growing search technique in [84].

- We first calculate the variance for each row in \mathbf{z} , and concatenate them into $\sigma^2(\mathbf{z}) \in \mathbb{R}^{M^2 N^2}$
- Then we find indices of the elements in $\sigma^2(\mathbf{z})$ with k largest variances and denote the indices by $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_k\} \in \mathbb{R}^k$;
- Finally, we construct the sampled feature matrix $\mathbf{z}' \in \mathbb{R}^{k \times N_{seg}}$ by keeping the rows in \mathbf{z} with same indices in \mathcal{I} , i.e., the p th row in \mathbf{z}' is the \mathcal{I}_p -th row of \mathbf{z} .

In this way we can remove the low frequency elements in the feature matrix.

With the extracted high-frequency structure signals, we use the column vector of \mathbf{z}' to calculate the similarity graph $\mathbf{A}_{\mathcal{M},sim}$. Let $\mathbf{a}^{(m)}$ be the m -th column vector of \mathbf{a}' . The similarity graph $\mathbf{A}_{\mathcal{M},sim}$ is calculated by

$$\mathbf{A}_{\mathcal{M},sim}[m, n] = e^{-\|\mathbf{z}'^{(m)} - \mathbf{z}'^{(n)}\|_2^2}. \quad (4.15)$$

We then convert the similarity graph $\mathbf{A}_{\mathcal{M},sim}$ into an SSSM as SMLGS, and find the cutting frames using the region growing search technique introduced in Section 4.4.1. The major steps of vertex-based MLG Motion Segmentation (VMLGS) is presented in Algorithm 4.2.

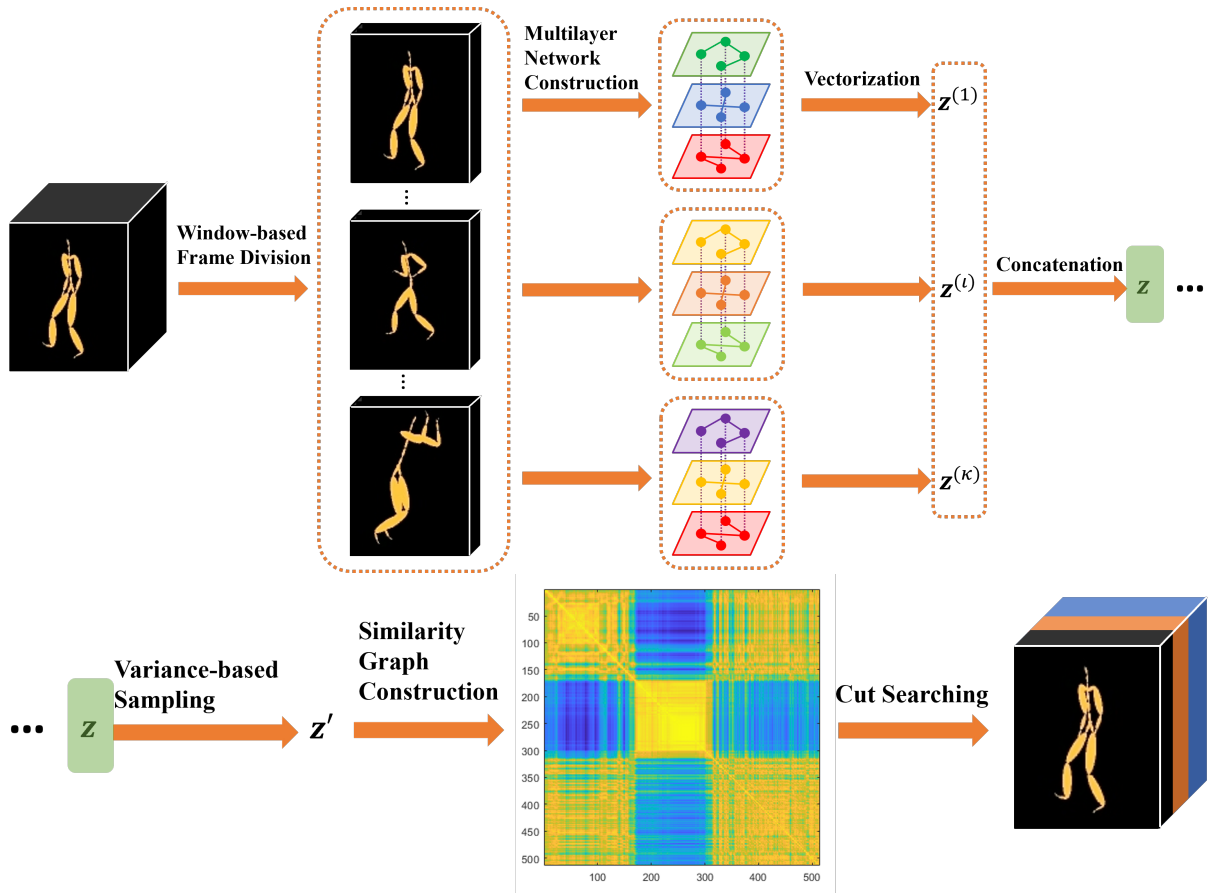


Figure 4.6: Block Diagram of Vertex-based MLG Motion Segmentation. The diagram is broken into two parts for clearer view, where the overlapping elements in two parts are labeled in green.

4.5 Experiments

We now present the experimental results of the proposed algorithms in both unsupervised and supervised setup compared to the existing clustering and recognition approaches.

4.5.1 Dataset

In our experiment, we test over two different datasets: 1) the CMU Graphics Lab Motion Capture Database; and 2) the Human Gait Database.

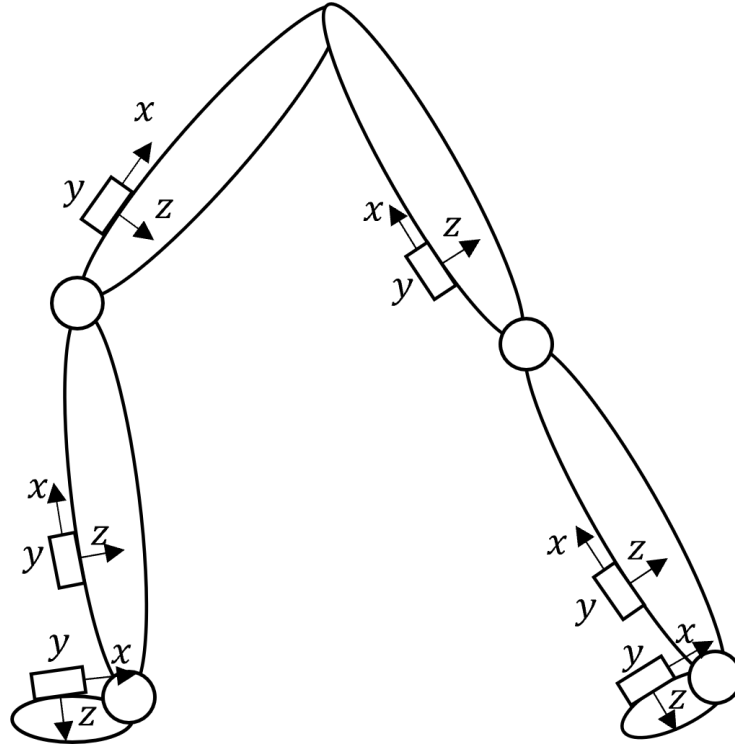


Figure 4.7: Example of the location of inertial measurement units in HuGaDB.

CMU Graphics Lab Motion Capture Database

The CMU graphics lab motion capture database¹ have 2605 trials in 6 categories and 23 subcategories. They are captured at 120 Hz with images of 4 megapixel resolution. An example motion trail in the database is shown in Fig. 4.2. We test all motion segmentation methods on trails 01 to 14 of subject 86, which have the human (supervisor)-labeled motion segmentation. In this work, we use the optimal cutting frame as the ground truth. We do not account for transitions between distinct actions in a manner similar to that of [63].

Human Gait Database (HuGaDB)

HuGaDB is an action segmentation dataset, where the subjects record typical lower limb activities, e.g. walking, running, and cycling [58]. 18 subjects are included in this dataset. MoCap was performed with 6 inertial measurement units (IMUs) at a sampling frequency of 60 Hz in HuGaDB.

¹<http://mocap.cs.cmu.edu/>

Table 4.1: Accuracy of motion segmentation on CMU 86 dataset

	SMLGS	VMLGS	SC	ACA	EUTS
86_01	0.9067	0.9558	0.7372	0.9212	0.9505
86_02	0.9387	0.9396	0.8916	0.8891	0.9469
86_03	0.9277	0.9488	0.8406	0.8953	0.9262
86_04	0.9138	0.9294	0.7539	0.8735	0.9267
86_05	0.8928	0.9228	0.6523	0.8963	0.9252
86_06	0.9058	0.9631	0.7325	0.9237	0.9068
86_07	0.9377	0.9537	0.8920	0.9214	0.9449
86_08	0.9527	0.9317	0.7808	0.9384	0.9682
86_09	0.9471	0.9235	0.8334	0.8761	0.9058
86_10	0.9709	0.9666	0.9626	0.8928	0.9238
86_11	0.9380	0.9603	0.9228	0.9147	0.9672
86_12	0.9150	0.9712	0.8730	0.8498	0.9275
86_13	0.7671	0.8612	0.8075	0.8206	0.6073
86_14	0.9136	0.9131	0.6049	0.7308	0.9216
Average	<u>0.9163</u>	0.9386	0.8061	0.8817	0.9106

Each IMU contains one accelerometer and one gyroscope. The IMUs were placed on the right and left thighs, shins and feet, as shown in Fig. 4.7. This dataset contains 364 IMU trials in 12 action categories. Since the accelerometer data and gyroscope data are different captures of the same motion, they should be processed in different ways. In our test, we further divide each trial into three datasets: accelerometer only (ACC) dataset, gyroscope only (GYRO) dataset, and one dataset containing both accelerometer and gyroscope data. We test all motion segmentation methods on all these three datasets to further investigate the robustness of motion segmentation methods on different kinds of data.

4.5.2 Unsupervised Motion Segmentation

We first evaluate the performance of proposed methods in unsupervised setup. Here, we compare our proposed method with four unsupervised motion methods: spectral clustering (SC), Aligned Cluster Analysis (ACA), Hierarchical Aligned Cluster Analysis (HACA) [63], and Efficient Unsupervised Temporal Segmentation (EUTS) [84] on CMU and HuGaDB datasets. For the CMU dataset, we convert data into 3D coordinates of all joints as input for our proposed algorithm,

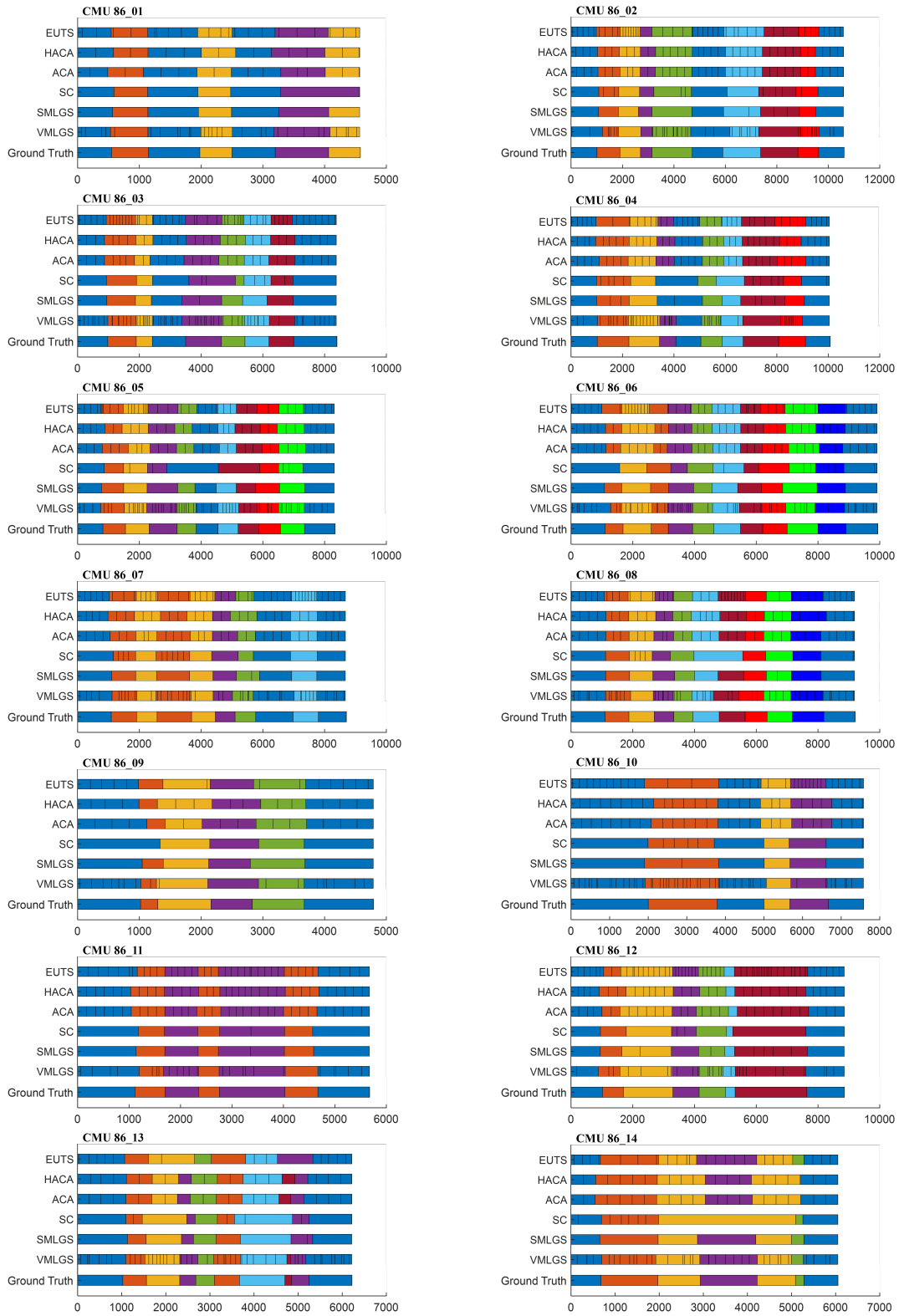


Figure 4.8: Segmentation result on CMU trial 86.

i.e., $\mathbf{x}_{\alpha i} \in \mathbb{R}^3$. For the HuGaDB dataset, we directly use the accelerometer data $\mathbf{a}_{\alpha i} \in \mathbb{R}^3$ and gyroscope data $\mathbf{g}_{\alpha i} \in \mathbb{R}^3$ as the input of our proposed algorithm, where $\mathbf{x}_{\alpha i} \in \mathbb{R}^6$. The parameters of the proposed method are fixed for all trials in one dataset. We vary the window size W_d from 2 to 10 in our experiment. We chose the parameters of all the existing methods based on the set of parameters provided in the original paper and codes with the best performance. To evaluate the clustering accuracy, we calculate the mean intersection over union (mIoU) on each trial.

The average accuracy and segmentation result on CMU datasets are shown in Table. 4.1 and Fig. 4.8, respectively. The average accuracy results on HuGaDB dataset are shown in Table. 4.2, and an example of segmentation on the first dataset in HuGaDB is shown in Fig. 4.10. The best performance is marked in bold font and the second best result is underlined. As shown in Fig. 4.8, our proposed spectrum-based (SMLGS) algorithm tends to segment each motion sequence into larger sections, whereas the vertex-based (VMLGS) algorithm cuts the motion sequence into finer sections. This distinction highlights the focus of SMLGS algorithm in capturing significant differences between distinct motions, whereas the VMLGS algorithm excels at detecting subtle variations among elements, even within the same motion. The average accuracy results presented in Table 4.1 indicate that VMLGS algorithm achieves superior overall accuracy. However, the increased number of segments generated by VMLGS may pose a greater challenge to the classifier in real-world applications. On the other hand, the number of segments generated by SMLGS closely aligns with the ground truth in the majority of CMU 86 datasets. From these test results, vertex-based MLG method (VMLGS) provides the best performance in most of the dataset while spectrum-based method (SMLGS) ranks the second best. The results demonstrate the strength of M-GSP in body motion analysis and the robustness of the proposed algorithms. From the visualization results, our proposed algorithms provide a clearer segment boundary and fewer segments, which is closer to the ground truth. This further demonstrates the benefits of applying M-GSP in body motion analysis.

Table 4.2: Average accuracy of motion segmentation on HuGaDB dataset

	acc	gyro	both
SC	0.6162	0.4738	0.4540
ACA	0.7167	0.5436	0.5341
HACA	0.7210	0.5276	0.5314
SMLGS	0.8171	<u>0.7000</u>	<u>0.8249</u>
VMLGS	<u>0.7667</u>	0.8495	0.8551

Table 4.3: Average accuracy on HuGaDB dataset with both accelerometer and gyroscope data.

