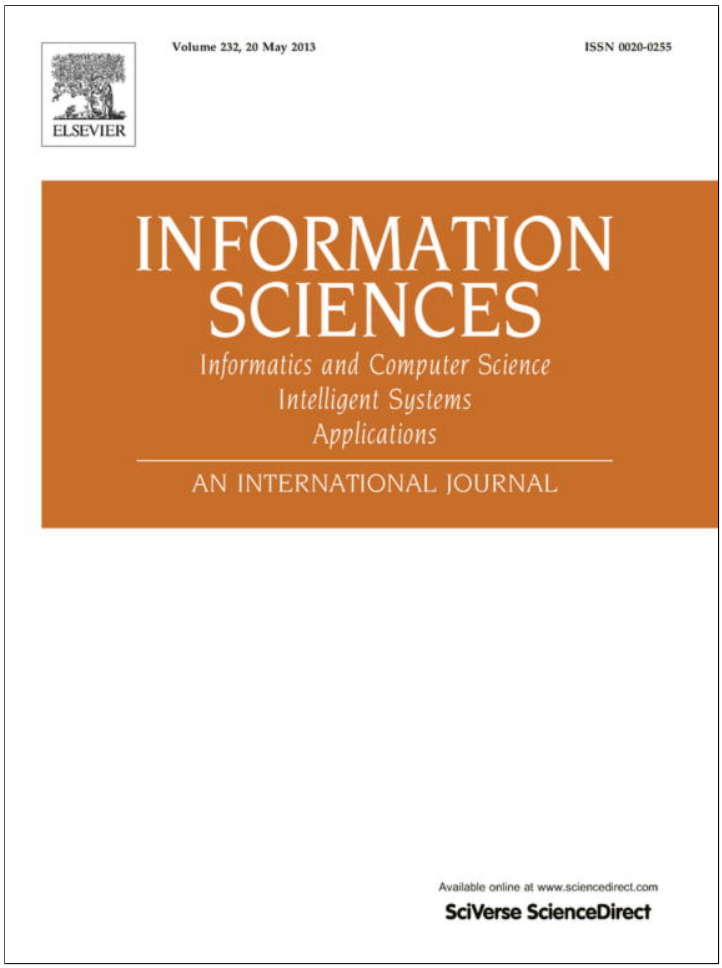


Information Sciences 232 (2013) 1–10

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

The alpha parallelogram predictor: A lossless compression method for motion capture data

Pengjie Wang [a,b,c], Zhigeng Pan [d,a], Mingmin Zhang [a,*], Rynson W.H. Lau [c], Haiyu Song [b]

[a] State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou 310027, China
[b] College of Computer Science & Engineering, Dalian Nationalities University, Dalian 116600, China
[c] Department of Computer Science, City University of Hong Kong, Hong Kong
[d] Digital Media and HCI Research Center, Hangzhou Normal University, Hangzhou 310023, China

ARTICLE INFO

Article history:
Received 26 July 2010
Received in revised form 11 October 2012
Accepted 16 January 2013
Available online 29 January 2013

Keywords:
Character animation
Motion capture
Lossless compression
Animation compression

ABSTRACT

Motion capture data in an uncompressed form can be expensive to store, and slow to load and transmit. Current compression methods for motion capture data are primarily lossy and cause distortions in the motion data. In this paper, we present a lossless compression algorithm for motion capture data. First, we propose a novel Alpha Parallelogram Predictor (APP) to estimate the DOF (degree of freedom) of each child joint from those of its immediate neighbors and parents that have already been processed. The prediction parameter of the predictor, which is referred to as the alpha parameter, is adaptively chosen from a carefully designed lookup table. Second, we divide the predicted and actual values into three components: sign, exponent and mantissa. We then compress their corrections separately with context-based arithmetic coding. Compared with other lossless compression methods, our approach can achieve a higher compression ratio with a comparable compression time. It can be used in situations where lossy compression is not preferred.

