Motion-aware Compression and Transmission of Mesh Animation Sequences

BAILIN YANG, LUHONG ZHANG, FREDERICK W.B. LI, JIANG XIAOHENG, DENG ZHIGANG, MENG WANG, MINGLIANG XU

With the increasing demand in using 3D mesh data over networks, supporting effective compression and efficient transmission of meshes has caught lots of attention in recent years. This paper introduces a novel compression method for 3D mesh animation sequences, supporting user-defined and progressive transmissions over networks. Our motion-aware approach starts with clustering animation frames based on their motion similarities, dividing a mesh animation sequence into fragments of varying lengths. This is done by a novel temporal clustering algorithm, which measures motion similarity based on the curvature and torsion of a space curve formed by corresponding vertices along a series of animation frames. We further segment each cluster based on mesh vertex coherence, representing topological proximity within an object under certain motion. To produce a compact representation, we perform intra-cluster compression based on Graph Fourier Transform (GFT) and Set Partitioning In Hierarchical Trees (SPIHT) coding. Optimized compression results can be achieved by applying GFT due to the proximity in vertex position and motion. We adapt SPIHT to support progressive transmission and design a mechanism to transmit mesh animation sequences with user-defined quality. Experimental results show that our method can obtain a high compression ratio while maintaining a low reconstruction error.

CCS Concepts: • **Information systems** → **Data mining**; • **Computing methodologies** → *3D imaging*; *Image compression*;

Additional Key Words and Phrases: 3D Mesh Animation, Progressive Transmission, Compression, Clustering, Graph Fourier Transform.

ACM Reference Format:
Bailin Yang, Luhong Zhang, Frederick W.B. Li, Jiang Xiaoheng, Deng Zhigang, Meng Wang, Mingliang Xu . 2018. Motion-aware Compression and Transmission of Mesh Animation Sequences. *ACM Trans. Intell. Syst. Technol.* x, x, Article xx (December 2018), 21 pages.
https://doi.org/10.1145/3300198

1 INTRODUCTION

Highly detailed three-dimensional (3D) meshes have been widely used in virtual reality, online virtual worlds, simulation, training, and education. A 3D mesh typically comprises connectivity information (topology) and geometry information (3D vertex positions). When storing and transmitting such highly detailed 3D meshes, large amounts of storage space and network bandwidth are required. It is even more challenging when processing mesh animation sequences, since their raw data sizes can be many multiples of that of a static mesh. Particularly, complicated mesh animations may comprise long sequences of highly accurate motion details. Their raw representations will result in large files, becoming very expensive to store and transmit. Hence, it is demanding to have a method significantly

Bailin Yang, Luhong Zhang, are with the School of Computer Science & Information Engineering, Zhejiang Gongshang University, Hangzhou, 310018, China. Frederick W.B. Li is with the Department of Computer Science, University of Durham, UK. Zhigang Deng is with the Department of Computer Science, University of Houston, Houston, Texas, USA. Meng Wang is School of Computer Science & Information Engineering, HeFei University of Technology,Jiang Xiaoheng, Mingliang XU is with the School of Information Engineering, Zhengzhou University, Zhengzhou, 450000, China.
Manuscript received xx, 2017; revised xx, 2017.
https://doi.org/10.1145/3300198

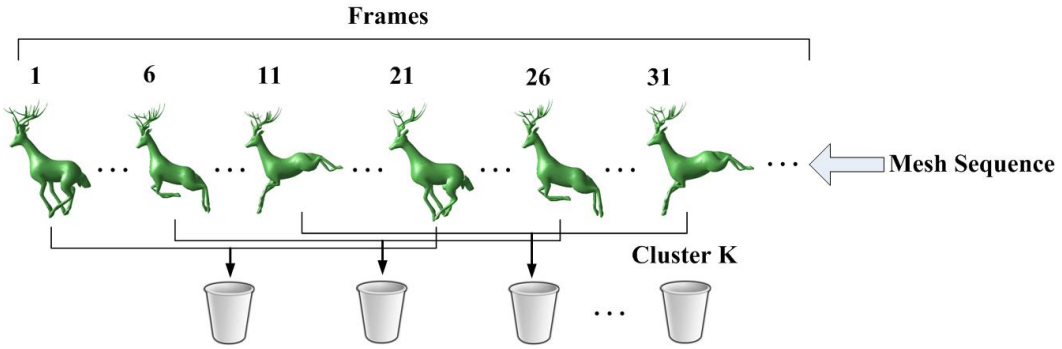


Fig. 1. Illustration of animation frame clustering based on motion similarity.

optimizing such representations, in order to support effective storage and transmission of mesh animation sequences.

We propose a novel approach to compress mesh animation sequences, exploiting spatial and temporal redundancies as well as the geometrical properties of curvature and torsion. The animated deer example in Fig. 1 illustrates the motivation of our compression method. Obviously, frames with similar motions are highly redundant and can be grouped into one cluster. We also observe that closely located frames may not be necessarily more correlated than frames at other parts of the animation sequence. For instance, frames #1 and #21 exhibit a similar motion, so do frames #11 and #31. We hence develop a novel compression method exploiting such motion similarity. In contrast, existing work typically only focused on identifying coherence between *neighboring* frames. Our method clusters the frames in a mesh animation sequence based on motion similarities between any frames to remove redundancy. The compression ratio achieved by our method can be easily escalated when the number of frames becomes larger.

Technically, we reduce temporal and spatial redundancies based on mesh motion similarity, generating a set of motion fragments. The motion similarity measure is designed based on curvature and torsion, since they jointly characterize the shape of a spatial curve, offering a mathematical formulation to model vertex motion in a 3D space. After temporal-spatial clustering, instead of transforming an entire animation sequence into components as many existing work did, we apply Graph Fourier Transform (GFT) [17] to each motion fragment independently. Since GFT is optimal for decomposing smooth objects, it works well with our generated motion fragments as mesh frames in each fragment are highly coherent. We then select significant GFT coefficients from all the fragments and encode them together to compress the animation sequence. We validate the effectiveness of our method through a set of comparisons with state-of-the-art methods. Our main contributions include:

- A new motion similarity metric through curvature and torsion, being effective to identify and reduce spatio-temporal redundancies in mesh animation sequences.
- A novel progressive transmission scheme for mesh animation sequences, which allows a coarse sequence with a minimal data size to be transmitted initially and streams fine details of mesh animation gradually. We also allow the reuse of a received mesh animation part to replace a required part without transmission if they are similar enough.

The rest of the paper is organized as follows. After reviewing previous works in Section 2, we give our method overview in Section 3. Sections 4 and 5 describe technical details about modeling of motion change and motion-based segmentation, respectively. Section 6 elaborates our compression

method and Section 7 presents a user-defined transmission mechanism. Experiment results are presented in Section 8. Finally, we conclude our paper in Section 9.

2 RELATED WORK

Given an input model, a main purpose of compression is to seek for a compact representation reducing the model data size, such that effective storage and fast transmission can be supported. A variety of researches have been conducted to compress mesh animation sequences. Generally, they utilize temporal redundancy (small difference between neighboring frames) [30, 31, 36, 37] and/or spatial redundancy (small difference between neighboring vertices in the same frame) [32, 35, 36] to reduce data size. We can broadly classify existing mesh animation compression methods into four categories, including Principal components analysis (PCA) based methods [42], prediction based methods [12, 18, 39], segmentation based methods [26, 29], and wavelet based methods [9, 33]. Meanwhile, according to their supported transmission mechanism, these methods can also be classified into single-rate and progressive compression.

PCA based methods transform mesh vertices from the Cartesian space to a new basis space formed by linearly uncorrelated eigen-vectors. Compression can be achieved by only retaining most significant components (i.e., those eigen-vectors with largest eigen-values) of the new basis space. Alexa and Müller [2] proposed the first PCA-based method to compress a mesh animation sequence based on decomposing the sequence into principal components by analyzing keyframe geometries globally. The principal component representation allows adaptive lossy compression of a mesh animation sequence with a factor up to 1:100. Later, various extensions have been further developed such as [36] that applied PCA to spatial clusters and [28] that applied PCA on temporal clusters. Ramanathan et al. [35] proposed a method to identify an optimized clustering to give the best compression ratio. In a nutshell, these methods produce a set of eigen-trajectories with their corresponding PCA coefficients. To further compress the resulting PCA coefficients, researchers combine PCA with Linear Predictive Coding (LPC) [23], or re-compress the resulting eigen-vectors by LPC [41]. To support progressive coding, [22] separately decomposed connectivity into progressive mesh and mesh geometry into PCA components, with vertex trajectories clustered by K-means to optimize compression ratio. Modeling connectivity with a global progressive mesh likely cannot match well with significant motion changes, causing undesired geometric distortions to reconstructed animation. Also, applying global PCA decomposition implicitly restricts reconstruction to be happened only if the data of an entire animation sequence is available (despite it can be lower-quality versions). This seriously limits the scalability for supporting long animation sequences. Recently, [25] attempted to decompose the global PCA computation process into local ones based on block of frames. This improved computational scalability and optimized output data size by constructing adaptive local eigen-space. [40] alternatively focused on improving the reconstructed output mesh quality by modifying the high-pass encoding scheme to limit error accumulation in the mesh encoding process.

