

Received March 21, 2020, accepted April 12, 2020, date of publication April 22, 2020, date of current version May 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2989430

Motion Recurring Pattern Analysis: A Lossless Representation for Motion Capture Databases

PENGJIE WANG¹, JIANG WANG², XIAOMING WEI¹, JIANA MENG¹, AND JING XUN¹

¹School of Computer Science, Dalian Minzu University, Dalian 116600, China

²PYDDot Technology Company, Ltd., Shenzhen 518000, China

Corresponding author: Xiaoming Wei (xmwei@dlnu.edu.cn)

This work was supported in part by the Liaoning Innovative Talents Support Plan under Grant LR2016071, and in part by China Postdoctoral Science Foundation under Grant 2014M561228.

ABSTRACT In this paper, we propose the motion recurring pattern analysis (MRPA) method for the lossless representation of a motion database at the segment level instead of the motion degree of freedom (DOF) level. First, we concatenate all the motions into a long sequence in the motion database, and we discover similar posture paths by building a matching trellis structure based on the randomized k-d tree. Second, horizontal segments of paths are suitably refined, based on a self-organizing map, to obtain the optimized segmentation for maximum compression gains. Third, by using the path as a connection agent, these segments are clustered into a forest of trees. With this forest structure, we obtain the prediction residuals (the differences between the nonroot branches and their parents), and the differences between neighboring residuals are encoded under floating-point compression. Relative to previous lossless compression methods, our approach can achieve a higher compression ratio with comparable decompression time costs.

INDEX TERMS Motion capture, animation compression, lossless compression, character animation.


I. INTRODUCTION

With the development of motion capture techniques, human motion data are widely used beyond the conventional fields of games and animation. These data are used in fields such as the automotive industry, arts, sports, virtual reality, and remote interaction in augmented reality. As the vast collections of motion capture data continue to grow, it becomes crucial to efficiently store and transmit these data. Previous works [1]–[20] on motion compression have focused on two types of methods, namely, methods that reduce the redundancy of the time domain and those that reduce the redundancy of the space domain. However, in a large motion database, there is a third way to reduce redundancy: the extraction of recurring similar motion patterns across a large database. These matched motion patterns can make coding more efficient. However, few works have addressed motion compression with motion pattern discovery and analysis.

Furthermore, although most state-of-the-art compression methods have shown satisfactory compression performance, they are based on lossy compression; hence, the original data are modified, resulting in errors. These errors can result in various perceptual artifacts, such as the well-known

foot-skating artifacts [21]. Although inverse kinematics (IK) can be introduced to reduce these errors [1], [2], [14], [17], the residual errors can still degrade the visual quality of the motion data, particularly for motion files undergoing multiple compression and decompression processes. Furthermore, the time-consuming IK process greatly degrades the decompression performance.

To address these problems, we propose in this paper a lossless compression method for a motion database by exploring the recurring pattern analysis in the motion database. Our method is composed of three steps. First, all motions in the database are concatenated into a long sequence. Then, for each node along this sequence, we search for its best matches, such as its nearest neighbors, in the database based on a randomized k-d tree. These matches are listed a column below the node of this sequence row, and a matching trellis is built. We then find continuous similar posture paths through the path growing process in the matching trellis. Second, a similar posture path can map two segments: a horizontal and a vertical segment. All the horizontal segments constitute a segmentation of the database sequence. However, this segmentation is not optimal, and overlaps exist among them. We propose a sequence segmentation method based on the self-organizing map (SOM) using horizontal segments as the input layer and equally split length segments as the output

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaoqing Pan .

layer. Third, with this optimized segmentation, each segment is either the root of a new tree or a descendant, a segment that is assigned to one branch of a tree based on its similarity with the existing trees. These trees constitute a forest. For each tree, we take the father node as the prediction of its descendants and obtain the prediction residuals as the difference between the prediction and real data. We then encode the residual differences for maximum compression gains. The contribution of our work can be summarized as follows:

- We propose a forest structure to best predict the motion segments instead of the motion degrees of freedom (DOFs). Then, the differences of the residuals are encoded with floating-point compression.
- We introduce the randomized k-d tree to find the nearest neighbor matches to improve the efficiency of the trellis building process.
- Based on the SOM, we propose a motion segmentation method that can suitably balance the similar posture paths for maximum compression gains.

II. RELATED WORK

In this section, we briefly summarize existing works on animation compression and floating-point compression.

A. ANIMATION COMPRESSION

Animation data are high-dimensional data that often present high spatial and temporal coherence. Nearly all animation compression methods exploit these two types of coherence to reduce the data size.

Previous works on animation compression have focused mainly on compressing animated meshes [22]–[28]. These works include [28], which adopted an octree to represent motion capture data; [23], [26], which performed vertex-wise motion prediction; [24], [25], which applied the principal component analysis (PCA) technique; [22], which used wavelet coding techniques; and [27], which encoded nonisomorphic animated mesh sequences.

The compression of motion capture data is a relatively new research area. Arican [1] used Bezier curves to represent motion clips and then applied a clustered PCA to reduce the dimensionality. Environmental contacts were encoded separately based on discrete cosine transformation (DCT). Khan *et al.* [9] proposed using the multidimensional quadratic Bezier curve break-and-fit method to compress motion capture data. Hou *et al.* [29] first segmented motion data into subsequences. Then, they exploited the strong low-rank characteristics within and among the subsequences of motion capture data to achieve a compact representation. Liu *et al.* [16] first segmented a motion sequence into subsequences of simple motions and then compressed each subsequence using PCA approximation and further compressed the resulting PCA data by selecting and storing only the keyframes. Vasa *et al.* [18] proposed combining PCA with Lagrange multipliers techniques to obtain a satisfactory balance of precision and distortion.

Tournier *et al.* [17] used a principal geodesics analysis (PGA) to build a descriptive model of pose data, keeping only the leading principal geodesics. This model was then used in an IK system to synthesize poses that very closely matched the end-joint constraints and the interior joint positions to the input data. Given this pose model, only the compressed end-joint trajectories and the position and orientation of the root joint need to be stored to recover the motion using IK. Zhu *et al.* [20] precompressed the motion data using the method proposed by Tournier *et al.* [17] to compress motion data. Then, they proposed a quaternion space sparse decomposition model that decomposes the rotational motion into the dictionary part and the weight part. Finally, these data were encoded with arithmetic coding. Hou *et al.* [8] proposed a human motion capture data tailored transform coding method by computing a set of data-dependent orthogonal bases, while Kruger *et al.* [12] and Hou *et al.* [30] proposed methods based on tensor decomposition.