1. Introduction

In recent years, motion capture data are widely used in many applications, in particular in movies and games. The advance in data-driven animation technologies rapidly produces huge collections of motion capture data, and it becomes necessary to design effective compression algorithms for the storage and transmission of these data. There are a number of compression methods proposed for motion capture data [1,2,6,10,23,27]. Although most of these compression methods have shown to produce good compression performances, they are all based on lossy compression and hence modify the original data. Scientists and engineers typically do not prefer the idea of having their data modified by a process beyond their control and therefore often refrain from using these lossy compression algorithms. In addition, any lossy compression methods for motion capture data, whether they are orientation-based or position-based, introduce errors to the motion data, causing various perceptual artifacts. For example, the well-known foot-skating artifact [18] is caused by the failure in preserving clean footprints. Although we may adopt inverse kinematics to reduce or minimize this error [1,2,27], the residual error can still degrade the visual quality of the motion data. Furthermore, if we want to edit/modify a motion file compressed with a lossy compression method, we will need to uncompress it before editing and then compress it again afterwards. Each time we compress the motion file, the lossy compression method will introduce additional errors to the file. Hence, a motion file needed to go through a number of such editing operations at different times will result in some compound errors.

* Corresponding author.
E-mail address: mingmin_zhang@yeah.net (M. Zhang).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.01.007

To address these problems, we propose in this paper a lossless compression method for motion capture data. To our knowledge, there has not been any work published that addresses the lossless compression of motion capture data. Our method can be summarized as follows. We first represent the motion capture data as a matrix, with all the elements split into several equal-sized clusters. For each cluster, we propose a novel Alpha Parallelogram Predictor (APP) to estimate a DOF (degree of freedom) value for each child joint from those of its immediate neighbors and parents that have already been processed. Although the APP is an extension of the parallelogram predictor for mesh compression [15,26], it is significantly different from the parallelogram predictor due to the introduction of a prediction parameter, referred to as the alpha parameter, and the indexing storage of alpha. Through the introduction of this alpha parameter, the APP can obtain more accurate prediction results and smaller correction values than those of the parallelogram predictor. As a result, it produces a better compression ratio. To address the storage overhead of the alpha values, we have carefully designed a lookup table from which the prediction parameter is chosen. In this way, we only need to store the indices to the lookup table to save storage space. Finally, after the prediction step, both the predicted and the actual values are separated into three components: sign, exponent and mantissa. The correction of each component is compressed by means of arithmetic coding. Our experiments show that the proposed method outperforms the lossless MPEG-FBA compression method and two general compression tools, gzip and Rar, in terms of compression ratio. The proposed method is completely lossless and nicely complements the lossy compression methods for motion capture data. The contributions of our work can be summarized as follows:

- We introduce an alpha parameter to the traditional parallelogram predictor [15,26] to improve the prediction accuracy and hence the compression ratio.
- To reduce the storage overhead of the alpha values, we propose to select the alpha values from a carefully designed lookup table.
- We have adapted the lossless float-point compression algorithm [16] to post-process the correction values obtained from the APP. As the APP can work better than the traditional parallelogram predictor, different context policies can be used for the actual exponent and mantissa corrections, based on the most frequently called sequence in Algorithm 1.

The remainder of this paper is organized as follows. Section 2 provides a review of related works. Section 3 describes the splitting of the motion capture data and the proposed APP, while Section 4 describes the floating-point compression method, which deals with the actual and predicted values produced by the APP. Section 5 presents some experimental results and evaluates the proposed method. Finally, Section 6 gives the conclusion remarks and discusses possible future works.

2. Related works

In this section, we briefly summarize existing works on animation compression and on floating-point compression, which are most relevant to our work presented here.

2.1. Animation compression

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

Previous works on animation compression mainly focus on compressing animated geometry meshes [3,11,12,14,17,19,29,30,33–35]. Zhang and Owen [35] adopts octrees to represent animation data. Yang et al. [33] and Ibarria and Rossignac [14] perform vertex-wise motion prediction for compression. Karni and Gotsman [17], Sattler et al. [29], and Sloan et al. [30] exploit principal component analysis (PCA) techniques for compression. Briceño et al. [3] generalize geometry image coding. Guskov and Khodakovsky [12] use wavelet coding techniques for compression. Gupta et al. [11] and Lengyel [19] detect parts of the mesh with rigid motion to encode only the transformation and the residuals. Yang et al. [34] encode non-isomorphic animated mesh sequences.

