

Spatio-temporal Segmentation Based Adaptive Compression of Dynamic Mesh Sequences

Xiaogang Jin, Hyewon Seo, Guoliang Luo, Zhigang Deng, Xin Zhao, Jin Xiaogang, Wei Zeng, Wenqiang Xie

▶ **To cite this version:**

Xiaogang Jin, Hyewon Seo, Guoliang Luo, Zhigang Deng, Xin Zhao, et al.. Spatio-temporal Segmentation Based Adaptive Compression of Dynamic Mesh Sequences. ACM Transactions on Multimedia Computing, Communications and Applications, 2020, 10.1145/3377475 . hal-03018359

HAL Id: hal-03018359
https://hal.science/hal-03018359

Submitted on 22 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Spatio-temporal Segmentation based Adaptive Compression of Dynamic Mesh Sequences

GUOLIANG LUO*, East China Jiaotong University, China
ZHIGANG DENG*, University of Houston, USA
XIN ZHAO, East China Jiaotong University, China
XIAOGANG JIN, State Key Lab of CAD&CG, Zhejiang University, China
WEI ZENG and WENQIANG XIE, Jiangxi Normal University, China
HYEWON SEO, University of Strasbourg, France

With the recent advances of data acquisition techniques, the compression of various dynamic mesh sequence data has become an important topic in computer graphics community. In this paper, we present a new spatio-temporal segmentation-based approach for the adaptive compression of the dynamic mesh sequences. Given an input dynamic mesh sequence, we first compute an initial temporal cut to obtain a small subsequence by detecting the temporal boundary of dynamic behavior. Then, we apply a two-stage vertex clustering on the resulting subsequence to classify the vertices into groups with optimal intra-affinities. After that, we design a temporal segmentation step based on the variations of the principle components within each vertex group prior to performing a PCA-based compression. Furthermore, we apply an extra step on the lossless compression of the PCA bases and coefficients to gain more storage saving. Our approach can adaptively determine the temporal and spatial segmentation boundaries in order to exploit both temporal and spatial redundancies. We have conducted extensive experiments on different types of 3D mesh animations with various segmentation configurations. Our comparative studies show the advantages of our approach for the compression of 3D mesh animations.

CCS Concepts: • **Computing methodologies → Computer graphics**; *Animation.*

Additional Key Words and Phrases: dynamic mesh sequences, animation compression, adaptive spatio-temporal segmentation, data compression

1 INTRODUCTION

With the rapid advancements of various technologies for 3D mesh animation acquisition, 3D mesh animation data is becoming another popular media type. While users can capture and generate 3D mesh animations with various tools, the amount of 3D mesh animation data has also been increasing. As a relatively new type of the growing data, many research efforts on the processing and analysis of 3D mesh animations have been conducted in recent years. Among them, compression is one of the key techniques for the storage, transfer, and display of 3D mesh animation data towards broad applications.

To date, researchers have developed a variety of efficient compression techniques for 2D video such as MPEG, H.263, and H.265. However, these compression methods for 2D video cannot be directly applied to 3D mesh animations due to the fundamental structure and representation differences between 2D video and 3D mesh animation data. For example, 3D shapes are often highly sensitive to vertex outliers, while an irregular pixel in an image may not be even visually noticeable. Therefore, with the strict requirements of 3D shape quality, efficient compression of 3D mesh animation data has become an increasingly important research topic.

*Corresponding authors.

Authors' addresses: Guoliang Luo, luoguoliang@ecjtu.edu.cn, East China Jiaotong University, 808 Shuanggang East AV., Nanchang, China; Zhigang Deng, University of Houston, USA, zdeng4@uh.edu; Xin Zhao, East China Jiaotong University, China; Xiaogang Jin, State Key Lab of CAD&CG, Zhejiang University, China; Wei Zeng; Wenqiang Xie, Jiangxi Normal University, China; Hyewon Seo, University of Strasbourg, France.

The key information of a 3D mesh animation is its dynamic behavior, which drives the deformations of different mesh surfaces. As reported in existing literature, we can achieve a better performance on the compression of 3D mesh animations with repetitive motions or rigid mesh segments, which contain significant redundancies either temporally or spatially [30, 47, 53]. Therefore, it is important to exploit the dynamic behaviors based on both spatial and temporal segmentation within a 3D mesh animation for effective data compression. However, due to the high complexity and the large data size, it remains a challenge to jointly explore the spatial and temporal segmentation to further improve the performance of compressing 3D mesh animations.

In this paper, we propose an adaptive spatio-temporal segmentation based model for the compression of 3D mesh animations. Specifically, we first introduce a *temporal segmentation* scheme that explores the temporal redundancy by automatically determining the optimal temporal boundaries. Then, we also introduce a novel *two-stage vertex clustering* approach to explore the spatial redundancy by automatically determining the number of the vertex groups with optimal intra-affinities. Based on the above adaptive spatio-temporal segmentation schemes, we develop a full scheme of its application for the compression of 3D mesh animations. Through many experiments, we show the effectiveness and efficiency of our approach compared to the state of the art mesh animation compression algorithms.

The contributions of this work can be summarized as follows:

- We develop an adaptive spatio-temporal segmentation approach for 3D mesh animations by exploiting the spatial and the temporal redundancies simultaneously based on the dynamic behaviors.
- We propose a compression model for 3D mesh animations by coupling with the novel adaptive spatio-temporal segmentation. Through extensive experiments as well as direct comparisons with state-of-the-art methods, we show the effectiveness and efficiency of our compression model.

The remainder of the paper is organized as follows. We first review previous and related works on the compression of 3D mesh animations in Section 2. In Section 3, we briefly give the overview of our compression method. Then, we present the details of our spatio-temporal segmentation model and its application to the compression of 3D mesh animations in Section 4. The experimental results by our approach with comparative studies are shown in Section 5. Finally, we conclude this work in Section 6.

2 RELATED WORKS

While motion capture data is becoming important in many areas including graphics, visualization, gaming, and medical applications, compression of motion capture data has been thoroughly studied using various techniques of wavelets [5], quadratic Bézier curve fitting [28], model-based indexing [6], motion pattern-based indexing [17], etc. To achieve a high efficiency, Kwak et al. proposed a hybrid scheme that is similar to hybrid video encoders, which contains predictive coding, DCT transform, quantization, and entropy coding steps [29]. Furthermore, Firouzmanesh et al. incorporate the factor of attention simulation in the model for fast compression [13]. In the work of [57], Wang et al. proposed a novel Alpha Parallelogram Predictor with context-based arithmetic coding to correct the predictions for the lossless compression of motion captured data. However, existing compression methods for motion captured data cannot be directly applied to compress dynamic mesh sequences due to significantly more intensive spatial redundancies of the dynamic meshes.

The compression of dynamic mesh sequences has been a persistent research topic in the past several decades. In [39], Maglo et al. classify existing 3D mesh animation compression methods into five different categories: prior segmentation based, PCA-based, spatio-temporal prediction based, wavelets based, and the MPEG framework based. In this paper, we group the existing methods into two general types: *non-segmentation based methods* and *segmentation based methods*. Below we first review state-of-the-art 3D mesh segmentation methods. Then, we focus on the non-segmentation based compression methods, and the segmentation based compression methods including spatial segmentation based methods and temporal segmentation based methods.

3D mesh segmentation: Au et al. [3] present a 3D mesh skeleton extraction method by mesh contraction while preserving the shape of the contracted mesh and the original topology. This method generates the skeleton-vertex correspondence, which automatically leads to the spatial segmentation of a 3D mesh. Ren et al. [43] present a region-growing based image segmentation, which gradually merges neighboring small regions based on region weights and boundary features. Hierarchical segmentation algorithms can also be applied for 3D meshes. Tsuchie et al. [50] present a 3D scanned mesh segmentation by using the Student-t mixture model to cluster the 3D vertices, which are featured with 3 principals, i.e., curvatures, coordinates, and normals. Nowadays, 3D mesh datasets are increasing available due to the advancement of data acquisition techniques, which enable us to apply advanced deep learning algorithms and tools for 3D model segmentation [9]. Kalogerakis et al. [25] present a learning based 3D mesh segmentation framework, where the conditional random field model based object functions are used to evaluate the consistency of triangle labels and the consistency among their neighborhoods. George et al. [14] present an active learning based framework to predict the 3D mesh segmentation, which couples with an quality measurement step to suggest the ordering of the segments to ensure high quality segmentation results. George et al. [15] present a robust conformal factor for convolutional neural networks (CNN) to improve efficiency of learning based methods for 3D mesh segmentation. While most of the existing 3D mesh segmentation methods aim to obtain smooth and finely shaped boundaries for functional or semantic parts, in this work, we focus on the compression and thus do not have strict requirements on segmentation boundary shapes.

Non-segmentation based compression: Among the existing methods, a large portion of the methods take a matrix form of the 3D mesh animation, on which many of classical data compression methods and algorithms can be applied, including Principal Component Analysis (PCA) [2, 22, 35], linear prediction encoders [26, 46, 47, 61], wavelet decomposition [18, 41], and the Moving Picture Experts Group (MPEG) framework [40]. PCA is a classical method that can decompose a large matrix as the product of two much smaller matrices, with minimal information loss. Following the work of [2], Lee et al. [32] apply PCA to 3D mesh animation data after removing its rigid transformations. Later, researchers have used the linear prediction theory to further encode the resulting coefficients from PCA [26, 52, 52, 54]. Similarly, researchers have proposed a Laplacian-based spatio-temporal predictor [53] or curvature-and-torsion based analysis [60] to encode the vertex trajectories for dynamic meshes. Moreover, Liu et al. [35] use a subspace optimization technique to accelerate the PCA iterations, and Hou et al. [22] formulate PCA to an optimization problem with constraints on the orthogonality and solve the problem with an inexact augmented Lagrangian multiplier method. The above non-segmentation compression methods improve either the efficiency or the effectiveness of PCA-based 3D mesh animation compression. However, they assume an entire sequence as the given input, and do not explicitly exploit the dynamic behaviors enclosed in the input animation.

Segmentation based compression: The key information of a 3D mesh animation is its enclosed dynamic behavior; therefore, it is important to exploit the dynamic behavior coherence in 3D mesh animations for effective compression, using either spatial segmentation or temporal segmentation methods.

Spatial segmentation based compression: The key to spatial segmentation of a 3D mesh animation is to understand its semantic behaviors, e.g., the body and limb movements of human actions. Many previous methods have been proposed to compute the spatial segmentation for 3D mesh animations, which can generate different spatial segmentation schemes for animations with different motions. These spatial segmentation results can be useful for the skeleton extraction/rigging for animation generation [11, 24, 27, 31] and semantic representation of the dynamic meshes towards shape similarity measurement [1, 33, 58], etc.

The spatial segmentation is also useful to reveal the spatial redundancies for the compression of the 3D mesh sequences. Hijiri et al. [20] separately compress the vertices of each object with the same movements to obtain an overall optimal compression rate. In order to adapt spatial segmentation for compression, Sattler et al. [44] proposed an iterative clustered PCA method to group the vertex trajectories that share similar Principal Component (PC) coefficients and then further compress each cluster separately. The main limitation is its heavy

computational cost. Similarly, Ramanathan et al. [42] compute the optimal vertex clustering for the optimal compression ratio. However, all the above methods assume the entire animation has been given at the beginning.