Bi-LSTM	86.1
TCN	88.3
ST-GCN	88.7
MS-TCN	86.8
MS-GCN	90.4
SMLGS (w. optimal parameters)	87.9
VMLGS (w. optimal parameters)	90.2

4.5.3 Supervised Motion Recognition

Although our algorithms are designed under unsupervised setup, we also test and provide comparison with supervised motion segmentation. To ensure a fair comparison, we tune the parameters of our algorithms based on the best performance from the training dataset. Here, we compare our proposed method with several existing supervised motion methods: (1) bidirectional long short term memory-based network (Bi-LSTM)[66], (2) temporal convolutional neural networks (TCN)[93], (3) spatial-temporal graph convolutional neural network (ST-GCN) [72], (4) multi-stage temporal convolutional neural networks (MS-TCN) [94], and (5) multi-stage spatial-temporal graph convolutional neural networks (MS-GCN) [73] on HuGaDB datasets.

We present the results in Table 4.3. From the results, our proposed algorithms have competitive performance against supervised learning machines, even without ever utilizing the label information for clustering. This demonstrate the effectiveness of our proposed method in terms of feature extraction from body motion sequences. Note that our clustering algorithms can be easily integrated with the supervised learning machines to further improve the performance. We shall investigate applications of M-GSP in deep learning in our future works.

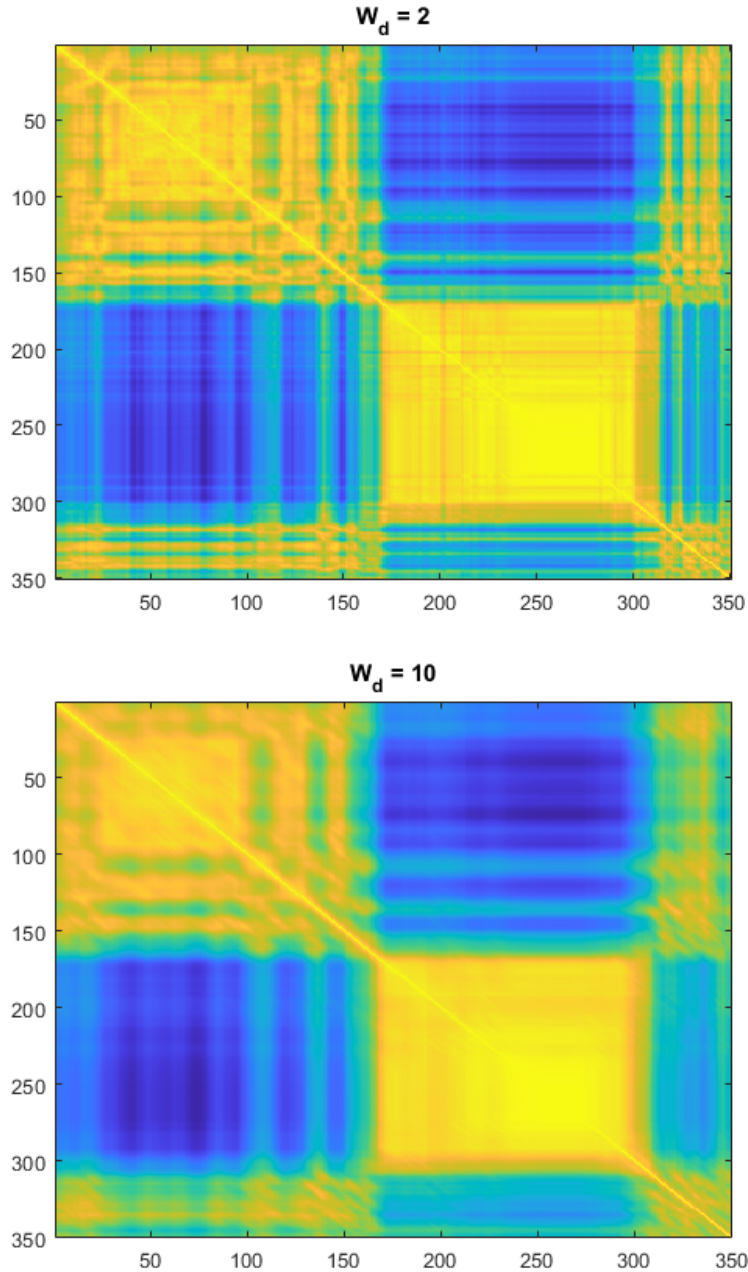


Figure 4.9: Example of the similarity matrix for different window sizes on HuGaDB dataset with GYRO data.

4.5.4 Ablation study on window size

In this section we test our proposed method under different window sizes to show its characteristics as a short-time processing method. We test the average accuracy on CMU dataset, ACC dataset in HuGaDB and GYRO dataset in HuGaDB. We only change the window size W_d of each segment

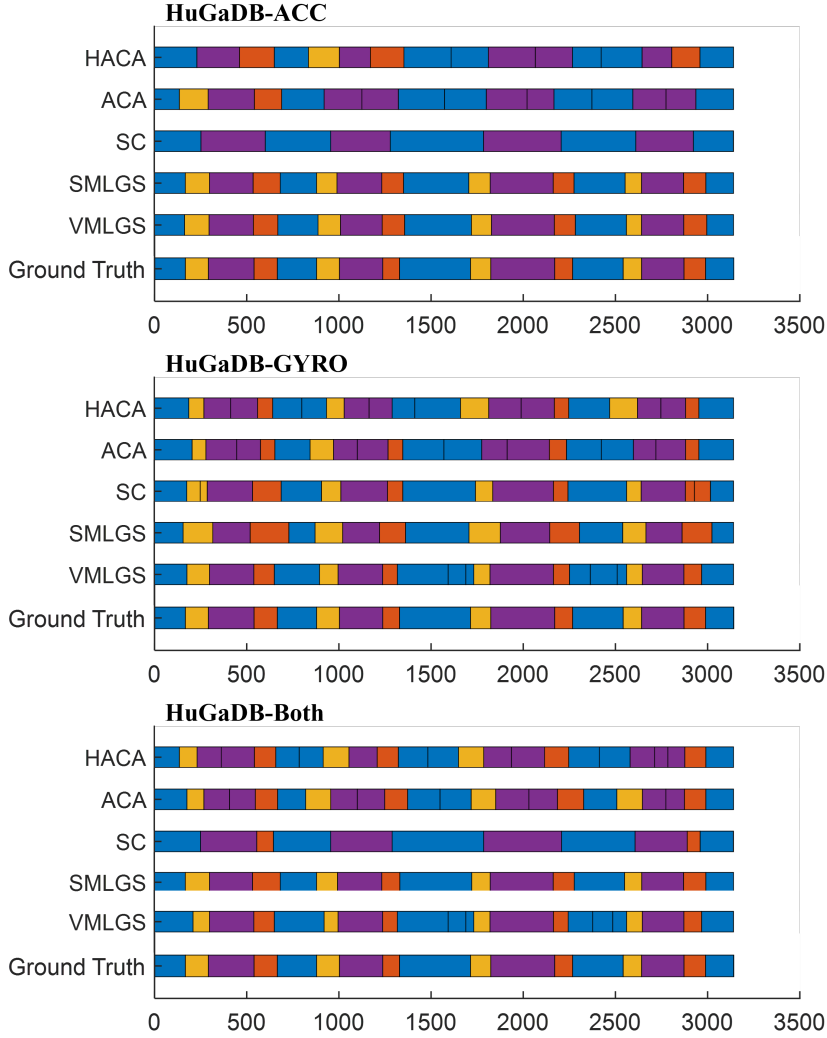


Figure 4.10: Segmentation result on HuGaDB trial 01 data 00.

where we extract the features while keeping all other parameters fixed. From Fig. 4.11, our result indicates that the optimal window size would vary for different datasets. It also shows that a larger window size will lower the average accuracy performance once it exceeds the optimal value. As shown in Fig. 4.9, the similarity matrix of window size $W_d = 10$ is smoother than the similarity matrix of $W_d = 2$, which makes it more difficult to accurately locate the cut frame between two body motions. In practice, the optimal window size may be estimated from a small subset within the entire dataset.

Note also that SMLGS performs better in the ACC dataset in HuGaDB while the VMLGS is more robust across different datasets. The reason is that SMLGS implements the spectrum

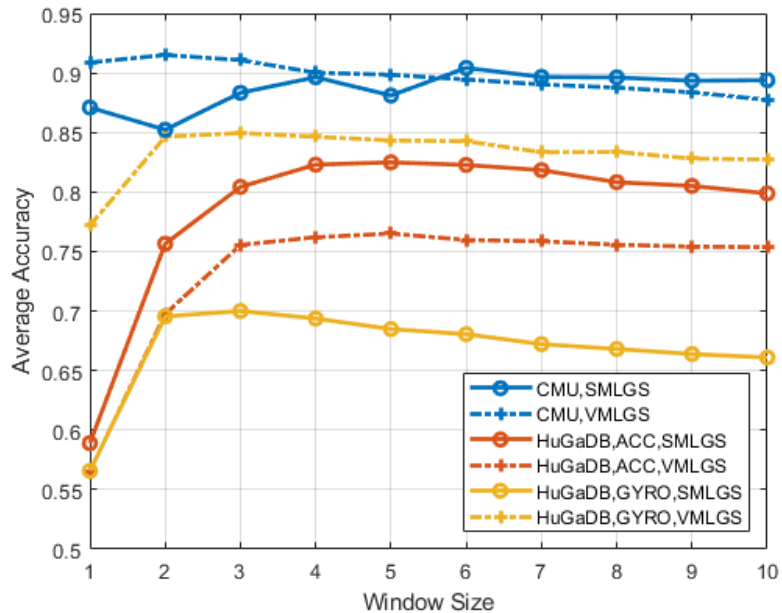
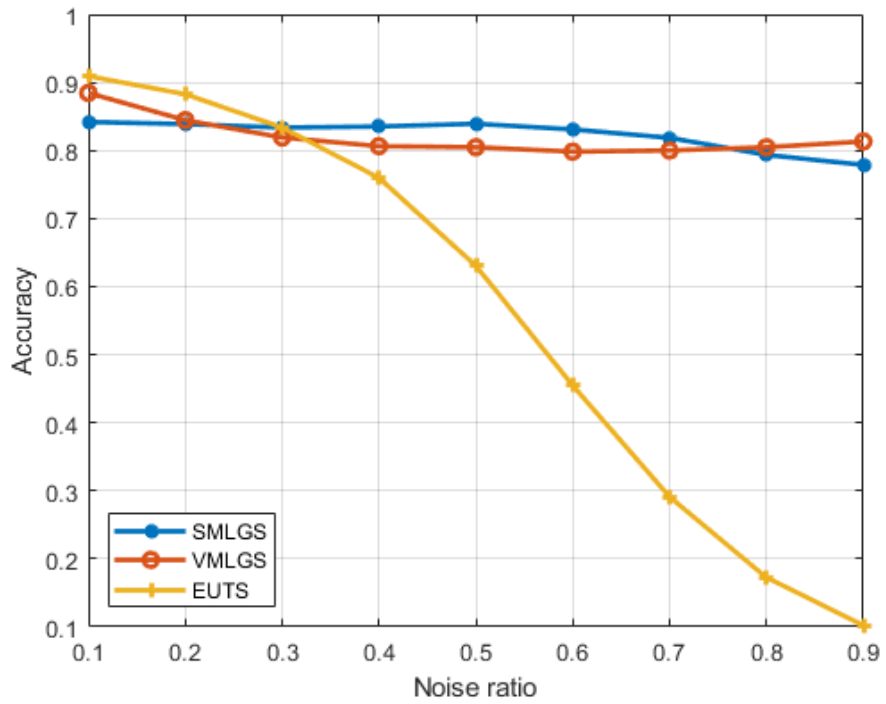


Figure 4.11: Average accuracy for different window sizes.

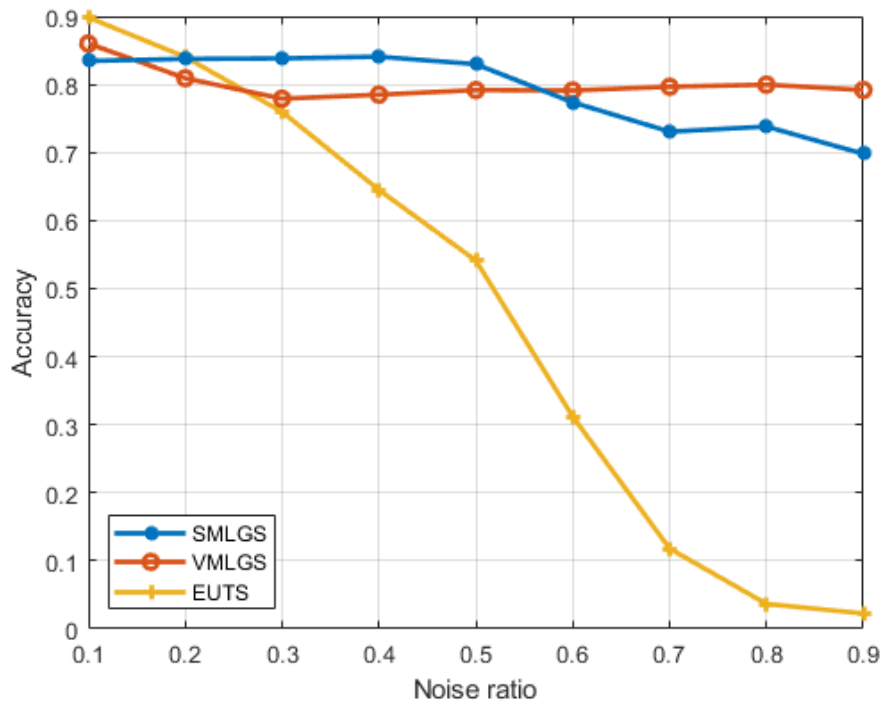
decomposition to extract additional features while introducing uncertainty during HOSVD. Thus, for simpler datasets, such as CMU data with only coordinate information, it could lead to superior performance with extracted spectral features. On the other hand, VMLGS evaluates the MLG via structural features, which is more stable and robust in complicated dataset, such as HuGaDB.

4.5.5 Robustness

In this part of test, we compare our proposed method with Efficient Unsupervised Temporal Segmentation (EUTS) [84] on CMU dataset with Gaussian noise. We add the Gaussian noise directly to the original dataset with varying standard deviation σ between 0.1 and 0.2. To test robustness against different level of noise, we add the Gaussian noise to 10% to 90% of the frames in the sequence. These noisy frames are randomly selected with equal probability and we repeat the process 10 times for each trial for each noisy frame ratio. In order to minimize the effect of randomness, we average the accuracy over these 10 noisy samples to arrive at the final result. As shown in Fig. 4.12, our proposed method exhibits stronger robustness against additive Gaussian noise.



(a)



(b)

Figure 4.12: Average accuracy on CMU trial 86 with different noisy frame ratio. (a) $\sigma = 0.1$; (b) $\sigma = 0.2$.

4.6 Conclusions

This chapter studies the use of M-GSP for body motion analysis. More specially, we have introduced the MLG models for body motion sequence, with which we propose two different M-GSP filter-based algorithms for motion segmentation. Our proposed methods are easier to implement, and show robustness across multiple datasets. Our experimental results have demonstrated the efficacy of the proposed methods and the potentials of M-GSP in motion analysis. This chapter have established M-GSP as an efficient tool to model multilateral relationship and to extract features in motion sequence applications. In our future works, we shall investigate new ways to integrate machine learning methods and M-GSP for better feature extraction. The interpretation of body motion from the perspective of graph Fourier space is another interesting direction for exploration.

Chapter 5

Efficient Eigenvalue Decomposition for Low-Rank Symmetric Matrices in Signal Processing

5.1 Introduction

Graph based signal processing methods, including graph signal processing (GSP) and graph neural networks (GNN), have garnered significant attention in recent years. This surge in interest is driven by the wide-ranging application in diverse domains, such as the Internet, social networks, financial data, sensor networks, traffic patterns, and biological systems [1, 4, 5]. All the network-structured data in these domains can be naturally modeled by graphs, which highlights the significance of employing graph-based methods for signal analysis and processing. In this field, the exploration of graph spectral domain, such as graph spectral analysis and filtering [24, 25], has become a key component of research. Additionally, analyzing the graph spectra facilitates feature extraction and signal processing, making it a fundamental tool of GSP. As a result, the eigenvalue decomposition (EVD) algorithms have become prevalent within the domain of GSP as the key to calculate the graph spectral domain from representation matrix of the data structure.

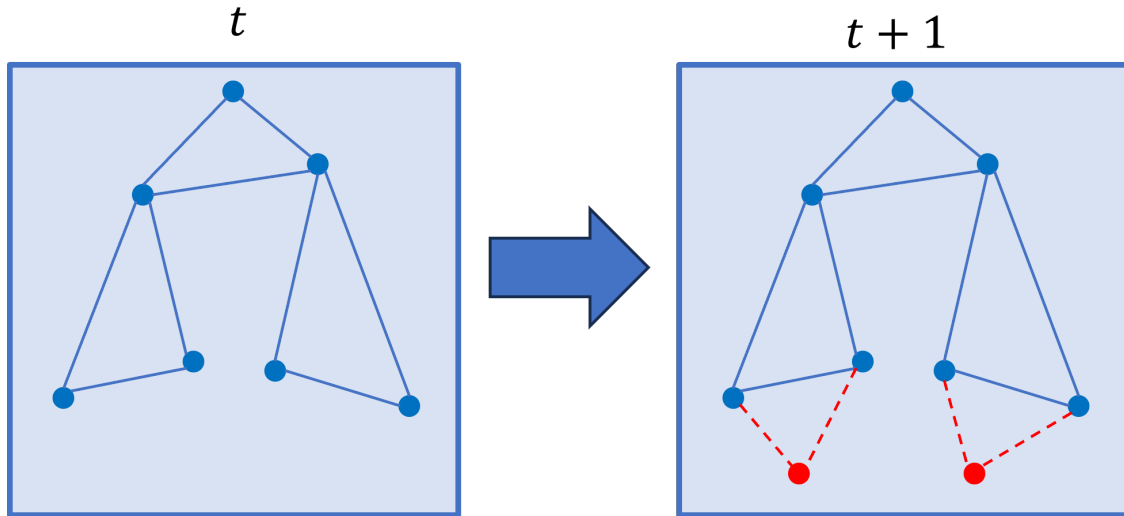


Figure 5.1: Example of the dynamic graph at time t and time $t + 1$. The additional vertices and edges at time $t + 1$ are labeled in red and red dash lines, respectively.