Compression of motion capture data is a relatively new area. Arican [1] uses Bezier curves to represent motion clips, and then applies clustered PCA to reduce dimensionality. Environmental contacts are encoded separately based on discrete cosine transformation (DCT). This method approximates the original motion with minor loss of visual quality. Liu and McMillan [23] first segment a motion sequence into subsequences of simple motions. They then compress each subsequence by PCA approximation and further compress the resulting PCA data by selecting and storing only the key frames.

Chattopadhyay et al. [6] propose a power aware algorithm for mobile devices that exploits motion data indexing concept. They derive the index of each floating-point number from the statistical distribution of the floating-point numbers in the motion matrix. Since these integer index numbers require much fewer bits to store, the motion capture data can be significantly compressed. Beaudoin et al. [2] adapt standard wavelet compression on joint angles by considering the process of selecting wavelet coefficients as a discrete optimization problem within a tractable search space. This search space is adapted to the nature of the data. For the contacts with the environment, they use optimized wavelet based compression and inverse kinematics correction. This method focuses on short and recomposable animation clips. Gu et al. [10] propose to compress human motion capture data based on constructing a hierarchical structure and motion pattern indexing. They first organize the 3D markers as a hierarchy, where each node corresponds to a meaningful part of the human body and is

therefore coded separately. For each meaningful part, a motion pattern database is built and the sequence of motion capture data can be efficiently represented as a series of motion pattern indices. This algorithm is especially effective for long sequences of motion capture data with repetitive motion styles. Tournier et al. [27] use principal geodesic analysis (PGA) to build a descriptive model of the pose data of a motion, keeping only the leading principal geodesics. This model is then used in an inverse kinematics system to synthesize poses that not only match the end-joint constraints, but also keep the interior joint positions very close to those of the input data. Given this pose model, only the compressed end-joint trajectories and the root joint position/orientation need to be stored in order to recover the motion using inverse kinematics.

2.2. Floating-point compression

Many different kinds of data sets may need to be represented in floating-point format. If floating-point numbers can be effectively compressed, high compression ratio of these data can be achieved. There have been a lot of methods developed for compressing and transmitting floating-point numbers, e.g., images [31,32], audio [9], 3D geometry data [16,28], high-dimensional scientific data [22] and linear streams [4,5,25].

Isenburg et al. [16] propose a lossless algorithm to encode floating-point geometry of triangle meshes. The algorithm first predicts a new position by using the parallelogram predictor. It then breaks each 32-bit floating-point number into separate parts and encodes them in different contexts with context-based arithmetic coding. Lindstrom and Isenburg [22] propose a fast and efficient compression algorithm for high-dimension floating-point data. After obtaining the predicted position using the Lorenzo predictor [13], 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 [4,5] and Ratanaworabhan et al. [25] propose methods for effective and lossless compression of sequences of 64-bit floating-point data. They first sequentially predict each value of the data sequence, and then do some bit-wised operations between the actual value and the predicted value. Their methods are very fast and can meet the high throughput demands of scientific computing environments.

3. The alpha parallelogram predictor

3.1. Splitting of motion capture data

Let us denote the motion capture data as a $n \times l$ dimensional matrix $\mathbf{M}$:

$$\mathbf{M} = (m_{i,j})_{n \times l} \tag{1}$$

where n is the number of frames (represented as rows in $\mathbf{M}$) and l is the number of DOFs (represented as columns in $\mathbf{M}$). Fig. 1 shows a simplified skeleton structure, with some negligible joints removed. Each row of $\mathbf{M}$ corresponds to a pose of the vir-