Temporal segmentation based compression: The objective of temporal segmentation is mainly to chop a 3D mesh animation into sub-sequences, each of which represents different dynamic behavior. Temporal segmentation has been exploited for the compression of motion capture data [16, 17, 44, 63], but the efficiencies of these methods for 3D mesh animation compression may be significantly decreased since 3D mesh surfaces typically have much more dense vertices and additional topology than motion capture data [38]. Given a mesh sequence, after partitioning the sequence into clusters with similar poses, researchers either apply PCA to compress each group to achieve the optimal compression ratio [36] or extract a key-frame of each cluster and encode the rest frames as the blending weights of the extracted key-frames [19]. Similarly, in [7], Chen et al. apply the manifold harmonic bases to characterize the primary poses (key-frames) and the deformation transfer technique to recover the geometric details of each frame within a cluster. This reduce the storage since only a small number of the key-frames and a few coefficients would be needed for animation decompression. Yang et al. [59] group the temporal frames with their motion trajectory changes, and then apply the spectral graph wavelet transform block encoding to convert the dynamic mesh sequence into a multi-resolution representation for the progressive streaming of the mesh sequence. Recently, Lalo et al. [30] proposed an adaptive Singular Value Decomposition (SVD) coefficient method for 3D mesh animation compression. They first divide a mesh sequence into temporal blocks of the same length and treat the first block with SVD. Then, the following blocks are treated with the adaptive bases from the previous block without solving the full SVD decomposition for each block, which reduces the computing time.

In summary, spatial and temporal segmentations can reveal the spatial and temporal redundancies within 3D mesh animations, respectively, which aid the development of effective compression algorithms.

To take the advantages of both spatial and temporal segmentations, the new compression scheme for 3D mesh animations presented in this work simultaneously exploits both spatial and temporal redundancies.

3 OVERVIEW OF OUR APPROACH

In general, dynamic mesh sequences mainly have two different forms: time-varying meshes and deforming meshes. A time-varying mesh may have different numbers of vertices and different topological connectivities at different frames, whereas a deforming mesh has a fixed topology across frames. Note that we can always compute the inter-frame vertex correspondence to convert a time-varying mesh into a deforming mesh [49]. For the sake of simplicity, we focus on the deforming mesh data in this work.

The following is a list of key notations used in this paper:

- γ^{init}: The temporal range to detect dynamic behavior for the full 3D mesh.
- γ^{max}: The temporal range to detect dynamic behavior for a vertex group (spatial segment).
- τ^{init}: The temporal boundary frame for the *Initial Temporal Cut* being detected within the temporal range of γ^{init}, i.e., $\tau^{init} \leqslant \gamma^{init}$.
- τ: The temporal boundary frame for the *Temporal Segmentation* being detected within the temporal range of γ^{max}, i.e., $\tau \leqslant \gamma^{max}$.
- b: The varying frame index while computing the candidate frame boundaries.
- ω: Information persistence rate for Principal Component Analysis (PCA).
- w: Window size while scanning for the temporal boundaries.
- δ^i: The i-th vertex group (spatial segment).
- N_g: The total number of vertex groups (spatial segment).

We define the trigger conditions for the two important steps in our method. (1) *Initial Temporal Cut*: given the maximal length γ^{init} if any dynamic behavior has been detected in the mesh sequence (with no more than γ^{init}

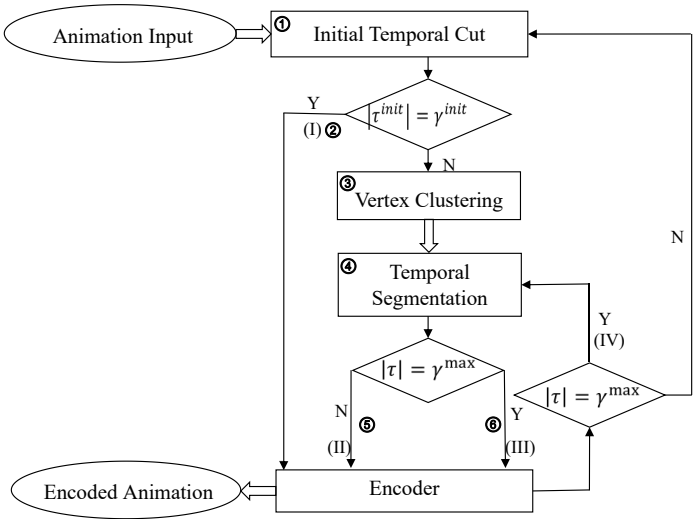


Fig. 1. Pipeline overview of our spatio-temporal segmentation scheme for compression. (I, II, III, and IV) are the 4 types of the segmented animation blocks, which are illustrated in Figure 2. Note that $|\tau^{init}|$ and $|\tau|$ denote the length of the detected initial temporal cut (see Section 4.1) and temporal segmentation (see Section 4.3), respectively; γ^{init} and γ^{max} denote the maximum range for the initial temporal cut and temporal segmentation, respectively. The numbers in circles correspond to the steps described in Section 3.

frames) (Section 4.1), and (2) *Temporal Segmentation*: given the maximal length γ^{max} if any dynamic behavior has been detected in any of the vertex groups (Section 4.2 and 4.3).

We briefly describe the pipeline of our segmentation scheme as follows. The algorithmic description is also shown in Figure 1.

①. We first conduct an *initial temporal cut* to produce a subsequence S by using a bi-directional boundary search algorithm. (Section 4.1).
②. If no distinct behaviors can be detected in S, i.e., the boundary frame $b = \gamma^{init}$, the subsequence S will be directly sent to the compressor (the case (I) in Figure 2). (Section 4.4)
③. Otherwise (i.e., distinct behaviors are detected in S), we perform *vertex clustering* on S for spatial segmentation. (Section 4.2)
④. Then, we continue to compute the temporal segmentation of each vertex group (spatial segment) within next γ^{max} frames, by observing the dynamic behaviors. (Section 4.3)
⑤. Through observing the number of Principle Components (PC), if we have detected distinct dynamic behaviors of any vertex group before γ^{max} is reached, the vertex trajectories of each group, up to the detected boundary frame, are sent to the compressor, separately (Section 4.4). After the compression, we repeat the process from step 1 (the case (II) in Figure 2).
⑥. Otherwise (i.e., we have not detected a temporal segmentation by γ^{max}), we also send the data of each vertex cluster to the compressor, separately (the case (III) in Figure 2). See Section 4.4. Afterwards, we reuse the previously obtained vertex clustering and continue observing the temporal segmentation in the remaining mesh frames. That is, we repeat the process from the step 4 for the remaining mesh frames (The case (IV) in Figure 2).

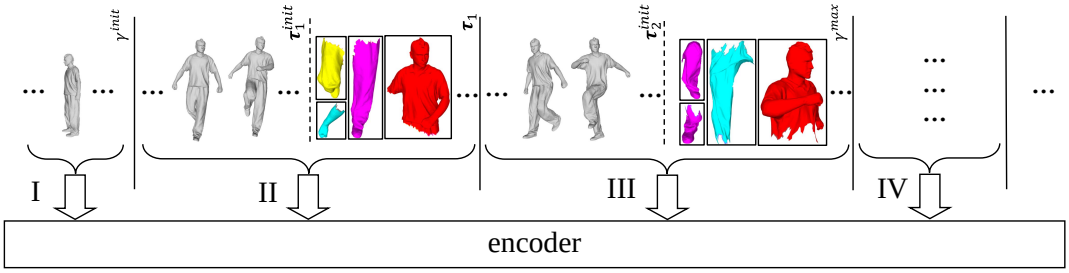


Fig. 2. 4 different types of mesh animation blocks sent to the encoder: (I) $|\tau^{init}| = \gamma^{init}$, (II) $|\tau^{init}| < \gamma^{init}$ and $|\tau| < \gamma^{max}$, (III) $|\tau^{init}| < \gamma^{init}$ and $|\tau| = \gamma^{max}$, and (IV) direct temporal segmentation based on the previous vertex grouping results.

4 SPATIO-TEMPORAL SEGMENTATION FOR COMPRESSION

We first describe our spatio-temporal segmentation model that consists of the initial temporal cut (Section 4.1), vertex clustering (Section 4.2), and temporal segmentation (Section 4.3). Then, we apply spatio-temporal segmentation results for the compression of 3D mesh animations in Section 4.4. Finally, we discuss different situations while processing a continuous mesh sequence as the input in Section 4.5.

4.1 Initial Temporal Cut

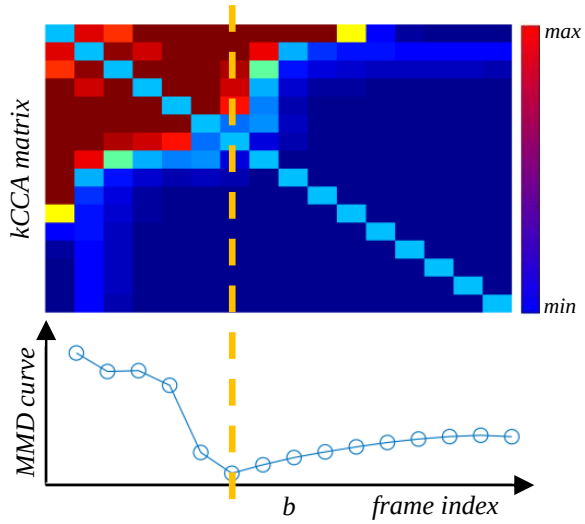


Fig. 3. An example of the initial temporal segmentation of the 'March' data, with the pairwise frame based kCCA matrix (Eq. 3) in the top panel and the MMN curve (Eq. 2) in the bottom panel. b is the detected boundary frame.

Let us denote a mesh animation as $(\{\mathbf{V}_i^f\},\ \mathbf{E})$, where $\mathbf{E}$ represents the connectivities among vertices, and $\mathbf{V}_i^f = (x_i^f, y_i^f, z_i^f)$ represents the 3D coordinates of the i-th vertex ($i = 1, \ldots, V$) at the f-th frame ($f = 1, \ldots, F$). Here V is the total number of vertices, and F is the number of frames of the animation sequence.

Given a mesh sequence, the objective of the initial temporal cut is to determine a boundary frame $\mathbf{V}^{|\tau^{init}|}$, so that the dynamic behavior in $[\mathbf{V}^1, \mathbf{V}^{|\tau^{init}|}]$ is distinctive from that in $[\mathbf{V}^{|\tau^{init}|+1}, \mathbf{V}^{\gamma^{init}}]$. To this end, we can formulate the initial temporal cut as the following optimization problem:

$$\min_{b \in [1, \gamma^{init}]} I([\mathbf{V}^1, \mathbf{V}^b], [\mathbf{V}^{b+1}, \mathbf{V}^{\gamma^{init}}]), \tag{1}$$

where b is a varying frame index and $I(\cdot, \cdot)$ computes the affinity between two mesh subsequences.