The pursuit of efficient EVD algorithm with low computational complexity has attracted intensive investigation from the community. The challenges of computational complexity and the peak memory utilization become problems for EVD algorithms when dealing with large-sized matrices. Despite the achievements of many existing works on efficiently approximating eigen-pairs of all scales of matrices [95], most algorithms are focused on static graphs. However, in many real-world applications, such as dynamic point clouds and social networks, the graph size keeps changing over time. One example of such dynamic graph is shown in fig. 5.1, where there are additional vertices at time $t + 1$, which will increase the dimension of the corresponding representation matrix, so that the eigen-pairs needs to be recomputed or approximated based on the result at time t . In these dynamic scenarios, the straightforward approach of recomputing eigen-pairs upon each alteration in the graph presents a significant computational burden. This demand for continual recomputation highlights the need to develop EVD algorithms that can effectively handle the evolving nature of dynamic graphs. In this work, we focus on the updating problem upon the largest k eigenvalues and the corresponding eigenvectors of symmetric adjacency matrices of undirected graphs as the graph size increases. The estimation results of the top k eigen-pairs are commonly used in applications such as spectral clustering and graph filtering.

Despite the study of eigen-updating algorithms has started many years ago, many existing works are focused on rank-one or rank- k EVD update problems [96–98], which can not be directly applied in real-world graph based applications. In these scenarios, there is no guarantee that each change of the graph has to be limited to rank-one or rank- k . Only a few studies are focused on such issues. The authors in [99, 100] proposed an eigen-pair updating algorithm based on the perturbation of matrices in a generalized eigenvalue system. This method needs to calculate the matrix inverse whenever updating the eigenvectors, which is computational costly. An efficient incremental eigen-approximation algorithm was proposed in [101]. However, compared with the general decomposition expression of the updated matrix given by [102], the proposed algorithm ignore the residual elements to further simplify the decomposition expression, which enlarges the approximation error of the result. This assumption also implies that the top K eigen-pairs of the original matrix will maintain their relative order to be in top K in the updated matrix, which may not hold in general cases. Another kinds of fast eigen-function tracking algorithms were proposed in [103], where the changes of the adjacency matrices are viewed as perturbations. Based on the matrix perturbation theory in [104], the authors proposed first-order and higher-order eigen-pair tracking algorithms, named TRIP-BASIC and TRIP, respectively. However, the approximation errors of the TRIP-BASIC algorithm is high, while the computational complexity of the TRIP is large.

To solve the eigen-updating problem efficiently, this chapter proposes an incremental EVD algorithm for low rank symmetric matrices (IEVD-LR). More specifically, our approach updates the top k eigen-pairs by iteratively increasing the matrix size one-by-one, instead of updating the increase of multiple dimensions at once. This strategy provides us two advantages with acceptable increase on the computational complexity of the algorithm. First, we have more control over the approximation errors on each iteration. Compared with the proposed algorithm in [101], we design a novel error correction branch whenever the approximation error exceed the tolerance in each iteration. Second, the intermediate variables of our proposed algorithm have smaller size, which reduces the peak memory usage. Consequently, our method exhibits smaller approximation errors

and less peak memory usage while maintaining the same order of computational complexity. Our experimental results validate its accuracy and efficiency on both synthetic and real-world datasets in applications like spectral clustering and graph filtering. We summarize our contributions as follows:

- To reduce the final approximation errors, we design a novel error correction branch whenever the approximation error exceed the tolerance in each iteration. To the best of our knowledge, we are the first to introduce such error correction algorithm in EVD approximation.
- To gain more flexibility of error control and reduce the memory cost, we adapt an iterative eigen-updating strategy for our IEVD-LR algorithm. In each iteration we increase the size of the matrix by one and update the tracking eigen-pairs.
- To demonstrate the accuracy and efficiency of our proposed IEVD-LR algorithm, we analyze its error performance, computational complexity and memory cost.
- Beyond the theoretical analysis, our experimental results validate the accuracy and efficiency of the proposed algorithm on both synthetic and real-world datasets in applications like spectral clustering and graph filtering.

We organize the rest of the chapter as follows. In Section 5.2, we first review the related works. Then, we present the proposed IEVD-LR algorithm in Section 5.3. Next, we present the experimental results of the proposed method on both synthetic and real-world datasets in applications such as spectral clustering and graph filtering in Section 5.4, before summarizing our work in Section 5.5.

5.2 Related Works

In this section we give an overview of the related works.

5.2.1 Fast Matrix Decomposition

Many existing works on EVD updating are focused on rank-one or rank-k EVD update problems [96–98], which can not be directly applied in real-world graph based applications. In these scenarios, there is no guarantee that each change of the graph has to be limited to rank-one or rank-k. Only a few studies are focused on such issues. The authors in [99, 100] considered a generalized eigenvalue system such that $\mathbf{B}\mathbf{x} = \lambda\mathbf{C}\mathbf{x}$, where both $\mathbf{B} \in \mathbb{R}^{N \times N}$ and $\mathbf{C} \in \mathbb{R}^{N \times N}$ are symmetric, and λ and \mathbf{x} are the eigenvalue and corresponding eigenvector that needs to be determined. They proposed the eigen-pair updating algorithm based on finding the perturbation of eigenvalue and eigenvectors according to the perturbation of matrices \mathbf{B} and \mathbf{C} . However, this method needs to calculate the matrix inverse whenever updating the eigenvectors, which is computational costly. An efficient incremental eigen-approximation algorithm was proposed in [101]. This algorithm assumes that the updated matrix $\mathbf{E} \in \mathbb{R}^{M \times M}$ can be written in the form that

$$\begin{aligned} \mathbf{E} &= [\mathbf{B} \quad \mathbf{C}]^\top [\mathbf{B} \quad \mathbf{C}] \\ &= \begin{bmatrix} \mathbf{B}^\top \mathbf{B} & \mathbf{B}^\top \mathbf{C} \\ \mathbf{C}^\top \mathbf{B} & \mathbf{C}^\top \mathbf{C} \end{bmatrix}, \end{aligned} \quad (5.1)$$

where $\mathbf{D} = \mathbf{B}^\top \mathbf{B} \in \mathbb{R}^{N \times N}$ is the original matrix before the size change, $\mathbf{B} \in \mathbb{R}^{K \times N}$ is the decomposition matrix of \mathbf{D} , $\mathbf{C} \in \mathbb{R}^{K \times M}$ is the difference between the decomposition matrix of \mathbf{E} and \mathbf{D} , and K is the highest rank of \mathbf{D} and \mathbf{E} . However, this decomposition expression is incomplete. The general expression of decomposition expression of the updated matrix, given by [102], should be

$$\begin{aligned} \mathbf{E} &= \begin{bmatrix} \mathbf{B} & \mathbf{C}_1 \\ 0 & \mathbf{C}_2 \end{bmatrix}^\top \begin{bmatrix} \mathbf{B} & \mathbf{C}_1 \\ 0 & \mathbf{C}_2 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{B}^\top \mathbf{B} & \mathbf{B}^\top \mathbf{C}_1 \\ \mathbf{C}_1^\top \mathbf{B} & \mathbf{C}_1^\top \mathbf{C}_1 + \mathbf{C}_2^\top \mathbf{C}_2 \end{bmatrix}. \end{aligned} \quad (5.2)$$

Comparing (5.1) and (5.2), the former decomposition expression drops the residual element of $\mathbf{C}_2^\top \mathbf{C}_2$, which enlarges the approximation error. Additionally, the former decomposition expression overlooks the existence of additional rows in the decomposition matrix. These rows could correspond to the new top K eigen-pairs of the updated matrix \mathbf{E} , and be kept in the final result. Therefore, the approximation errors of the proposed algorithm is enlarged.

Another kinds of fast eigen-function tracking algorithms were proposed in [103], where the changes of the adjacency matrices are viewed as the perturbation. Based on the matrix perturbation theory in [104], the authors proposed first order and higher order eigen-pairs tracking algorithms, named TRIP-BASIC and TRIP, respectively. However, the approximation errors of the TRIP-BASIC algorithm is in the order of the norm of the perturbation of adjacency matrix. The approximation error which will be high when the norm of perturbation is large and there is no control on that. On the other hand, the computational complexity of the TRIP is $O(k^4)$ for each iteration, where k is the number of eigenvalues that are tracking. Such complexity is too high for a large k .

5.3 Method and Analysis

In this section we will introduce our incremental EVD algorithm for low rank symmetric matrices (IEVD-LR), we assume the input matrix of the algorithm is low rank in this section for better understanding while keeping the mathematical integrity of the expression.

5.3.1 Basic Incremental eigenvalue decomposition

Here we describe one iteration of our IEVD-LR algorithm, which only increases the matrix size by one and updates the eigen-pairs. In practice applications where we need to increase the matrix size by ℓ from time t to time $t + 1$, we will run the following iteration by ℓ times, as shown in Fig. 5.2. In each iteration, one new pair of α_i and d_i are used as the inputs of the algorithm to update the EVD result. In this way we will have better control on the approximation errors in each iteration

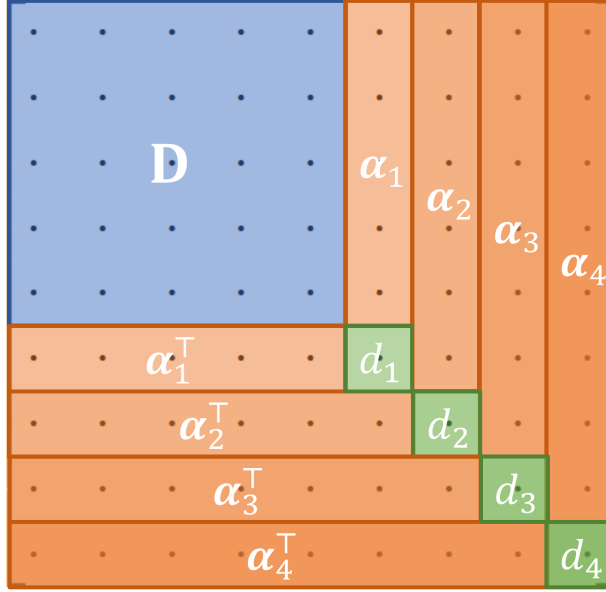


Figure 5.2: Example of iterative update of our IEVD-LR algorithm. \mathbf{D} is the original low rank symmetric matrix whose EVD matrices are known. In each iteration of the IEVD-LR algorithm, one new pair of α_i and d_i are used as the inputs of the algorithm to update the EVD result.

and lower peak memory usage.

Assuming we already have the truncated eigenvalue decomposition of rank k for the matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ as follows:

$$\mathbf{D} = \mathbf{Q}\Sigma\mathbf{Q}^\top, \quad (5.3)$$

where $\Sigma \in \mathbb{R}^{k \times k}$ is a diagonal matrix with the k ordered eigen values of \mathbf{D} on the diagonal, i.e., $\Sigma = \text{diag}(\lambda_1, \dots, \lambda_k)$, $\mathbf{Q} \in \mathbb{R}^{n \times k}$ is the matrix of the corresponding k eigenvectors of $\mathbf{D} \in \mathbb{R}^{n \times n}$. with $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}_k$,

Our objective is to perform an eigenvalue decomposition update only employing $\mathbf{Q} \in \mathbb{R}^{n \times k}$, $\alpha \in \mathbb{R}^n$, and $d \in \mathbb{R}$ for the following matrix \mathbf{E}

$$\mathbf{E} = \begin{bmatrix} \mathbf{D} & \alpha \\ \alpha^\top & d \end{bmatrix}. \quad (5.4)$$

We compute the residual vector \mathbf{e} of $\boldsymbol{\alpha} \in \mathbb{R}^n$ by projecting it onto the subspace spanned by the columns of $\mathbf{Q} \in \mathbb{R}^{n \times k}$, i.e.,

$$\mathbf{e} = \boldsymbol{\alpha} - \mathbf{Q}\mathbf{Q}^\top \boldsymbol{\alpha}. \quad (5.5)$$

The norm of the residual vector \mathbf{e} corresponds to the approximation error of the final result. Therefore, we want to correct the error when it is too large. On the other hand, when the approximation error is small, there is no need to make such correction with additional complexity cost. In our algorithm, we use a predefined tolerance ε to determine whether the approximation error is small or not.

(a) Case 1: Small Residual Vector Norm

If the norm of the residual vector is small, i.e., $p = \|\mathbf{e}\| < \varepsilon$, then we have

$$\boldsymbol{\alpha} \approx \mathbf{Q}\mathbf{Q}^\top \boldsymbol{\alpha}, \quad (5.6)$$

so that the matrix \mathbf{E} can be approximated by

$$\begin{aligned} \mathbf{E} &\approx \begin{bmatrix} \mathbf{D} & \mathbf{Q}\mathbf{Q}^\top \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^\top \mathbf{Q}\mathbf{Q}^\top & d \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{Q} & 0 \\ 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} \boldsymbol{\Sigma} & \mathbf{Q}^\top \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^\top \mathbf{Q} & d \end{bmatrix}}_{\mathbf{Y}_1} \begin{bmatrix} \mathbf{Q} & 0 \\ 0 & 1 \end{bmatrix}^\top \\ &= \begin{bmatrix} \mathbf{Q} & 0 \\ 0 & 1 \end{bmatrix} \mathbf{Q}_1 \boldsymbol{\Sigma}_1 \mathbf{Q}_1^\top \begin{bmatrix} \mathbf{Q} & 0 \\ 0 & 1 \end{bmatrix}^\top. \end{aligned} \quad (5.7)$$

Here $\mathbf{Q}_1 \boldsymbol{\Sigma}_1 \mathbf{Q}_1^\top$ is the EVD of the middle matrix \mathbf{Y}_1 . \mathbf{Q}_1 and $\boldsymbol{\Sigma}_1$ can be computed by standard EVD algorithm. Given that the size of \mathbf{Y}_1 is $(k + 1) \times (k + 1)$ and $k \ll n$, we can greatly reduce the

computational cost compared with direct recomputing the EVD of \mathbf{E} .

To further reduce the computational complexity as well as the memory cost, we can perform truncation by ignoring the smallest eigenvalue of \mathbf{Y}_1 when it is small. In our algorithm we use another predefined threshold ε_λ to determine if the eigenvalue is small. A typical value of ε_λ is 0.1ε . If the smallest eigenvalue of \mathbf{Y}_1 is small, then we perform truncation and this suggest the follow update

$$\begin{aligned}\mathbf{Q} &\leftarrow \begin{bmatrix} \mathbf{Q} & 0 \\ 0 & 1 \end{bmatrix} \mathbf{Q}_1(:, 1:k), \\ \Sigma &\leftarrow \Sigma_1(1:k, 1:k).\end{aligned}\tag{5.8}$$

Otherwise, we consider the following full-dimension update

$$\begin{aligned}\mathbf{Q} &\leftarrow \begin{bmatrix} \mathbf{Q} & 0 \\ 0 & 1 \end{bmatrix} \mathbf{Q}_1, \\ \Sigma &\leftarrow \Sigma_1.\end{aligned}\tag{5.9}$$

The updated \mathbf{Q} and Σ will be the input of the next iteration of the IEVD-LR algorithm as the approximated EVD of \mathbf{E} .