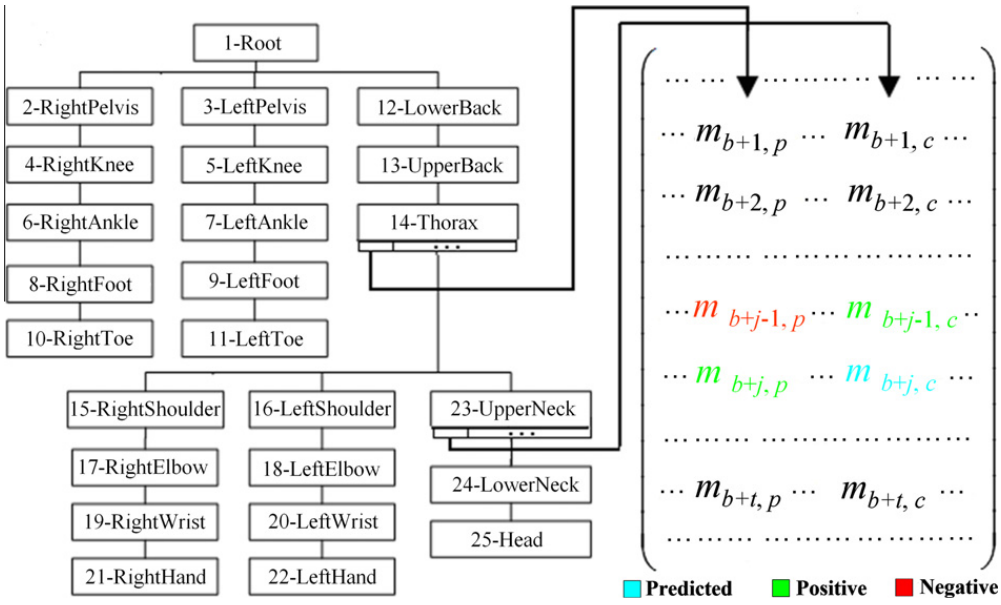


Fig. 1. The hierarchy of a virtual human skeletal structure (left) and an example of APP (right).

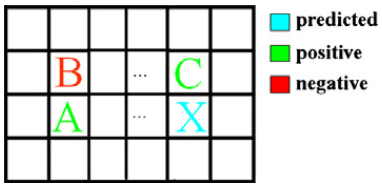


Fig. 2. The parallelogram predictor for 2D data.

tual human at a certain time frame. Each column of **M** represents either a rotation in Euler angle or a displacement of the model from a fixed origin.

The motion data should be split into some equal sized clusters before prediction, in order to have fewer prediction parameters and take up less storage space. We will discuss the splitting parameter trade-off in Section 5.2.

The n rows of motion data, represented as matrix **M**, is split into *ClstNm* clusters of equal size. If each cluster has t (referred to as the splitting parameter) rows, the number of clusters is then:

$$ClstNm = \lceil n/t \rceil \tag{2}$$

where $\lceil\ \rceil$ is the ceiling operator. The number of rows of the last cluster is:

$$LstClsRow = n \bmod t \tag{3}$$

3.2. Prediction rule

From Refs. [15,26], given the actual values of A and B as shown in Fig. 2, the parallelogram predictor computes a predicted value, P_X, of the actual value, X, as follows:

$$P_X = C + A - B \tag{4}$$

In order to minimize the difference between P_X and X, we propose here to modify the parallelogram predictor by including a parameter α as follows:

$$P'_X = C + \alpha(A - B) \tag{5}$$

Theoretically, we may compute an ideal α value for each cluster. However, in order to minimize the storage cost of these α values, we construct an α lookup table, from which an α value closest to the ideal α value can be chosen to perform the prediction. In this way, we only need to store the indices to the lookup table to save storage space.

Fig. 1 shows the hierarchical structure of a virtual human skeletal model and an example of the APP. The right hand side of Fig. 1 only shows two columns and some rows of cluster i. We assume that c is a DOF index of child joint C and p is the DOF index of the parent of joint C. The numbers of DOFs of joint 23 and joint 14 are indexed by c and p, respectively. For cluster i, we can predict $m_{b+j,c}$ by using the following formula:

$$\begin{aligned} P_{b+j,c} &= m_{b+j-1,c} + \alpha(m_{b+j,p} - m_{b+j-1,p}) \\ b &= t(i-1) \end{aligned} \tag{6}$$

where

$$\begin{cases} j \in [2,t] & \text{when } i = 1 \\ j \in [1,t] & \text{when } 1 < i < ClstNm \\ j \in [1, LstClstRow] & \text{when } i = ClstNm \end{cases}$$

Note that when c in Eq. (6) represents the DOF index of the root joint, $m_{b+j,p}$ and $m_{b+j-1,\,p}$ will become zero. Note also that j can start from 1 when i is not equal to 1. This means that the APP is seamless because the last row of cluster $i-1$ is used to predict the first row of cluster i. Fig. 3 shows an example of how the seamless prediction works. We assume that the motion data only contain 12 frames, divided into three clusters of four frames each. The last frame of cluster 1 (marked by a red dash box) is used to predict the first frame of cluster 2 and so on.