Beaudoin *et al.* [2] adapted standard wavelet compression on joint angles by considering the process of selecting wavelet coefficients to be a discrete optimization problem within a tractable search space adapted to the nature of the data. For contacts with the environment, they used optimized wavelet-based compression and IK correction. This algorithm focuses on short and recomposable animation clips. Lee *et al.* [13] proposed a human motion compression framework based on a multiresolution wavelet. Firouzmanesh *et al.* [5] proposed the perceptually guided compression method, which incorporates wavelets with attention stimulating factors.

Chattopadhyay *et al.* [3] proposed a power-aware algorithm for mobile devices, which exploits the motion data indexing concept. They derived each floating number index from the statistical distribution of the floating-point numbers in the motion matrix. Since this integer index number takes much fewer data bits, the motion capture data can be significantly compressed. Han *et al.* [7] presented a motion capture compression method based on simple polynomial curve-fitting techniques to efficiently store and transfer motion databases with mobile phones.

Gu *et al.* [6] proposed a method for compressing human motion capture data based on hierarchical structure construction and motion pattern indexing. They first organized the 3D markers as a hierarchy in which each node corresponds to a meaningful part of the human body and is therefore coded separately. For each meaningful part, the motion pattern database is built, and the sequence of the motion capture data can be efficiently represented as a series of motion pattern indices. Park [31] presented a motion rearrangement method that shares a similar concept with the method proposed by Gu *et al.* Chew *et al.* [32] presented a fuzzy clustering algorithm for virtual character animation representation. They proposed mapping a virtual character animation as an image and using a modified motion filter to minimize the visual discontinuity and distortion.

Kwak *et al.* [10] presented their version of low-delay motion capture compression using a reordered data frame and then made predictions from both long-term and short-term reference frames. In 2017, Kwak *et al.* [11] improved their method by proposing bit allocation to long-term and short-term reference frames and postprocessing of temporal low-pass filtering. Based on the parallelogram predictor [33], [34] in geometric compression, Wang *et al.* [19], [35] proposed an alpha parallelogram predictor for effectively predicting the DOFs of motion capture data. They stored the alpha parameters with a carefully designed lookup table and the prediction residuals were then encoded using the adapted floating-point compression method [36].

Lin *et al.* [14] proposed the repeat motion analysis method to achieve a compact representation of motion capture data. Based on the self-distance metric of the match web [37], they extracted primary clips and repeated clips and they also fit the trajectories of the projected coefficient or coefficient differences using Catmull-Rom splines. Because the spline-based approximation at the subspace might not suitably preserve high-frequency motions, such as a foot contacting with the floor, these contact trajectories were recorded separately for subsequent motion decompression with IK.

B. FLOAT-POINT COMPRESSION

Many kinds of data sets are often represented in floating-point format. If floating-point numbers can be compressed effectively, a higher compression ratio can be achieved. A variety of methods have been developed for compressing and transmitting such numbers in the context of images [38], [39], large-dimensional scientific data [40], audio [41], 3D geometric data [36] and linear streams [42]–[44].

Isenburg *et al.* [36] proposed a lossless algorithm to encode the floating-point geometry of triangular meshes. In this method, a new position is first predicted by using the parallelogram predictor. The predicted and actual floating-point values are broken into their signs, exponents, and mantissas and then their corrections are then compressed separately with context-based arithmetic coding. Since the prediction quality varies with the exponent, the exponent is used to select different arithmetic contexts. Lindstrom *et al.* [40] proposed a fast and efficient compression algorithm for large-dimension floating-point data. In this approach, after the predicted position is obtained using the Lorenzo predictor [45], the predicted and the actual floating-point values are mapped to unsigned integers. The correction is then calculated and encoded with an arithmetic coder. Burtscher and Ratanaworabhan [42], [43] and Ratanaworabhan *et al.* [44] proposed methods for the effective and lossless compression of sequences of 64-bit floating-point data. They first sequentially predicted each value of the data sequence and then performed bitwise operations between the actual and predicted values. These methods are very fast and can meet the high-throughput demands of scientific computing environments.

Our method is fundamentally different from all of the above methods in that we develop a completely lossless compact representation by mining similar segments across a motion database. Our method is mainly inspired by [46] and [6] but is different from those methods in four aspects. First, we propose a forest structure for best predicting motion segments, and encode the residual segments with floating-point compression. Second, we propose a SOM-based motion segmentation method that can suitably balance the similar posture paths for maximum compression gains. Third, instead of searching in the self-distance matrix or locality-sensitive hashing tree, our method introduces randomized k-d trees to accelerate the process. Fourth, our method is completely lossless. Without time-consuming IK during decompression, which occurs with the method proposed by [6], our method can decompress motion segments in real time.

III. COMPRESSION PIPELINE

Our compression pipeline consists of four steps: trellis building and motion path discovery, SOM-based database sequence segmentation refining, forest construction and encoding of forest structure and residuals.

A. TRELLIS BUILDING AND MOTION PATH DISCOVERY

Our goal is to find continuous paths through an efficient tree-growing process to effectively exploit the recurring patterns in a motion database. Furthermore, we need an efficient k-nearest neighbor search method for trellis building. Based on this trellis structure, we can start the process of motion path discovery.

As defined in [47], there are two forms of the nearest neighbor search based on the randomized k-d tree structure: the k-nearest neighbor search and the radius nearest neighbor search. The former returns the predefined k neighboring frames, while the latter returns the neighboring frames where the distance is smaller than a predefined threshold radius. Because we wish to obtain the nearest neighbors that are similar within a predefined threshold, we adopt the radius nearest neighbor method and the Manhattan distance similar to that presented in [47], as shown in Equation (1).

$$RNN\{m, M, R\} = \{n | n \in M, Dis(m, n) < R\}, \quad (1)$$

where $Dis(m, n)$ is the distance between frame m and n , as shown in Equation (2):

$$Dis(m, n) = \sum_{k=1}^J w_k ||m_k - n_k||, \quad (2)$$

where $J = 32$ in our test set. Similar to the approach taken in [48], w_k is set to 0.7 for important joints, such as the hip, lower back, upper back, humerus, radius and femur, whereas w_k is set to 0.3 for supplementary joints, such as toes, feet, and fingers.