(b) Case 2: Large Residual Vector Norm

If the norm of the residual vector is large, i.e., $p = \|\mathbf{e}\| \geq \varepsilon$, we have the following fundamental theorem

Theorem 1. Let $\tilde{\mathbf{e}} = \mathbf{e}/p$, then the updated matrix \mathbf{E} can be expressed by the identity

$$\begin{aligned} \mathbf{E} &= \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} \Sigma & 0 & \mathbf{Q}^\top \alpha \\ 0 & 0 & p \\ \alpha^\top \mathbf{Q} & p & d \end{bmatrix}}_{\mathbf{Y}_2} \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix}^\top \\ &= \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{Q}_2 \Sigma_2 \mathbf{Q}_2^\top \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix}^\top \end{aligned} \quad (5.10)$$

Proof. The right-hand side of (5.10) can be written as

$$\begin{aligned} &\begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Sigma & 0 & \mathbf{Q}^\top \alpha \\ 0 & 0 & p \\ \alpha^\top \mathbf{Q} & p & d \end{bmatrix} \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix}^\top \\ &= \begin{bmatrix} \mathbf{Q}\Sigma & 0 & \mathbf{Q}\mathbf{Q}^\top \alpha + \tilde{\mathbf{e}}p \\ \alpha^\top \mathbf{Q} & p & d \end{bmatrix} \begin{bmatrix} \mathbf{Q}^\top & 0 \\ \tilde{\mathbf{e}}^\top & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{Q}\Sigma\mathbf{Q}^\top & \mathbf{Q}\mathbf{Q}^\top \alpha + \mathbf{e} \\ \alpha^\top \mathbf{Q}\mathbf{Q}^\top + \mathbf{e}^\top & d \end{bmatrix} \end{aligned} \quad (5.11)$$

Substitute (5.5) into (5.11), we have

$$\begin{aligned} &\begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Sigma & 0 & \mathbf{Q}^\top \alpha \\ 0 & 0 & p \\ \alpha^\top \mathbf{Q} & p & d \end{bmatrix} \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix}^\top \\ &= \begin{bmatrix} \mathbf{Q}\Sigma\mathbf{Q}^\top & \alpha \\ \alpha^\top & d \end{bmatrix} \\ &= \mathbf{E}, \end{aligned} \quad (5.12)$$

which completes the proof. \square

It is easy to see that 0 must be the eigenvalue of \mathbf{Y}_2 , hence the last column of \mathbf{Q}_2 represents an unused subspace dimension and should be suppressed. Similar to the case when the norm of the residual vector is small, when the second smallest eigenvalue of \mathbf{Y}_2 is small, the truncation update is

$$\begin{aligned} \mathbf{Q} &\leftarrow \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{Q}_2(:, 1:k), \\ \Sigma &\leftarrow \Sigma_2(1:k, 1:k). \end{aligned} \quad (5.13)$$

Otherwise, we perform the full-dimensional update

$$\begin{aligned} \mathbf{Q} &\leftarrow \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{Q}_2(:, 1:k+1), \\ \Sigma &\leftarrow \Sigma_2(1:k+1, 1:k+1). \end{aligned} \quad (5.14)$$

For the case of multiple dimension increase, we will update the EVD result by multiple iterations using the introduced algorithm. Assuming we have the EVD matrices $\mathbf{Q}_{\text{init}} \in \mathbb{R}^{n \times k}$ and $\Sigma_{\text{init}} \in \mathbb{R}^{k \times k}$ for the rank k initial matrix $\mathbf{D}_{\text{init}} \in \mathbb{R}^{n \times n}$, the final matrix $\mathbf{E}_{\text{final}} \in \mathbb{R}^{(n+\ell) \times (n+\ell)}$, where $\ell \geq 1$, we summarize our proposed basic IEVD-LR algorithm in Algorithm 5.1.

5.3.2 Fast Incremental eigenvalue decomposition

We can reduce the computational complexity of the basic IEVD-LR algorithm. To do this, we adapt a new decomposition model. Instead of performing rotations on the large eigenvector matrices, we maintain the eigenvalue decomposition (5.3) in the form of five matrices

$$\mathbf{D} = \tilde{\mathbf{Q}} \hat{\mathbf{Q}} \Sigma \hat{\mathbf{Q}}^\top \tilde{\mathbf{Q}}^\top, \quad (5.15)$$

where $\tilde{\mathbf{Q}} \in \mathbb{R}^{n \times k}$ and $\hat{\mathbf{Q}} \in \mathbb{R}^{k \times k}$ such that the product of these two matrices $\mathbf{Q} = \tilde{\mathbf{Q}} \hat{\mathbf{Q}}$ is orthonormal. The expansive outer matrices $\tilde{\mathbf{Q}}$ and $\tilde{\mathbf{Q}}^\top$ solely capture the eigen subspace's span.

Algorithm 5.1 Basic Incremental Eigenvalue Decomposition Algorithm (IEVD-LR-BASIC)

Input: Decomposition matrix \mathbf{Q}_{init} , diagonal eigenvalue matrix $\mathbf{\Sigma}_{\text{init}}$, additional vectors $\{\boldsymbol{\alpha}_i, i \in 1, \dots, \ell\}$, additional singular values $\{d_i, i \in 1, \dots, \ell\}$ tolerance ε for residual vector and threshold ε_λ for eigenvalue;

Output: Decomposition matrix \mathbf{Q} and diagonal eigenvalue matrix $\mathbf{\Sigma}$ as the approximated EVD of $\mathbf{E}_{\text{final}}$;

1. Set \mathbf{Q} and $\mathbf{\Sigma}$ as \mathbf{Q}_{init} and $\mathbf{\Sigma}_{\text{init}}$ respectively;

for i from 1 to ℓ **do**

2. Calculate the residual vector \mathbf{e}_i and its norm using (5.5);

if $p = \|\mathbf{e}\| < \varepsilon$ **then**

3. Calculate the EVD of \mathbf{Y}_1 defined in (5.7);

if the smallest eigenvalue of \mathbf{Y}_1 is smaller than ε_λ **then**

4. Update \mathbf{Q} and $\mathbf{\Sigma}$ using (5.8);

else

5. Update \mathbf{Q} and $\mathbf{\Sigma}$ using (5.9);

end if

else

6. Calculate the EVD of \mathbf{Y}_2 defined in (5.10);

if the second smallest eigenvalue of \mathbf{Y}_2 is smaller than ε_λ **then**

7. Update \mathbf{Q} and $\mathbf{\Sigma}$ using (5.13);

else

8. Update \mathbf{Q} and $\mathbf{\Sigma}$ using (5.14);

end if

end if

end for

They are extended by adding rows to $\tilde{\mathbf{Q}}$ when the rank of the update matrix remains the same, and by appending both new rows and columns when the rank of the update matrix increases. The transforms of these subspace bases that make $\mathbf{\Sigma}$ diagonal are maintained in the much smaller $\hat{\mathbf{Q}}$ matrix, whose size only extends when the rank of the update matrix increases. This makes the update much faster and eliminates the numerical error that would accumulate if the bases specified by the tall $\tilde{\mathbf{Q}}$ matrix was rotated on each update.

With the new expression of the decomposition form in (5.15), we can write down the new update equations of our IEVD-LR algorithm.

(a) Case 1: Small Residual Vector Norm

If the norm of the residual vector is small, i.e., $p = \|\mathbf{e}\| < \varepsilon$, based on the preceding discussion in Section. 5.3.1, it is imperative to ensure that the product $\mathbf{Q} = \tilde{\mathbf{Q}}\hat{\mathbf{Q}}$ remains orthogonal. Referring to equations (5.8) and (5.9), we can deduce that the right-side update must adhere to the following form

$$\tilde{\mathbf{Q}}_{\text{new}} \hat{\mathbf{Q}}_{\text{new}} = \begin{bmatrix} \tilde{\mathbf{Q}}_{\text{old}} \hat{\mathbf{Q}}_{\text{old}} & 0 \\ 0 & 1 \end{bmatrix} \mathbf{X}, \quad (5.16)$$

where $\tilde{\mathbf{Q}}_{\text{old}}$ and $\hat{\mathbf{Q}}_{\text{old}}$ correspond to the original matrix \mathbf{D} , and $\tilde{\mathbf{Q}}_{\text{new}}$ and $\hat{\mathbf{Q}}_{\text{new}}$ correspond to the updated matrix \mathbf{E} . \mathbf{X} is a variable matrix that takes different values in different cases. In cases where the rank increases, let $\mathbf{X} = \mathbf{Q}_1$, and when the rank remains unchanged, set $\mathbf{X} = \mathbf{Q}_1(:, 1 : k)$.

We can reduce the computational complexity of (5.16) by keeping on updating a small pseudo-inverse matrix $\hat{\mathbf{Q}}_{\text{old}}^+$. When the rank does not increase, we can further reduce the complexity by splitting $\mathbf{Q}_1(:, 1 : k) \in \mathbb{R}^{(k+1) \times k}$ into the form

$$\mathbf{Q}_1(:, 1 : k) = \begin{bmatrix} \mathbf{W} \in \mathbb{R}^{k \times k} \\ \mathbf{w} \in \mathbb{R}^{1 \times k} \end{bmatrix}, \quad (5.17)$$

where submatrix \mathbf{W} is a linear transform that will be applied to $\hat{\mathbf{Q}}_{\text{old}}$, and row-vector \mathbf{w} is the subspace projection of the new data vector. By substituting (5.17) into (5.16), the resulting right-side update becomes

$$\begin{aligned} \hat{\mathbf{Q}}_{\text{new}} &\leftarrow \hat{\mathbf{Q}}_{\text{old}} \mathbf{W}; \\ \hat{\mathbf{Q}}_{\text{new}}^+ &\leftarrow \mathbf{W}^+ \hat{\mathbf{Q}}_{\text{old}}^+; \\ \tilde{\mathbf{Q}}_{\text{new}} &\leftarrow \begin{bmatrix} \tilde{\mathbf{Q}}_{\text{old}} \\ \mathbf{w} \hat{\mathbf{Q}}_{\text{new}}^+ \end{bmatrix}. \end{aligned} \quad (5.18)$$

Conveniently, the pseudo-inverse \mathbf{W}^+ can be computed in $\mathcal{O}(k^2)$ -time using only

matrix-vector and vector-vector products via the identity

$$\mathbf{W}^+ = \mathbf{W}^\top + \frac{\mathbf{w}^\top}{1 - \|\mathbf{w}\|^2} (\mathbf{w}\mathbf{W}^\top). \quad (5.19)$$

When the update is rank-increasing, the right-side update in (5.16) can be replaced by

$$\begin{aligned} \widehat{\mathbf{Q}}_{\text{new}} &\leftarrow \begin{bmatrix} \widehat{\mathbf{Q}}_{\text{old}} & 0 \\ 0 & 1 \end{bmatrix} \mathbf{Q}_1; \\ \widehat{\mathbf{Q}}_{\text{new}}^+ &\leftarrow \mathbf{Q}_1^\top \begin{bmatrix} \widehat{\mathbf{Q}}_{\text{old}}^+ & 0 \\ 0 & 1 \end{bmatrix}; \\ \widetilde{\mathbf{Q}}_{\text{new}} &\leftarrow \begin{bmatrix} \widetilde{\mathbf{Q}}_{\text{old}} & 0 \\ 0 & 1 \end{bmatrix}. \end{aligned} \quad (5.20)$$

(b) Case 2: Large Residual Vector Norm

If the norm of the residual vector is large, i.e., $p = \|\mathbf{e}\| \geq \varepsilon$, referring to equations (5.13) and (5.14), the update must satisfy

$$\widetilde{\mathbf{Q}}_{\text{new}} \widehat{\mathbf{Q}}_{\text{new}} = \begin{bmatrix} \widetilde{\mathbf{Q}}_{\text{old}} \widehat{\mathbf{Q}}_{\text{old}} & \widetilde{\mathbf{e}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{X},$$

where \mathbf{X} is a variable matrix that takes different values in different cases. When the rank remains unchanged, set $\mathbf{X} = \mathbf{Q}_2(:, 1 : k)$ and split $\mathbf{X} = \begin{bmatrix} \mathbf{W} \in \mathbb{R}^{(k+1) \times k} \\ \mathbf{w} \in \mathbb{R}^{1 \times k} \end{bmatrix}$. In cases where the rank increases, let $\mathbf{X} = \mathbf{Q}_2(:, 1 : k + 1)$ and split $\mathbf{X} = \begin{bmatrix} \mathbf{W} \in \mathbb{R}^{(k+1) \times (k+1)} \\ \mathbf{w} \in \mathbb{R}^{1 \times (k+1)} \end{bmatrix}$.

The update is simply

$$\begin{aligned}
\widehat{\mathbf{Q}}_{\text{new}} &\leftarrow \begin{bmatrix} \widehat{\mathbf{Q}}_{\text{old}} & 0 \\ 0 & 1 \end{bmatrix} \mathbf{W}; \\
\widehat{\mathbf{Q}}_{\text{new}}^+ &\leftarrow \mathbf{W}^+ \begin{bmatrix} \widehat{\mathbf{Q}}_{\text{old}}^+ & 0 \\ 0 & 1 \end{bmatrix}; \\
\widetilde{\mathbf{Q}}_{\text{new}} &\leftarrow \begin{bmatrix} \begin{bmatrix} \widetilde{\mathbf{Q}}_{\text{old}} & \widetilde{\mathbf{e}} \end{bmatrix} \\ \mathbf{w} \widehat{\mathbf{Q}}_{\text{new}}^+ \end{bmatrix}.
\end{aligned} \tag{5.21}$$

For the case of multiple dimension increase, we will update the EVD result by multiple iterations using the introduced algorithm. Assuming we have the EVD matrices $\mathbf{Q}_{\text{init}} \in \mathbb{R}^{n \times k}$ and $\mathbf{\Sigma}_{\text{init}} \in \mathbb{R}^{k \times k}$ for the rank k initial matrix $\mathbf{D}_{\text{init}} \in \mathbb{R}^{n \times n}$, the final matrix $\mathbf{E}_{\text{final}} \in \mathbb{R}^{(n+\ell) \times (n+\ell)}$, where $\ell \geq 1$, we summarized our proposed fast IEVD algorithm for multiple dimension increase case in Algorithm 5.2.

Algorithm 5.2 Fast Incremental Eigenvalue Decomposition Algorithm (IEVD-LR-FAST)

Input: Decomposition matrices \mathbf{Q}_{init} , diagonal eigenvalue matrix Σ_{init} , additional vectors $\{\alpha_i, i \in 1, \dots, \ell\}$, additional singular values $\{d_i, i \in 1, \dots, \ell\}$, tolerance ε for residual vector and threshold ε_λ for eigenvalue;

Output: Decomposition matrix \mathbf{Q} and diagonal eigenvalue matrix Σ as the approximated EVD of $\mathbf{E}_{\text{final}}$;

1. Set $\tilde{\mathbf{Q}}$ and Σ as \mathbf{Q}_{init} and Σ_{init} respectively, and set both $\hat{\mathbf{Q}}$ and $\hat{\mathbf{Q}}^+$ as identity matrix \mathbf{I} ;
for i from 1 to ℓ **do**

2. Calculate the residual vector \mathbf{e}_i and its norm using (5.5);

if $p = \|\mathbf{e}\| < \varepsilon$ **then**

3. Calculate the EVD of \mathbf{Y}_1 defined in (5.7);

if the smallest eigenvalue of \mathbf{Y}_1 is smaller than ε_λ **then**

4. Calculate \mathbf{W} and \mathbf{W}^+ based on \mathbf{Q}_1 using (5.17) and (5.18);

5. Update $\tilde{\mathbf{Q}}$, $\hat{\mathbf{Q}}$, $\hat{\mathbf{Q}}^+$ and Σ using (5.19);

else

6. Update $\tilde{\mathbf{Q}}$, $\hat{\mathbf{Q}}$, $\hat{\mathbf{Q}}^+$ and Σ using (5.20);

end if

else

6. Calculate the EVD of \mathbf{Y}_2 defined in (5.10);

7. Calculate \mathbf{W} and \mathbf{W}^+ based on \mathbf{Q}_2 using (5.18);

8. Update $\tilde{\mathbf{Q}}$, $\hat{\mathbf{Q}}$, $\hat{\mathbf{Q}}^+$ and Σ using (5.21);

end if

end for

9. Calculate \mathbf{Q} using $\tilde{\mathbf{Q}}$ and $\hat{\mathbf{Q}}$;

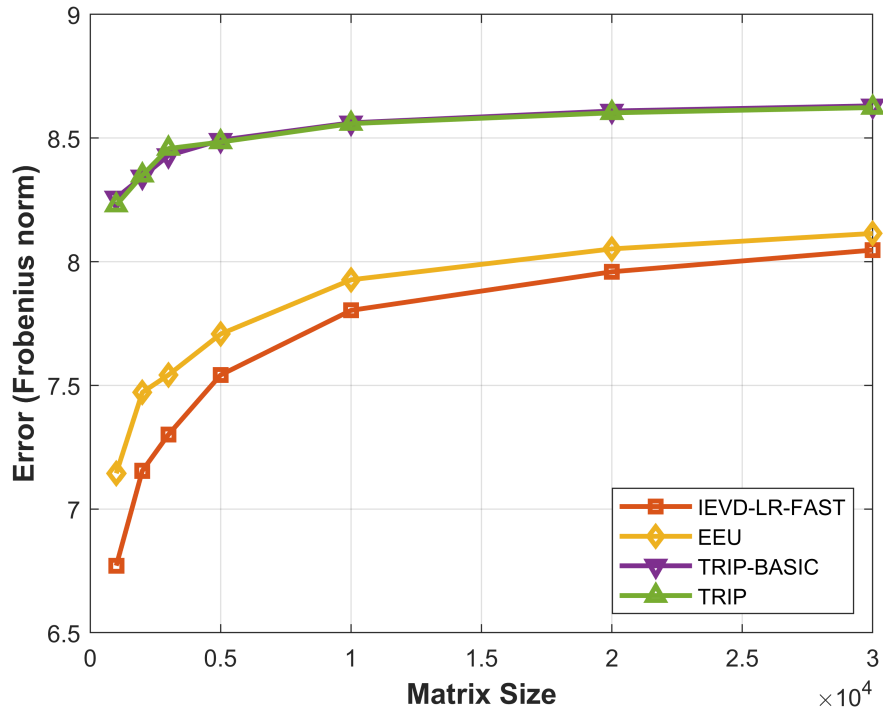
5.4 Experiments

We now present the experimental results of the proposed fast IEVD algorithm, comparing its performance in both synthetic and real datasets against existing EVD approximation approaches. Specifically, we evaluate our IEVD-LR-FAST algorithm alongside other state-of-the-art methods, including efficient eigen-updating (EEU) algorithm presented in [101], the TRIP-BASIC and TRIP algorithm outlined in [103], as well as the conventional eigen-decomposition (EIG) algorithm.