Prediction based methods exploit the typical fixed connectivity property of mesh animation sequence, utilizing the recovered vertex positions at previous frames to predict the new vertex positions at the next frame. Different from PCA based methods, prediction methods exploit local coherence and thus are computationally efficient. The first work along this direction [10] only used Delta encoding: the vertex position to encode was predicted based on the position of the same vertex at the previous frame; the Delta, the vector between the two positions, was then encoded. Bajaj et al. [7] introduced a second-order predictor that encoded the difference between consecutive delta predictions. Alternatively, Stefanoski and Ostermann [38] proposed a scalable predictive coding (SPC) to decompose mesh animation sequences in temporal and spatial layers and to perform prediction in the space of rotation-invariant coordinates, compensating local rigid motion for effective

encoding. Later, it has been improved by applying both weighted spatial prediction and weighted refinement based on the angular relations of triangles between current and previous frames [8]. Recently, Ahn et al. [1] improved the multilayer prediction of the above SPC method [38] to achieve 30% performance gain.

Segmentation-based compression for mesh animation sequences is to divide mesh vertices into groups of similar motions, followed by encoding each group with a small number of representative vectors or parameters [26]. Gupta et al. [15] used multilevel k-way partitioning algorithm on the basis of proximity in the connectivity and the number of parts specified by a user. Amjoun et al. [4] partitioned mesh vertices into clusters by applying k-means clustering [21], where vertex motions can be described by unique 3D affine transforms. A region growing based method has been proposed in [4], which was simple to implement, but the results were seriously dependent on the choice of initial seed vertices. Alternatively, [36] clustered vertex trajectories by integrating Lloyd's algorithm and PCA, followed by compressing each cluster independently.

Wavelet based methods have also been used to deal with mesh animation compression, aiming to de-correlate geometric data and to generate a sequence of detail coefficients. Guskov and Khodakovsky [16] used wavelets to perform multiresolution analysis along an animation sequence and applied delta coding on wavelet coefficients to improve the compression ratio. Payan and Antonini [34] proposed a temporal wavelet filtering together with an efficient bit allocation process. Alternatively, [43] performed remeshing on a mesh animation sequence facilitating wavelet transform, but being inherently prone to geometric distortions, which likely propagate along the entire sequence. Despite pre-defined wavelet coefficient weightings were proposed to fix the problem, it was still hard to guarantee proper coding and reconstruction of sequences with complicated mesh motions.

In addition, a number of previous research efforts [5, 6, 20, 42] have been focused on connectivity-based compression. Their core idea is to exploit connectivity information of a mesh. Unlike these approaches, our method does not focus on mesh connectivity but geometry information. This is critical as when a mesh animation sequence goes large in terms of frame numbers, geometry information will become the dominated part of data in the sequence.

3 METHOD OVERVIEW

This section gives an overview of our method. A mesh animation sequence is typically formed by F frames of meshes $M_1, \ldots, M_F$ sharing the same topology (connectivity). Positions of vertices of the j-th frame are represented as a set of vectors $v_{ij} = (v_{ij}^x, v_{ij}^y, v_{ij}^z)$, where $i = 1, \ldots, N$, $j = 1, \ldots, F$, and N is the number of vertices of a frame. Our method focuses on compressing such geometry information. Mesh connectivity information is separately compressed by [14], as it is defined once and shared among all frames, making the data size relatively insignificant, particularly for long animation sequences.

Fig. 2 illustrates the pipeline of our compression process. We start with performing temporal (frame) clustering to obtain K clusters of mesh frames, exploiting motion similarity, which will be discussed in Section 4. We then partition mesh vertices of each cluster into S segments through spatial segmentation. Thereafter, each segment obtained is relatively smooth both spatially and temporally, since the vertices of its mesh frames are highly proximal exhibiting similar motions. This allows us to apply Graph Fourier Transform (GFT) [17], converting each segment into orthogonal components. Each component is characterized by a GFT coefficient C_{ij}, where i and j are the indices with their ranges determining by the number of vertices contained in a segment. Eventually, we may select a subset of most significant coefficients and apply Set Partitioning in Hierarchical Trees (SPIHT) [13] to encode these coefficients. On the other hand, the connectivity information as well as the clustering

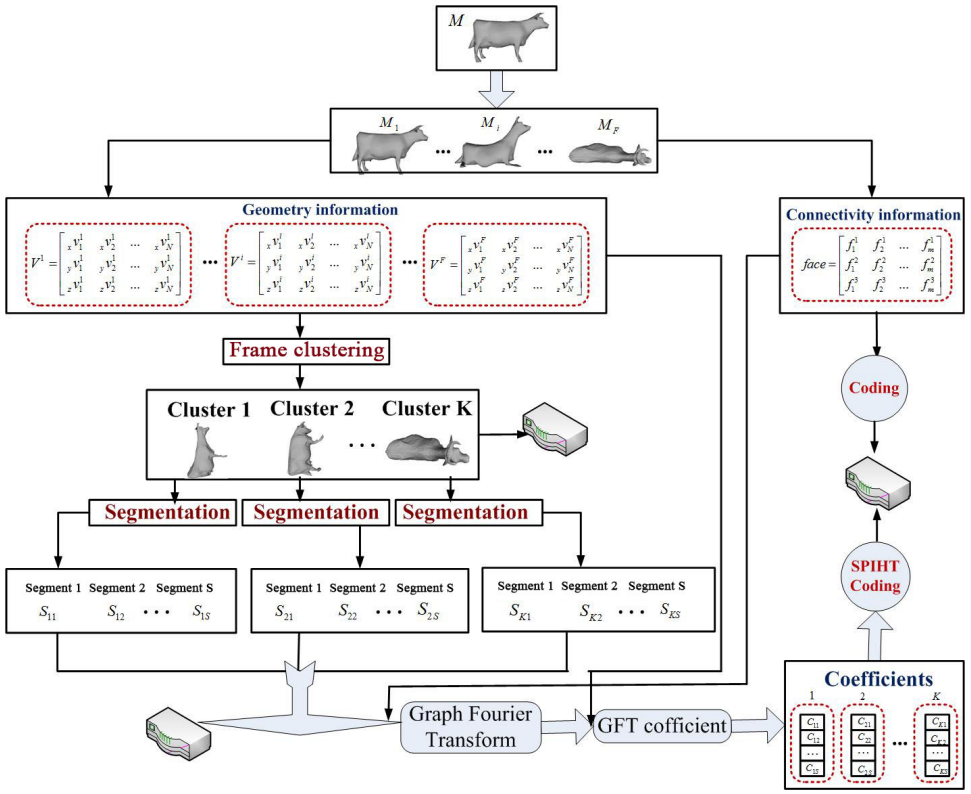


Fig. 2. The compression process in our approach

and segmentation information (i.e., frame/vertex correspondence to cluster/segment) will also be compressed by GFT.

Decompression is the inverse process of compression, which is illustrated in Fig. 3. To proceed, we firstly perform SPIHT decoding to recover GFT coefficients and the information of clustering and segmentation. These will be processed by GFT to obtain corresponding GFT bases. Together with connectivity information recovered by [14], decompressed geometry information of the mesh animation sequence can then be reconstructed.

4 MODELING OF MOTION CHANGE

Differential geometry literature shows that curvature and torsion measure how a spatial curve bends. Curvature measures the degree of curve bending along the tangential direction near a point. Torsion measures how sharply a spatial curve twists out of the plane of curvature, i.e., the degree of curve distortion. Imagine when a vertex moves in a space, its motion essentially forms a spatial curve, while the curve shape represents vertex motion change. Therefore, curvature and torsion are suitable parameters for evaluating motion change and thus the motion similarity between two frames. Based on this, we define *motion strength* to measure the degree of motion change of a vertex, calculating as the weighted average of curvature and torsion. Its definition will be described later in this section.