Available techniques for computing $I(\cdot, \cdot)$ can be classified into two categories: 1) front-to-end uni-direction boundary candidate search, and 2) bi-directional boundary candidate search. Although the uni-directional search method may have the advantage on efficiency, our bi-directional search method is more robust in detecting the temporal cut between two successive dynamic behaviors [4, 16]. Inspired by the kernelized Canonical Correlation Analysis (kCCA) approach [21, 45], and its successful application to semantic temporal cut for motion capture data [16], we formulate the initial cut to a Maximum-Mean Discrepancy (MMD) problem as follows:

$$\min_{b \in [1, \gamma^{init}-w]} -(H_1 - H_2 + H_3), \tag{2}$$

where $H_1 = \frac{1}{|T_1|^2} \sum_{i,j}^{|T_1|} K(\mathbf{v}^{b_i \to b_i+w}, \mathbf{v}^{b_j \to b_j+w}), H_2 = \frac{1}{|T_1||T_2|} \sum_{i}^{|T_1|} \sum_{j}^{|T_2|} K(\mathbf{v}^{b_i \to b_i+w}, \mathbf{v}^{b_j \to b_j+w}), H_3 = \frac{1}{|T_2|^2} \sum_{i,j}^{|T_2|} K(\mathbf{v}^{b_i \to b_i+w}, \mathbf{v}^{b_j \to b_j+w})$, and T_1 is the subsequence $[\mathbf{V}^1, \ldots, \mathbf{V}^b]$, T_2 is the subsequence $[\mathbf{V}^{b+1}, \ldots, \mathbf{V}^{\gamma^{init}-w}]$, w is a predefined parameter to ensure smooth kernels.

The kernel function in Eq. 2 is defined as follows:

$$K(\mathbf{v}^{b_i \to b_i+w}, \mathbf{v}^{b_j \to b_j+w}) = \exp(-\lambda \| \mathbf{v}^{b_i \to b_i+w} - \mathbf{v}^{b_j \to b_j+w} \|^2) \tag{3}$$

where λ is the kernel parameter for $K(\cdot)$ [51]. Due to the symmetric property of the kCCA, i.e., $K(\mathbf{A}, \mathbf{B}) = K(\mathbf{B}, \mathbf{A})$, we obtain a symmetric kCCA matrix for the animation block.

Finally, we can obtain a boundary frame $\mathbf{V}^{|\tau^{init}|}$ for the initial cut by solving the objective function in (Eq. 2). Note that $|\tau^{init}| = b + w$ due to the usage of a smoothing window. Meanwhile, we denote the detected initial temporal cut as $\tau^{\mathbf{init}}$. Figure 3 shows one of the initial cuts for a 3D mesh animation data, with $\gamma^{init} = 20$ and $w = 5$.

The complexity of the above bi-directional search for the initial temporal cut is $O(|\gamma^{init}|^2)$, which is less efficient than the uni-directional methods with $O(|\gamma^{init}|)$. However, in our context, we compute the initial temporal cut within a short mesh sequence $[1, \gamma^{init}]$, which is a small cost on the computation and thus will not cause notable delay to the overall compression framework. The settings of γ^{init} for different experimental data are presented in Table 1.

4.2 Vertex Clustering

In this section, we describe a vertex clustering (spatial segmentation) algorithm based on a two-stage, bottom-up hierarchical clustering algorithm to obtain optimal spatial affinities within segments.

4.2.1 Initial Vertex Clustering. After the initial temporal cut, τ^{init} is obtained; we then compute the vertex clustering based on the dynamic behaviors of different vertices. The pipeline of our approach is shown in Figure 4(I,II,III).

In this initial vertex clustering step, we first segment a dynamic mesh based on rigidity, which can be described as follows.

- *Compute the Maximal Edge-length Change (MEC) for all edge pairs.* Similar to [34, 58], we compute MEC within $|\tau^{init}|$ frames for each vertex pair, see Figure 4(I).
- *Binary labeling of vertices.* We fit the MEC of all the edges as an exponential distribution *epd*, see the top of Figure 4(I). Then, with the aid of the inverse cumulative distribution function of *epd*, we can determine

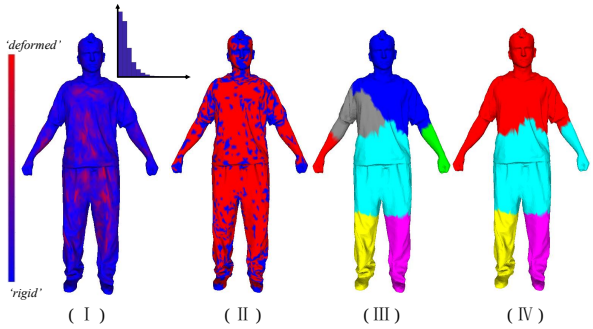


Fig. 4. Pipeline of the vertex clustering within an initial temporal cut of the '*March*' data: (I) Maximal Edge-length Change (MEC) for all the edge pairs and their distribution, (II) Binary labeling of vertices, (III) the rigid clusters resulted from the initial vertex clustering, and (IV) the rigid cluster grouping results.

a user-specified percent of the edges as the rigid edges ($\rho = 20\%$ in our experiments). Thus, the vertices that are connected to the rigid edges are called the *rigid vertices*, and the remaining vertices are called the *deformed vertices* in this work, see Figure 4(II).

- *Identify the rigid regions.* Based on the above binary labeling results, we merge the topologically connected rigid vertices into rigid regions, which become initial rigid vertex clusters. We also compute the center of each cluster as the average vertex trajectory among all vertices for each cluster.
- *Rigid clusters growing.* Starting with the above rigid clusters, we repeatedly merge the connected neighboring deformed vertices into the rigid cluster with the most similar trajectories, and update the center of the corresponding rigid cluster.

The initial vertex clustering is completed till every deformed vertex has been merged into a rigid cluster δ^i ($i = 1, \ldots, k$, k is the total number of the clusters), see Figure 4(III).

4.2.2 Rigid Cluster Grouping. In the second-stage vertex clustering, we further classify the rigid clusters to N_g groups with high internal affinities. In [44], Sattler et al. proposed an iterative clustered PCA based model for animation sequence compression. Inspired by this work, we design the second-stage vertex clustering by iteratively classifying and assigning each rigid cluster to the group with the minimal reconstruction error until the grouping remains unchanged. Since the iterative clustered-PCA is performed on the initial vertex cluster, it works very efficiently, unlike the heavy computation to perform on each individual vertex in [44].

The reconstruction error of a rigid cluster δ^j can be defined as follows:

$$\|\delta_j - \widetilde{\delta}_j\|^2 = \|\delta_j - (C[j] + \widehat{\delta}_j)\|^2, \tag{4}$$

where $\widetilde{\delta}_j$ is the reconstructed cluster by using PCA, $C[j]$ is the center of each group ($j = 1, \ldots, N_g$), and $\widehat{\delta}_j$ is the reconstruction by using the PCA components (see Eq. 5). Note that we have $C[j]$ in Eq. 4 because PCA contains the centering (mean subtraction) of the input data for the covariance matrix calculation.

Figure 4 illustrates the process of the rigid cluster grouping with the 'March' data. The full process of the rigid cluster grouping is also described in Algorithm 1. As an example result in Figure 4(IV), the relatively less moved rigid clusters, '*head*', '*chest*' and '*right-arm*', are classified into the same group. Note that we obtain large vertex groups because the input mesh are smooth on the surface (see Table 1), unlike the motion Capture data containing sparse vertex trajectories that may lead to small groups. Moreover, the computational cost for the initial vertex clustering presented in Algorithm 1 is relatively small because both the number of clusters k and the number of groups N_g are small.

Algorithm 1 Rigid Cluster Grouping

procedure RCGrouping(group number:N_g, clusters: $\delta^i, i = 1, \ldots, k$)
  for $i = 1 \rightarrow N_g$ **do** ▷ *group initialization*
    $L[\delta^i] \leftarrow i$ ▷ *initial group label*
    $C[i] \leftarrow \bar{\delta^i}$ ▷ *initial group center*
  end for
  while $L' \neq L$ **do**
    $L' \leftarrow L$
    for $i = 1 \rightarrow k$ **do** ▷ *k clusters*
      for $j = 1 \rightarrow N_g$ **do** ▷ *N_g groups*
        $d[j] = \|\delta^i - \widetilde{\delta^i_j}\|^2$ ▷ *Eq. 4*
      end for
      $L[\delta^i] = \min_j d[j]$ ▷ *assign δ^i to the closest group*
    end for
    update the group centers $C[i], \forall\ i \in [1\ N_g]$
  end while
end procedure

4.3 Temporal Segmentation

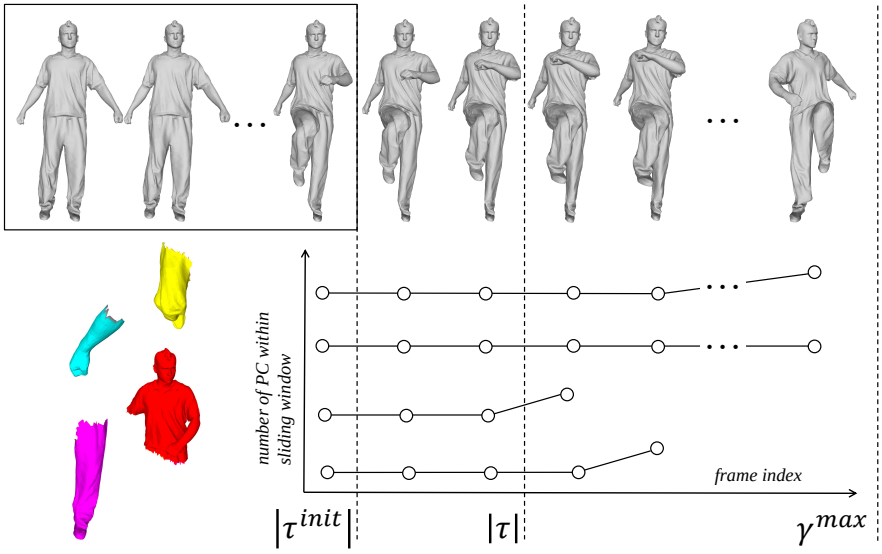


Fig. 5. Illustration of the temporal segmentation. The top row shows a sampled mesh sequence, with a bounding box as a sliding window. The size of the window is the length of the initial temporal cut, i.e., $|\tau^{init}|$. The bottom-left shows the vertex grouping of the initial temporal segment, and the bottom-right contains the change of the number of PCs for each vertex group in the sliding window. γ^{max} is the maximal possible delay, and $|\tau|$ is the detected temporal segmentation boundary.