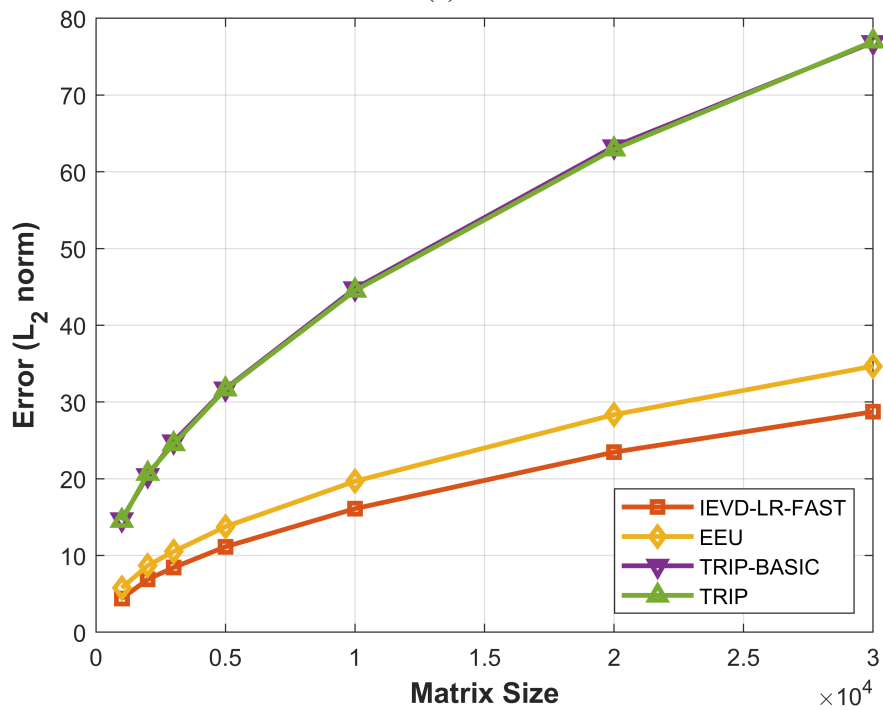
5.4.1 Synthetic dataset

In this subsection, we evaluate the error performance and runtime of our proposed IEVD-LR-FAST algorithm using positive definite symmetric matrices. The reason for selecting such matrices is to demonstrate the algorithm’s versatility across different matrix sizes. We generate these random symmetric matrices as follows: we first generate an n -by- n size random matrix whose elements are randomly sampled from uniform distribution from 0 to 1. Sequentially, we calculate the mean of this random matrix and its transpose to form a symmetric matrix. We then calculate the EVD matrices \mathbf{Q} and $\mathbf{\Sigma}$ of the symmetric matrix using conventional EIG algorithm. Next, we set the elements of $\mathbf{\Sigma}$ to its absolute value. Finally, we use \mathbf{Q} and updated $\mathbf{\Sigma}$ to generate a positive definite symmetric matrix.

Throughout our experiments, we set the final matrix of all EVD algorithms to an n -by- n size, while the initial matrix \mathbf{D}_{init} is configured as the top left submatrix of size $0.9n$ -by- $0.9n$. We focus on updating top K eigen-pairs to match the task in Section 5.4.2, i.e., the input decomposition matrices \mathbf{Q}_{init} and $\mathbf{\Sigma}_{\text{init}}$ are truncated by keeping the top K eigen-pairs of the initial matrix \mathbf{D}_{init} . After getting the estimation result of the EVD algorithm, we truncate the output matrices \mathbf{Q} and $\mathbf{\Sigma}$ in the same way, and compare them with the ground truth result given by EIG algorithm on the final matrix. To reduce the impact of the randomness, we conduct 100 Monte Carlo runs for each matrix size and use the average estimation error as the metric. For measuring the estimation errors on eigenvector matrix \mathbf{Q} , we first reorder the columns in \mathbf{Q} to find the best match with the ground truth. We then calculate the Frobenius norm of the difference between the reordered matrix and the ground truth. In this way we can avoid the order mismatch of the eigenvectors caused by the small turbulence on the estimation of eigenvalues. On the other hand, we use the L_2 norm to measure the difference of eigenvalues with the ground truth. The average estimation errors of all EVD algorithms are shown in Fig. 5.3. Our proposed IEVD-LR-FAST has the lowest estimation errors on all matrix sizes ranging from 1,000 to 30,000, which demonstrate the accuracy of our proposed algorithm. Considering the fact that the main difference between the EEU algorithm and our proposed IEVD-LR-FAST algorithm is the error correction branch, this performance



(a)



(b)

Figure 5.3: Estimation errors of the EVD algorithm on synthetic dataset. (a) Average estimation errors of eigenvector matrix \mathbf{Q} , measured by Frobenius norm. (b) Average estimation errors of eigenvector matrix Σ , measured by L_2 norm.

improvement demonstrate the effect of the error correction branch in our proposed algorithm. Also, the TRIP and the TRIP-BASIC algorithms perform bad compared with other algorithms, showing that these two algorithms are not suitable for EVD updating problem when the size of the matrix is increasing.

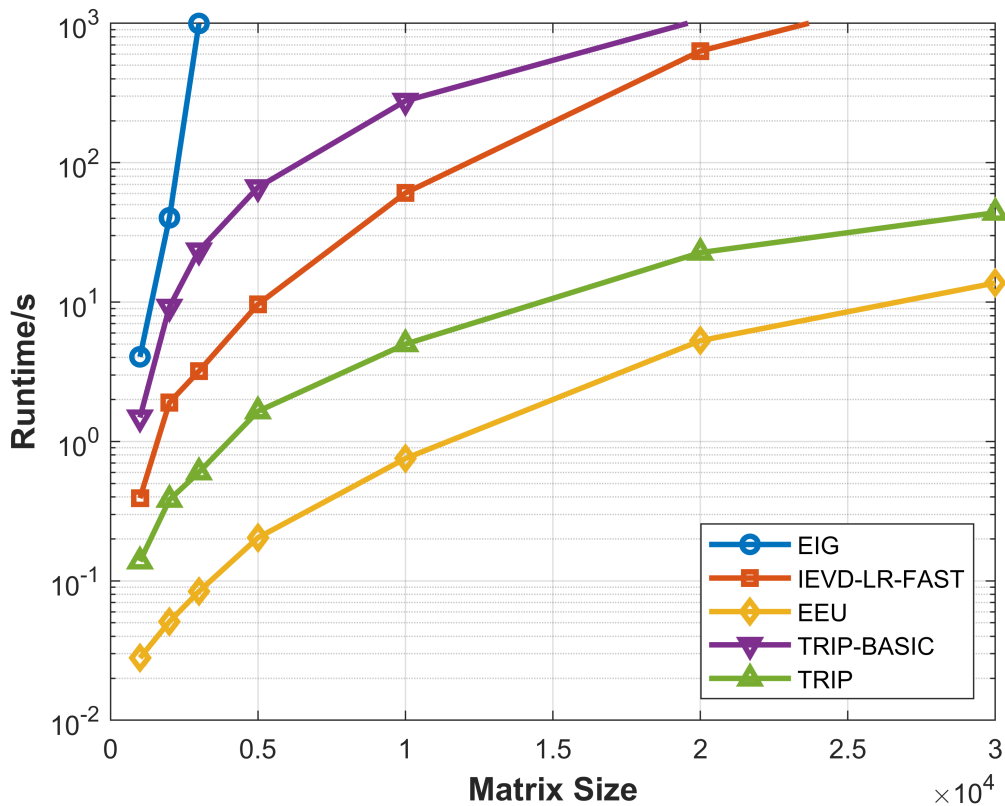


Figure 5.4: Average runtime of the EVD algorithm on synthetic dataset.

We also collect the average runtime of all EVD algorithms for different matrix sizes. All the algorithms are implemented in MATLAB and executed on the same computer equipped with i5-9600K CPU at 3.7 GHz. As shown in Fig. 5.4, all EVD algorithms are much faster than the conventional EIG algorithm. The runtime of our proposed IEVD-LR-FAST algorithm is smaller than TRIP-BASIC, but larger than TRIP and EEU algorithms. Such higher runtime of our proposed algorithm is partially caused by the high number of loops in the implementation. Considering both TRIP and EEU are updating the matrix changes in one step, they have lower number of loops than our proposed IEVD-LR-FAST algorithm.

5.4.2 Real Multimedia Datasets

In this subsection, we assess the error performance and runtime of our proposed IEVD-LR-FAST algorithm on real multimedia datasets. Specifically, we subject the EVD algorithms on dynamic point clouds, wireframe datasets and hyperspectral images as examples for graph-based applications such as spectral clustering and low-pass filtering. Despite the adjacency matrices for graphs built on real dataset are typically high rank, our focus lies on updating top K eigen-pairs, considering the fact that the top K eigen-pairs are commonly used in spectral clustering and low-pass graph filtering algorithms. To ensure a fair comparison, we standardize the comparison process by truncating the output matrices \mathbf{Q} and Σ of our proposed IEVD-LR-FAST algorithm by selecting the first K column vectors and diagonal elements, respectively.

In our experiments on all three datasets, we begin by constructing the adjacency matrix based on the attribute similarities, using it as the final matrix for all EVD algorithms. Subsequently, we randomly drop some nodes of the data, resulting in a truncated adjacency matrix with the corresponding rows and columns removed. We then compute the top K eigen-pairs of this truncated adjacency matrix using conventional EIG algorithm as the input of all EVD algorithms. Finally, we use the output of these EVD algorithms for spectral clustering and low-pass filtering to assess their error performance.

For experiments on spectral clustering, we employ the approximated EVD results \mathbf{Q} to cluster the data, comparing the results with the cluster outcomes using the conventional EIG algorithm. The number of clustering errors serves as the metric to assess error performance. Because spectral clustering algorithm only uses the estimated eigenvectors, these results reflect the accuracy of estimating the eigenvectors of EVD algorithms.

In low-pass filtering experiments, we use both ideal graph-based filter and Haar-like graph-based filter based on the approximated top K eigen-pairs of EVD algorithms to process the graph signal based on the vector of the first attribute on all nodes. The ideal filter uses all K estimated eigenvectors, while the Haar-like filter also uses the corresponding estimated eigenvalues. The ground truth for the filtered spatial domain signal is generated using eigen-pairs

from the conventional EIG algorithm, and the L_2 norm of the difference between the filtered spatial domain signal using EVD algorithms and ground truth as the metrics to assess error performance. The results of ideal graph-based filters reflect the accuracy of estimating eigenvectors of EVD algorithms, while the results of Haar-like graph-based filters reflect the accuracy of estimating both eigenvectors and the exact value of all eigenvalues.

(a) Dynamic point clouds

Dynamic point clouds have various applications in autonomous driving [105], virtual reality [106] and medical imaging [107]. An example of dynamic point cloud with additional nodes in successive frames is shown in Fig. 5.5. In this example, the points on the right foot are not captured by the sensors in the first frame. These additional points in the second frame resulting in the addition of nodes and edges in the new graph, which corresponds to the additional rows and columns in the new adjacency matrix. As shown in the Fig. 5.5(b), dynamic graph is a nature method to capture the underling relationship of points in dynamic point cloud. As a result, graph-based analyzing and processing methods, such as spectral clustering and graph-based filtering, are popular way of processing dynamic point clouds. The authors in [111] utilize spectral clustering to separate the motions flows in their proposed object detection and extraction framework on mobile lidar data. A robust dynamic point cloud segmentation routine is proposed in [112], where the spectral clustering is used to generate the initial segmentation. A graph-based low-pass filtering method is proposed in [113] for point cloud denoising.

In this chapter, we only focus on the cases when the size of the point cloud increases. Therefore, we use the static point clouds in Mythological creatures database [108, 109] to simulate a dynamic point cloud with addition of nodes in successive frames. We conduct 10 Monte Carlo runs on each point cloud. In each round we first build the ϵ -neighborhood graph on the point cloud with the radius of neighborhood ϵ as three times of the intrinsic distance d_r as (2.1). The distance between two points is measured by the L_2 norm of the difference on attributes of these two points. Then we calculate the adjacency matrix of the constructed graph using the Gaussian kernel. This adjacency

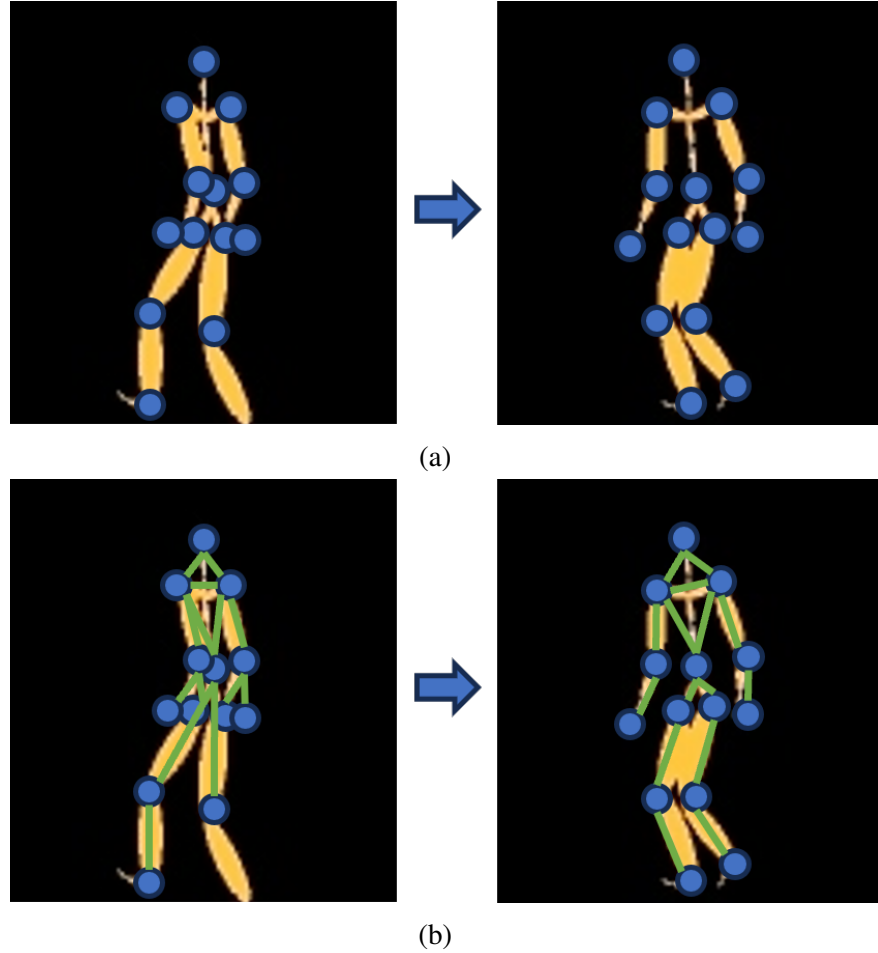


Figure 5.5: Example of dynamic point cloud with additional nodes in successive frames. (a) Example of dynamic point cloud, the points on the right foot does not captured by the sensors in the first frame. (b) Example of the graphs built on the dynamic point cloud, the additional points in the second frame resulting in the addition of nodes and edges in the new graph, corresponding to the additional rows and columns in the new adjacency matrix.

matrix is the final matrix of all EVD algorithm. Next, we randomly drop 1% of the nodes in the point cloud, and remove their corresponding rows and columns in the adjacency matrix. We then calculate the top K eigen-pairs of this truncated adjacency matrix \mathbf{A}_{init} to generate the input decomposition matrix \mathbf{Q}_{init} and $\mathbf{\Sigma}_{init}$. In our test the number of tracking eigen-pairs K is set to 100. We summarize this EVD updating algorithm in Algorithm 5.3.

After getting the approximated EVD results \mathbf{Q} and $\mathbf{\Sigma}$ of the final matrix, we use them to do spectral clustering, Haar-like low-pass filtering and ideal low-pass filtering compare them with the ground truth results using the top K eigen-pairs of the full-sized adjacency matrix. For spectral

Algorithm 5.3 Fast Incremental Eigenvalue Decomposition Updating Algorithm for Dynamic Point Cloud

Input: Coordinates of the updated points \mathbf{P}_u , adjacency matrix of the point cloud at the current frame \mathbf{A}_{init} , number of tracking eigen-pairs K , intrinsic resolution d_r ;

Output: Decomposition matrix \mathbf{Q} and diagonal eigenvalue matrix $\mathbf{\Sigma}$ as the approximated EVD of the adjacency matrix of the updated matrix $\mathbf{A}_{\text{final}}$;

1. For each updated point whose coordinates are in \mathbf{P}_u , calculate the additional vector α_i as 2.1, set all singular values $\{d_i\}$ as 1.;
 2. Calculate \mathbf{Q}_{init} and $\mathbf{\Sigma}_{\text{init}}$ containing the top K eigen-pairs of \mathbf{A}_{init} using conventional EIG algorithm;
 3. Use Algorithm 5.2 to calculate the approximated EVD results \mathbf{Q} and $\mathbf{\Sigma}$ for the updated frame;
 4. Trim \mathbf{Q} and $\mathbf{\Sigma}$ by keeping the top K eigen-pairs.
-

Table 5.1: Average clustering errors and low-pass filter errors on dynamic point clouds

	IEVD-LR-FAST	EEU	TRIP-BASIC	TRIP
Number of Clustering Errors	31.0270	41.3378	101.7568	101.2838
L2 (Haar)	51.6431	60.4235	111.5380	111.2226
L2 (Ideal)	68.1637	84.0457	187.3133	186.9413

clustering results, the error performance is measured by the number of points that are incorrectly clustered. For low-pass filtering results, the error performance is measured by the L_2 norm between the filtered signals using approximated results and ground truth.