Based on the randomized k-d tree structure and frame distance definition, a trellis structure is built. In this structure,

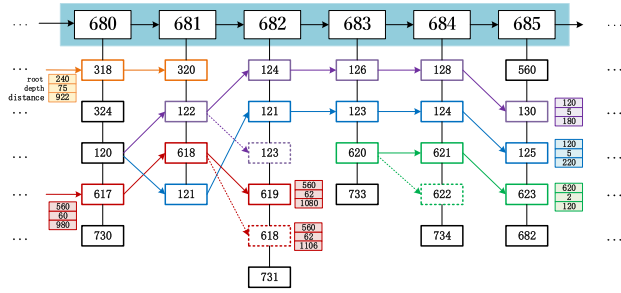


FIGURE 1. Trellis building and path growing.

the top row is the database sequence, while the columns are their best matches found in the database by using the randomized k-d tree nearest neighbor searching method [47]. In the following text, we call the top row of the trellis top row and the columns of the trellis *columns*.

Tree growth initiates at the 1st column of the trellis. One node i in the *column* can find the nodes of the neighboring columns within the interval $[i, i+2]$. As shown in Figure 1, the same-color continuous solid arrows constitute a path, while the dotted-line arrows indicate invalid paths, which are either not long enough or do not have the least average distance. Each path is described by root, depth, path distance, where the path distance is the accumulated distances of the matched pairs in the path; the root is the path start frame number, which corresponds to a node in the *columns* of the trellis; and the depth is the path length along the *top row*. This path information is stored in the current leaf nodes of the growing tree and passed to their descendants with the updated values. In Figure 1, the red path $\{560, 60, 980\}$ is stored in 617. Then, this information is passed to 618 (the second *column*) and further to 619 or 618 (the third *column*). We can obtain the updated path information as $\{560, 62, 1080\}$ and $\{560, 62, 1106\}$.

Different from the previous method presented in [46], the path distance is added to the path information to evaluate the average distances for paths that share the same start frame number. For these paths, we keep only the ones with the least average distance and the largest depth. As shown in Figure 1, for two red paths $\{560, 62, 1080\}$ and $\{560, 62, 1106\}$, the former has the lower average distance and is kept.

Another example is the two paths starting from 120. One is the purple path with $\{120, 5, 180\}$, and the other is the light blue path with $\{120, 5, 220\}$. The purple one has the lower average distance in the current situation. However, since the two paths might extend beyond frame 685 along the *top row*, we cannot determine which one should be kept at this stage.

After the path growing process, we keep only valid paths, which are the paths whose depth values are greater or equal to the threshold *ValidPathThr*. We collect all the paths from the database sequence into a path set as follows:

$$P = (p_1, p_2, \dots, p_n)^T, \quad (3)$$

where $p_i = (h_i, v_i)$, $h_i = (h_{i1}, h_{i2})$ and $v_i = (v_{i1}, v_{i2})$. h_{i1} and h_{i2} are temporal indices from the nodes of the top row of

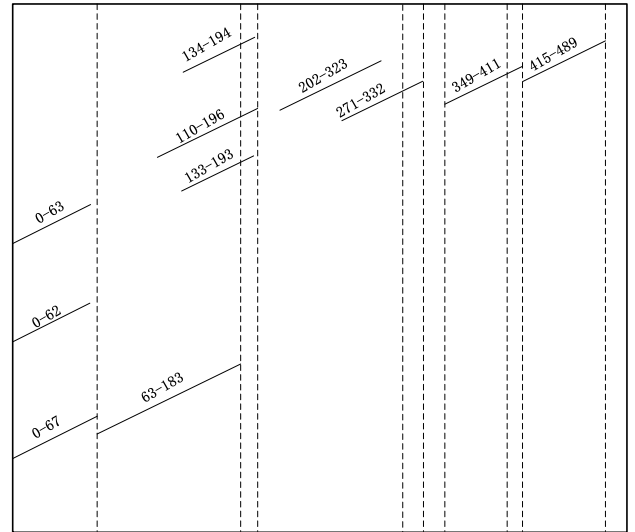


FIGURE 2. Example of a path set. The horizontal lines are h -segments, and the vertical lines are v -segments.

the trellis, as shown in Figure 1. They denote the start and end frame of a segment of the i^{th} paths. We call this kind of segment an h -segment, which corresponds to each of the horizontal lines in Figure 2. Moreover, v_{i1} and v_{i2} are the counterpart temporal indices from the nodes of the *columns* of the trellis in Figure 1. They denote the start and end frame of a segment of i^{th} paths. We call this kind of segment a v -segment, which corresponds to each of the vertical lines in Figure 2.

Different from the previous method, in our path growing process, we follow the rule that the h -segment h_i and the v -segment v_i cannot overlap. That is, either $h_{i2} < v_{i1}$ or $v_{i2} > h_{i1}$. Assuming that the red path in Figure 1 represents the i^{th} paths in the database, we obtain $h_{i1} = 682 - 62 = 620$, $h_{i2} = 682$, $v_{i1} = 560$, and $v_{i2} = 619$. Node 619 in the third *column* has the descendant 620. However, this sequence stops extending to the next *column* because such an extension would violate the rule that the h -segment and the v -segment cannot overlap.