Curvature and torsion capture between-frame motions of mesh vertices. They serve as a foundation to divide a mesh animation into logical parts exhibiting similar motions. This allows us to identify both

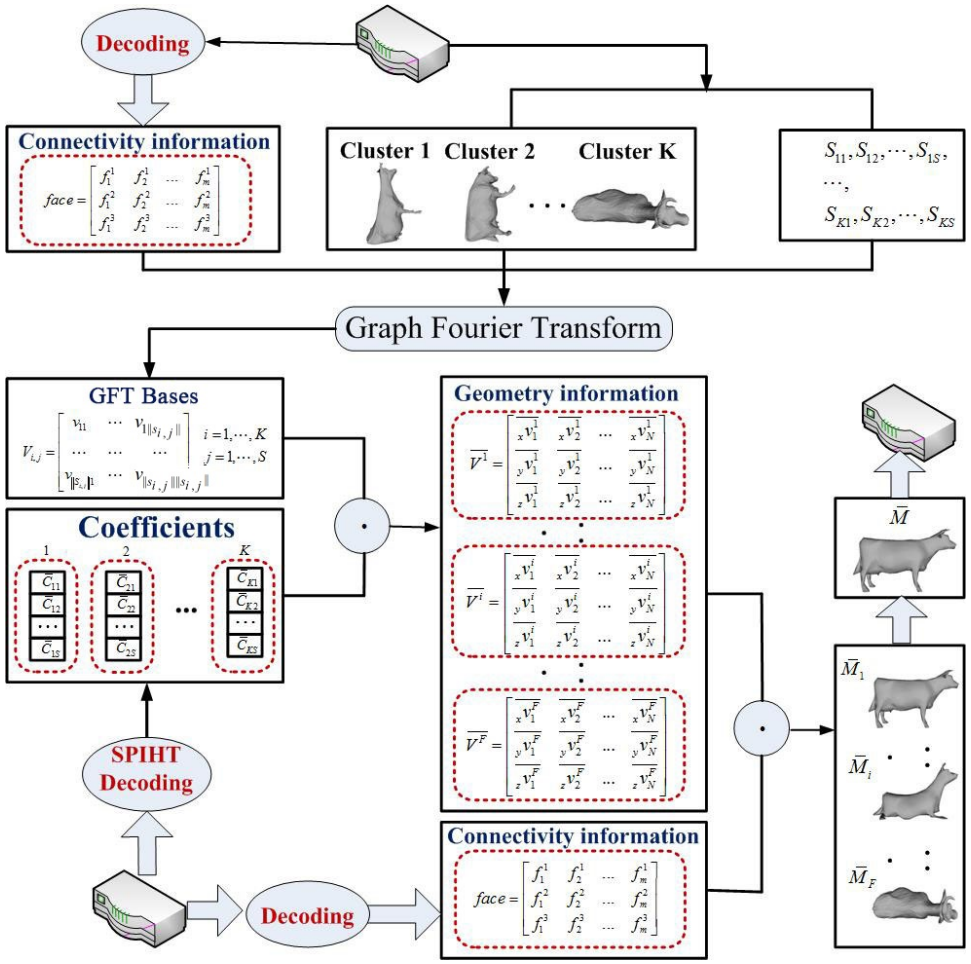


Fig. 3. The decompression process in our approach

intra-mesh and inter-mesh motion redundancy while methods based on standard vertex coordinates usually do not explore intra-mesh motion redundancy. Using our method is particularly beneficial for compressing models with parts of similar motions, e.g., leg motions of animal walk.

To implement, for each vertex in a mesh frame, we connect it with all its instances across all mesh frames to obtain a spatial curve modeling the motion of this vertex. Mathematically, the motion of a vertex v_{ij} is then formulated by curvature and torsion of the spatial curve, represented by k_{ij} and τ_{ij}, respectively, as follows:

$$k_{ij} = \frac{\|\overrightarrow{r'_{ij}} \times \overrightarrow{r''_{ij}}\|}{\|\overrightarrow{r'_{ij}}\|^3} \tag{1}$$

$$\tau_{ij} = \frac{(\overrightarrow{r'_{ij}}, \overrightarrow{r''_{ij}}, \overrightarrow{r'''_{ij}})}{(\overrightarrow{r'_{ij}} \times \overrightarrow{r''_{ij}})^2} \tag{2}$$

$\overrightarrow{r_{ij}}$ represents as a spatial curve describing the motion change of the i-th vertex in the vicinity of frame j. We approximate the derivatives of the above equations using differential geometry. To simplify the calculation, we set the step length to be 1. This allows us to calculate the first order derivative $\overrightarrow{r'_{ij}}$ of the spatial curve based on the values at two neighboring frames, i.e. frame i and frame $i+1$. Similarly, the second order derivative $\overrightarrow{r''_{ij}}$ can be calculated based on the two neighboring first order derivatives, and we can perform similar calculations to obtain the third order derivative $\overrightarrow{r'''_{ij}}$. On top of curvature and torsion, we define motion strength $R_{i,j}$ as in Eqn. (3). We experimentally set $\beta = 0.5$, since we assume curvature and torsion have are of equal importance.

$$R_{ij} = \beta * k_{ij} + (1 - \beta) * \tau_{ij} \tag{3}$$

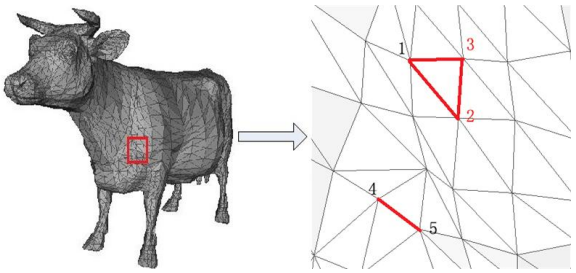


Fig. 4. Example Cow model with 2904 vertices and 5804 triangles. (Left) Wireframe rendering of the model. (Right) Zoomed view of the highlighted model part.

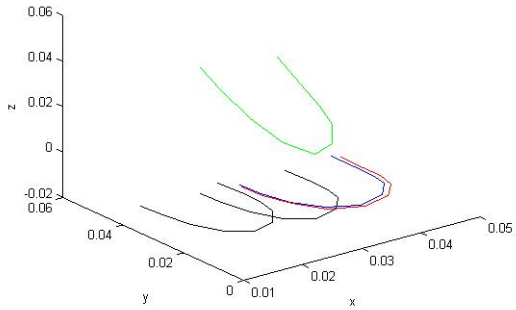
Table 1. motion strength of the five selected vertices as in Fig. 4

Index	1	2	3	4	5
motion strength	58.01	59.47	60.05	89.68	77.19

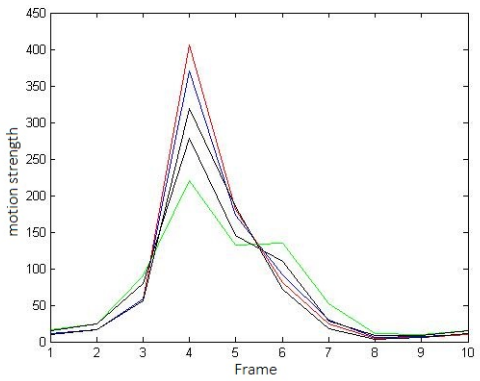
Fig. 4 shows an example of a cow mesh frame. The wireframe rendering on the left depicts the topological structure of the mesh, while the zoomed view on the right shows a highlighted mesh part with five labeled vertices. The corresponding values of motion strength of these vertices are shown in Table 1. We can see that the 2nd and the 3rd vertices exhibit similar motion strength, while the 2nd and the 4th vertices exhibit quite different motion strength. In order to validate whether similarity/difference in motion strength can represent spatial curve trajectory similarity/difference, we plot the trajectories of the five vertices as exhibited in the first ten mesh frames in Fig. 5(a). Fig. 5(b) plots the computed motion strength values of the five vertices. The trajectories of the 2nd, 3rd and 4th vertices are plotted in red, blue and green, respectively. We can clearly see that the motion trajectories in red and blue are very similar, while those in red and green are quite different. By observing the changes of corresponding motion strength values along the mesh frames, it shows similarities/differences of these values agree with the above findings, supporting the validation.

5 MOTION-BASED SEGMENTATION

With the mathematical motion change modeling based on curvature and torsion, mesh animations are allowed to group into segments, such that each contains vertices exhibiting similar motions along certain mesh frames. We apply K-means clustering [21] for this purpose due to its simplicity and efficiency. This motion-based segmentation process is critical for achieving a better compression



(a)



(b)

Fig. 5. (a) Trajectories and (b) motion strength of five selected cow model vertices during the first ten frames. Color codes of the 1st to 5th vertices are black, red, blue, green and black, respectively.

performance and particularly enabling user-defined transmission, which is not supported by previous methods.