After obtaining a set of the spatio-temporal segments $L(\delta)_j (j = 1, \ldots, N_g)$ for the initial temporal cut τ^{init}, we further introduce a temporal segmentation step as follows:

Algorithm 2 Temporal Segmentation

procedure TempSeg(mesh sequence: $\mathbf{V}^{1\rightarrow\gamma^{max}}$,initial cut:$\tau^{init}$)
  for $j = 1 \rightarrow N_g$ **do**
    $PC_{j_} = \mathbf{PCA}(\mathbf{V}_j^{1\rightarrow|\tau^{init}|})$ ▷ *#PCs of the j-th vertex group within* τ^{init}
  end for
  for $i = |\tau^{init}| + 1 \rightarrow \gamma^{max}$ **do**
    for $j = 1 \rightarrow N_g$ **do**
      $PC_j = \mathbf{PCA}(\mathbf{V}_j^{i\rightarrow i+|\tau^{init}|-1})$
      if $PC_j \neq PC_{j_}$ **then**
        break
      end if
    end for
  end for
  $b = i$ ▷ *boundary index*
end procedure

- For each vertex group, we stop observing the number of Principle Components (PC) once it is changed within the current sliding window. In this way, we can obtain a Num-of-PCs curve for each vertex group, see the bottom of Figure 5.
- To this end, similar to [26], the temporal segmentation boundary is determined as the first frame where any Num-of-PCs curve has changes, see the bottom-right of Figure 5.

The full description of the temporal segmentation is presented in Algorithm 2, with the complexity of $O(N_g\gamma^{max})$, where γ^{max} is a user-specified parameter that denotes the *maximal length of temporal segments*. Note that the computational cost of the PCA decomposition increases exponentially with the input data size. In order to balance the computational cost and the effectiveness of PCA, we set an adaptive γ^{max} for each of the input data, see Table 1. The detailed discussion of the involved parameters γ^{max} and the implementation of the algorithm are described below.

• *Maximal length of temporal segments*, γ^{max}. The computational cost of the PCA decomposition increases exponentially with the input data size. In order to balance between the computational cost and the effectiveness of PCA, we set an adaptable γ^{max} for each of the input data, see Table 1.

• *Parallel computing*. The temporal segmentation presented in Algorithm 2 is designed for each vertex group (spatio-temporal segment), and the vertex groups are independent of each other. Thus, we can implement the temporal segmentation for each vertex group in parallel. The computational time statistics in Table 1 show the efficiency improvement through parallelization.

4.4 Compression

After the above spatio-temporal segmentation, we apply PCA to compress each segment with a pre-defined threshold on the information persistence rate $\omega \in [0\ 1]$, which is used to determine the number of PCs to retain after the PCA decomposition.

• *Encoder*. For a vertex group $L(\delta)_j^i$, i.e., the j-th vertex group within the i-th temporal segment τ_i, we denote its compression as follows:

$$\widehat{\mathbf{X}} \overset{PCA}{\approx} \mathbf{A}_j^i \times \mathbf{B}_j^i, \tag{5}$$

where $\mathbf{X} = \mathbf{V}^{\tau_i}_{L(\delta)^i_j}$ for simplicity, $\mathbf{A}^i_j$ is the score matrix of dimensions $3|\mathbf{V}_{L(\delta)^i_j}| \times k^i_j$, $\mathbf{B}^i_j$ is the coefficient matrix of dimensions $k^i_j \times |\tau_i|$, and $\widehat{\mathbf{X}}$ denotes a centered matrix of $\mathbf{X}$ by subtracting the mean vectors $\overline{\mathbf{X}}$, i.e., $\widehat{\mathbf{X}} = \mathbf{X} - \overline{\mathbf{X}}$.

Boundary consistency. As can be seen in Figure 6(II), boundary inconsistencies may occur due to the independently chosen PCs from the two neighboring vertex groups. In order to alleviate the potential inconsistency along the boundary, we extend the range of each vertex group with η-ring neighbors ($\eta = 2$ in the middle and bottom of Figure 6(I)), and drop the extended η-ring region after the PCA decomposition. In this way, the PCA basis of each vertex group inherently encode the features of its extended neighbors, which can enhance the boundary consistency, see Figure 6(III).

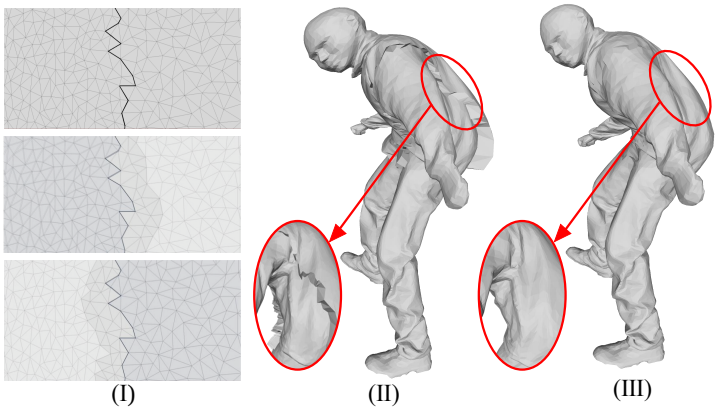


Fig. 6. Boundary consistency by overlapping the neighboring vertex groups. (I) Top: an example of segmentation boundary, Middle and Bottom: merge 2-ring neighbor vertices for each vertex group before sending to the encoder. (II) and (III) are the decoded 3D model without/with boundary consistency, respectively.

Bitstream encoding. As the final step of the compression pipeline, we choose the well-known fast lossless compression method ZLib [12], which combines the Huffman coding [23] and the LZ77 compression [64] that both can be approximated to the limits of information entropy [62]. More specifically, we first concatenate the PCA components into a bitstream XC_{bs}. Then, we apply ZLib for the encoding, i.e.,

$$X_{bs} = \mathbf{ZLib}(XC_{bs}). \tag{6}$$

• *Decoder.* We first apply the inverse of the ZLib method to uncompress for the PCA components, i.e.,

$$XC_{bs} = \mathbf{unZLib}(X_{bs}), \tag{7}$$

where **unZLib** is the corresponding decoder of the compression method **ZLib**. Then, with the uncompressed score matrix and the coefficient matrix, we can approximate each of the spatio-temporal segment by using the Eq. 5. Finally, we restore the original animation by concatenating the spatio-temporal segments in order.

Note that the complexity of both **ZLib** and **unZLib** is $O(n)$, where n is the length of the operated data [12].

4.5 Sequential Processing

As discussed in Section 3, our spatio-temporal segmentation scheme generates four possible animation blocks that are further sent to the encoder for compression (see Figure 2), which leads to four types of the sequential processing to the successive mesh sequence:

(I) $|\tau^{init}| = \gamma^{init}$. This indicates no distinct behavior has been detected at the initial temporal cut step (Section 4.1). In this case, the animation block $[\mathbf{V}^1, \mathbf{V}^{\gamma^{init}}]$ will be directly sent to the encoder. Moreover, we need to re-compute a spatio-temporal segmentation for the successive mesh sequence.

(II) $|\tau^{init}| < \gamma^{init}$ and $|\tau| < \gamma^{max}$. This indicates the vertex clustering has been conducted and a temporal segmentation boundary has been detected at $\mathbf{V}^{\tau}$. In this case, each vertex group of the animation block will be sent to the encoder, separately. Moreover, we will re-compute a spatio-temporal segmentation for the successive mesh sequence.

(III) $|\tau^{init}| < \gamma^{init}$ and $|\tau| = \gamma^{max}$. This indicates the vertex clustering has been conducted and a temporal segmentation boundary has not been detected within the range $[\mathbf{V}^1, \mathbf{V}^{\gamma^{max}}]$. In this case, each vertex group of the animation block will be sent to the encoder, separately. Moreover, we will only need to re-compute the *temporal segmentation* for the successive mesh sequence.

(IV) Otherwise, we can directly reuse the obtained (previous) vertex grouping results, compute the temporal segmentation, and then perform the PCA-based compression for each vertex group. If the new boundary $|\tau| < \gamma^{max}$, we will need to re-compute a spatio-temporal segmentation for the successive mesh sequence; otherwise (i.e., $|\tau| = \gamma^{max}$), it will again become the case (IV) for the successive mesh sequence.

5 EXPERIMENT RESULTS AND DISCUSSION

In this section, we first present the experimental data and the used evaluation metrics in Section 5.1. Then, we describe our experimental results in Section 5.2. In addition, we conducted comparative studies in Section 5.3. Given our spatio-temporal segmentation based compression scheme shown in Figure 1, both our approach and the comparative approaches were implemented with *Matlab*. All the experiments were performed on the same computer with Intel Core i5-6500 CPU @3.2GHz (4 cores) with 12G RAM. More results can be found in the supplemental demo video.

5.1 Experimental Setup

Table 1 shows the details of our experimental data. Among them,

- '*March*', '*Jump*' and '*Handstand*' were created by Daniel Vlasic at the Computer Graphics Group at MIT, through driving a 3D template with multi-view video [56].
- '*Horse*' was generated through deformation transfer by Robert W. Sumner when he was in MIT [48].
- '*Flag*' and '*Cloth*' are dynamic open-edge mesh sequences (Courtesy of Frederic Cordier from Université de Haute-Alsace) [10].
- 'Samba' is a dancing lady animation shared by Ing. Libor Váša from Department of Computer Science and Engineering at University of West Bohemia.

We applied the following metrics for quantitative analysis:

Bits per vertex per frame (bpvf). Similar to [8, 47], we also used bpvf to measure the performance of compression approaches. Note that we assume the vertex coordinates are originally stored as single-precision floating numbers, i.e., $8bits/Byte \times 4Bytes = 32$. Thus, after the PCA decomposition in Section 4.4, we can calculate the quantization of the basis and coefficients as follows:

$$Q = 32 \sum_{i,j} (3|\mathbf{V}_{L(\delta)^i_j}| \times k^i_j + k^i_j \times |\tau_i| + |\tau_i|), \tag{8}$$

where k^i_j denotes the number of the principal dimensions of the j-th vertex group within the i-th temporal segment. Finally, after the encoding of the PCA decomposition components, we can estimate the bpvf of our

Table 1. The results and performances by our model with different configurations of parameters: w and γ^{init} for the *Initial Temporal Cut* (Section 4.1), γ^{max} for the *Temporal Segmentation* (Section 4.3) and N_g for the vertex clustering (Section 4.2). s and s_p are the timings in seconds (unit) of the single-thread and paralleled implementations, respectively, % denotes the percentage of the time savings for each data, i.e., $100 \cdot (s - s_p)/s$. The last column s_d shows the decoding timings.