3.3. The ideal α and constructing the α lookup table

When using Eq. (6) for prediction, the prediction error of DOF index c of cluster i is:

$$\varepsilon_{i,c} = \sum_{j=1}^{t} (m_{b+j,c} - P_{b+j,c})^2 = \sum_{j=1}^{t} ((m_{b+j,c} - m_{b+j-1,c}) + \alpha(m_{b+j-1,p} - m_{b+j,p}))^2 \tag{7}$$

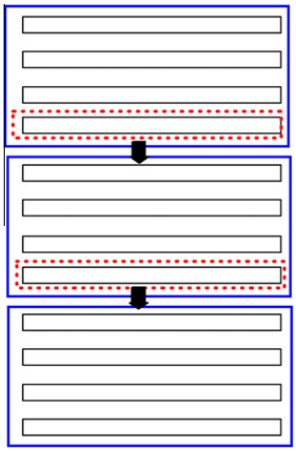


Fig. 3. An example to illustrate seamless prediction.

where $m_{b+j,c}$ is the actual value corresponding to the prediction value $P_{b+j,c}$. The minimum prediction error $\varepsilon_{i,c}$ can be computed as:

$$\min(\varepsilon_{i,c}) = \sum_{j=1}^{t}(m_{b+j,c} - m_{b+j-1,c})^2 - \frac{\left(\sum_{j=1}^{t}(m_{b+j,p} - m_{b+j-1,p})(m_{b+j-1,c} - m_{b+j,c})\right)^2}{\sum_{j=1}^{t}(m_{b+j,p} - m_{b+j-1,p})^2} \tag{8}$$

when

$$\alpha = \frac{\sum_{j=1}^{t}(m_{b+j,p} - m_{b+j-1,p})(m_{b+j-1,c} - m_{b+j,c})}{\sum_{j=1}^{t}(m_{b+j,p} - m_{b+j-1,p})^2} \tag{9}$$

The α value computed from Eq. (9) is referred to as the ideal α value, since we can obtain a minimum error when using it for prediction. Unfortunately, each ideal α value will occupy 32 bits to store. With a lot of α values to store, it will offset the compression ratio significantly. For example, with a motion sequence of 640 frames and 62 DOFs, if each cluster contains 10 frames, we will have a total of $64 * 62$ or 3968 α values to store. As each α value occupies 32 bits, the total data size is then $3968 * 4$ or 15.5k Bytes. Using the proposed algorithm, we can compress the motion data to 64k Bytes. However, as we also need to store the α values, the overall file size becomes 79.5k Bytes. Storing these ideal α values is equivalent to adding 24% overhead on the motion data, which significantly reduces the compression ratio. We need an effective method to compress them.

To address this problem, we first construct an α lookup table. Given an ideal α value, we simply look up the nearest α value from this lookup table and use it for prediction with Eq. (6). In this way, we only need to store the indices to the lookup table. Assuming we use k bits to index $2^k\alpha$ values in the lookup table, the index data size is:

$$IdxSize = \lceil ClstNm \times l \times k/8 \rceil \text{ Bytes} \tag{10}$$

where l is the number of DOFs, and $ClstNm$ is the number of clusters obtained from Eq. (2). Using the same example above, if k is set to 3, the $IdxSize$ will be 1.45k, which is only 9% of the total data size of the ideal α values. Storing these index values is equivalent to adding only 2.3% overhead on the motion data.

4. Floating-point compression

To further save storage space, we divide the predicted value, $P_{b+j,c}$, computed from Eq. (6) into three components: sign (Sgn_p), exponent (Exp_p) and mantissa (Mts_p). Similarly, we also divide the actual value, $m_{b+j,c}$, into three components: sign (Sgn_a), exponent (Exp_a) and mantissa (Mts_a). Their corrections are compressed separately using context-based arithmetic coding.

Algorithm 1 shows the pseudo code of our floating-point compression algorithm. In Step 4, bits ($Mts_a - Mts_p$) computes the bit length of ($Mts_a - Mts_p$). Our method is different from Isenburg's method [16] in that we encode the actual exponent and mantissa (Steps 3 and 5 in Algorithm 1) without using context-based arithmetic coding, since the APP performs better than the parallelogram predictor and in most of cases our algorithm will be executing Steps 1, 2 and 4. If context-based arithmetic coding is used in Step 5, the memory usage of the context for 23-bit mantissa will be significant.