5.1 Temporal clustering

To start the process, we perform temporal clustering grouping vertices based on their motion similarity to facilitate temporal redundancy discovery. This is motivated by the idea that an animation sequence may exhibit similar model deformations from time to time, e.g., existence of cyclic or repetitive motions, inducing significant data redundancy. In such situations, similar frames may be found at different parts of a mesh animation sequence, where these frames may not be necessarily adjacent to each other. Based on our motion change modeling, motion information of a mesh animation sequence can be formulated with $\mathbf{T}$ as in Eqn. (4), representing the details of mesh motion change over time along the animation sequence. Each column models the motion of an individual mesh vertex, while each row presents global mesh motion change in the vicinity of a particular frame, where $i = 1, \ldots, N$ and $j = 1, \ldots, F - 1$.

$$\mathbf{T} = \begin{pmatrix} k_{11} & \tau_{11} & \dots & k_{i1} & \tau_{i1} \\ k_{12} & \tau_{12} & \dots & k_{i2} & \tau_{i2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ k_{1j} & \tau_{1j} & \dots & k_{ij} & \tau_{ij} \end{pmatrix} \tag{4}$$

To implement temporal clustering, our strategy is to group frames with similar motion change into the same cluster, i.e., putting similar rows of matrix $\mathbf{T}$ (Eqn. (4)) into one group based on K-means. Simple Euclidean distance $\|v_j - v_h\|$ is used to measure the motion similarity between two rows of matrix T, where $v_i = (k_{1j}, \tau_{1j}, \dots, k_{Nj}, \tau_{Nj})$ given that $j, h = 1, \dots, F$ and $j \neq h$.

5.2 Spatial Segmentation

Temporal clustering facilitates the discovery of data redundancy by exploiting mesh frame coherence. In fact, data redundancy also exists within the mesh model itself, which could be significant if mesh model complexity is high. Although some existing methods also worked along this line, they have usually focused on exploring spatial redundancy from one mesh frame, e.g., the first frame, and reused the obtained result for an entire animation sequence. In fact, this might not fit to other frames due to posture/motion changes, producing non-optimal results. We therefore extend spatial coherence to incorporate motion changes, becoming motion-aware.

To proceed with spatial segmentation, we formulate a motion-aware spatial representation for a mesh animation sequence with matrix $\mathbf{R}$ as in Eqn. (5), which consists of N rows and $3 + 2S$ columns, where S is the number of mesh frames contained in a temporal cluster.

$$\mathbf{R} = \begin{pmatrix} \alpha\widehat{x_1} & \alpha\widehat{y_1} & \alpha\widehat{z_1} & \frac{1-\alpha}{2}k_{11} & \frac{1-\alpha}{2}\tau_{11} & \dots & \frac{1-\alpha}{2}k_{1S} & \frac{1-\alpha}{2}\tau_{1S} \\ \alpha\widehat{x_2} & \alpha\widehat{y_2} & \alpha\widehat{z_2} & \frac{1-\alpha}{2}k_{21} & \frac{1-\alpha}{2}\tau_{21} & \dots & \frac{1-\alpha}{2}k_{1S} & \frac{1-\alpha}{2}\tau_{2S} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha\widehat{x_N} & \alpha\widehat{y_N} & \alpha\widehat{z_N} & \frac{1-\alpha}{2}k_{N1} & \frac{1-\alpha}{2}\tau_{N1} & \dots & \frac{1-\alpha}{2}k_{1S} & \frac{1-\alpha}{2}\tau_{NS} \end{pmatrix} \tag{5}$$

In $\mathbf{R}$, each row encloses the motion-aware representation of each vertex, where $\widehat{x_i}$, $\widehat{y_i}$ and $\widehat{z_i}$ denote the average coordinates of the i-th vertex within a given set of mesh frames, k_{ij} and τ_{ij} respectively denote the curvature and torsion of the i-th vertex of the j-th frame, and α denotes the weight of coordinates. We do not incorporate all mesh vertex coordinates of the entire animation sequence since this will enormously escalate the matrix data size, which is redundant, as we already incorporate a more concise representation of them, namely the curvature and torsion. In addition, including only the averages coordinate of each vertex is possible since a mesh animation sequence comes with a fixed mesh connectivity, and that each vertex has a well-defined correspondence across all frames. During segmentation, we consider three factors to determine the similarity among mesh vertices, namely vertex position, curvature, and torsion. Considering the magnitudes of these three factors are different, we apply min-max normalization to their corresponding values under each column of the matrix $\mathbf{R}$, i.e., updating each of these values u by $u = (u - u_{min})/(u_{max} - u_{min})$, where u_{min} and u_{max} are the smallest and largest u in a column.

Regarding the motion-aware component of the matrix $\mathbf{R}$, we set the weights of both curvature and torsion to $(1 - \alpha)/2$ since their importances are approximately equivalent. On the other hand, we set α as the weight for each component of the averaged vertex coordinates, allowing us to adjust the importance between geometrical and motion feature of a mesh animation sequence. Specific values of α are determined by our experiments as presented in Section 7. As shown in Eqn. (5), since each row of matrix $\mathbf{R}$ represents the motion of a vertex across a set of frames, if the corresponding rows of two vertices are similar, it would mean their geometrical and motion characteristics are

comparable. Consequently, we perform K-means clustering to group similar rows of matrix **R** by minimizing within-cluster sum of squares, which is measured by evaluating the Euclidean distance between different rows. Unfortunately, it is not trivial to automatically determine the optimal number of clusters. We determine a suitable segment number for each mesh animation through experiments by trading off reconstruction error against compression rate, as will be described in Section 7.2.

Our motion-aware spatial segmentation essentially allows some spatially distant vertices performing the same motion to be grouped together. This could be useful in terms of increasing the compression ratio, since it helps reduce the number of segments produced. Moreover, for mesh animation retrieval applications as will be discussed in Section 7.4.3, being able to group spatially proximal vertices will be more favorable, since they provide logical representations of mesh part motions. We refer this as preserving spatial continuity, which is controlled by α in the matrix **R**. Section 7.2 shows results about how different values of α affect the spatial continuity.

6 COMPRESSION

Considering the segments produced by the temporal-spatial clustering are relatively smooth in terms of their contents, we hence apply Graph Fourier Transform (GFT) [17] to encode these segments. In this way, most of the obtained coefficients would be close to zero, while those coefficients with relatively large values represent low-frequency information of the corresponding segments. To compress the results, we only choose to store those relatively large coefficients, essentially removing high-frequency information. Note that we can retain more coefficients, incorporating certain high-frequency information to improve the reconstruction quality if needed. To achieve progressive transmission, we choose the Set Partitioning In Hierarchical Trees (SPIHT) algorithm [13] to encode the selected coefficients.

6.1 Graph Fourier Transform (GFT)

We apply GFT to encode the results from temporal-spatial clustering, i.e., the segments obtained in Section 5.2. We choose GFT over other types of transforms due to the following reasons: (i) From the perspective of progressive transmission, although some methods such as Fourier and discrete cosine transforms [9, 16] also support progressive transmission, they require the input to be organized in a regular grid, being incapable to transform any input arbitrary meshes without remeshing. Instead, GFT can directly handle any input mesh connected by a graph, including meshes presented in this work. (ii) Since each segment is relatively smooth, applying GFT can produce optimal results.

Suppose we need to apply GFT to $S_{i,j}$, i.e. the j-th segment of the i-th cluster, where $i = 1, \ldots, K$, $j = 1, \ldots, S$. First, we compute the Laplacian matrix $L = D - A$ of $S_{i,j}$, where A is an adjacency matrix encoding the connectivity of every pair of vertices within $S_{i,j}$. D is a diagonal matrix with each diagonal element encoding the degree (number of immediate neighbors) of each vertex within $S_{i,j}$. L, D, A are all square matrices and their orders are $\|S_{i,j}\| \times \|S_{i,j}\|$, where $\|S_{i,j}\|$ denotes the number of vertices of the j-th segment in the i-th cluster. Then, we compute the eigenvectors V of matrix L, where V is also a square matrix and its orders are $\|S_{i,j}\| \times \|S_{i,j}\|$. Subsequently, we project the x, y, z coordinates of all the vertices in the same segment to the corresponding bases to obtain GFT coefficients. Finally, we retain the first k_n group of coefficients, corresponding to the most significant bases. Determining k_n is related to the expected compression quality defined by the user and the resource availability. Further discussion of the selection of k_n is presented in Section 7.