Animations	Parameters				Rate	Distortion(%)		Timing			
(V/F)	w	γ^{init}	γ^{max}	N_g	$bpvf$	$STED$	$KGError$	s	s_p	%	s_d
March	5	15	50	4	2.53	5.83	6.01	**75.69**	**70.08**	**7.41**	0.29
(10002/250)	5	20	50	4	2.44	**5.28**	5.98	93.72	88.2	5.89	**0.26**
	5	20	100	4	2.44	5.31	**5.91**	104.79	98.08	6.40	**0.26**
	5	20	50	8	**2.38**	5.44	6.15	97.15	92.79	4.49	0.28
Jump	5	15	50	4	4.00	**6.22**	7.07	**63.24**	**61.39**	2.93	**0.17**
(10002/150)	5	20	50	4	**3.94**	9.39	**6.58**	100.33	98.40	1.92	0.18
	5	20	100	4	**3.94**	9.39	**6.58**	101.17	96.95	**4.17**	0.19
	5	20	50	8	**3.94**	9.39	**6.58**	103.45	100.92	2.45	0.18
Handstand	5	15	50	4	2.38	9.00	5.20	**51.25**	**48.02**	**6.30**	0.20
(10002/175)	5	20	50	4	2.31	**7.63**	**5.09**	69.29	66.51	3.89	0.18
	5	20	100	4	**2.14**	8.07	5.22	70.06	66.61	4.92	**0.16**
	5	20	50	8	2.25	8.05	5.12	71.99	68.92	4.26	0.18
Horse	3	9	20	4	7.93	**4.38**	4.88	20.29	19.05	**6.11**	0.06
(8431/49)	3	12	20	4	**6.42**	4.61	**3.72**	**17.00**	16.21	4.65	**0.05**
	3	12	30	4	**6.42**	4.61	**3.72**	17.09	**16.07**	5.97	**0.05**
	3	12	20	8	7.31	4.52	4.21	22.43	21.55	3.92	0.06
Flag	10	30	100	4	0.87	**2.28**	7.89	**120.48**	**104.95**	**12.89**	0.62
(2750/1001)	10	40	100	4	0.79	2.63	7.89	195.13	178.64	8.45	**0.61**
	10	40	150	4	**0.73**	2.71	7.94	210.52	192.56	8.53	**0.61**
	10	40	100	8	0.79	2.67	**7.87**	198.25	180.23	9.09	0.69
Cloth	10	30	100	4	**0.54**	0.89	3.01	**14.49**	**13.19**	8.97	**0.03**
(2750/200)	10	40	100	4	0.63	**0.84**	1.95	31.64	29.13	7.93	**0.03**
	10	40	150	4	0.63	0.86	1.97	33.98	28.20	**17.01**	0.04
	10	40	100	8	0.63	0.90	**1.94**	32.40	28.95	10.47	**0.03**
Samba	5	15	50	4	**1.40**	**0.04**	**4.87**	**16.57**	**21.42**	29.34	0.13
(9971/175)	5	20	50	4	1.62	0.09	6.16	56.59	63.88	12.87	**0.11**
	5	20	100	4	1.43	0.09	6.17	53.97	73.50	**36.19**	**0.11**
	5	20	50	8	1.62	0.09	6.16	66.42	68.92	3.76	**0.11**

approach as follows:

$$bpvf = \frac{|X_{bs}|}{|XC_{bs}|} \cdot \frac{Q}{V \times F}, \tag{9}$$

where $|XC_{bs}|$ and $|X_{bs}|$ denotes the quantities of the bitstream before and after applying the lossless compressor **ZLib**, respectively.

Reconstruction errors. After compression, we can reconstruct the animation with the decoder described in Section 4.4. In order to measure the difference between the reconstructed animation and the original animation, we use two well-known metrics, namely, the *Spatiotemporal edge difference (STED)* error proposed by Váša et al. [55] and the *KGError* proposed by Karni et al. [26]. The STED error can be defined as the weighted spatial and

temporal errors as follows [55]:

$$STED = \sqrt{STED_s(d_-)^2 + c_-^2 \cdot STED_t(\omega_-, dt_-)^2}, \tag{10}$$

where d_- denotes the local spatial range, c_- is a weighting parameter, ω_- is the local temporal range and dt_- is the temporal distance value. We apply the default parameter settings based on the studies by Váša et al. in [55]. Moreover, the KGError can be defined as follows [26]:

$$KGError = 100 \cdot \frac{\|\mathbf{F} - \widehat{\mathbf{F}}\|_f}{\|\mathbf{F} - \mathbf{E}(\mathbf{F})\|_f}, \tag{11}$$

where $\|\cdot\|_f$ denotes the Frobenius norm, $\mathbf{F}$ and $\widehat{\mathbf{F}}$ are the original animation coordinates and the reconstructed animation coordinates ($3V \times F$), respectively. Furthermore, $\mathbf{E}(\mathbf{F})$ denotes the averaged centers of all the frames, and thus $\mathbf{F} - \mathbf{E}(\mathbf{F})$ denotes the center-subtracted animation.

5.2 Experimental Results

In this part, we present and discuss both the segmentation results and the compression results by our approach.

Spatio-temporal segmentation results. Figure 7 shows some samples of the spatio-temporal segmentation results of our experimental data (more results can be found in our supplemental materials). As can be seen in this figure, given the maximal number of spatial segments (groups) $N_g = 4$, our approach is able to automatically determine the optimal number of vertex groups (i.e., exploiting the spatial redundancy) for different dynamic behaviors (i.e., exploiting the temporal redundancy) for all the data. For example:

- The segmentation results of the '*March*', the '*Jump*', and the '*Handstand*' data are representatives of the local dynamic behaviors of different mesh regions. As can be clearly seen in Figure 7(6), our segmentation approach can not only determine the number of segments automatically, but also divide the mesh based on the local movements and group the disconnected regions with similar behaviors.
- The '*Cloth*' animation in Figure 7(2) is firstly segmented into 4 different highly deformed regions while dropping onto the table. Then, our approach generates 3 segments, i.e., 2 waving corner regions with deformed wrinkles and 1 relatively static large region.
- From the segmentation results of the '*Horse*' animation in Figure 7(3), we can observe the 4 *legs* are classified into the same group when moving towards the same direction; otherwise, they form different spatial groups. Similarly, the 'tail' is grouped with the 'trunk' region if the absence of distinct movements, or it is divided into two groups if bended.

Parallel computing. As described in Section 4.3, the temporal segmentation is applied on each vertex group independently, which can be accelerated through parallel computing. In our experiments, we implemented the temporal segmentation step with parallel computing on 4 cells. The computational time is shown in the column 's_p' in Table 1. Compared to the single thread implementation (column 's' in Table 1), the average efficiency has been improved by 9.94%, while it can be improved even up to 17.01% for the '*Cloth*' data. It is noteworthy that the decompression time 's_d' is less than 0.3s for all the experimental data. This is important for those applications that require a fast decompression such as bandwidth-limited animation rendering and display.

Compression results. Table 1 shows the different configurations of our spatio-temporal segmentation model for the compression of the experimental data ($\omega = 0.99$). For each of the data with different parameters, we highlight the best 'Rate', 'STED' 'KGError', and 'Timing' in bold fonts. We present and discuss the compression results in reference to the following different parameters:

- w. This parameter is a smoothing parameter for the initial temporal cut (Section 4.1). This parameter can be empirically chosen based on the target frame rate and the mesh complexity.

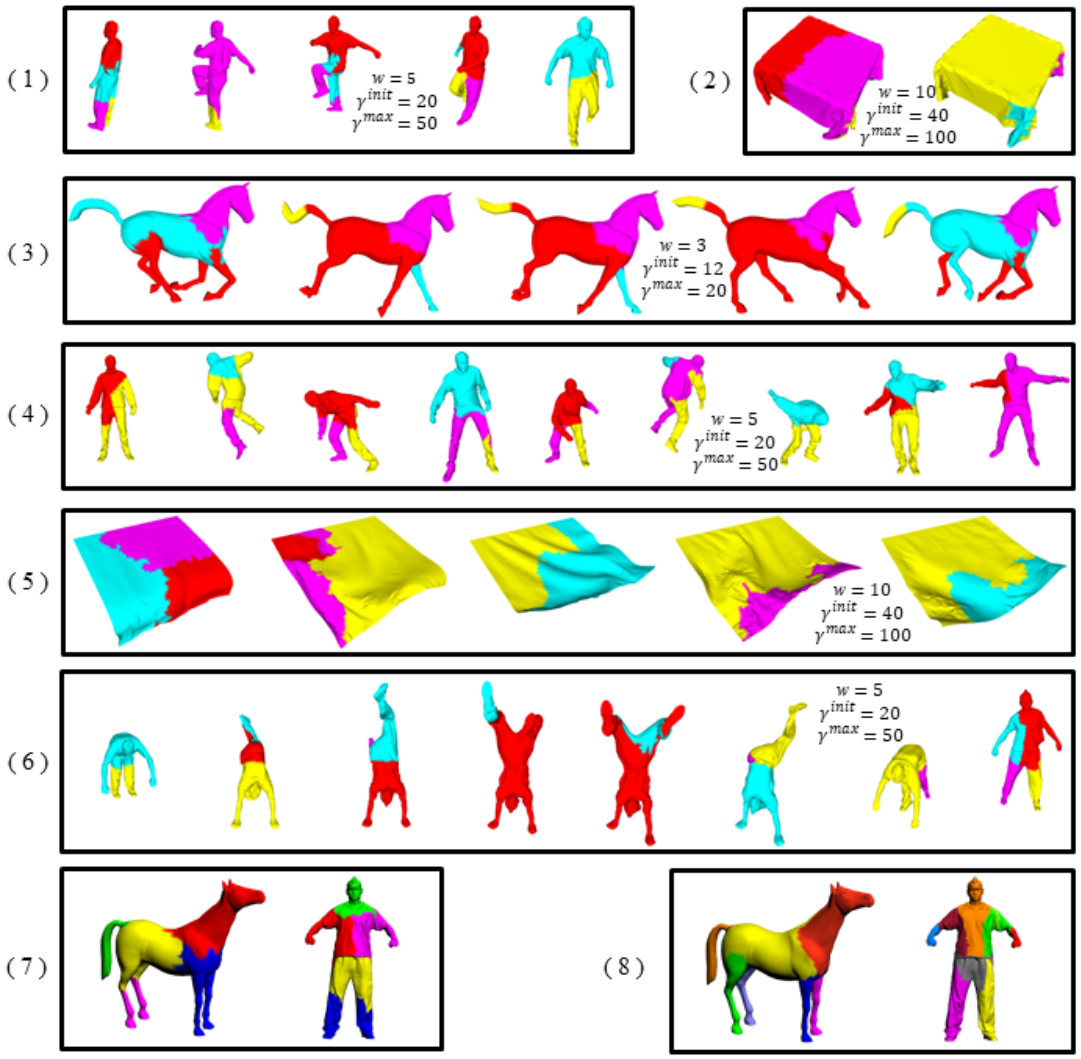


Fig. 7. The spatio-temporal segmentation results of the experimental data: (1) 'March', (2) 'Cloth', (3) 'Horse', (4) 'Jump', (5) 'Flag' and (6) 'Handstand'. The maximal possible number of the vertex groups $N_g = 4$ for all the data. (7)(8) are the segmentation results of 'Horse' and 'Handstand' data, by using Ren et al.'s [43] and Au et al.'s [3] methods, respectively. Note that colors only indicate the intra-segment (not inter-segment) disparities. See more results in the supplemental materials.