B. SOM-BASED DATABASE SEQUENCE SEGMENTATION

In the preceding section, we obtained the path set for the database sequence. However, overlaps exist among h -segments, resulting in increased memory overhead. Furthermore, overlaps mean that we did not obtain the best segmentation that can balance all paths. Figure 2 shows an example of a path set, where h -segments $[63-183]$, $[133-193]$, $[110-196]$ and $[134-194]$ overlap. We seek an optimal segmentation based on these h -segments to achieve the maximum compression gains. We propose refining the motion sequence segmentation based on the SOM by taking all h -segments as the input layer and an evenly segmented segmentation as the output layer. In Figure 2, the horizontal lines are the h -segments and the vertical lines are the v -segments. The dotted lines are major segment points in the horizontal direction. Here, we use evenly tilted lines to

indicate different paths. First, we find the longest segment among the h -segments and denote its segment number as t . The database sequence is then equally divided into m segments as Equation (4) shows.

$$m = \lceil \frac{l}{h_{t2} - h_{t1} + 1} \rceil, \tag{4}$$

where l is the size of the database (i.e., the total number of frames of this database). We designate this segment set as follows:

$$S = (s_1, s_2, \dots, s_m)^T, \tag{5}$$

where $s_j = (s_{j1}, s_{j2})$ and s_{j1}, s_{j2} are the start and end frame numbers of the j^{th} segment, respectively.

Second, for each h -segment h_i from P , we search for a segment s_j from S that has the largest IoU (intersection over union) with h_i . This IoU is a similarity metric as shown in Equation (6).

$$IoU(h_i, s_j) = \frac{overlap(h_i, s_j)}{h_{i2} - h_{i1} + 1 + s_{j2} - s_{j1} + 1 - overlap(h_i, s_j)}. \tag{6}$$

Here, the overlap is defined as follows:

$$overlap(x_i, y_j) = \begin{cases} x_{i2} - x_{i1} + 1, & \text{when } x_{i1} \geq y_{j1} \text{ and } x_{i2} \leq y_{j2} \\ x_{i2} - x_{j1} + 1, & \text{when } x_{i1} < y_{j1} \text{ and } x_{i2} \leq y_{j2} \\ x_{j2} - x_{i1} + 1, & \text{when } x_{i1} \geq y_{j1} \text{ and } x_{i2} > y_{j2} \\ x_{j2} - x_{j1} + 1, & \text{when } x_{i1} < y_{j1} \text{ and } x_{i2} > y_{j2}. \end{cases}$$

Third, we adjust the length of the segment of the output layer by moving its start and end point. That is,

$$s_j = s_j + k \times (h_i - s_j). \tag{7}$$

In Figure 3, because $h_{i2} - s_{j2}$ is negative, s_{j2} decreases. A new segment is formed with frames from updated $s_{j2} + 1$ to $s_{(j+1)1} - 1$. If the length of this segment, $s_{(j+1)1} - s_{j2} - 1$, is larger than or equal to a threshold $LengthThr$, then the segment is added to S as a new segment. Otherwise, it is combined into s_{j+1} . Moreover, if $h_{i2} - s_{j2}$ is positive, then segment s_{j+1} decreases. If $s_{(j+1)2} - s_{(j+1)1} + 1$ is less than the threshold $LengthThr$, this segment is combined into s_{j+2} .

We iterate the second and third steps through all h -segments. We decrease the learning rate adaptively for more iterations until k is equal to or smaller than zero, and we obtain the optimal refined segmentation set S .

C. FOREST CONSTRUCTION

For efficiently exploiting the recurring patterns among the paths in the path set, we need to cluster the h -segments in S from the SOM into many trees. Typically, each tree has a primary segment (the root) and many similar segments (the branches). Since each path has an h -segment and a v -segment, we use the path as a connection agent for organizing these segments into a forest structure.

First, we insert the longest h -segment into F , and this segment becomes the root of the first tree in F . Second,

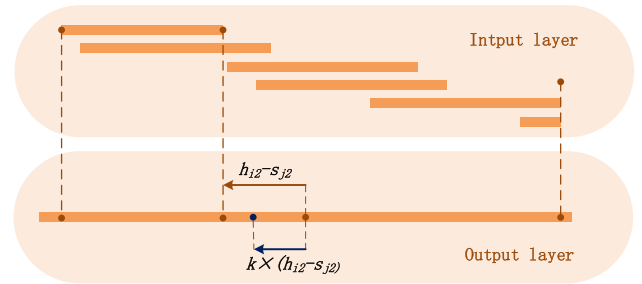


FIGURE 3. Sequence segmentation refining with a SOM.

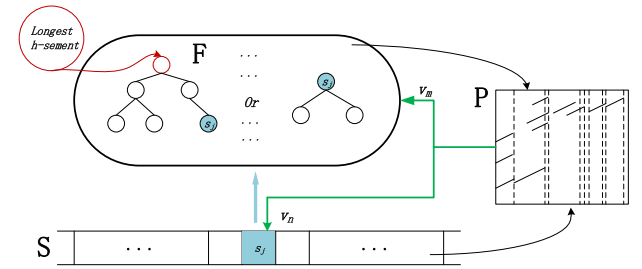


FIGURE 4. Forest construction process where the paths in set P act as connection agents.

assuming that F already contains trees, as Figure 4 shows, for a specific h -segment s_j in S , we either insert it into one tree of F , or take it as the root of a new tree, as indicated by the blue arrow in Figure 4. We iterate the second step for each h -segment in S until all segments of S are processed. In the final step, we evaluate the tree similarity according to Equation (6) between any pair segments and merge the trees when there is a pair whose similarity values are larger than the threshold $IoUThr$.

In this paragraph, we will provided the detailed process of the second step. For an h -segment s_i in F , we obtain an h -segment h_j , which has the largest $IoU(h_j, s_i)$ with h -segment s_i , according to Equation (6). Then, the counterpart segment of h_j , e.g., the v -segment v_m , is obtained by looking up the path set P (as indicated by the green arrow in Figure 4). The same process is applied to s_j in S , as Figure 4 shows, and the counterpart segment is noted as v_n . We can obtain the similarity metric between v_m and v_n as $IoU(v_m, v_n)$ from Equation (6).

We iterate all the nodes in F and evaluate their similarity with s_j and keep the node that has the largest $IoU(v_m, v_n)$. We denote this h -segment as s_max . If the largest $IoU(v_m, v_n)$ is larger than or equal to the threshold $IoUThr$, s_j is inserted into the tree that s_max belongs to as a descendant segment. Otherwise, s_j forms the root of a new tree.