6.2 SPIHT Coding

In this part, we describe how we encode the selected GFT coefficients. Instead of applying some classical coding algorithms such as Huffman coding and predictive coding, we adopt embedded coding. Embedded coding encodes information based on importance and supports multiple bit rates

for decompression. It is done by encoding most important information into the initial part of the output code stream and gradually adding information into the stream based on the reduction in importance. This allows lower rate streams to be embedded in a high rate code stream. Hence, embedded coding supports progressive transmission, where the encoder can stop at any point and the decoder can also stop at any time when receiving the code stream. In this way, embedded coding can precisely control the coding rate. Once the coding distortion or bit rate meets user requirements, we can stop the decoding process accordingly. So the embedded coding is suitable to support progressive transmission of mesh animations in this work.

The most representative algorithms of embedded coding are EZW, SPIHT and Set Partition Embedded block (SPECK) [24], where EZW coding algorithm is important to wavelet encoding technology. It achieves effective organization of wavelet coefficients by introducing zero-tree data structure. SPIHT coding algorithm makes some improvements on data structure on the basis of Embedded Zerotree Wavelets Encoding (EZW) and achieves a better performance. Similar to EZW, SPIHT is based on the zero-tree structure and additionally puts a tree node and all successor nodes into a cluster. It finishes the embedded coding by initializing, sorting, refining, and updating the quantization step. SPIHT coding is not only simple in data structure but can also achieve a high coding efficiency without any training; it also produces a good reconstruction quality.

7 RESULTS AND APPLICATIONS

In this section, we show the experiment results of our method and compare our method with state-of-the-art mesh animation compression methods.

7.1 Animation Data

In our experiments, we used eight well-known mesh animation sequences as the test set, including cow, dance, deer, snake, chicken, man, tiger and dog [3]. The geometric features of these models are described in Table 2. It is clear that major portion of data size is contributed by geometry information rather than topology information in each test model. Note that the data sizes of geometry and topology information are uncompressed sizes.

Table 2. Test models used in our experiments

Animation Sequence	cow	snake	deer	dance	chicken	man	dog	tiger
Number of vertices	2904	9179	2969	7061	3030	1070	1179	587
Number of faces	5804	18354	5832	14118	5664	2176	3504	1126
Number of frames	204	134	201	201	400	301	201	291
Geometry information (KB)	12689	72499	5722	48230	19064	3340	4060	1821
Topology information (KB)	21	71	22	48	22	8	13	5

7.2 Compression and Reconstruction

To compare the reconstruction errors of different methods, an effective metric is required. We choose the Karni-Gotsman error (KG_{error}) [23] because it is specifically designed for mesh animation sequences. The metric works on matrices representing mesh animation, where matrix columns describe trajectories of respective vertices of the animation. It is used for evaluating the amount of distortions caused by compression for a mesh animation sequence. Also, the state-of-the-art methods [22, 28, 42] in our comparison relied on this metric to evaluate their results. There are other metrics available [11], such as D_a, 4D Hausdorff and PerceptualDiff. They are not suitable metrics due to lack of rotation invariance, high computation cost and pixel-based measurement, respectively. Alternatively, STED measures both local vertex and temporal edge changes, serving as a perceptual

metric. This metric does not provide a fair measurement to our method, since we involve mesh animation simplification for supporting progressive transmission. Therefore, KG_{error} is preferable for assessing the effectiveness of our algorithm, which is defined as follows:

$$KG_{error} = 100 \times \frac{||B - B'||}{||B - E(B)||}\% \tag{6}$$

As in Eqn. (6), B is a matrix composed of $3N$ rows and F columns of geometry information, where N is the number of vertices and F is the number of frames in the animation sequence. B' is also a $3N \times F$ matrix, representing the reconstructed animation sequence. $E(B)$ is a matrix of the same dimensions as B, in which the values have been replaced by per-frame averages. The reconstruction error is calculated with Frobenius norm.

To compare the effectiveness of compression, we define r as in Eqn. (7) to represent the bit size per vertex per frame (bpvf):

$$r = \frac{M}{N \cdot F} \tag{7}$$

where M represents the total amount of bits after compression, N is the number of all the vertices, and F is the number of frames in a mesh animation sequence. In our method, we first perform a temporal clustering along an animation sequence. To perform spatial segmentation, we then divide mesh vertices of each cluster into S segments. In order to investigate the effects of different S, Table 3 shows the compression rate (r) and the reconstruction error (KG_{error}) of different animation sequences when different numbers of segments are applied. We attempt to identify a good number of segments for each mesh animation by considering the tradeoff between r and KG_{error}. For instance, both the cow and snake animations reduce in KG_{error} when their segment numbers are being increased, which is benefitical. Meanwhile, the compression rates of them become larger, being less favorable. We observe that when the segment number of both mesh animations are increased further after 6 segments, KG_{error} will no longer drop significantly while r may still grow gradually. We conclude that 6 is the best segment number for both the cow and snake animations. For dance and deer animations, the best segment numbers are 10 and 7, respectively. From the table, we can see that with the increase in the number of segments, the reconstruction error is reduced and the compression rate becomes larger in general. This makes sense because the more segments we divide, data within each segment become more coherent, which help improve the reconstruction quality.

Fig. 6 shows the influence of k_n on the reconstruction error of the test sequences, where k_n denotes the number of GFT bases and their corresponding coefficients used for mesh animation reconstruction. As shown in the figure, the reconstruction error becomes smaller with the increase in k_n. The reconstruction quality of a compressed animation sequence will be increased when larger k_n is used. When $k_n = 1$, it means only one GFT basis and the corresponding coefficient are chosen for reconstructing the coarsest version of mesh animation. Such reconstruction is suffered from a large error since no other GFT bases and their corresponding coefficients are involved. However, since our work adopts GFT rather than other transformation methods, mesh animation reconstruction error can be reduced significantly when a relatively small k_n is used, e.g., $k_n = 7$. In practical applications, we should take both user-desired compression quality and resource availability into account when we determine k_n. For example, we should select a relatively large k_n if we want to reconstruct fine animation details of a heart movement for medical visualization. On the other hand, if an application emphasizes on performance rather then producing very high quality animation, such as online games [27], we may select a relatively small k_n to improve compression rate, given that a certain level of reconstruction quality can be achieved.

Table 3. Compression performance with different number of segments

Animation Sequence	S	KG_{error}	r
cow	3	0.0957	0.48
	6	0.0783	0.56
	9	0.0752	0.65
dance	4	0.0782	0.33
	10	0.0574	0.42
	12	0.0752	0.44
deer	3	0.0815	0.63
	7	0.0504	0.72
	9	0.0498	0.75
snake	3	0.0492	0.47
	6	0.0325	0.55
	9	0.0321	0.64
chicken	2	0.0621	0.53
	4	0.0489	0.61
	6	0.0494	0.69
man	2	0.0183	0.39
	3	0.0177	0.43
	4	0.0175	0.49
tiger	4	0.0415	0.37
	7	0.0236	0.48
	9	0.0231	0.56
dog	2	0.0523	0.62
	3	0.0369	0.69
	5	0.0375	0.83

As described in Section 5, α represents the weight of the geometric information during the compression process. Fig. 7 illustrates the influence of weight α on the reconstruction quality. In the spatial segmentation, preserving spatial coherence offers better performance to user-defined and progressive transmission, so we set $\alpha \geq 0.5$. Here, we use the proportion of topologically unconnected vertices to measure the reconstruction quality. Such vertices refer to those locating at topologically disjointed mesh parts, while spatial coherence measures proximity of mesh vertices in each segment. From the figure, we observe that for the majority of the test animation sequences, the proportion of topologically unconnected vertices will not be further reduced when $\alpha > 0.7$, indicating α can no longer significantly affect the proportion. Because adjacent vertices' motions must be spatially similar, the weight of curvature and torsion also contribute to the weight of vertex coordinates.

7.3 Comparison with State-of-the-Art Methods

We choose three state-of-the-art methods [22, 28, 42] for comparison. The work of [42] compresses geometry data by removing redundancy among vertices. [28] alternatively groups similar frames together and perform intra-cluster compression. On the other hand, [22] supports progressive coding.