- γ^{init}. If we increase γ^{init} for the initial temporal cut, the computing time may be significantly increased since the time complexity of the initial temporal cut (Section 4.1) is $O(|\gamma^{init}|^2)$. On the other hand, its influence on the distortion is limited. Moreover, $bpvf$ tends to decrease for most of the experimental data (except the '*Cloth*' data).
- γ^{max}. As can be seen in Table 1, the change of γ^{max} does not significantly affect any of $bpvf$, distortion, and the computing time. This is because most of the temporal segmentation boundaries are found before

reaching γ^{max}. Note that γ^{max} is used in our compression scheme to simulate a block-by-block progressive compression. That is, in the case that a *Temporal Segmentation* (Section 4.3) boundary cannot be found in a long sequence, we send the scanned data for compression to avoid the compressor being idle for too long time (case (III) in Figure 2).

- N_g. By increasing N_g from 4 to 8, we do not observe the significant changes of the evaluation metrics. This is because our 2-stage vertex clustering can automatically converge to the optimal number of vertex groups. Moreover, the multi-thread implementation of our approach significantly improves the computational efficiency (see the 'Timing' column in Table 1). Therefore, in general N_g tends to be set to a small number. In fact, based on the previous studies [26, 36], N_g cannot be a big number because the bit rate will increase sharply due to the additional groups' basis. In our experiments, we empirically set $N_g = 4$ because our experimental computer has a CPU of 4 cores.

5.3 Comparative Studies

In this section, we first adapt the existing compression methods for the temporal block-wise compression and compare with our adaptive spatio-temporal segmentation based compression method in Section 5.3.1. We show the improvements of our method by comparing to the previous method presented in [37]. Then, in Section 5.3.2, we further demonstrate the effectiveness of our method by comparing to the recent advanced dynamic mesh compression methods based on the measurement metric STED [55]. Although the method in [53] also compresses an animation both spatially and temporally by using a Laplacian-based spatio-temporal predictor, they require the animation to be given in advance to compute an averaged pose.

5.3.1 Comparisons with the adapted methods. We compared our method with Sattler et al.'s method in [44], which is a non-sequential processing compression method. Additionally, we adopted the idea in [30] which cuts an animation into temporal blocks of the same size. Then, we can simulate the sequential processing of the existing compression methods, including Karni et al.'s method in [26] and the PCA-based methods, to compress each block in order. We call the adapted approaches as the 'Adapted Soft' and the 'Adapted PCA'. In order to make fair comparisons, the block size of the adapted methods is approximately set to the average of $|\tau|$ for each of the experimental data. Note that we have not included an 'Adapted Simple' method, which can be obtained by similarly adapting Sattler et al.'s method, into the comparison due to the extremely high computational cost of Sattler et al.'s method [44], which is unsuitable for sequential processing. More importantly, with the additional step on the lossless compression of the PCA bases and coefficients in Section 4.4, our method (red solid line in Figure 8) has been significantly improved, compared to our previous method in [37] (red dotted line in Figure 8).

Rate-Distortion curves (KG Error versus bpvf). Figure 8 shows the comparisons of an example between our method and the other methods. As can be seen with the KGError in the left of Figure 8, our method shows a significantly better performance than the adapted methods. That is, with the same $bpvf$ in the range of [2, 6.5], our method can always reconstruct the '*Cloth*' animation with a much smaller KG Error. Note that Karni et al.'s method has a better performance when $bpvf < 2$. This is because the 'Cloth' data contains a large portion of nearly static poses, which means the animation has significant temporal redundancies. Thus, the non-sequential precessing method by Karni et al. takes this advantage by treating the entire animation. However, our method runs much more efficiently: on average, 14.5 seconds consumed by our method, 32.5 seconds consumed by Karni et al.'s method, and 4421.9 seconds consumed by Sattler et al.'s method. Moreover, our method also provides a fine option for users who prefer high qualities after compression with slightly more storage cost, e.g., $bpvf > 2$.

5.3.2 Comparisons with the recent advanced methods. We also compared our method with several state-of-the-art animation compression methods, including the temporal block-wise method by Lalo et al. [30], the trajectory-prediction based methods by Váša et al. [53, 54], and the linear prediction based method by Karni et al. [26]. For

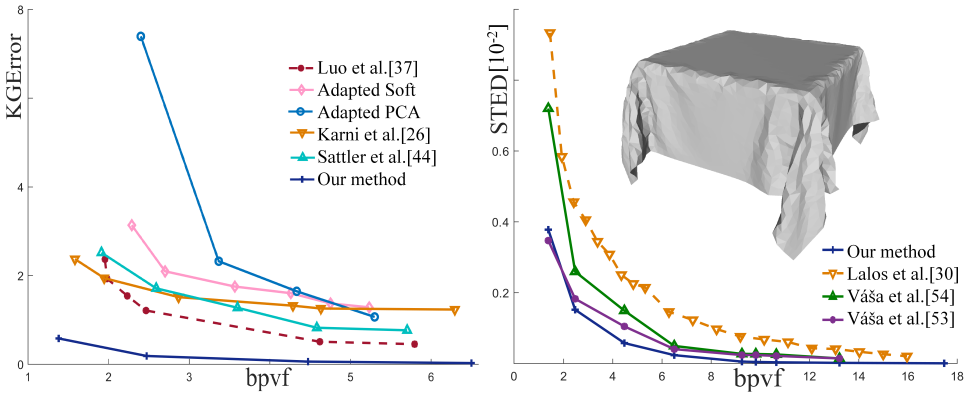


Fig. 8. **Left** shows the KGError comparisons on the 'Cloth' animation between our model and the previous model in I3D'19 [37] with the same specifications ($N_g = 4, \gamma^{init} = 40, \gamma^{max} = 100$), and the 'Adapted Soft' (block size = 100), the 'Adapted PCA' (block size = 100), Karni et al.'s method [26] and Sattler et al.'s method [44]. **Right** shows the STED error comparisons on the 'Cloth' animation between our method ($N_g = 4, \gamma^{init} = 40, \gamma^{max} = 80$) and the existing methods with the block size of 80.

the purpose of a fair comparison, we adapted the trajectory-prediction based method and the linear prediction based method to the block-wise compression with the block size of γ^{max}. The comparative results are described as follows.

Rate-Distortion curves (STED versus bpvf). The right of Figure 8 show the distortion comparisons of an example between our method and the other methods, summarized below.

- Compared to the block-wise method [30], our adaptive block-wise model shows better performances in the right of Figures 8. This is because our approach can automatically compute the adaptive block size and the number of vertex groups by exploiting both the temporal and the spatial redundancies.
- Compared to the trajectory-prediction based methods in [53, 54], our approach shows a better performance in the right of Figure 8. This is because our compression model utilizes the adaptive spatio-temporal segmentation, which automatically preserves the STED within the spatio-temporal segments.
- Regarding the computational time, our approach took **46 seconds**, while the trajectory-prediction based methods in [53, 54] took several minutes. Note that we have not included the time comparison with the method in [26] and [44], because they took hours of computation and returned higher reconstruction errors.

Reconstruction errors. Figure 9 shows the heat-map visualizations of the reconstruction errors using our approach and the other methods. Note that the heat-map is colored based on the per-vertex Euclidean distances. Overall, using our compression model, we can obtain lower distortions with a smaller bpvf. We describe the comparative results in details as follows:

- *Comparisons with the temporal block-wise method.* From the comparison between our method and the temporal block-wise method [30], the heat-map shows much lower per-vertex distortions on the fast-moving regions, e.g., swing hands of the 'March' data and the stretching legs/tails of the 'Horse' data. This is because our model exploits the spatial redundancy with a spatial segmentation within each temporal block, while the temporal block-wise method directly compresses the entire block.
- *Comparisons with the linear prediction based methods.* The linear prediction method in [26] first decomposes the animation data with PCA, and then achieves compression by applying the linear prediction analysis to

the decomposed components. Compared to [30], [26] can avoid high distortions on the fast-moving regions. However, this method can cause distortions in the relatively rigid regions due to the information loss by the linear predictors.

- *Comparisons with the trajectory-prediction based methods.* As can be seen from the 'Cloth' data in the right of Figure 9, the vertex distortions even occur on the rigid table-top surface of the mesh using the trajectory-prediction based compression methods [53, 54], as they do not explicitly constrain the spatial affinities, i.e., the spatial segmentation.
- *Our method.* Based on the above findings, our method avoids local extreme reconstruction errors using the specially-designed spatio-temporal segmentation to exploit both the spatial and the temporal redundancies. This advantage becomes more significant when periodically dynamic behaviors either spatially or temporally occur in the animation. In addition, our method runs much more efficiently, compared to the trajectory-prediction based compression methods [53, 54].

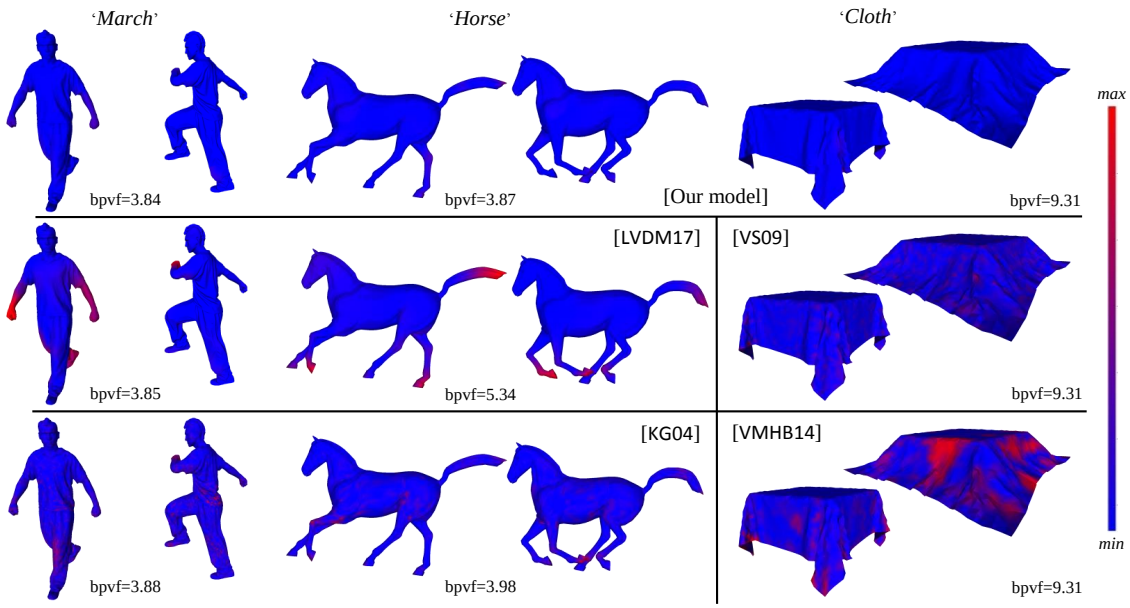


Fig. 9. The reconstruction errors of the compression by using our approach, Lalo et al.'s method in [30], Karni et al.'s methods in [26], Váša et al.'s method in [54] and in [53]. The colorbar indicates the reconstruction errors from low (blue) to high (red).

5.3.3 More comparisons with the spatial segmentation based methods. While there exist relatively few 3D mesh animation compression methods in the literature, we further adapt the well-known advanced spatial segmentation methods for comparative studies, including the skeleton extraction based [3] and the hierarchical region-growing segmentation method [43]. Note that we did not choose to compare our method with the recent learning based segmentation methods, because:

- On the one hand, we have a limited amount of 3D mesh animation data, which may be insufficient to well train learning based methods;

- On the other hand, many of the learning based methods require manual labelling and computing the correspondences among the training meshes, which requires extra human operations that may not lead to ideal segmentation results due to human interventions.