D. ENCODING OF THE FOREST STRUCTURE AND RESIDUALS

After the above tree construction process, we obtain a forest structure. Based on this forest structure, we obtain the prediction residuals (the differences between the non-root branches and their parents). Finally, the differences

between neighboring residuals are encoded by introducing a floating-point compression method [36]. The root frame data are encoded by employing a nonprediction mode of float-point compression. For the tree structure, we just save the temporal frame indices of the h -segment, the number of the trees children for the nonleaf node and frame indices for leaf nodes according to the width-first policy.

IV. RESULTS AND DISCUSSION

In this section, we present the experimental results to demonstrate the performance of the proposed method. Our focus is to show the relationships between compression ratios under different parameter values. We also demonstrate the comparison results with previous studies. In this paper, we define the compression ratio as the original file size divided by the compressed file size.

All the data in our experiment are from the CMU Graphics Lab Motion Capture Database, which has 2605 trials in 6 categories and 23 subcategories. We use the five motion database (run, modern dance, jumping, salsa dance, various activities) and three steps Climb and Walk on uneven terrain. Before we present our results, we first define some terms that we used for evaluation. The first term is the compression ratio, which is defined as the original file size before compression divided by the compressed file size, including motion data and index data. The second term is the predicted frame number, which is defined as the number of frames predicted in the hierarchal tree structure. The last term is the index entry number, which is defined as the number of parent/child pair indices need to be saved to maintain the tree structure. For the best compression gains, we expect a larger predicted frame number and a smaller index number for better compression performance.

A. COMPRESSION PERFORMANCE WITH DIFFERENT PARAMETERS

Our algorithm has three associated parameters: $ValidPathThr$, $LengthThr$ and $IoUThr$. $ValidPathThr$ is a threshold for determining whether a path is collected into a path set as defined in subsection III-A. $LengthThr$ is a threshold for determining whether a new interval can be a new segment or combined into a neighboring segment, as defined in subsection III-B. Finally, $IoUThr$ is a threshold for determining whether a segment will form the root of a new tree or contribute branches to an existing tree, as defined in subsection III-C. We discuss the trade-off between these parameters and show the results of the compression ratio under different parameter value combinations.

1) COMPRESSION PERFORMANCE UNDER DIFFERENT $ValidPathThr$

If $ValidPathThr$ is smaller, then there will be more paths and, hence, more segments in the input layer of the SOM. This condition will lead to a tree with more branches. In contrast, if $ValidPathThr$ is larger, then there will be fewer paths, and therefore, we will obtain a tree with fewer branches.

TABLE 1. Number of input segments and predicted frames with parameter $ValidPathThr$.

$ValidPathThr$	The number of input segments of SOM	The predicted frame number
10	50	141
60	23	222
90	4	152

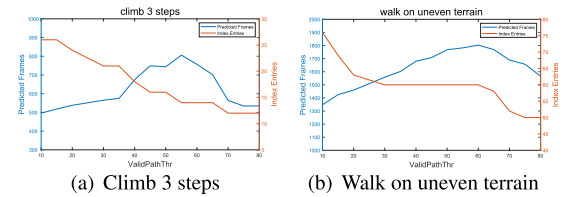


FIGURE 5. Predicted frame number and index entry number with parameter $ValidPathThr$.

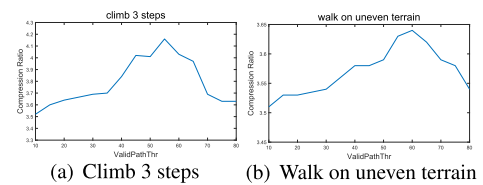


FIGURE 6. Compression ratio against the parameter $ValidPathThr$.

Clearly, the prediction performance is poor with a very small $ValidPathThr$ because we cannot fully exploit the recurring similar motion patterns in the motion database. The smallest value of $ValidPathThr$ is 1. In this case, the compression method cannot find the recurring patterns in the motion database at all. However, if we have a very large $ValidPathThr$, then we cannot obtain enough paths for the tree structure; hence, the predicted frames will be unsatisfactory. Furthermore, $ValidPathThr$ is inversely proportional to the number of input segments of the SOM. We give an example of 344 frames of walking motion in Table 1. A larger number of input segments of the SOM typically implies a larger tree. Figure 5 shows the index entry number for $ValidPathThr$. The index entry number decreases with $ValidPathThr$ because larger parameter values are associated with more branches in the tree. Figure 5 also shows the predicted frames number against the parameter $ValidPathThr$. The relationship is consistent with our discussion above. For the trade-off, the best value for $ValidPathThr$ might fall in the interval of 55-60. The curve of the compression ratio with the parameter supports our choice, as shown in Figure 6.

2) ANALYSIS AND DISCUSSION FOR $LengthThr$

In our experiment, we find that the parameter $LengthThr$ has little impact on the compression performance as shown in Table 2, because the reason is that the main segment set remains the same regardless of what value we set for $LengthThr$. Table 2 shows the resulting segments for different values for $LengthThr$ for 343 frames of walking motion. The input

TABLE 2. Segment number with parameter LengthThr.

LengthThr	Segments after SOM
5	[0-119], [120-219], [220-250], [251-342]
10	[0-119], [120-219], [220-250], [251-342]
15	[0-119], [120-219], [220-250], [251-342]
20	[0-119], [120-231], [232-342]
25	[0-119], [120-231], [232-342]
30	[0-119], [120-231], [232-342]
40	[0-119], [120-231], [232-342]

TABLE 3. Compression ratio with different LengthThr values.

Motion	Frame no.	Compression ratio with different LengthThrs						
		10	15	20	30	40	45	50
Basketball	722	4.20	4.21	4.21	4.21	4.19	4.19	4.19
Cartwheel	2460	4.21	4.21	4.22	4.21	4.22	4.20	4.20

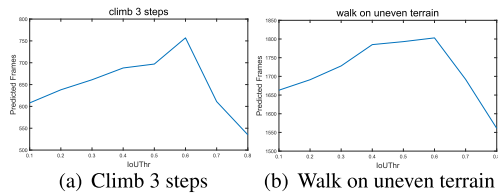


FIGURE 7. Predicted frame number against parameter IoUThr.