Algorithm 1. APP floating-point compression

Input: $m_{b+j,c}$, $P_{b+j,c}$

1. $m_{b+j,c}$ and $P_{b+j,c}$ are divided into sign, exponent and mantissa: Sgn_a, Exp_a, Mts_a and Sgn_p, Exp_p, Mts_p, respectively.
2. **if** (Sgn_a == Sgn_p) && ($|Exp_a - Exp_p| < 4$) **then**
   Encode ($Exp_a - Exp_p$) using context *expo_diff*
3. **else**
   Encode Exp_a
   endif
4. **if** (Sgn_a == Sgn_p) & & (Exp_a == Exp_p) &&
   (bits ($Mts_a - Mts_p$) < 12) **then**
   Encode ($Mts_a - Mts_p$) using context MTS [Exp_a]
5. **else**
   Encode Mts_a
   endif

Fig. 4 shows three examples, with all the floating-point values shown in hexadecimal form. The calling sequences indicate the steps being executed.

5. Results and discussions

In this section, we present a number of experiments to demonstrate the performance of the proposed method. Our focus is to show the relationship between the average prediction error and the compression ratio under different parameter values.

Before we present the experiments, we first define three metrics that we use for evaluation, *MCR* (Motion Compression Ratio), *TCR* (Total Compression Ratio), and *APE* (Average Prediction Error), as follows:

$$MCR = \frac{RawSize}{CompMotionSize} \tag{11}$$

$$TCR = \frac{RawSize}{CompMotionSize + IdxSize} \tag{12}$$

where *RawSize* is the original file size before compression, *CompMotionSize* is the motion data size after compression and *IdxSize* is the index data size after compression as obtained from Eq. (10).

$$APE = \frac{\sum_{i=1}^{n}\sum_{j=1}^{l}|m_{i,j} - P_{i,j}|}{n \cdot l} \tag{13}$$

where n is the total number of frames and l is the number of DOFs. $m_{i,j}$ is the actual value represented as the (i,j) element of matrix **M** as defined in Eq. (1). $P_{i,j}$ is the correspondent predicted value calculated from Eq. (6).

5.1. Compression ratio with different parameters

Our algorithm has three associated parameters: t, k and the α lookup table, where t is the number of frames per cluster (or the splitting parameter) and k is the number of bits used to index the α lookup table. In this subsection, we first investigate the statistical distribution of the ideal α value, which is important in determining the parameter trade-offs. We will then show the results on compression ratio under different combinations of the parameter values.

The ideal α values from 1021 different motion clips with different splitting parameters are collected and summarized as shown in Fig. 5. Most α values, up to about 50%, fall within the interval of $[-0.5,0.5]$, 21% within the interval of $[-1,-0.5)\cup(0.5,1)$, 18% within the interval of $[-2,-1)\cup(1,2]$, and 7% within the interval of $[-5,-2)\cup(2,5]$. Finally, the remaining 4% fall within the interval of $(-\infty,-5)\cup(5,+\infty]$.

Fig. 6 shows the distribution of the ideal α values of a 641-frame dancing clip, with splitting parameter $t = 10$. There are a total of 91% of the ideal α values falling within the interval of $[-2,2]$. We can also see from Fig. 6 that the closer to the x-axis, the denser the ideal α values are. This shows that the distribution of the ideal α values is non-uniform, which agrees with the non-uniform distribution shown in Fig. 5. The non-uniform distribution of the ideal α values suggests that the α lookup table should also have a non-uniform distribution.

Table 1 shows three non-uniformly spaced α lookup tables and a uniformly spaced α lookup table, with several different parameter combinations. Fig. 7 shows the corresponding *TCR* results under these parameter combinations.