Segmentation results. Figure 7-(7) and Figure 7-(8) show the spatial segmentation results for 'Horse' and 'Handstand' by using the two spatial segmentation methods, respectively. As shown in our spatio-temporal segmentation results for 'Horse' in Figure 7-(3) and 'Handstand' in Figure 7-(6), our approach does not have strict requirements on segmentation boundary shapes and we obtain different segmentation results based on motions. This is because our approach groups the vertices with large affinities to achieve high compression rates, unlike most of the existing 3D mesh segmentation methods that aim to obtain smooth and finely shaped boundaries for functional or semantic parts.

Compression performance. In order to adapt the spatial segmentation methods in [3] and [43] for the compression of 3D mesh animation, we process the spatial segments separately with our compression technique presented in Section 4.4. Figure 10 shows the Rate-Distortion curves of our approach and the comparative methods by experimenting on different data. As can be seen from the figure, our compression approach is robust and outperforms most of the compared methods.

- With the 'Horse' data, our compression approach mostly outperforms the comparative methods. Although Váša et al.'s method in [54] returns less KGError with $bpvf < 4$, our compression approach returns much less STED error.
- For the 'Handstand' data, our approach shows better performance than the comparative methods [3, 43, 54], especially when $bpvf > 1.8$. By comparing to Váša et al.'s method in [53], our approach shows less errors when $bpvf > 2.6$.
- For the 'Samba' data, our approach overtakes all the comparative methods by measuring with the STED error. With the KGError metric, our approach shows competitive performance against the methods in [3, 43, 54] when $bpvf > 3$ and outperforms the method [53] when $bpvf > 4.3$.

Therefore, based on the above experimental results on various of data, we can observe that

- our spatio-temporal segmentation based compression approach shows more robust performance, given our approach returns both less KGError and STED errors on most of the data;
- our approach outperforms the comparative methods when $bpvf$ becomes large. This means when a user prefers decoded precision rather than the quantization, our compression approach is the best choice among the comparative methods.

5.4 Limitations

The main limitation of our current model is the configuration of the parameters needed for the spatio-temporal segmentation scheme. To investigate this issue, we have conducted experimental analysis on the parameters in Section 5.2. Based on our analysis, the tuning of the parameters only has limited influence on the compression results. Using the 'Horse' data in Table 1 as an example, the compression does not change when we modify γ^{max} from 20 to 30. This is because our method often detects a temporal segmentation boundary before reaching γ^{max}, case (II) of Figure 2 in Section 4.5.

Another limitation of our model is the computational cost. Although we have implemented some parts of our spatio-temporal segmentation model through parallel computing and its computational time is superior to those of the existing non-sequential processing based compression methods, it requires further design for a frame-by-frame segmentation update scheme towards the real-time compression of 3D mesh animations in the future.

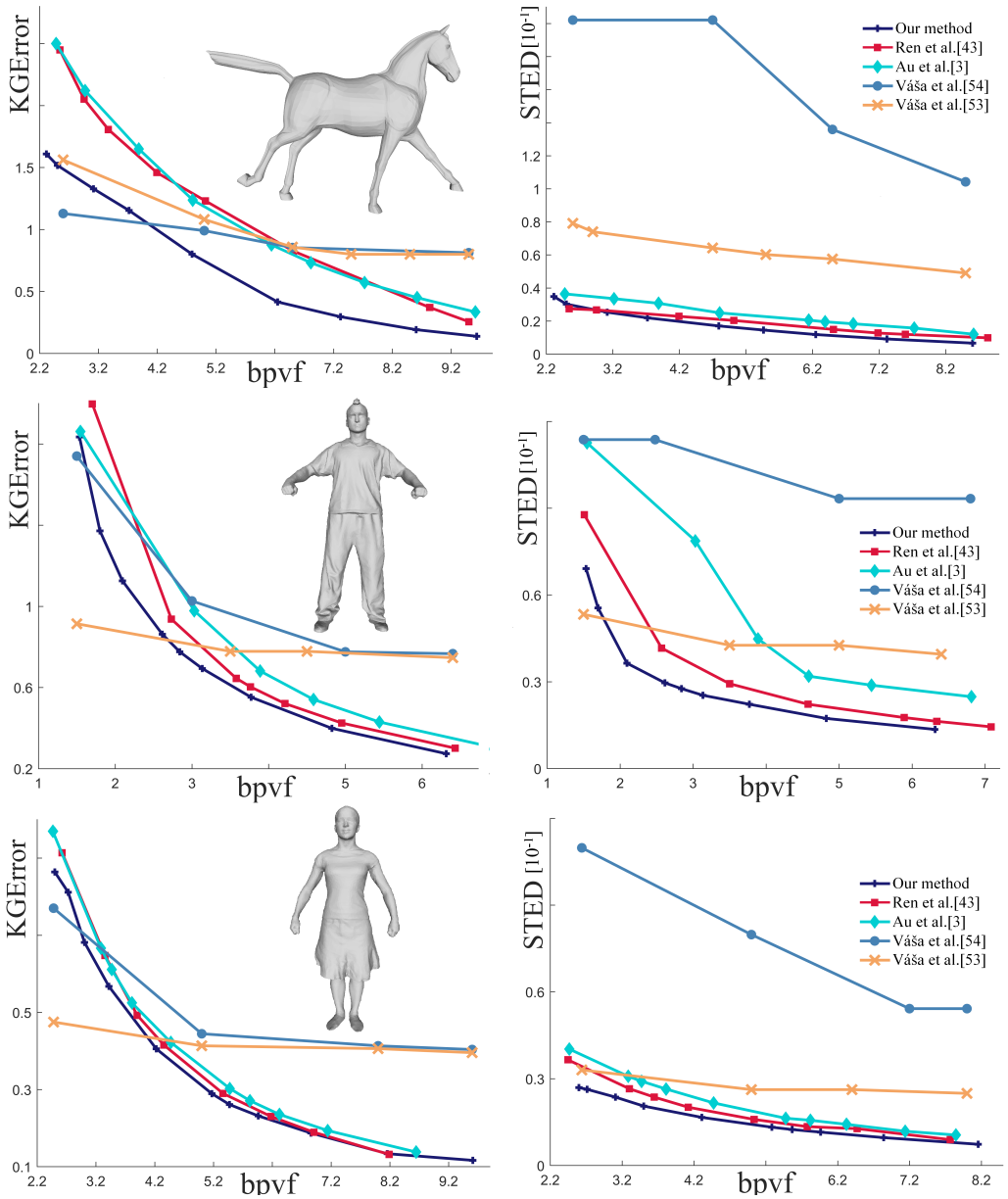


Fig. 10. The KGError (Left) and STED error (Right) comparisons among our approach, the spatial segmentation adapted methods by Ren et al. [43], Au et al. [3] and Váša et al.'s compression methods [53, 54], by using the 'Horse' (Top), 'Handstand' (Middle) and 'Samba' (Bottom) data.

6 CONCLUSION

In this paper, we have presented a new 3D mesh animation compression model based on spatio-temporal segmentation. Our segmentation scheme utilizes the process of an *initial temporal cut*, two-stage *vertex clustering*, and *temporal segmentation*, which are greedy processes to exploit the temporal and spatial redundancies. As discussed in the experimental results, with the user-specified parameters of the corresponding 3D mesh animation data, our compression scheme can automatic determine the optimal number of temporal segments and the optimal number of vertex groups based on global motions and the local movements of input 3D mesh animations. That is, our segmentation methods can automatically optimize the temporal redundancies and the spatial redundancy for compression. Our experiments on various animations demonstrated the effectiveness of our compression scheme. In the future, we would like to extend our spatio-temporal segmentation scheme to handle various motion representations, which can be potentially used for various motion-based animation searching, motion editing, and so on.

ACKNOWLEDGEMENTS

This work has been in part supported by the National Natural Science Fundation of China (No.61962021, 61602222, 61972344, 61732015), the China Postdoctoral Science Foundation (2019M662261), the Key Research and the Development Program of Jiangxi Province (No.20192BBE50079), the Key Research and the Development Program of Zhejiang Province (No.2018C01090). Zhigang Deng is in part supported by US NSF IIS-1524782.

REFERENCES

[1] Andreas A Vasilakis and Ioannis Fudos. 2014. Pose partitioning for multi-resolution segmentation of arbitrary mesh animations. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 293–302.

[2] Marc Alexa and Wolfgang Müller. 2000. Representing Animations by Principal Components. *Computer Graphics Forum* 19, 3 (2000), 411–418.

[3] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. 2008. Skeleton extraction by mesh contraction. In *ACM Transactions on Graphics (TOG)*, Vol. 27. 44.

[4] Jernej Barbic, Alla Safonova, Jiayu Pan, Christos Faloutsos, Jessica K Hodgins, and Nancy S Pollard. 2004. Segmenting motion capture data into distinct behaviors. (2004), 185–194.

[5] Philippe Beaudoin, Pierre Poulin, and Michiel van de Panne. 2007. Adapting wavelet compression to human motion capture clips. In *Proceedings of Graphics Interface 2007 on*. 313–318.

[6] Siddhartha Chattopadhyay, Suchendra M. Bhandarkar, and Kang Li. 2007. Human Motion Capture Data Compression by Model-Based Indexing: A Power Aware Approach. *IEEE Transactions on Visualization and Computer Graphics* 13, 1 (2007), 5–14.

[7] Chengju Chen, Qing Xia, Shuai Li, Hong Qin, and Aimin Hao. 2018. High-fidelity Compression of Dynamic Meshes with Fine Details using Piece-wise Manifold Harmonic Bases. In *Proceedings of Computer Graphics International 2018 on*. 23–32.

[8] Jiong Chen, Yicun Zheng, Ying Song, Hanqiu Sun, Hujun Bao, and Jin Huang. 2017. Cloth compression using local cylindrical coordinates. *Visual Computer* 33, 6-8 (2017), 801–810.

[9] Xiaobai Chen, Aleksey Golovinskiy, and Thomas A. Funkhouser. 2009. A benchmark for 3D mesh segmentation. In *ACM Transactions on Graphics (TOG)*, Vol. 28. 73.

[10] Frederic Cordier and Nadia Magnenatthalmann. 2005. A Data-Driven Approach for Real-Time Clothes SimulationâĂă. *Computer Graphics Forum* 24, 2 (2005), 173–183.

[11] Edilson de Aguiar, Christian Theobalt, Sebastian Thrun, and Hans-Peter Seidel. 2008. Automatic Conversion of Mesh Animations into Skeleton-based Animations. *Computer Graphics Forum* 27, 2 (2008), 389–397.

[12] Peter Deutsch and Jean-Loup Gailly. 1996. ZLIB Compressed Data Format Specification version 3.3. *RFC* 1950 (1996), 1–11.