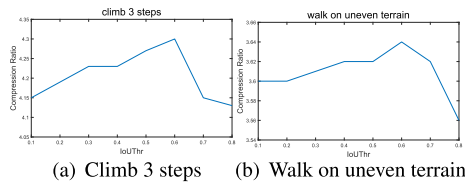


FIGURE 8. Compression ratio with parameter IoUThr.

layer mainly determines the resulting segment set, although the value of LengthThr might result in little change. However, this change has nearly no influence on the compression ratio as shown in Table 3. In this paper, we set LengthThr to 30 to reduce the computation costs.

3) COMPRESSION PERFORMANCE UNDER DIFFERENT IoUThr

For the third parameter, IoUThr, a smaller IoUThr value might result in a tree having more branches. However, a very small IoUThr might result in collecting segments that are not very similar. This condition leads to poor compression performance. On the other hand, a significantly larger IoUThr will cause the tree to have significantly fewer branches, and the predicted frame number will be small, which also results in poor compression performance. Figure 7 shows the curve of the predicted frame with different IoUThr values. Figure 8 shows the relation between the compression ratio and IoUThr. A value of 0.6 is very favorable for IoUThr.

Note that IoUThr has no impact on the index size, which is mainly determined by ValidPathThr. Therefore, we did not characterize its relationship with its index entry number as ValidPathThr.

TABLE 4. Compression ratios of six compression schemes.

Motion database	Frame No.	Compression ratio					
		Gzip	Rar	L3C	Non-SOM	APP	MRPA
run	3472	2.79	2.82	5.17	3.56	3.62	4.01
modern dance	18620	2.78	2.87	5.21	3.64	3.78	3.96
jumping	21566	2.96	3.07	4.61	4.20	4.41	5.02
salsa dance	30931	2.74	2.86	5.22	3.41	3.68	4.12
various activities	116152	2.80	2.96	5.26	4.34	4.08	4.98

B. COMPARISON WITH OTHER LOSSLESS COMPRESSION METHODS

In this section, we compare the performance of our MRPA method with the previous lossless compression method alpha parallelogram predictor (APP) [19] and selected widely used compression tools, namely, Gzip and Rar. We also compare our method with the method that uses our compression pipeline without the SOM-based segmentation (non-SOM) and a lossless deep-learning based image compression method (L3C) [49] by adapting our data to their implementation accordingly. Table 4 compares the six lossless schemes in terms of their compression ratios using different motion databases. Each database has a concatenation of many motions from the CMU motion capture database. Each database is mainly composed of motions labeled by their names. The database “run” has 12 motions, “modern dance” has 20 motions, “salsa dance” has 15 motions, “jump” has 32 motions, and “various activities” has 15 motions of different activities, such as walking, jumping, squatting, sitting, and punching. Table 4 demonstrates that our method outperforms the APP method, Non-SOM method, L3C method and conventional compression tools in terms of the compression ratio due to the reduction in redundancy at the segment level and the establishment of an efficient index structure.

Although L3C outperforms our method in term of compression ratio, the decompression time of their method is about two order of magnitude more than that of our method, which hinder its use in real-time applications. Please note that experiments on L3C were performed on a PC with Core i7 CPU and 16 GB RAM, while experiments on other methods were performed on a PC with i5 CPU and 8 GB RAM. Motion databases are typically used in computer games, virtual reality, and computer animation, among other applications. In these applications, motion databases are typically stored and transmitted in a compressed form and are decompressed on demand in real time when needed. Therefore, for a compression scheme, the decompression efficiency is much more important than the compression time. From 2011 to 2016, we received successive funding for the development of a compression system for the popular computer game QQ Dance, which has more than 100 million players. All the character motions of this game are from motion capture data. Usually, one player picks a song and the game chooses a couple of dance segments from a compressed motion database and concatenates them into a motion that matches the song. In this application scenario, the decompression time is very

TABLE 5. Decompression and compression times of five compression schemes.

Motion database	Frame no.	Decompress time (s)					Compress time (s)				
		Gzip	Rar	APP	L3C	MRPA	Gzip	Rar	APP	L3C	MRPA
run	3472	0.27	0.22	0.07	10.17	0.10	0.51	0.54	0.39	10.36	0.44
modern dance	18620	0.29	0.28	0.35	55.05	0.61	1.85	1.08	2.03	55.77	2.91
jumping	21566	0.30	0.32	0.38	64.49	0.51	1.48	1.26	0.2.34	63.10	4.56
Salsa dance	30931	0.42	0.48	0.58	92.21	0.69	2.66	2.02	3.39	93.26	3.71
various activities	116152	1.21	1.13	2.07	346.93	2.33	11.53	9.72	12.64	342.55	13.44

important for a smooth experience. Moreover, 33 frames have to be played within one second for a smooth experience. That is, each frame is 33 ms, including data decompression, and decompression must occur in real time. In the game, at most there are 100 characters dancing in the same scene. This maximum scene of a frame has to be decompressed within 3 ms for real-time experience. The decompression rate of our method is 0.022 ms/frame on average, according to Table 5, and can meet the requirement of this kind of system.

The decompression times of our method (MRPA) are comparable to that of the APP and the conventional compression tool Gzip. Please note that we optimized the decompression of the APP method in terms of I/O for a fair comparison. The decompression pipeline of our method is much simpler than the compression pipeline because there are no time-consuming processes, such as path discovery, SOM-based segmentation and the clustering process of the compression process, which is why our decompression is much more efficient than compression.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a compact representation for a motion database by employing effective recurring pattern discovery and analysis. We have shown that by using our MRPA method, we can reorganize the motion database at the segment level into a forest structure; moreover, the prediction process can be more efficient at this segment level along the tree than prediction processes using conventional DOF prediction methods.

This increased efficiency is one reason why we can feature a very small index size and less decompression time relative to the previous method [19]. We have demonstrated that our method outperforms the APP method, Non-SOM method and conventional compression tools (in terms of the compression ratio) due to the reduction in redundancy at the segment level and the establishment of an efficient index structure. Our method is lossless, while current compression methods are primarily lossy and distort the motion data. We have demonstrated that our method outperforms conventional compression tools in terms of the compression ratio. The decompression complexity of our method is also low, thus resulting in decompression times comparable to those of previous methods.