The average result is shown in Table. 5.1. Our proposed IEVD-LR-FAST algorithm has the lowest error in all three metrics, which demonstrate the accuracy of our proposed algorithm on both eigenvalues and eigenvectors in dynamic point clouds. Compared with EEU algorithm, our proposed IEVD-LR-FAST algorithm has significant performance improvement. Considering the fact that the main difference between the EEU algorithm and our proposed IEVD-LR-FAST algorithm is the error correction branch, this performance improvement demonstrate the effect of the error correction branch in our proposed algorithm on dynamic point clouds. One example of the spectraling result using IEVD-LR-FAST algorithm is shown in Fig. 5.6, where the cluster error points are labeled in pink. As shown in Fig. 5.6(c), our proposed IEVD-LR-FAST algorithm can keep the majority of the clustering result to be the same as the ground truth.

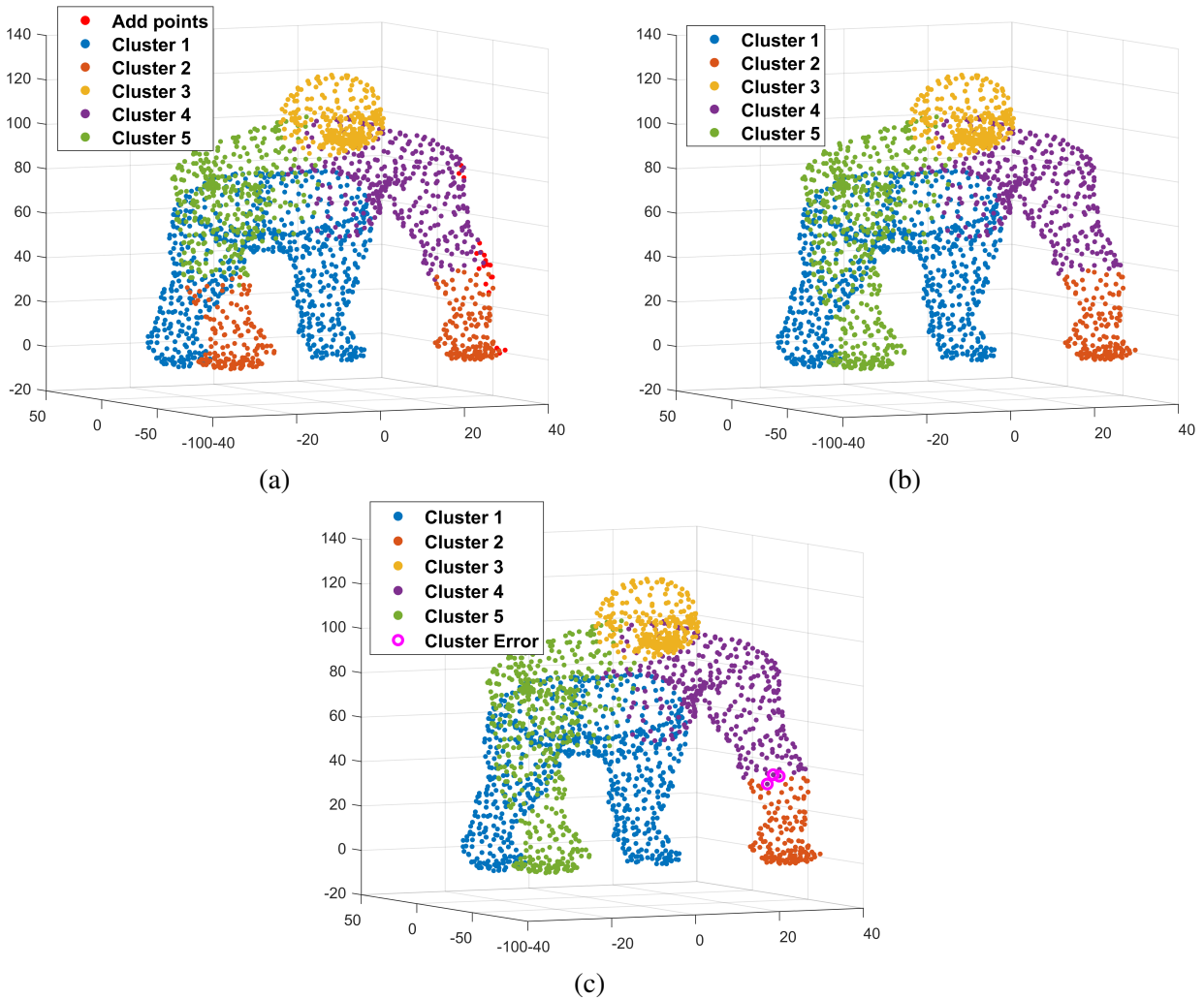


Figure 5.6: Example of spectral clustering result in our experiments. (a) Clustering result of the original point cloud using the ground truth EVD result, updated points are labeled in red. (b) Clustering result of the updated point cloud using ground truth. (c) Clustering result of the updated point cloud using estimated EVD results from IEVD-LR-FAST algorithm, clustering errors are marked in pink.

(b) Large-scale point cloud

In real-world applications, point clouds may contain millions of points. Traditional EVD algorithms need an enormous amount of memory to process such a large-scale matrix directly, which makes the process of estimating its eigen-pairs very difficult. However, our proposed IEVD-LR-FAST algorithm only requires a small amount of memory, which provides a solution for these problems. In this experiment we test a simple algorithm on estimating the top K eigen-pairs



Figure 5.7: Boxer point cloud in 8i Voxelized Surface Light Field (8iVSLF) Dataset.

of the adjacency matrix of a large-scale point cloud. We first select a part of the large-scale point cloud containing N_0 points. Subsequently, we build the adjacency matrix on this part based on its intrinsic distance with the Gaussian kernel. Then we calculate the top K eigenpairs of this adjacency matrix as the input of the IEVD-LR-FAST algorithm. Next, we add one point in the original point cloud each time to the existing part, while using the same intrinsic distance value to calculate the corresponding additional column vector α and d in the updated adjacency matrix. α and d are used in the IEVD-LR-FAST algorithm to update the approximation of the top K eigenpairs. Although this simple algorithm may have large estimation errors on the final result, especially when N_0 is small compared with the number of points in the original point cloud, our purpose is to demonstrate that IEVD-LR-FAST can be used to estimate the eigenpairs of large-scale matrix with limit on the memory. On the other hand, how to further improve the accuracy of this simple algorithm is beyond the scope of this chapter. Therefore, we do not compare the error performance of this simple algorithm on large-scale matrices.

To demonstrate the memory efficiency of our proposed algorithm, we test it on the Boxer point

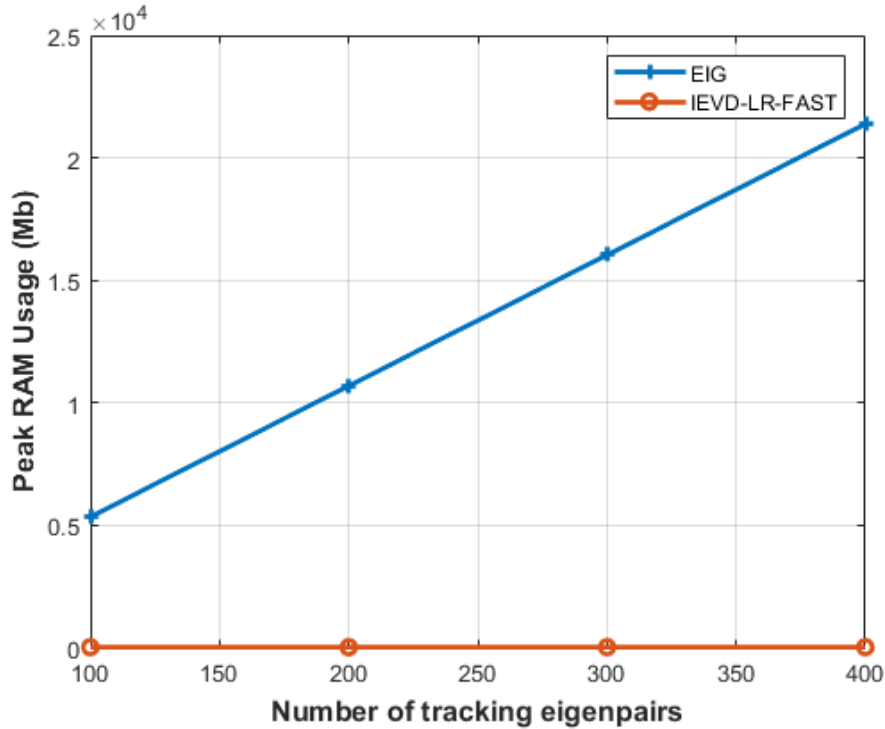


Figure 5.8: Peak memory usage of the EVD algorithms on Boxer point cloud.

cloud in 8i Voxelized Surface Light Field (8iVSLF) Dataset [50], which contains about 3.49 million nodes, as shown in Fig. 5.7. We measure the peak memory usage by using the built-in profiler in Matlab. The peak memory usage result is shown in Fig. 5.8. Compared with the conventional EIG algorithm, our proposed IEVD-LR-FAST has much lower peak memory usage. This is because once initial eigenpairs are calculated, we can free the memory of the adjacency matrix.

(c) Motion capture data

Human motion analysis has recently emerged as an active research field, stemming from its broad applications in many areas, ranging from human-robot interaction to autonomous driving [56]. Motion capture data contains a sequence of data frames captured by sensors mounted on the human body, as shown in Fig. 5.9. If we consider the attributes of all nodes in one frame as the signal on one vertex, we can build a temporal correlation matrix on motion capture data. With the number of data frames increase through out the time, the size of the temporal correlation matrix increases

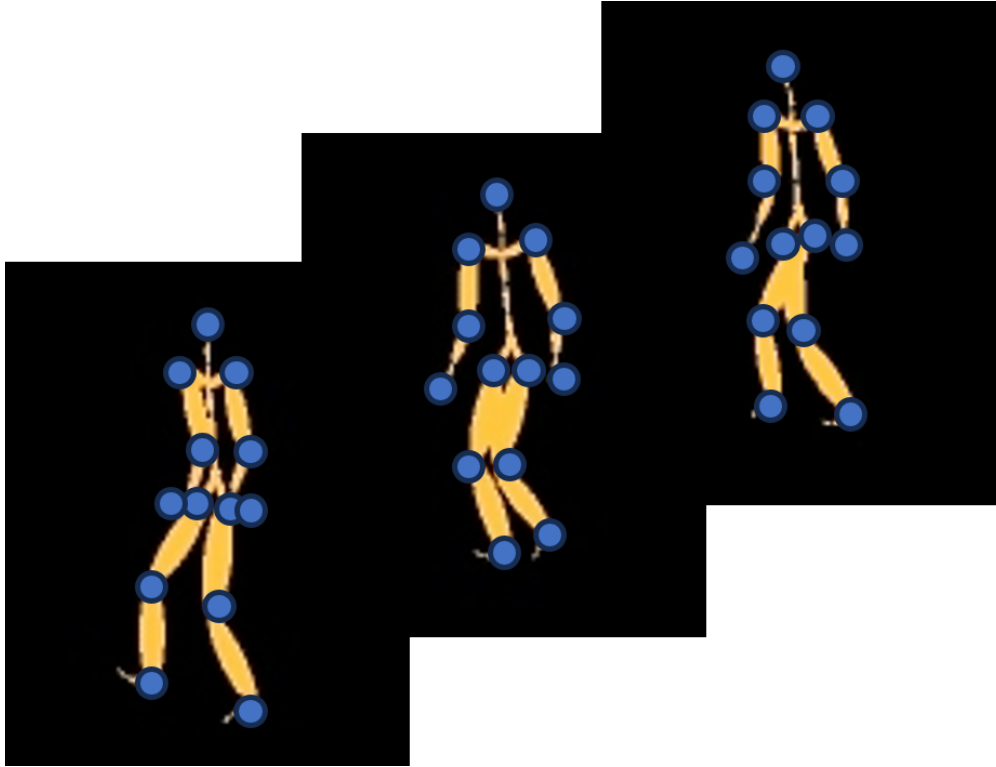


Figure 5.9: Example of motion capture data.

as well, which provides a background for EVD algorithms. One attracting problem on motion capture data is the motion segmentation, which aims to separate the motion capture sequence into segments, each corresponds to one kinds of motion. Of all existing works on motion segmentation, spectral clustering is one of the basic methods. The authors in [63] used spectral clustering as the basic comparison in their experiments. On the other hand, many works develop new clustering methods based on spectral clustering. The authors in [114] employed two methods of spectral clustering, t-nearest neighbors and the Nystrom method, to cluster motion capture data for getting behavioral segmentation. Spectral clustering was also used in [64] as a part of the proposed framework of sparse subspace clustering based on Riemannian manifold structure. Graph filters are also use in motion capture data analysis. A irregular-aware graph filters and graph Fourier transform were proposed in [115] by considering the irregular relationships between the data points on applications like motion capture data.

In this part, we test the EVD algorithms on trails 01 to 14 of subject 86 in CMU graphics lab

Algorithm 5.4 Fast Incremental Eigenvalue Decomposition Updating Algorithm for Motion Capture data

Input: Attributes of the updated frames \mathcal{A}_u , adjacency matrix of the point cloud at the current frame \mathbf{A}_{init} , number of tracking eigen-pairs K , intrinsic resolution d_r ;

Output: Decomposition matrix \mathbf{Q} and diagonal eigenvalue matrix $\mathbf{\Sigma}$ as the approximated EVD of the adjacency matrix of the updated matrix $\mathbf{A}_{\text{final}}$;

1. For each updated frame whose attributes are in \mathcal{A}_u , calculate the additional vector α_i as 2.1, set all singular values $\{d_i\}$ as 1.;
 2. Calculate \mathbf{Q}_{init} and $\mathbf{\Sigma}_{\text{init}}$ containing the top K eigen-pairs of \mathbf{A}_{init} using conventional EIG algorithm;
 3. Use Algorithm 5.2 to calculate the approximated EVD results \mathbf{Q} and $\mathbf{\Sigma}$ for the updated frame;
 4. Trim \mathbf{Q} and $\mathbf{\Sigma}$ by keeping the top K eigen-pairs.
-

motion capture database¹. Similar to Section 5.4.2, we conduct 10 Monte Carlo runs on each trail. In each run, the temporal correlation matrix is based by the L2 distance between all data values of two frames, and using a Gaussian kernel to calculate the correlation value. This correlation matrix is the final matrix of all EVD algorithm. Next, we randomly drop 1% of the frames in the wireframe sequence, and remove its corresponding rows and columns in the correlation matrix. We then calculate the top K eigen-pairs of this truncated correlation matrix to generate the input decomposition matrix \mathbf{Q}_{init} and $\mathbf{\Sigma}_{\text{init}}$. In our test K is still set to 100. After getting the approximated EVD results \mathbf{Q} and $\mathbf{\Sigma}$ of the final matrix, we use them to do spectral clustering, Haar-like low-pass filtering and ideal low-pass filtering compare them with the ground truth results using the top K eigen-pairs of the full-sized correlation matrix. We summarize this EVD updating algorithm in Algorithm 5.4.

The average result is shown in Table. 5.2. Our proposed IEVD-LR-FAST algorithm has the lowest error in all three metrics, which demonstrate the accuracy of our proposed algorithm on motion capture data.