[13] Amirhossein Firouzmanesh, Irene Cheng, and Anup Basu. 2011. Perceptually Guided Fast Compression of 3-D Motion Capture Data. *IEEE Transactions on Multimedia* 13, 4 (2011), 829–834.

[14] David George, Xianghua Xie, Yu-Kun Lai, and Gary K. L. Tam. 2018. A Deep Learning Driven Active Framework for Segmentation of Large 3D Shape Collections. *arXiv preprint arXiv:1807.06551* (2018).

[15] David George, Xianghua Xie, and Gary K. L. Tam. 2018. 3D mesh segmentation via multi-branch 1D convolutional neural networks. *Graphical Models graphical Models and Image Processing computer Vision, Graphics, and Image Processing* 96 (2018), 1–10.

[16] Dian Gong, Gérard Medioni, Sikai Zhu, and Xuemei Zhao. 2012. Kernelized temporal cut for online temporal segmentation and recognition. In *European Conference on Computer Vision*. Springer, 229–243.
[17] Qin Gu, Jingliang Peng, and Zhigang Deng. 2009. Compression of Human Motion Capture Data Using Motion Pattern Indexing. *Computer Graphics Forum* 28, 1 (2009), 1–12.
[18] Igor Guskov and Andrei Khodakovsky. 2004. *Wavelet compression of parametrically coherent mesh sequences*. Eurographics Association. 183–192 pages.
[19] Mohammadali Hajizadeh and Hossein Ebrahimnezhad. 2016. Predictive compression of animated 3D models by optimized weighted blending of key-frames. *Computer Animation and Virtual Worlds* 27, 6 (2016), 556–576.
[20] Toshiki Hijiri, Kazuhiro Nishitani, Tim Cornish, Toshiya Naka, and Shigeo Asahara. 2000. A spatial hierarchical compression method for 3D streaming animation. In *Symposium on Virtual Reality Modeling Language*. 95–101.
[21] Thomas Hofmann, Bernhard Scholkopf, and Alexander J Smola. 2008. Kernel methods in machine learning. *Annals of Statistics* 36, 3 (2008), 1171–1220.
[22] Junhui Hou, Lap Pui Chau, Nadia Magnenat-Thalmann, and Ying He. 2017. Sparse Low-Rank Matrix Approximation for Data Compression. *IEEE Transactions on Circuits & Systems for Video Technology* 27, 5 (2017), 1043–1054.
[23] David A. Huffman. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.
[24] Doug L. James and Christopher D. Twigg. 2005. Skinning mesh animations. *international conference on computer graphics and interactive techniques* 24, 3 (2005), 399–407.
[25] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. 2010. Learning 3D mesh segmentation and labeling. *international conference on computer graphics and interactive techniques* 29, 4 (2010), 102.
[26] Zachi Karni and Craig Gotsman. 2004. Compression of soft-body animation sequences. *Computers & Graphics* 28, 1 (2004), 25–34.
[27] Ladislav Kavan, Peter-Pike J. Sloan, and Carol O'Sullivan. 2010. Fast and Efficient Skinning of Animated Meshes. *Computer Graphics Forum* 29, 2 (2010), 327–336.
[28] Murtaza Ali Khan. 2016. An efficient algorithm for compression of motion capture signal using multidimensional quadratic Bézier curve break-and-fit method. *Multidimensional Systems and Signal Processing* 27, 1 (2016), 121–143.
[29] Choong-Hoon Kwak and Ivan V. Bajic. 2011. Hybrid low-delay compression of motion capture data. In *2011 IEEE International Conference on Multimedia and Expo*. 1–6.
[30] Aris S. Lalos, Andreas A. Vasilakis, Anastasios Dimas, and Konstantinos Moustakas. 2017. Adaptive compression of animated meshes by exploiting orthogonal iterations. *The Visual Computer* 33, 6-8 (2017), 1–11.
[31] Binh Huy Le and Zhigang Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. *Acm Transactions on Graphics* 33, 4 (2014), 1–10.
[32] Pai-Feng Lee, Chi-Kang Kao, Juin-Ling Tseng, Bin-Shyan Jong, and Tsong-Wuu Lin. 2007. 3D animation compression using affine transformation matrix and principal component analysis. *IEICE TRANSACTIONS on Information and Systems* 90, 7 (2007), 1073–1084.
[33] Tong-Yee Lee, Yu-Shuen Wang, and Tai-Guang Chen. 2006. Segmenting a deforming mesh into near-rigid components. *The Visual Computer* 22, 9 (24 Aug 2006), 729. https://doi.org/10.1007/s00371-006-0059-6
[34] Tong Yee Lee, Yu Shuen Wang, and Tai Guang Chen. 2006. Segmenting a deforming mesh into near-rigid components. *Visual Computer* 22, 9-11 (2006), 729.
[35] Xin Liu, Zaiwen Wen, and Yin Zhang. 2012. Limited Memory Block Krylov Subspace Optimization for Computing Dominant Singular Value Decompositions. *Siam Journal on Scientific Computing* 35, 3 (2012), 1641–1668.
[36] Guoliang Luo, Frederic Cordier, and Hyewon Seo. 2013. Compression of 3D mesh sequences by temporal segmentation. *Computer Animation & Virtual Worlds* 24, 3-4 (2013), 365–375.
[37] Guoliang Luo, Zhigang Deng, Xiaogang Jin, Xin Zhao, Wei Zeng, Wenqiang Xie, and Hyewon Seo. 2019. 3D Mesh Animation Compression based on Adaptive Spatio-temporal Segmentation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 10.
[38] Guoliang Luo, Gang Lei, Yuanlong Cao, Qinghua Liu, and Hyewon Seo. 2017. Joint entropy-based motion segmentation for 3D animations. *The Visual Computer* 33, 10 (2017), 1279–1289.
[39] Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot. 2015. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *Comput. Surveys* 47, 3 (2015), 44.
[40] K. Mamou, T. Zaharia, F. Preteux, N. Stefanoski, and J. Ostermann. 2008. Frame-based compression of animated meshes in MPEG-4. In *IEEE International Conference on Multimedia and Expo*. 1121–1124.
[41] Frédéric Payan and Marc Antonini. 2007. Temporal wavelet-based compression for 3D animated models. *Computers & Graphics* 31, 1 (2007), 77–88.
[42] Subramanian Ramanathan, Ashraf A. Kassim, and Tiow Seng Tan. 2008. Impact of vertex clustering on registration-based 3D dynamic mesh coding. *Image & Vision Computing* 26, 7 (2008), 1012–1026.
[43] Zhile Ren and Gregory Shakhnarovich. 2013. Image Segmentation by Cascaded Region Agglomeration. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2011–2018.

[44] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. 2005. Simple and efficient compression of animation sequences. In *ACM Siggraph/eurographics Symposium on Computer Animation*. 209–217.
[45] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. 2007. A Hilbert space embedding for distributions. In *In Algorithmic Learning Theory: 18th International Conference*. Springer-Verlag, 13–31.
[46] Nikolce Stefanoski, Xiaoliang Liu, Patrick Klie, and Jorn Ostermann. 2007. Scalable Linear Predictive Coding of Time-Consistent 3D Mesh Sequences. In *3dtv Conference*. 1–4.
[47] Nikolče Stefanoski and Jörn Ostermann. 2010. SPC: fast and efficient scalable predictive coding of animated meshes. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 101–116.
[48] Robert W Sumner and Jovan Popovic. 2004. Deformation transfer for triangle meshes. *international conference on computer graphics and interactive techniques* 23, 3 (2004), 399–405.
[49] Art Tevs, Alexander Berner, Michael Wand, Ivo Ihrke, Martin Bokeloh, Jens Kerber, and Hans-Peter Seidel. 2012. Animation Cartography - Intrinsic Reconstruction of Shape and Motion. *ACM Trans. Graph.* 31, 2 (April 2012).
[50] Shoichi Tsuchie, Tikara Hosino, and Masatake Higashi. 2014. High-quality vertex clustering for surface mesh segmentation using Student-t mixture model. *Computer-aided Design* 46 (2014), 69–78.
[51] Steven Van Vaerenbergh. 2010. *Kernel methods for nonlinear identification, equalization and separation of signals*. Ph.D. Dissertation. University of Cantabria. Software available at https://github.com/steven2358/kmbox.
[52] Libor Váša and Guido Brunnett. 2013. Exploiting Connectivity to Improve the Tangential Part of Geometry Prediction. *IEEE Transactions on Visualization and Computer Graphics* 19, 9 (2013), 1467–1475.
[53] Libor Váša, Stefano Marras, Kai Hormann, and Guido Brunnett. 2014. Compressing dynamic meshes with geometric laplacians. *Computer Graphics Forum* 33, 2 (2014), 145–154.
[54] Libor Váša and Vaclav Skala. 2009. COBRA: Compression of the Basis for PCA Represented Animations. *Computer Graphics Forum* 28, 6 (2009), 1529–1540.
[55] Libor Váša and Vaclav Skala. 2011. A Perception Correlated Comparison Method for Dynamic Meshes. *IEEE Transactions on Visualization & Computer Graphics* 17, 2 (2011), 220–30.
[56] Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popovic. 2008. Articulated mesh animation from multi-view silhouettes. *international conference on computer graphics and interactive techniques* 27, 3 (2008), 97.
[57] Pengjie Wang, Zhigeng Pan, Mingmin Zhang, Rynson W.H. Lau, and Haiyu Song. 2013. The alpha parallelogram predictor: A lossless compression method for motion capture data. *Information Sciences* 232 (2013), 1–10.
[58] Stefanie Wuhrer and Alan Brunton. 2010. Segmenting animated objects into near-rigid components. *Visual Computer* 26, 2 (2010), 147–155.
[59] Bailin Yang, Zhaoyi Jiang, Jiantao Shangguan, Frederick W.B. Li, Chao Song, Yibo Guo, and Mingliang Xu. 2018. Compressed dynamic mesh sequence for progressive streaming: Compressed Dynamic Mesh Sequence Progressive Streaming. *Computer Animation and Virtual Worlds* (2018).
[60] Bailin Yang, Luhong Zhang, W.B. Frederick Li, Xiaoheng Jiang, Zhigang Deng, Meng Wang, and Mingliang Xu. 2018. Motion-aware Compression and Transmission of Mesh Animation Sequences. *ACM Transactions on Intelligent Systems and Technologies* (2018), (accepted in December 2018).
[61] Jeong Hyu Yang, Chang Su Kim, and Sang Uk Lee. 2002. Compression of 3-D triangle mesh sequences based on vertex-wise motion vector prediction. *IEEE Transactions on Circuits & Systems for Video Technology* 12, 12 (2002), 1178–1184.
[62] Yazhou Yuan, Yu Zhang, Zhixin Liu, and Xinping Guan. 2017. Lossless coding scheme for data acquisition under limited communication bandwidth. *Digital Signal Processing* 69 (2017), 204–211.
[63] Mingyang Zhu, Huaijiang Sun, and Zhigang Deng. 2012. Quaternion space sparse decomposition for motion compression and retrieval. In *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*. Eurographics Association, 183–192.
[64] Jacob Ziv and Abraham Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23, 3 (1977), 337–343.