The compression times of our method are outperformed by that of the previous method because our compression pipeline has to undergo trellis building, the path discovery process

and the SOM-based segmentation process. Although we have introduced randomized k-d trees for finding the nearest neighbors instead of the time-consuming self-distance matrix method [14] and the space-consuming locality-sensitive hashing (LSH) method [46], the time complexity of our method is still higher than that of the previous method [19]. The adoption of a more time-efficient SOM method, such as the parallel SOM on a graphic processing unit (GPU) [50]–[52], will be necessary. We plan to explore this issue in the future.

ACKNOWLEDGMENTS

Our work is supported by CCF-Tencent Open Fund, Liaoning Innovative Talents Support Plan (Grant No. LR2016071), China Postdoctoral Science Foundation (No. 2014M561228) and successive funding from Beijing Yonghang Co., Ltd.

REFERENCES

- [1] O. Arikan, "Compression of motion capture databases," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 890–897, Jul. 2006.
- [2] P. Beaudoin, P. Poulin, and M. van de Panne, "Adapting wavelet compression to human motion capture clips," in *Proc. Graph. Interface (GI)*, 2007, pp. 313–318.
- [3] S. Chattopadhyay, S. M. Bhandarkar, and K. Li, "Human motion capture data compression by model-based indexing: A power aware approach," *IEEE Trans. Vis. Comput. Graphics*, vol. 13, no. 1, pp. 5–14, Jan. 2007.
- [4] B.-S. Chew, L.-P. Chau, and K.-H. Yap, "Progressive transmission of motion capture data for scalable virtual character animation," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2009, pp. 1461–1464.
- [5] A. Firouzmanesh, I. Cheng, and A. Basu, "Perceptually guided fast compression of 3-D motion capture data," *IEEE Trans. Multimedia*, vol. 13, no. 4, pp. 829–834, Aug. 2011.
- [6] Q. Gu, J. Peng, and Z. Deng, "Compression of human motion capture data using motion pattern indexing," *Comput. Graph. Forum*, vol. 28, no. 1, pp. 1–12, Mar. 2009.
- [7] Y. Han, "Computer animation in mobile phones using a motion capture database compressed by polynomial curve-fitting techniques," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, pp. 1008–1016, Aug. 2008.
- [8] J. Hou, L.-P. Chau, N. Magnenat-Thalmann, and Y. He, "Human motion capture data tailored transform coding," *IEEE Trans. Vis. Comput. Graphics*, vol. 21, no. 7, pp. 848–859, Jul. 2015.
- [9] M. A. Khan, "An efficient algorithm for compression of motion capture signal using multidimensional quadratic Bézier curve break-and-fit method," *Multidimensional Syst. Signal Process.*, vol. 27, no. 1, pp. 121–143, Jan. 2016.
- [10] C.-H. Kwak and I. V. Bajic, "Hybrid low-delay compression of motion capture data," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2011, pp. 1–6.
- [11] C.-H. Kwak and I. V. Bajic, "Online MoCap data coding with bit allocation, rate control, and motion-adaptive post-processing," *IEEE Trans. Multimedia*, vol. 19, no. 6, pp. 1127–1141, Jun. 2017.
- [12] B. Krüger, J. Tautges, and A. Weber, "Multi-mode representation of motion data," in *Proc. GRAPP (AS/IE)*, 2007, pp. 21–29.
- [13] C. Lee and J. Lasenby, "An efficient wavelet-based framework for articulated human motion compression," in *Proc. Int. Symp. Vis. Comput.*, in Lecture Notes in Computer Science, vol. 5358, 2008, pp. 75–86.