(d) Hyperspectral image

Hyperspectral image has been widely used in applications like earth observation and industrial scanning [110]. As shown in Fig. 5.10, hyperspectral image contains images of multiple bandwidth

¹<http://mocap.cs.cmu.edu/>

Table 5.2: Average clustering errors and low-pass filter errors on motion capture data

	IEVD-LR-FAST	EEU	TRIP-BASIC	TRIP
Number of Clustering Errors	8.2143	22.1429	65.1244	63.1054
L2 (Haar)	0.4568	0.5810	0.8157	0.7982
L2 (Ideal)	0.4569	0.5812	0.8136	0.8065



Figure 5.10: Example of the hyperspectral image with RGB colors.

on the same object. The pixels in hyperspectral image can be viewed as the vertices of a graph, while the values of all bandwidth form attributes on the vertices. One common sensor of capturing the hyperspectral image is called pushbroom hyperspectral imager, which scans a line at each moment in time. Therefore, the size of the hyperspectral image is increasing through out the time, and this scenario is suitable for the EVD algorithms. Spectral clustering and graph-based filters are widely used on hyperspectral images. A fast spectral clustering was proposed in [116] for unsupervised hyperspectral image classification. The authors in [117] proposed a spatial-spectral

Algorithm 5.5 Fast Incremental Eigenvalue Decomposition Updating Algorithm for hyperspectral image

Input: Attributes of the updated points \mathcal{A}_u , adjacency matrix of the point cloud at the current frame \mathbf{A}_{init} , number of tracking eigen-pairs K , intrinsic resolution d_r ;

Output: Decomposition matrix \mathbf{Q} and diagonal eigenvalue matrix $\mathbf{\Sigma}$ as the approximated EVD of the adjacency matrix of the updated matrix $\mathbf{A}_{\text{final}}$;

1. For each updated frame whose attributes are in \mathcal{A}_u , calculate the additional vector α_i as 2.1, set all singular values $\{d_i\}$ as 1.;
 2. Calculate \mathbf{Q}_{init} and $\mathbf{\Sigma}_{\text{init}}$ containing the top K eigen-pairs of \mathbf{A}_{init} using conventional EIG algorithm;
 3. Use Algorithm 5.2 to calculate the approximated EVD results \mathbf{Q} and $\mathbf{\Sigma}$ for the updated frame;
 4. Trim \mathbf{Q} and $\mathbf{\Sigma}$ by keeping the top K eigen-pairs.
-

clustering with anchor graph for HSI data clustering. On the other hand, the authors in [118] designed a linear function to combine the different graph filters in their proposed framework so that the graph filter can be adaptively determined by training different weight matrices.

In this part, we test the EVD algorithms on the Indian Pines data [119]. We follow the similar workflow as Section 5.4.2 and Section 5.4.2. The major difference of our experiment for hyperspectral image is that we drop the last 5% columns of the original hyperspectral image to simulate the real situation of using a pushbroom hyperspectral imager. We first build the ϵ -neighborhood graph on the hyperspectral image with the radius of neighborhood as three times of the intrinsic distance, where the intrinsic distance is defined as the mean of the distances between all pixels and its nearest neighbor. The distance between two pixels is measured by the L_2 norm of the difference on attributes of these two pixels. Then we calculate the adjacency matrix of the constructed graph using the Gaussian kernel. This adjacency matrix is the final matrix of all EVD algorithm. Next, we drop the last 5% columns of the original hyperspectral image, and remove its corresponding rows and columns in the adjacency matrix. We then calculate the top K eigen-pairs of this truncated adjacency matrix to generate the input decomposition matrix \mathbf{Q}_{init} and $\mathbf{\Sigma}_{\text{init}}$. In our test K is set to 100. We summarize this EVD updating algorithm in Algorithm 5.5.

After getting the approximated EVD results \mathbf{Q} and $\mathbf{\Sigma}$ of the final matrix, we use them to do spectral clustering, Haar-like low-pass filtering and ideal low-pass filtering compare them with the ground truth results using the top K eigen-pairs of the full-sized adjacency matrix. For spectral

Table 5.3: Average clustering errors and low-pass filter errors on hyperspectral images

	IEVD-LR-FAST	EEU	TRIP-BASIC	TRIP
Number of Clustering Errors	70.5	71.5	1529.5	1875.6
L2 (Haar)	13820	13918	55670	62845
L2 (Ideal)	14756	14872	75525	84772

clustering results, the error performance is measured by the number of pixels that are incorrectly clustered. For low-pass filtering results, the error performance is measured by the L_2 norm between the filtered signals using approximated results and ground truth.

The average result is shown in Table. 5.3. Our proposed IEVD-LR-FAST algorithm has the lowest error in all three metrics, which demonstrate the accuracy of our proposed algorithm on hyperspectral images.

5.5 Conclusion

This chapter studies the EVD approximation algorithm for low rank matrices. More specially, we have proposed the basic basic IEVD-LR algorithm with the error correction branch to improve the estimation accuracy. We have also proposed a faster algorithm named IEVD-LR-FAST by introducing a five-matrices eigen-decomposition form into the model. Our proposed methods are easier to implement, and have shown robustness across both synthetic and real-world datasets. Our experimental results have demonstrated the efficacy on runtime and memory and the accuracy of the proposed method. This chapter have established IEVD-LR-FAST algorithm as an efficient tool to approximate EVD results on dynamic point clouds, motion capture data and hyperspectral images. In our future works, we shall extend this algorithm to process high-order tensors. The fast EVD algorithm on sparse symmetric matrices with high rank is another interesting direction for exploration.

Chapter 6

Summary and Future Direction

In this chapter, we summarize our contributions we have made in this dissertation, and propose certain potential directions related to the application of advanced graph approaches.

6.1 Summary of Key Findings

In this dissertation, we investigate the applications of advanced graph signal processing approaches on multimedia data and the low cost eigen-decomposition updating algorithm for dynamic graphs. These advanced graph signal processing approaches aiming to capture the underlying multilateral interactions between data points, with the restriction of limited computational power. The key findings addressed in this dissertation is summarized as follows.

- The ‘divide and conquer’ strategy proved effective in mitigating storage challenges, allowing for the processing of high-dimensional graphs in smaller, more manageable subsets. Through the systematic partitioning and processing of smaller subsets of data, our findings reveal that this approach not only successfully overcomes storage challenges but also significantly enhances the scalability and efficiency of the algorithms. The modular processing framework demonstrates promise as a viable solution for handling large and complex datasets in real-world applications.

- Algorithmic optimizations significantly reduced computational complexity of eigen-updating problem, especially for the case where the dimension of dynamic graph is increasing, enhancing the overall efficiency and scalability of high-dimensional graph processing. By refining algorithms with conditions derived from practical applications, our results indicate a substantial reduction in computational demands. This optimization makes the processing of multimedia datasets, including point clouds and motion capture data, more feasible with little compromising on accuracy. Our findings highlight the adaptability of high-dimensional graph processing algorithms to specific data characteristics, providing a valuable contribution to the field.
- The dissertation also aimed to bridge theoretical foundations with practical applications by applying high-dimensional graph processing algorithms to specific multimedia data types, such as point cloud data and motion capture data. Through extensive experimentation, our results showcase the successful application of advanced graph-based algorithms to these unique datasets. The adaptability and effectiveness of these algorithms underscore their potential for addressing challenges in multimedia data analysis, thereby contributing to the advancement of algorithmic exploration in these domains.

The key findings in our exploration of ‘divide and conquer’ strategies, optimization of computational complexity, and application of advanced graph-based algorithms to multimedia data contribute to a holistic understanding of high-dimensional graph processing. The integration of these findings lays the groundwork for more effective and efficient approaches to analyzing diverse and complex datasets in multimedia applications.

6.2 Future Directions

While our research makes contributions in addressing challenges, it is important to acknowledge its limitations. Factors such as dataset specificity and problem sensitivity highlight the need for continued refinement and adaptation in different kinds of multimedia datasets. Building on the

insights gained from this dissertation, there are several promising directions for future research:

- **Exploration of Advanced Graph based Processing on Other Multimedia Data** Based on our success on applying advanced graph-based processing algorithms to static point cloud in Chapter 3 and time-varying motion capture data in Chapter 4, our exploration can extend to further applications of high-dimensional graph processing techniques on diverse multimedia datasets with varying structures. The dynamic nature of certain multimedia systems, such as dynamic point clouds, has garnered increased attention in recent years. Compared with motion capture data, the number of data points in dynamic point clouds changes over time, introducing a new challenge in efficiently modeling and adapting our advanced graph-based processing algorithms to accommodate such dynamic systems. This expansion of our research scope allows us to contribute valuable insights to the broader area of high-dimensional graph processing in more general dynamic multimedia environments.
- **Algorithmic Optimization for Dynamic Multimedia scenarios** The IEVD-LR algorithm, proposed in Chapter 5, focused on the scenarios where the graph size increases over time. To extend our research focus beyond this specific context, future research directions may explore optimization strategies for more general contextual conditions. For example, in multimedia datasets such as dynamic graphs, the number of data points may decrease over time, which can not be directly solved by our proposed IEVD-LR algorithm. Additionally, most adjacency matrices of ϵ -neighborhood graph, one of the common type of graph constructed on multimedia datasets, are sparse. In our future work, we aim to consider both the sparse property of the adjacency matrix and the challenge posed by a decreasing graph size. This expansion of our research allows us to contribute an eigenvalue updating algorithm to a broader range of scenarios.
- **Algorithmic Optimization for High-dimensional Graphs** The IEVD-LR algorithm proposed in Chapter 5 primarily addressed the eigen-space updating problem for representation matrix of graphs. Another direction of extending the IEVD-LR algorithm

is to consider the singular space updating problem for representation tensor of high-dimensional graphs, initialized by orthogonal CP decomposition or higher-order singular value decomposition result. This expansion of our research allows us to integrate the low-complexity algorithm with the high-dimensional graph based processing algorithms, including those proposed in Chapter 3 and Chapter 4. Such integration will reduce the complexity obstacle of high-dimensional graph based processing algorithms on complex, large-scaled multimedia datasets.

Funding Acknowledgement

The material conducted in this dissertation was supported by the National Science Foundation under Grant No. 1824553, No. 2029027, No. 2029848 and No. 2009001.

Bibliography

- [1] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura and P. Vandergheynst, “Graph Signal Processing: Overview, Challenges, and Applications,” *Proc. IEEE*, vol. 106, no. 5, pp. 808-828, May 2018.
- [2] T.N. Kipf, and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2016, arXiv:1609.02907.
- [3] S.Zhang, M. Wang, S. Liu,P. Y. Chen, and J. Xiong, “Fast learning of graph neural networks with guaranteed generalizability: one-hidden-layer case,” in *International Conference on Machine Learning*, Nov. 2020, pp. 11268-11277.
- [4] W. Hu, J. Pang, X. Liu, D. Tian, C. -W. Lin and A. Vetro, “Graph Signal Processing for Geometric Data and Beyond: Theory and Applications,” *IEEE Transactions on Multimedia*, vol. 24, pp. 3961-3977, 2022.
- [5] R. Li et al., “Graph Signal Processing, Graph Neural Network and Graph Learning on Biological Data: A Systematic Review,” *IEEE Reviews in Biomedical Engineering*, vol. 16, pp. 109-135, 2023.
- [6] H. E. Egilmez and A. Ortega, “Spectral anomaly detection using graph-based filtering for wireless sensor networks,” in *Proceedings of 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, 2014, pp. 1085-1089.

- [7] J.D. Medaglia, W. Huang, E.A. Karuza, et al., “Functional alignment with anatomical networks is associated with cognitive flexibility,” *Nature Human Behaviour*, vol. 2, pp. 156-164, 2018.
- [8] W. Huang, T. A. W. Bolton, J. D. Medaglia, D. S. Bassett, A. Ribeiro and D. Van De Ville, “A Graph Signal Processing Perspective on Functional Brain Imaging,” *Proceedings of the IEEE*, vol. 106, no. 5, pp. 868-885, May 2018.
- [9] J. Pang, G. Cheung, A. Ortega and O. C. Au, “Optimal graph laplacian regularization for natural image denoising,” in *Proceedings of 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, South Brisbane, QLD, Australia, 2015, pp. 2294-2298.
- [10] U. von Luxburg, “A tutorial on spectral clustering,” *Stat. Comput.*, vol. 17, no. 4, pp. 395-416, 2007.
- [11] S. Zhang, S. Cui and Z. Ding, “Hypergraph Spectral Analysis and Processing in 3D Point Cloud,” *IEEE Trans. Image Process.*, vol. 30, pp. 1193-1206, 2021.
- [12] S. Zhang, S. Cui and Z. Ding, “Hypergraph-Based Image Processing,” in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Abu Dhabi, United Arab Emirates, Oct. 2020, pp. 216-220.
- [13] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. Zhang, “Edge-aware point set resampling,” *ACM Trans. Graph.*, vol. 32, no. 1, pp. 1-12, Feb. 2013.
- [14] S. Zhang, Z. Ding and S. Cui, “Introducing Hypergraph Signal Processing: Theoretical Foundation and Practical Applications,” *IEEE Internet Things J.*, vol. 7, no. 1, pp. 639-660, Jan. 2020.
- [15] S. Barbarossa and M. Tsitsvero, “An introduction to hypergraph signal processing,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Shanghai, China, Mar. 2016, pp. 6425-6429.

- [16] S. Zhang, S. Cui and Z. Ding, "Hypergraph Spectral Clustering for Point Cloud Segmentation," *IEEE Signal Process. Lett.*, vol. 27, pp. 1655-1659, 2020.
- [17] S. Zhang, Q. Deng and Z. Ding, "Signal Processing over Multilayer Graphs: Theoretical Foundations and Practical Applications," *IEEE Internet of Things Journal*, Jul. 2023.
- [18] S. Zhang, Q. Deng and Z. Ding, "Image Processing via Multilayer Graph Spectra," 2021, arXiv :2108.13639.
- [19] S. Zhang, Q. Deng and Z. Ding, "Multilayer graph spectral analysis for hyperspectral images," *EURASIP Journal on Advances in Signal Processing*, vol. 1, no. 92, pp. 1-25, Oct. 2022.
- [20] Q. Deng, S. Zhang and Z. Ding, "Point Cloud Resampling via Hypergraph Signal Processing," *IEEE Signal Processing Letters*, vol. 28, pp. 2117-2121, 2021.
- [21] Q. Deng, S. Zhang and Z. Ding, "An Efficient Hypergraph Approach to Robust Point Cloud Resampling," *IEEE Transactions on Image Processing*, vol. 31, pp. 1924-1937, 2022.
- [22] Q. Deng, S. Zhang and Z. Ding, "Body Motion Segmentation via Multilayer Graph Processing for Wearable Sensor Signals," in preparation for IEEE Open Journal of Signal Processing.
- [23] Q. Deng, Y. Zhang, M. Li, S. Zhang and Z. Ding, "Efficient Eigenvalue Decomposition for Low-Rank Symmetric Matrices in Graph Signal Processing: An Incremental Approach" in preparation for IEEE Transactions on Signal Processing.
- [24] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs: frequency analysis," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3042-3054, Jun., 2014.
- [25] A. Sandryhaila, and J. M. F. Moura, "Discrete signal processing on graphs: graph filters," in *Proceedings of 2013 IEEE ICASSP*, Vancouver, Canada, May 2013, pp. 6163-6166.

- [26] T. Kong, Y. Tian and H. Shen, "A Fast Incremental Spectral Clustering for Large Data Sets," in *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, Gwangju, Korea (South), 2011, pp. 1-5.
- [27] A. Banerjee, A. Char and B. Mondal, "Spectra of general hypergraphs," *Linear Algebra Appl.*, vol. 518, pp. 14-30, Apr. 2017.
- [28] A. Bretto, *Hypergraph theory: An introduction* (Mathematical Engineering). Cham, Switzerland: Springer, 2013.
- [29] A. Afshar *et al.* "CP-ORTHO: An orthogonal tensor factorization framework for spatio-temporal data," in *Proc. 25th ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Redondo Beach, CA, USA, Jan. 2017, pp. 1-4.
- [30] J. Pan, and M. K. Ng, "Symmetric orthogonal approximation to symmetric tensors with applications to image reconstruction," *Numer. Linear Algebra Appl.*, vol. 25, no. 5, p. e2180, Apr. 2018.
- [31] T. G. Kolda, "Orthogonal tensor decompositions," *SIAM J. Matrix Anal. Appl.*, vol. 23, no. 1, pp. 243-255, Jul. 2006.
- [32] S. Zhang, H. Zhang, H. Li and S. Cui, "Tensor-based Spectral Analysis of Cascading Failures over Multilayer Complex Systems," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Monticello, IL, USA, 2018, pp. 997-1004
- [33] L. De Lathauwer, B. De Moor and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253-1278, 2000.

- [34] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha and M. Beetz, "Towards 3D point cloud based object maps for household environments," *Robot. Auto. Syst.*, vol. 56, no. 11, pp. 927-941, Nov. 2018.
- [35] Z. C. Marton, R. B. Rusu and M. Beetz, "On fast surface reconstruction methods for large and noisy point clouds," in *Proc. IEEE Int. Conf. Robot. Autom.*, Kobe, Japan, May 2009, pp. 3218-3223.
- [36] R. Schnabel, S. Möser and R. Klein, "A Parallely Decodeable Compression Scheme for Efficient Point-Cloud Rendering," in *Proc. Eurograph. Symp. Point-Based Graph.*, Prague, Czech Republic, Sep. 2007, pp. 119-128.
- [37] S. Gumhold, X. Wang and R. S. MacLeod, "Feature Extraction From Point Clouds," in *Proc. IMR*, Newport Beach, USA, Oct. 2001, pp. 293-305.
- [38] R. Q. Charles, H. Su, M. Kaichun and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 77-85.
- [39] Z. Yang, Y. Sun, S. Liu and J. Jia, "3DSSD: Point-Based 3D Single Stage Object Detector," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 11037-11045.
- [40] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler and M. Pollefeys, "Semantic3D.Net: A new Large-scale Point Cloud Classification Benchmark," 2017, *arXiv:1704.03847*.
- [41] S. Chen, D. Tian, C. Feng, A. Vetro and J. Kovačević, "Fast Resampling of Three-Dimensional Point Clouds via Graphs," *IEEE Trans. Signal Process.*, vol. 66, no. 3, pp. 666-681, Feb. 2018.