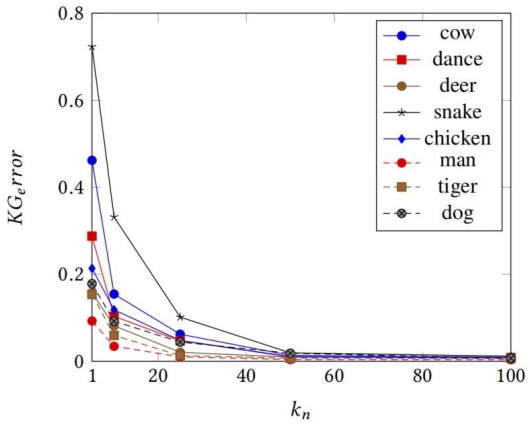


Fig. 6. The influence of different k_n on reconstruction quality

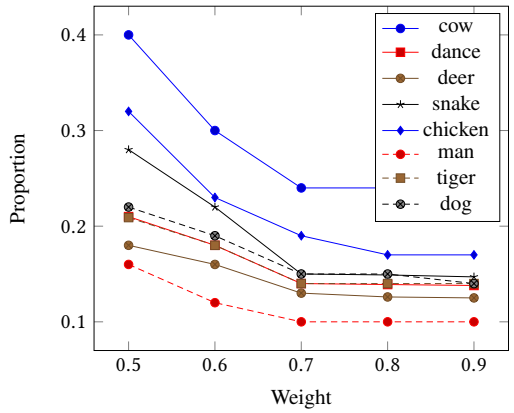
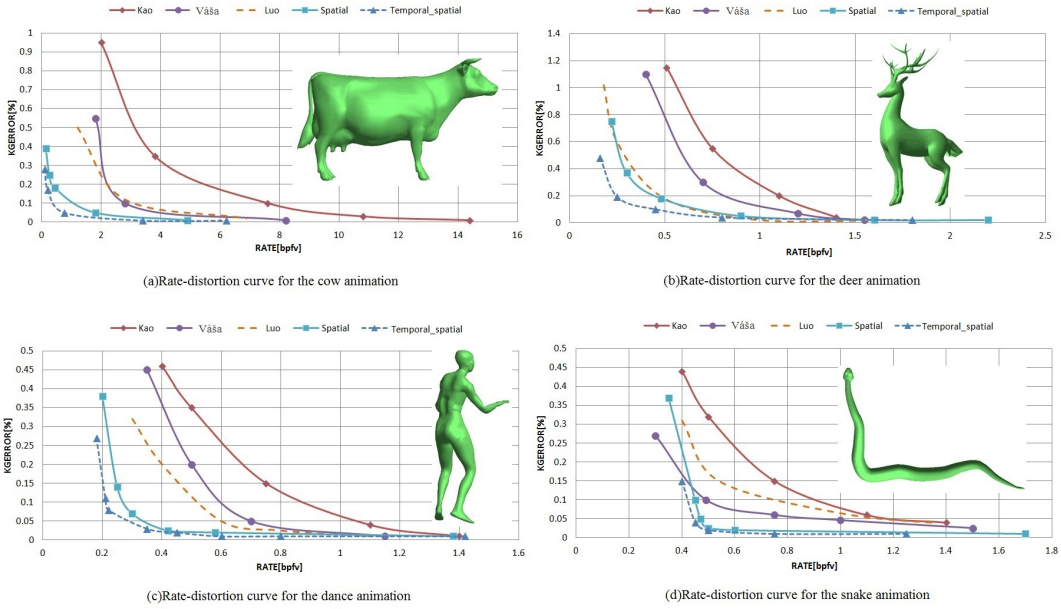


Fig. 7. The influence of different α values on the proportion of topologically unconnected vertices in the reconstructed results

We choose rate-distortion (R-D) curves to measure the quality of a mesh animation sequence compression algorithm. Fig. 8 and Fig. 9 compare R-D curves of our method against the state-of-the-art methods by running tests on the aforementioned 8 test models. In order to investigate the importance of temporal clustering, we further compare our method with and without temporal clustering. In the figure, "Temporal_spatial" denotes running our method with both temporal clustering and spatial segmentation, while "Spatial" denotes running our method without performing temporal clustering. Unsurprisingly, in terms of R-D curve, "Temporal_spatial" is clearly better than "Spatial", validating the contributions of our proposed method.

Figs. 8 and 9 also clearly show that our method performs better than the existing methods [22, 28, 42]. Our method consistently produces a lower KG error under different compression rates for all the tested animation sequences. Also, our method generally produces a higher compression rate than the existing methods, under the same reconstruction quality. However, the result of the snake model by our method is not good enough when $r < 0.45$. It is because the topology information of the snake model occupies a relatively large storage space, substantially reducing the available data rate



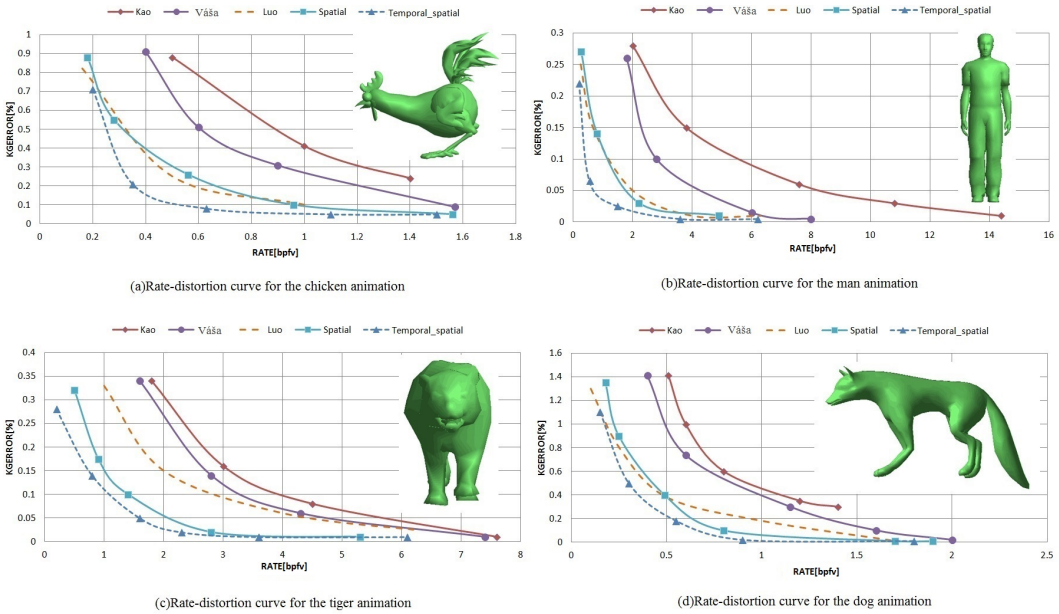
(a)Rate-distortion curve for the cow animation

(b)Rate-distortion curve for the deer animation

(c)Rate-distortion curve for the dance animation

(d)Rate-distortion curve for the snake animation

Fig. 8. Performance comparison in terms of the Rate-Distortion curve (R-D curve) among Kao's [22], Váša's [42], Luo's [28], and our method without and with temporal clustering.



(a)Rate-distortion curve for the chicken animation

(b)Rate-distortion curve for the man animation

(c)Rate-distortion curve for the tiger animation

(d)Rate-distortion curve for the dog animation

Fig. 9. Performance comparison in terms of the Rate-Distortion curve (R-D curve) among Kao's [22], Váša's [42], Luo's [28], and our method without and with temporal clustering.

for holding the geometry information. Hence, our method had a relatively high reconstruction error

in such a situation. When handling the snake model, Luo's method [28] also performs worse than Váša's method [42]. Because Luo's method divides a mesh animation into clusters and compresses each by exploiting its intra-similarity. Without a good amount of geometry information available, such a compression cannot perform well. With our method, this is not a critical issue. Particularly, when the length of an animation sequence becomes large, its topology information will become insignificant as such information occupies a constant data size regardless the length of the sequence. In other words, more space will be available for storing geometry information. We also observe that our method works best with the deer animation. Because this animation is featured with many repeated animation frames, which benefits from the temporal clustering of our method.

On the other hand, we also show compression and decompression timing of different methods in Table 4. All the methods were implemented with Matlab and ran on the same off-the-shelf computer with Intel Xeon 3.90 GHz PC and 16 GB RAM. As shown in Table 4, our method runs much faster than [42] in term of both compression and decompression, but not as efficient as [28] for compression. Because our method can support progressive transmission, we can transmit the base animation from the server to a client first, and then transmit more details to refine the reconstruction quality. Note that [22] is not involved in this part of comparison since the method focuses only on progressive coding rather than compression.