- [14] I.-C. Lin, J.-Y. Peng, C.-C. Lin, and M.-H. Tsai, "Adaptive motion data representation with repeated motion analysis," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 4, pp. 527–538, Apr. 2011.
- [15] Y. Lin and M. D. McCool, "Nonuniform segment-based compression of motion capture data," in *Proc. Int. Symp. Vis. Comput.* Berlin, Germany: Springer, 2007, pp. 56–65.
- [16] G. Liu and L. McMillan, "Segment-based human motion compression," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*. Aire-la-Ville, Switzerland: Eurographics Association, 2006, pp. 127–135.
- [17] M. Tourner, X. Wu, N. Courty, E. Arnaud, and L. Revéret, "Motion compression using principal geodesics analysis," *Comput. Graph. Forum*, vol. 28, no. 2, pp. 355–364, Apr. 2009.
- [18] L. Váša and G. Brunnett, "Rate-distortion optimized compression of motion capture data," *Comput. Graph. Forum*, vol. 33, no. 2, pp. 283–292, May 2014.
- [19] P. Wang, Z. Pan, M. Zhang, R. W. H. Lau, and H. Song, "The alpha parallelogram predictor: A lossless compression method for motion capture data," *Inf. Sci.*, vol. 232, pp. 1–10, May 2013.
- [20] M. Zhu, H. Sun, and Z. Deng, "Quaternion space sparse decomposition for motion compression and retrieval," in *Proc. 11th ACM SIGGRAPH/Eurograph. Conf. Comput. Animation*. Aire-la-Ville, Switzerland: Eurographics Association, 2012, pp. 183–192.
- [21] L. Kovar, J. Schreiner, and M. Gleicher, "Footskate cleanup for motion capture editing," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation (SCA)*, 2002, pp. 97–104.
- [22] I. Guskov and A. Khodakovsky, "Wavelet compression of parametrically coherent mesh sequences," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation (SCA)*. Aire-la-Ville, Switzerland: Eurographics Association, 2004, pp. 183–192.
- [23] L. Ibarria and J. Rossignac, "Dynamack: Space-time compression of the 3D animations of triangle meshes with fixed connectivity," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*. Aire-la-Ville, Switzerland: Eurographics Association, 2003, pp. 126–135.
- [24] Z. Karni and C. Gotsman, "Compression of soft-body animation sequences," *Comput. Graph.*, vol. 28, no. 1, pp. 25–34, Feb. 2004.
- [25] M. Sattler, R. Sarlette, and R. Klein, "Simple and efficient compression of animation sequences," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation (SCA)*, 2005, pp. 209–217.
- [26] J.-H. Yang, C.-S. Kim, and S.-U. Lee, "Compression of 3-D triangle mesh sequences based on vertex-wise motion vector prediction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 12, pp. 1178–1184, Dec. 2002.
- [27] J.-H. Yang, C.-S. Kim, and S.-U. Lee, "Semi-regular representation and progressive compression of 3-D dynamic mesh sequences," *IEEE Trans. Image Process.*, vol. 15, no. 9, pp. 2531–2544, Sep. 2006.
- [28] J. Zhang and C. Owen, "Octree-based animated geometry compression," in *Proc. Data Compress. Conf. (DCC)*, 2004, pp. 508–517.
- [29] J. Hou, L.-P. Chau, Y. He, and N. Magnenat-Thalmann, "Low-rank based compact representation of motion capture data," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 1480–1484.
- [30] J. Hou, L.-P. Chau, N. Magnenat-Thalmann, and Y. He, "Scalable and compact representation for motion capture data using tensor decomposition," *IEEE Signal Process. Lett.*, vol. 21, no. 3, pp. 255–259, Mar. 2014.
- [31] J. P. Park, K. H. Lee, and J. Lee, "Finding syntactic structures from human motion data," *Comput. Graph. Forum*, vol. 30, no. 8, pp. 2183–2193, Dec. 2011.
- [32] B.-S. Chew, L.-P. Chau, and K.-H. Yap, "A fuzzy clustering algorithm for virtual character animation representation," *IEEE Trans. Multimedia*, vol. 13, no. 1, pp. 40–49, Feb. 2011.
- [33] M. Isenburg and P. Alliez, "Compressing polygon mesh geometry with parallelogram prediction," in *Proc. IEEE Vis. (VIS)*, Oct. 2002, pp. 141–146.
- [34] C. Touma and C. Gotsman, "Triangle mesh compression," in *Proc. Graph. Interface*. Mississauga, ON, Canada: Canadian Information Processing Society, 1998, pp. 26–34.
- [35] P. Wang, R. W. H. Lau, M. Zhang, J. Wang, H. Song, and Z. Pan, "A real-time database architecture for motion capture data," in *Proc. 19th ACM Int. Conf. Multimedia (MM)*, 2011, pp. 1337–1340.
- [36] M. Isenburg, P. Lindstrom, and J. Snoeyink, "Lossless compression of predicted floating-point geometry," *Comput.-Aided Des.*, vol. 37, no. 8, pp. 869–877, Jul. 2005.
- [37] L. Kovar and M. Gleicher, "Automated extraction and parameterization of motions in large data sets," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 559–568, Aug. 2004.
- [38] B. E. Usevitch, "JPEG2000 extensions for bit plane coding of floating point data," in *Proc. Data Compress. Conf. (DCC)*, 2003, p. 451.
- [39] B. E. Usevitch, "JPEG2000 compatible lossless coding of floating-point data," *EURASIP J. Image Video Process.*, vol. 2007, no. 1, p. 22, 2007.
- [40] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Trans. Vis. Comput. Graphics*, vol. 12, no. 5, pp. 1245–1250, Sep. 2006.
- [41] F. Ghido, "An efficient algorithm for lossless compression of IEEE float audio," in *Proc. Data Compress. Conf. (DCC)*, 2004, pp. 429–438.
- [42] M. Burtcher and P. Ratanaworabhan, "High throughput compression of double-precision floating-point data," in *Proc. Data Compress. Conf. (DCC)*, 2007, pp. 293–302.
- [43] M. Burtcher and P. Ratanaworabhan, "FPC: A high-speed compressor for double-precision floating-point data," *IEEE Trans. Comput.*, vol. 58, no. 1, pp. 18–31, Jan. 2009.
- [44] P. Ratanaworabhan, J. Ke, and M. Burtcher, "Fast lossless compression of scientific floating-point data," in *Proc. Data Compress. Conf. (DCC)*, 2006, pp. 133–142.
- [45] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-core compression and decompression of large n-dimensional scalar fields," *Comput. Graph. Forum*, vol. 22, no. 3, pp. 343–348, Sep. 2003.
- [46] J. Meng, J. Yuan, M. Hans, and Y. Wu, "Mining motifs from human motion," in *Eurographics (Short Papers)*, K. Mania and E. Reinhard, Eds. The Eurographics Association, 2008.
- [47] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, Nov. 2014.
- [48] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 491–500, Jul. 2002.
- [49] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Practical full resolution learned lossless image compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10629–10638.
- [50] N. E. B. A. Khalid, M. F. B. Mustapha, A. B. Ismail, and M. B. Manaf, "Parallel self-organizing map using shared virtual memory buffers," in *Advanced Topics in Intelligent Information and Database Systems*. 2017, p. 49.
- [51] S. Q. Khan and M. A. Ismail, "Design and implementation of parallel SOM model on GPGPU," in *Proc. 5th Int. Conf. Comput. Sci. Inf. Technol.*, Mar. 2013, pp. 233–237.
- [52] M. F. Mustapha, N. E. A. Khalid, M. Manaf, and A. Ismail, "Evaluating parallel self-organizing map processing using graphic processing unit," *Adv. Sci. Lett.*, vol. 23, no. 6, pp. 5232–5236, Jun. 2017.



PENGJIE WANG is currently a Professor with the School of Computer Science, Dalian Minzu University, Dalian, China. His research interests include computer vision, computer graphics, and data compression.



JIANG WANG received the degree from the School of Computer Science, Dalian Minzu University, Dalian, China. He is currently a Co-Founder of PYDDot Technology Company, Ltd., Guiyang, China. His research interests include big data, the IoT, and computer vision.



XIAOMING WEI is currently a Professor with the School of Computer Science, Dalian Minzu University, Dalian, China. His research interests include computer vision, collaborative design, and so on.



JING XUN is currently pursuing the master's degree with the School of Computer Science, Dalian Minzu University, Dalian, China. Her research interests include computer vision and data compression.

...



JIANA MENG is currently a Professor with the School of Computer Science, Dalian Minzu University, Dalian, China. Her research interests include computer vision and natural language processing.