From Fig. 7, we can see that k3-uniform is the worst in terms of compression ratio. This shows that a uniformly spaced α lookup table is not a good choice. Fig. 7 also shows that k3-non-uniform and k4-non-uniform have nearly the same *TCR* results, although the α lookup table with $k = 4$ is larger than that with $k = 3$. The reason is that, although a larger lookup table

$m_{b+j,c}$	$P_{b+j,c}$	Calling Sequence
-0.140821 1 7d 103361 Sgn_a Exp_a Mts_a	-0.140834 1 7d 1033c9 Sgn_p Exp_p Mts_p	①②④
0.011 0 78 343958	-0.021 1 79 2c0831	①③⑤
12.8 0 82 4ccccd	12.74 0 82 4bd708	①②⑤

Fig. 4. Examples to illustrate how the floating-point compression algorithm works.

guarantees a higher *MCR* (Motion Compression Ratio), a larger k also leads to a higher storage cost of the indexing data, which to some extent offsets the increase in *MCR*. Fig. 8 shows that the *MCR* increases less significantly as k increases from 3 to 4 than as k increases from 2 to 3. This even leads to the situation that the *MCR* at $k = 3$ and $t = 3$ is higher than the *MCR* at $k = 4$ and $t = 18$.

Fig. 8 also demonstrates that with the same parameter k, the smaller parameter t, the higher the *MCR* is. We will discuss this issue in the following subsection.

5.2. Average prediction error and compression ratio

In Fig. 9, we plot the compression ratio (left axis) and the prediction error (right axis) against splitting parameter t. We can see that as t increases, the *MCR* decreases while the *APE* (Average Prediction Error) increases. This is because when t is smaller, we obtain an ideal α value that can better balance all frames within the cluster as shown by Eq. (7). This leads to a smaller *APE* and hence a higher *MCR*. It means that we get the highest *MCR* when t is set to 1, i.e., we have one ideal α value for each frame. However, in that situation, the size of the indexing data will be very large, which will cause a low *TCR* (Total Compression Ratio).

The selection of t is a trade-off between the size of the indexing data and the *APE*, as shown in Fig. 9. If t is small, the *APE* will be small, which will lead to a high *MCR* but the indexing data will be large too. On the other hand, if t is large, the *APE* will be high leading to a small *MCR* as well as a lower *TCR*, although the size of the indexing data will be small. This explains that there is an optimal value for t at which we obtain the highest *TCR*.

5.3. Comparison with other compression tools

In this experiment, we compare the performance of our method with some popular compression tools, MPEG-4 FBA (Face Body Animation) [24], gzip, and Rar. We also compare with the method that uses the parallelogram predictor but without the alpha predictor component (Non-Alpha PP).

MPEG-4 provides two alternative methods for coding FBA, the linear prediction (LP) scheme and the discrete cosine transform (DCT) scheme. Both schemes have quantization steps. Since the quantization process introduces information loss, we omit this step in our implementation of these schemes. In the linear prediction scheme, the data of an I-frame is coded without prediction. For the subsequent P-frames, the values of a P-frame, F_i, are predicted from the values of the previously decoded I-frame or P-frame, F_{i-1}. The prediction error is then coded using arithmetic coding. In the DCT scheme, DCT encoding

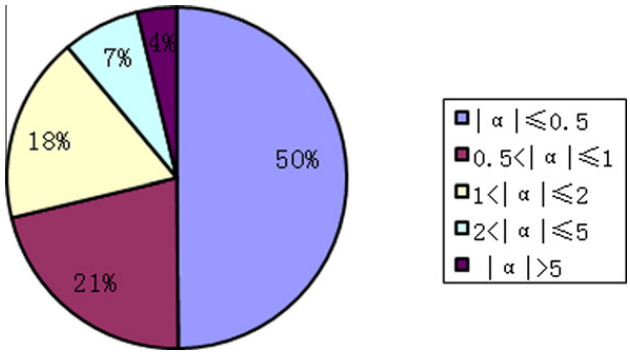


Fig. 5. The statistical distribution of the ideal α values.