Table 4. Compression and decompression time [seconds]

		Cow	Deer	Dance	Snake	Chicken	Man	Tiger	Dog
[42]	Compression	26.23	25.13	24.61	20.46	23.44	18.64	23.84	25.76
	Decompression	3.24	4.52	4.73	4.94	3.14	2.75	3.72	5.13
[28]	Compression	0.67	0.69	0.73	0.58	0.87	0.53	0.97	0.72
	Decompression	0.61	0.58	0.74	0.77	0.82	0.57	0.92	0.75
Our method	Compression	7.97	7.82	8.43	8.12	8.45	6.24	7.14	7.93
	Decompression	0.48	0.43	0.64	0.47	0.57	0.41	0.58	0.45

In our method, we have to transmit three types of information from the server to a client, namely topology information, temporal-spatial clustering results, and selected coefficients, in order to reconstruct the base mesh animation. Here, we select 2% coefficients to reconstruct the base mesh animation. During the progressive reconstruction, we transmit more coefficients, e.g., we transmit fine details by adding 20% coefficients each time. As shown in Table 5, if we only transmit the information for the base mesh animation, our method uses significantly less time on decompression than [28]. It is reasonable because when we reconstruct the base mesh animation at the client side, we can transmit more coefficients to improve the reconstruction quality at the same time.

Table 5 compares the time used for progressive transmission by our method and Kao's method [22], which is the most recent method supporting progressive coding for mesh animation to our knowledge. We conducted experiments under a local area network with a bandwidth of 4.49Mbit/sec. Table 5(a) shows the breakdown of model transmission time with our method. The first column ("Base mesh anim. (2%)") shows the time used to transmit topology information, temporal-spatial clustering results, and the selected 2% of coefficients of each model we have tested. The next column ("Add 20%") shows the added time for transmitting 20% of coefficients on top of the base mesh animation to improve rendering quality. Moreover, each of the remaining columns shows both the added transmission time for sending a specific percentage of additional coefficients on top of the base mesh animation and the added transmission time (in the parentheses) for sending 20% of additional coefficients on top of that of the previous column. Because the coefficients are arranged in a descending order according to their importance, the coefficients added subsequently will be more closer to zero, becoming computationally more efficient for mesh animation reconstruction. In this way, we can achieve real time transmission to a certain extent.

Table 5. Time used for progressive transmission [milliseconds]

	Base mesh anim. (2%)	Add 20%	Add 40%	Add 60%
Cow	320	67	115(48)	138(23)
Deer	355	75	120(45)	152(32)
Dance	451	84	135(51)	171(36)
Snake	487	63	95(32)	109(14)
Chicken	347	65	108(43)	129(21)
Man	302	61	86(25)	95(9)
Tiger	324	75	127(52)	165(38)
Dog	315	69	99(30)	126(27)

(a) Our Method

	Base mesh anim. (20%)	Add 40%	Add 60%
Cow	284	849(565)	1102(253)
Deer	292	870(578)	1095(225)
Dance	341	986(645)	1206(220)
Snake	374	1107(733)	1397(290)
Chicken	293	868(575)	1069(201)
Man	260	771(511)	991(220)
Tiger	288	873(585)	1065(192)
Dog	279	836(557)	1073(237)

(b) Kao's method [22]

In contrast, Kao's method [22] performs progressive coding of mesh topology to support progressive transmission. Both topology information and geometry information are therefore required to progressively transmit. In contrast, our method maintains static topology information, which is only required to transmit once. As a result, [22] signficantly needs more time to support progressive transmission than our method during the course of mesh animation reconstruction, as shown in Table 5(b). It also cannot generate a much concise base mesh animation than our method, where they require 20% of geometry information to be encoded in the base mesh animation.

7.4 Selected Applications

7.4.1 User-defined Transmission. User-defined transmission is a mechanism to control how much data is transmitted. It arranges a suitable amount of mesh data to transmit. When the reconstruction quality satisfies the user requirements, we can stop transmitting more data. Having such a mechanism is important, as the requirement of reconstruction quality varies among different applications. Our temporal-spatial clustering method can generate segments, each of which comprises vertices exhibiting similar motions both in space and time. So, it may be unnecessary for the user to always request all frames of a segment for transmission as long as the user has already received at least one frame from the segment. The other frames can be approximated based on that frame if the reconstructed quality is acceptable.

To support user-defined transmission, we classify those segments obtained by temporal-spatial clustering into different classes, namely classes of drastic motion, gentle motion, and relatively steady motion. Our method handles the three classes differently. For a drastic motion class, we transmit all of its frames, and for the classes of gentle and relatively steady motions, we transmit them with an interval of k_1 and k_2 frames, respectively. Missing frames will be substituted by an available frame

which is of high proximity in the same segment. The values of k_1 and k_2 can be application-specific, specified by the user. For example, in a mobile application, we can dynamically compute them based on the available network bandwidth, display resolution, and cache size. In this way, we can significantly reduce the amount of transmitted data.

7.4.2 Progressive Transmission. Since the SPIHT algorithm can produce an embedded bit stream, we can decompress and reconstruct the mesh even if there is an interruption of flow at any point. Therefore, we adapt the SPIHT algorithm to support progressive transmission. In our method, we support progressive transmission by controlling the selected number (k_n) of GFT coefficients to satisfy the reconstruction requirement specified by users. The encoder running at the server side can stop at any point and the decoder at the client side can also stop at any time while receiving the data stream. More GFT coefficients can be transmitted to enhance the reconstruction result if more time or network bandwidth resources are available. Specifically, at the client side, the user reconstructs the animation based on the information received from the server side, which includes topology information, temporal-spatial clustering results, and some selected coefficients. In order to support progressive transmission, we first transmit a data packet including topology information, temporal-spatial clustering results, and a small number of coefficients to the client side, and then progressively improve the reconstruction quality by transmitting more coefficients. One progressive transmission example by our approach is shown in Fig. 10. For the cow animation, its topology information and temporal-spatial results occupy about 19KB storage space, all the others are coefficients. Obviously, with the increase in the transmitted data, the reconstructed animation can be refined gradually.

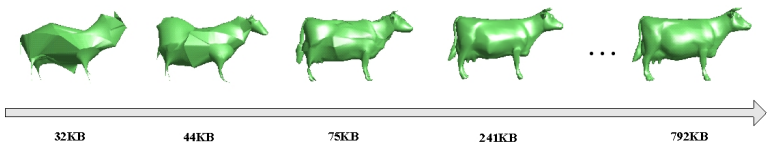


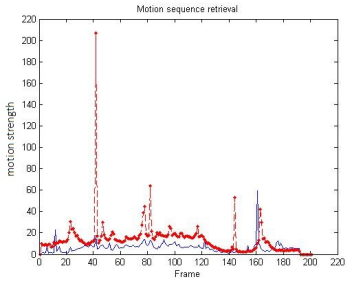
Fig. 10. A progressive transmission example (the cow model) by our approach

7.4.3 Mesh Animation Retrieval. Mesh animation retrieval has caught increasing attention in the community. Motion patterns of different mesh animations could be very similar. For example, given the dancing mesh animations of two different people, although their topology and geometry information are different, their motions could be very similar. Such animations should be grouped togther to a certain extent.

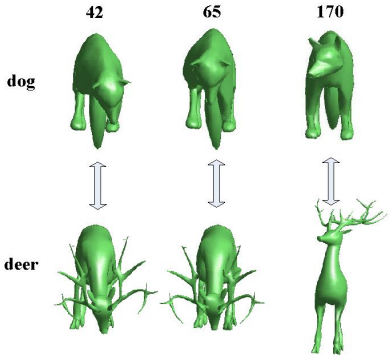
Fig. 11(a) plots the motion strength across all frames of dog and deer animations. The curves in red and blue are for the dog and the deer models, respectively. From the motion strength of the two different animations, we can see their motions are similar though their shapes are substantially different. From the motion aspect, they could be grouped into one cluster. According to Fig. 11(a), we found that the frames with the biggest, the average, and the smallest differences in motion strength of the two animations are the #42nd, #65th, and #170th frames, respectively. The corresponding graphs are shown in Fig. 11(b), from which we can see their motions are indeed very similar.

8 DISCUSSION AND CONCLUSION

We introduce a novel compression method to improve the compression rate by removing temporal-spatial redundancies from an animated mesh sequence. In our method, curvature and torsion are chosen to measure the motion of a vertex. Instead of clustering frames based on their temporal



(a) motion strength of all frames in the dog (red) and the deer (blue) animations.



(b) Comparisons of the frame with the biggest difference, the frame with the average difference, and the frame with the smallest difference between the dog and the deer animations.

Fig. 11. Illustration of potential mesh animation retrieval application, using the dog and the deer animations as an example.

adjacency, we cluster frames by motion similarity. Our method also ensures mesh coherence in the spatial domain, which is favorable to support user-defined transmission.

In addition, we perform intra-cluster compression based on GFT and SPIHT coding. Because every segment within a cluster exhibits similar motion, GFT can effectively reduce the amount of data using to encode those segments. Also, SPIHT is applied to code the GFT coefficients to support progressive transmission. Our method is computationally efficient and outperforms state-of-the-art methods for compressing mesh animation sequences.