- [42] Z. Chen, T. Zhang, J. Cao, Y. J. Zhang and C. Wang, "Point cloud resampling using centroidal Voronoi tessellation methods," *Comput.-Aided Des.*, vol. 102, pp. 12-21, Sep. 2018.
- [43] S. Orts-Escolano, V. Morell, J. García-Rodríguez and M. Cazorla, "Point cloud data filtering and downsampling using growing neural gas," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Dallas, TX, USA, Aug. 2013, pp. 1-8.
- [44] A. Anis, P. A. Chou and A. Ortega, "Compression of dynamic 3D point clouds using subdivisional meshes and graph wavelet transforms," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Shanghai, China, Mar. 2016, pp. 6360-6364.
- [45] B. Kathariya, A. Karthik, Z. Li and R. Joshi, "Embedded binary tree for dynamic point cloud geometry compression with graph signal resampling and prediction," in *Proc. IEEE Vis. Commun. Image Process. (VCIP)*, St. Petersburg, FL, USA, Dec. 2017, pp. 1-4.
- [46] S. Chen, D. Tian, C. Feng, A. Vetro and J. Kovačević, "Contour-enhanced resampling of 3D point clouds via graphs," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, New Orleans, LA, USA, Mar. 2017, pp. 2941-2945.
- [47] C. Weber, S. Hahmann and H. Hagen, "Sharp feature detection in point clouds," in *Proc. Shape Modeling Int. Conf.*, Aix-en-Provence, France, Jun. 2010, pp. 175-186.
- [48] D. Bazazian, J. R. Casas and J. Ruiz-Hidalgo, "Fast and Robust Edge Extraction in Unorganized Point Clouds," in *Proc. Int. Conf. Digit. Image Comput., Techn. Appl. (DICTA)*, Adelaide, SA, Australia, Nov. 2015, pp. 1-8.
- [49] A. K. Cherri, and M. A. Karim, "Optical symbolic substitution: Edge detection using Prewitt, Sobel, and Roberts operators," *Appl. Opt.*, vol. 28, no. 21, pp. 4644-4648, Nov. 1989.

- [50] M. Krivokuća, P. A. Chou, and P. Savill, “8i Voxelized Surface Light Field (8iVSLF) Dataset,” *document m42914, ISO/IEC JTC1/SC29 WG11(MPEG)* Ljubljana, Jul. 2018.
- [51] “ScanLAB Projects: Bi-plane point cloud dataset,” Accessed: Oct. 15, 2021. [Online]. Available: <https://www.epfl.ch/labs/mmspg/jpeg/>
- [52] M. Kazhdan, M. Bolitho and H. Hoppe, “Poisson surface reconstruction,” in *Proc. 4th Eurograph. Symp. Geometry Process.*, vol. 7, Jun. 2006, pp. 1-10.
- [53] H. Edelsbrunner and E. P. Mücke, “Three-dimensional alpha shapes,” *ACM Trans. Graph.*, vol. 13, no. 1, pp. 43-72, Jan. 1994.
- [54] A. X. Chang, *et al.*, “ShapeNet: An information-rich 3d model repository,” 2015, *arXiv:1512.03012*.
- [55] T. B. Moeslund and E. Granum, “A survey of computer vision-based human motion capture,” *Comput. Vis. Image Underst.*, vol. 81, no. 3, pp. 231-268, Mar. 2001.
- [56] Y. Desmarais, D. Mottet, P. Slangen and P. Montesinos, “A review of 3d human pose estimation algorithms for markerless motion capture,” *Comput. Vis. Image Underst.*, vol. 212, Nov. 2021.
- [57] J. Sedmidubsky, P. Elias, P. Budikova and P. Zezula, “Content-Based Management of Human Motion Data: Survey and Challenges,” *IEEE Access*, vol. 9, pp. 64241-64255, Apr. 2021.
- [58] R. Chereshevnev and A. Kertész-Farkas, “Hugadb: Human gait database for activity recognition from wearable inertial sensor networks,” in *Proc. Int. Conf. Anal. Images, Soc. Netw. Texts*, 2017, pp. 131-141.
- [59] L. Wang, Z. Ding and Y. Fu, “Low-Rank transfer human motion segmentation,” *IEEE Trans. Image Process.*, vol. 28, no. 2, pp. 1023-1034, Feb. 2019.

- [60] L. Xi, W. Chen, X. Wu, Z. Liu and Z. Li, "Online Unsupervised Video Object Segmentation via Contrastive Motion Clustering," *IEEE Trans. Circuits Syst. Video Technol.*, doi: 10.1109/TCSVT.2023.3288878.
- [61] X. Xu, L. Zhang, L. -F. Cheong, Z. Li and C. Zhu, "Learning Clustering for Motion Segmentation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 3, pp. 908-919, March 2022.
- [62] S. Lin, A. Yang, T. Lai, J. Weng and H. Wang, "Multi-motion Segmentation via Co-attention-induced Heterogeneous Model Fitting," *IEEE Trans. Circuits Syst. Video Technol.*, doi: 10.1109/TCSVT.2023.3298319.
- [63] F. Zhou, F. De la Torre and J. K. Hodgins, "Hierarchical Aligned Cluster Analysis for Temporal Clustering of Human Motion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 3, pp. 582-596, Mar. 2013.
- [64] G. Xia, H. Sun, L. Feng, G. Zhang and Y. Liu, "Human Motion Segmentation via Robust Kernel Sparse Subspace Clustering," *IEEE Trans. Image Process.*, vol. 27, no. 1, pp. 135-150, Jan. 2018.
- [65] A. Byravan and D. Fox, "SE3-nets: Learning rigid body motion using deep neural networks," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, 2017, pp. 173-180.
- [66] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," in *Proceedings of 2005 IEEE International Joint Conference on Neural Networks*, vol. 4, pp. 2047-2052, Aug. 2005.
- [67] S. Li, Y. A. Farha, Y. Liu, M. -M. Cheng and J. Gall, "MS-TCN++: Multi-Stage Temporal Convolutional Network for Action Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 6, pp. 6647-6658, Jun. 2023.

- [68] Z. Wang, Z. Gao, L. Wang, Z. Li, and G. Wu, “Boundary-Aware Cascade Networks for Temporal Action Segmentation,” in *Proceedings of European Conference on Computer Vision (ECCV)*, 2020.
- [69] Y. Li, Z. Dong, K. Liu, L. Feng, L. Hu, J. Zhu, L. Xu, S. Liu et al., “Efficient two-step networks for temporal action segmentation,” *Neurocomputing*, vol. 454, pp. 373-381, 2021.
- [70] H. Ahn and D. Lee, “Refining action segmentation with hierarchical video representations,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16302-16310.
- [71] L. Xu, Q. Wang, X. Lin and L. Yuan, “An efficient framework for few-shot skeleton-based temporal action segmentation,” *Computer Vision and Image Understanding*, vol. 232, pp. 103707, 2023.
- [72] S. Yan, Y. Xiong and D. Lin, “Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, pp. 1-9, Apr. 2018.
- [73] B. Filtjens, B. Vanrumste and P. Slaets, “Skeleton-Based Action Segmentation with Multi-Stage Spatial-Temporal Graph Convolutional Neural Networks,” *IEEE Transactions on Emerging Topics in Computing*, pp. 1-11, Dec. 2022.
- [74] P. Ghosh, Y. Yao, L. Davis, and A. Divakaran, “Stacked spatio-temporal graph convolutional networks for action segmentation,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.
- [75] K. Cheng, Y. Zhang, C. Cao, L. Shi, J. Cheng, and H. Lu, “Decoupling GCN with dropgraph module for skeleton-based action recognition,” in *Proceedings of European Conference on Computer Vision (ECCV)*, Aug. 2020, pp. 536-553.

- [76] S. Miao, Y. Hou, Z. Gao, M. Xu, and W. Li, "A central difference graph convolutional operator for skeleton-based action recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 7, pp. 4893-4899, 2021.
- [77] C. Wu, X.-J. Wu, and J. Kittler, "Graph2net: Perceptually-enriched graph learning for skeleton-based action recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 4, pp. 2120-2132, 2021.
- [78] Y. -H. Li, K. -Y. Liu, S. -L. Liu, L. Feng and H. Qiao, "Involving Distinguished Temporal Graph Convolutional Networks for Skeleton-Based Temporal Action Segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, doi: 10.1109/TCSVT.2023.3285416.
- [79] Y. Zhou, G. Gallego, X. Lu, S. Liu and S. Shen, "Event-Based Motion Segmentation With Spatio-Temporal Graph Cuts," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 4868-4880, Aug. 2023.
- [80] F. Grassi, A. Loukas, N. Perraudin and B. Ricaud, "A time-vertex signal processing framework: scalable processing and meaningful representations for time-series on graphs," *IEEE Transactions on Signal Processing*, vol. 66, no. 3, pp. 817-829, Feb. 2018.
- [81] J. S. Stanley, E. C. Chi and G. Mishne, "Multiway Graph Signal Processing on Tensors: Integrative Analysis of Irregular Geometries," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 160-173, Nov. 2020
- [82] J. B. Kim, H. S. Park, M. H. Park, and H. J. Kim, "A real-time region-based motion segmentation using adaptive thresholding and K-means clustering," in *AI 2001: Advances in Artificial Intelligence: 14th Australian Joint Conference on Artificial Intelligence Adelaide*, Australia, Dec. 2001, pp. 213-224.

- [83] F. Lauer and C. Schnorr, "Spectral clustering of linear subspaces for motion segmentation," in *2009 IEEE 12th International Conference on Computer Vision*, Kyoto, Japan, 2009, pp. 678-685.
- [84] B. Krüger, A. Vögele, T. Willig, A. Yao, R. Klein and A. Weber, "Efficient Unsupervised Temporal Segmentation of Motion Data," *IEEE Transactions on Multimedia*, vol. 19, no. 4, pp. 797-812, Apr. 2017.
- [85] Y. Bai, L. Wang, Y. Liu, Y. Yin, H. Di and Y. Fu, "Human Motion Segmentation via Velocity-Sensitive Dual-Side Auto-Encoder," *IEEE Transactions on Image Processing*, vol. 32, pp. 524-536, 2023.
- [86] X. Dong, D. Thanou, L. Toni, M. Bronstein and P. Frossard, "Graph Signal Processing for Machine Learning: A Review and New Perspectives," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 117-127, Nov. 2020.
- [87] M. Onuki, S. Ono, M. Yamagishi and Y. Tanaka, "Graph Signal Denoising via Trilateral Filter on Graph Spectral Domain," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 2, pp. 137-148, Jun. 2016.
- [88] Q. Deng, S. Zhang and Z. Ding, "An Efficient Hypergraph Approach to Robust Point Cloud Resampling," *IEEE Transactions on Image Processing*, vol. 31, pp. 1924-1937, Feb. 2022.
- [89] K. Pena-Pena, D. L. Lau and G. R. Arce, "t-HGSP: Hypergraph Signal Processing Using t-Product Tensor Decompositions," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 9, pp. 329-345, 2023.
- [90] S. Barbarossa and S. Sardellitti, "Topological Signal Processing Over Simplicial Complexes," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2992-3007, 2020.

- [91] L. Herda, P. Fua, R. Plankers, R. Boulic and D. Thalmann, "Skeleton-based motion capture for robust reconstruction of human motion," in *Proceedings Computer Animation 2000*, Philadelphia, PA, USA, 2000, pp. 77-83
- [92] G. Xia, P. Xue, H. Sun, Y. Sun, D. Zhang and Q. Liu, "Local Self-Expression Subspace Learning Network for Motion Capture Data," *IEEE Transactions on Image Processing*, vol. 31, pp. 4869-4883, Jul. 2022.
- [93] C. Lea, M. D. Flynn, R. Vidal, A. Reiter and G. D. Hager, "Temporal Convolutional Networks for Action Segmentation and Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1003-1012, Jul. 2017.
- [94] Y. Abu Farha and J. Gall, "MS-TCN: Multi-stage temporal convolutional network for action segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3570-3579, Jun. 2019.
- [95] Q. Zhang, Y. Tian, T. Wang, F. Yuan and Q. Xu, "ApproxEigen: An approximate computing technique for large-scale eigen-decomposition," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Austin, TX, USA, 2015, pp. 824-830.
- [96] K. . -B. Yu, "Recursive updating the eigenvalue decomposition of a covariance matrix," *IEEE Transactions on Signal Processing*, vol. 39, no. 5, pp. 1136-1145, May 1991.
- [97] R. D. DeGroat and R. A. Roberts, "Efficient, numerically stabilized rank-one eigenstructure updating (signal processing)," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 2, pp. 301-316, Feb. 1990.
- [98] R. Bru, R. Canto, and A. M. Urbano, "Eigenstructure of rank one updated matrices," *Linear Algebra and its Applications*, vol. 485, pp. 372-391, 2015.

- [99] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. Huang, “Incremental spectral clustering with application to monitoring evolving blog communities,” in *SIAM International Conference on Data Mining*, Citeseer, 2007.
- [100] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang, “Incremental spectral clustering by efficiently updating the eigen-system,” *Pattern Recognit.*, vol. 43, no. 1, pp. 113-127, 2010.
- [101] C. Dhanjal, R. Gaudel, and S. Cl  men  on, “Efficient eigen-updating for spectral graph clustering,” *Neurocomputing*, vol. 131, pp. 440-452, May 2014.
- [102] J. T. Kwok and H. Zhao, “Incremental eigen decomposition,” in *Proc. ICANN*, Istanbul, Turkey, Jun. 2003, pp. 270-273.
- [103] C. Chen and H. Tong, “Fast eigen-functions tracking on dynamic graphs,” in *Proc. SIAM Int. Conf. Data Mining*, 2015, pp. 559-567.
- [104] G. W. Stewart and J.-G. Sun, *Matrix Perturbation Theory*. Boston, MA, USA: Academic, 1990.
- [105] S. Chen, B. Liu, C. Feng, C. Vallespi-Gonzalez and C. Wellington, “3D Point Cloud Processing and Learning for Autonomous Driving: Impacting Map Creation, Localization, and Perception,” *IEEE Signal Processing Magazine*, vol. 38, no. 1, pp. 68-86, Jan. 2021.
- [106] W. Zhu, Z. Ma, Y. Xu, L. Li and Z. Li, “View-Dependent Dynamic Point Cloud Compression,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 2, pp. 765-781, Feb. 2021.
- [107] D. Rempe, T. Birdal, Y. Zhao, Z. Gojcic, S. Sridhar, and L. J. Guibas, “CaSPR: Learning canonical spatiotemporal point cloud representations,” in *Proc. Conf. Neural Inf. Process. Syst.*, 2020, pp. 13688-13701.
- [108] A. M. Bronstein, M. M. Bronstein, and R. Kimmel, “Numerical geometry of non-rigid shapes,” Springer, 2008.

- [109] A. M. Bronstein, M. M. Bronstein, A. M. Bruckstein, and R. Kimmel, "Analysis of two-dimensional non-rigid shapes," *Intl. J. Computer Vision (IJCV)*, vol. 78, no. 1, pp. 67-88, June 2008.
- [110] V. Lodhi, D. Chakravarty, and P. Mitra, "Hyperspectral imaging system: Development aspects and recent trends," *Sensing and Imaging*, vol. 20, pp. 1-24, 2019.
- [111] C. Jiang, D. P. Paudel, D. Fofi, Y. Fougerolle and C. Demonceaux, "Moving Object Detection by 3D Flow Field Analysis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 1950-1963, April 2021.
- [112] D. Wang et al., "Separating tree photosynthetic and non-photosynthetic components from point cloud data using dynamic segment merging," *Forests*, vol. 9, no. 5, p. 252, 2018.
- [113] R. Watanabe, K. Nonaka, E. Pavez, T. Kobayashi and A. Ortega, "Graph-Based Point Cloud Color Denoising with 3-Dimensional Patch-Based Similarity," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Rhodes Island, Greece, 2023, pp. 1-5.
- [114] X. Yu, W. Liu, and W. Xing, "Behavioral segmentation for human motion capture data based on graph cut method," *J. Vis. Lang. Comput.*, vol. 43, pp. 50-59, Dec. 2017.
- [115] B. Girault, A. Ortega and S. S. Narayanan, "Irregularity-Aware Graph Fourier Transforms," *IEEE Transactions on Signal Processing*, vol. 66, no. 21, pp. 5746-5761, 1 Nov.1, 2018.
- [116] Y. Zhao, Y. Yuan, and Q. Wang, "Fast spectral clustering for unsupervised hyperspectral image classification," *Remote Sens.*, vol. 11, no. 4, pp. 399, Feb. 2019.
- [117] Q. Wang, Y. Miao, M. Chen and Y. Yuan, "Spatial-Spectral Clustering With Anchor Graph for Hyperspectral Image," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1-13, 2022.

- [118] Y. Ding et al., “AF2GNN: Graph convolution with adaptive filters and aggregator fusion for hyperspectral image classification,” *Inf. Sci.*, vol. 602, pp. 201-219, Jul. 2022.
- [119] M. F. Baumgardner, L. L. Biehl, and D. A. Landgrebe, “220 Band AVIRIS Hyperspectral Image Data Set: June 12, 1992 Indian Pine Test Site 3,” *Purdue University Research Repository*, 2015.