A limitation of our approach is raised from compressing mesh animations without repeated posture or motion patterns, where our temporal clustering method may not significantly improve compression effectiveness. But in fact, if the length of an animation sequence is long enough, similar model poses or motions may likely exist, and that the sequence can be effectively compressed by our method. Another limitation is that we generate GFT bases directly at the client side, which consumes about two seconds in our experiments. In future work, we plan to improve spatial segmentation in order to make this bases recovery more efficient. Besides, the number of segments is required to manually specify in our current method. As shown in Table 3, the segment number is a major factor to determine compression performance, we therefore would also like to investigate a way to identify segment number automatically [19].

REFERENCES

[1] Jae-Kyun Ahn, Yeong Jun Koh, and Chang-Su Kim. 2013. Efficient fine-granular scalable coding of 3D mesh sequences. *IEEE Transactions on Multimedia* 15, 3 (2013), 485–497.

[2] Marc Alexa and Wolfgang Müller. 2000. Representing animations by principal components. In *Computer Graphics Forum*, Vol. 19. Wiley Online Library, 411–418.

[3] Rachida Amjoun. 2009. *Compression of static and dynamic three-dimensional meshes*. Ph.D. Dissertation. Universität Tübingen.

[4] Rachida Amjoun, Ralf Sondershaus, and Wolfgang Straßer. 2006. Compression of complex animated meshes. In *Advances in Computer Graphics*. Springer, 606–613.

[5] Rachida Amjoun and Wolfgang Straßer. 2009. Single-rate near lossless compression of animated geometry. *Computer-Aided Design* 41, 10 (2009), 711–718.

[6] Marco Attene, Bianca Falcidieno, Michela Spagnuolo, and Jarek Rossignac. 2003. Swingwrapper: Retiling triangle meshes for better edgebreaker compression. *ACM Transactions on Graphics (TOG)* 22, 4 (2003), 982–996.

[7] Chandrajit L Bajaj, Valerio Pascucci, and Guozhong Zhuang. 1999. Single resolution compression of arbitrary triangular meshes with properties. *Computational Geometry* 14, 1 (1999), 167–186.

[8] M Oguz Bici and Gozde B Akar. 2011. Improved prediction methods for scalable predictive animated mesh compression. *Journal of Visual Communication and Image Representation* 22, 7 (2011), 577–589.

[9] Jae W Cho, Sébastien Valette, Ju H Park, Ho Y Jung, and Rémy Prost. 2010. 3-D mesh sequence compression using wavelet-based multi-resolution analysis. *Appl. Math. Comput.* 216, 2 (2010), 410–425.

[10] Mike M Chow. 1997. Optimized geometry compression for real-time rendering. In *Visualization'97., Proceedings*. IEEE, 347–354.

[11] M. Corsini, M. C. Larabi, G. Lavoué, L. Petříša, and K. Wang. 2013. Perceptual Metrics for Static and Dynamic Triangle Meshes. *Computer Graphics Forum* 32, 1 (2013), 101–125.

[12] Clément Courbet and Céline Hudelot. 2011. Taylor prediction for mesh geometry compression. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 139–151.

[13] T. W. Fry and S. A. Hauck. 2005. SPIHT Image Compression on FPGAs. *IEEE Transactions on Circuits and Systems for Video Technology* 15, 9 (Sept 2005), 1138–1147.

[14] Stefan Gumhold and Wolfgang Straßer. 1998. Real time compression of triangle mesh connectivity. In *ACM Computer graphics and interactive techniques*. ACM, 133–140.

[15] Sumit Gupta, Kuntal Sengupta, and Ashraf A Kassim. 2002. Compression of dynamic 3D geometry data using iterative closest point algorithm. *Computer Vision and Image Understanding* 87, 1 (2002), 116–130.

[16] Igor Guskov and Andrei Khodakovsky. 2004. Wavelet compression of parametrically coherent mesh sequences. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 183–192.

[17] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis* 30, 2 (2011), 129–150.

[18] Lawrence Ibarria and Jarek Rossignac. 2003. Dynapack: space-time compression of the 3D animations of triangle meshes with fixed connectivity. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 126–135.

[19] Timor Kadir and Michael Brady. 2001. Saliency, scale and image description. *International Journal of Computer Vision* 45, 2 (2001), 83–105.

[20] Felix Kälberer, Konrad Polthier, Ulrich Reitebuch, and Max Wardetzky. 2005. FreeLence-Coding with Free Valences. In *Computer Graphics Forum*, Vol. 24. Wiley Online Library, 469–478.

[21] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. 2002. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 7 (2002), 881–892.

[22] C. K. Kao, B. S. Jong, and T. W. Lin. 2010. Representing Progressive Dynamic 3D Meshes and Applications. In *Pacific Conference on Computer Graphics and Applications*. 5–13.

[23] Zachi Karni and Craig Gotsman. 2004. Compression of soft-body animation sequences. *Computers & Graphics* 28, 1 (2004), 25–34.

[24] Umut Konur, Uluğ Bayazit, Fikret Gürgen, and Ozgur Örcay. 2006. Compressing Mesh Geometry using Spectral Methods and a Set Partitioning Approach. In *IEEE Signal Processing and Communications Applications*. IEEE, 1–4.

[25] 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 (June 2017), 811–821.

[26] Jerome Edward Lengyel. 1999. Compression of time-dependent geometry. In *ACM Symposium on Interactive 3D graphics*. ACM, 89–95.

[27] Frederick W. B. Li, Rynson W. H. Lau, Danny Kilis, and Lewis W. F. Li. 2011. Game-on-demand:: An Online Game Engine Based on Geometry Streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications* 7, 3, Article 19 (Sept. 2011).

[28] Guoliang Luo, Frederic Cordier, and Hyewon Seo. 2013. Compression of 3D mesh sequences by temporal segmentation. *Computer Animation and Virtual Worlds* 24, 3-4 (2013), 365–375.

[29] Khaled Mamou, Titus Zaharia, and Françoise Prêteux. 2006. A skinning approach for dynamic 3D mesh compression. *Computer Animation and Virtual Worlds* 17, 3-4 (2006), 337–346.

[30] Khaled Mamou, Titus Zaharia, Françoise Prêteux, Nikolce Stefanoski, and Jörn Ostermann. 2008. Frame-based compression of animated meshes in MPEG-4. In *IEEE International Conference on Multimedia and Expo*. IEEE, 1121–1124.

[31] K Muller, A Smolic, M Kautzner, and P Eisert. 2005. Predictive compression of dynamic 3D meshes. In *IEEE Image Processing*. I–621–4.

[32] K. Muller, A. Smolic, M. Kautzner, and T. Wiegand. 2006. Rate-Distortion Optimization in Dynamic Mesh Compression. *Applied Energy* 87, 4 (2006), 1122–1133.
[33] Frédéric Payan and Marc Antonini. 2005. Wavelet-based compression of 3d mesh sequences. In *ACIDCA-ICMI'2005*.
[34] Frédéric Payan and Marc Antonini. 2007. Temporal wavelet-based compression for 3D animated models. *Computers & Graphics* 31, 1 (2007), 77–88.
[35] Subramanian Ramanathan, Ashraf A Kassim, and Tiow-Seng Tan. 2008. Impact of vertex clustering on registration-based 3D dynamic mesh coding. *Image and Vision Computing* 26, 7 (2008), 1012–1026.
[36] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. 2005. Simple and efficient compression of animation sequences. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM, 209–217.
[37] Nikolce Stefanoski and Jorn Ostermann. 2006. Connectivity-Guided Predictive Compression of Dynamic 3D Meshes. In *International Conference on Image Processing*. 2973–2976.
[38] 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.
[39] 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.
[40] L. Váša and J. Dvořák. 2018. Error Propagation Control in Laplacian Mesh Compression. *Computer Graphics Forum* 37, 5 (2018), 61–70.
[41] Libor Váša and Václav Skala. 2009. Cobra: Compression of the basis for PCA represented animations. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 1529–1540.
[42] Libor Váša and Václav Skala. 2010. Geometry-Driven Local Neighbourhood Based Predictors for Dynamic Mesh Compression. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1921–1933.
[43] Jeong-Hyu Yang, Chang-Su Kim, and Sang Uk Lee. 2005. Progressive coding of 3D dynamic mesh sequences using spatiotemporal decomposition. *IEEE International Symposium on Circuits and Systems* (2005), 944–947 Vol. 2.

Received August 2017; revised August 2018; accepted December 2018