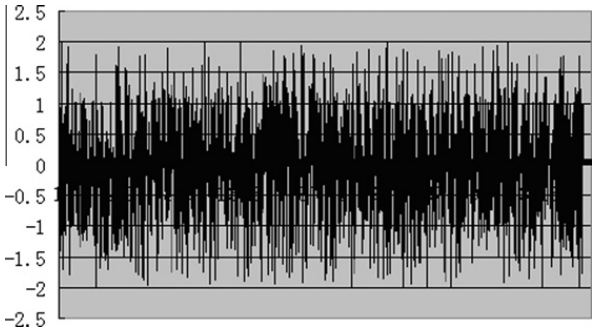
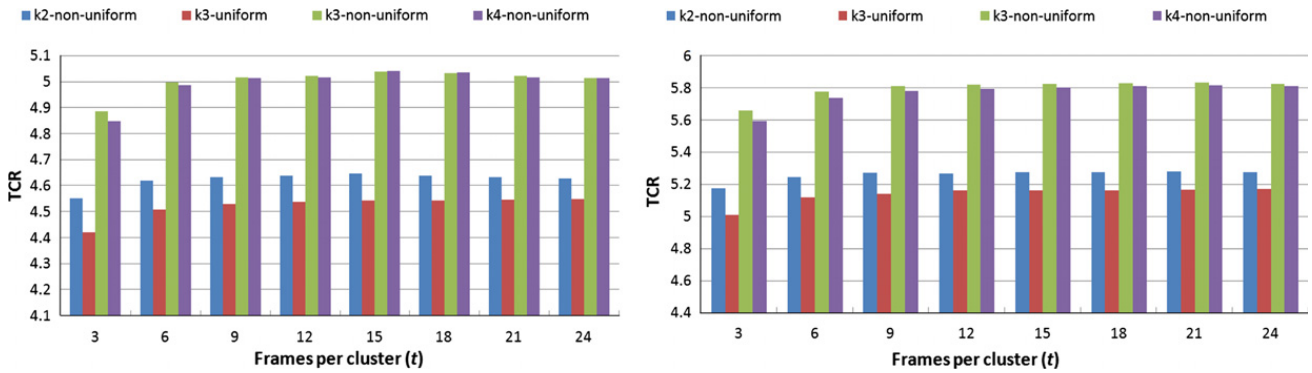


Fig. 6. The distribution of the ideal α values of a 641-frame dancing clip.

Table 1
Four parameter combinations.

Name	k	α Lookup table
k2-non-uniform	2	−2, −0.5, 0, 0.5
k3-uniform	3	−1.75, −1.25, −0.75, −0.25, 0.25, 0.75, 1.25, 1.75
k3-non-uniform	3	−1, −0.5, −0.25, 0, 0.25, 0.5, 1, 2
k4-non-uniform	4	−2, −1.25, −1, −0.75, −0.5, 0.25, −0.125, 0, 0.125, 0.25, 0.5, 0.75, 1, 1.25, 2, 5



(a) Sitting (watching TV) *(b) Dancing (break dancing)*

Fig. 7. The TCR results under different parameter combinations as shown in Table 1.

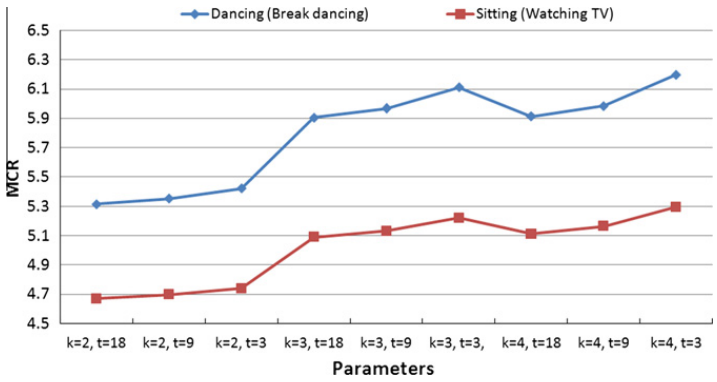


Fig. 8. The MCR results under different parameter combinations for motion clips.

is applied to 16 consecutive frames. After this process, DC and AC coefficients are coded separately. While a DC value is coded using the previous DC coefficient as a prediction, the AC coefficients are directly coded. Table 2 compares the six lossless schemes in terms of compression ratio using different motion clips. In general, gzip and Rar perform better than the DCT and the LP schemes. However, our method (k3-non-uniform) performs significantly better than the other four methods, gzip by 60%, Rar by 56%, the DCT scheme by 99%, and the LP scheme by 85%. The reason is that our alpha parallelogram predictor can exploit both temporal coherency by predicting from immediate neighbors and spatial coherency by predicting from the

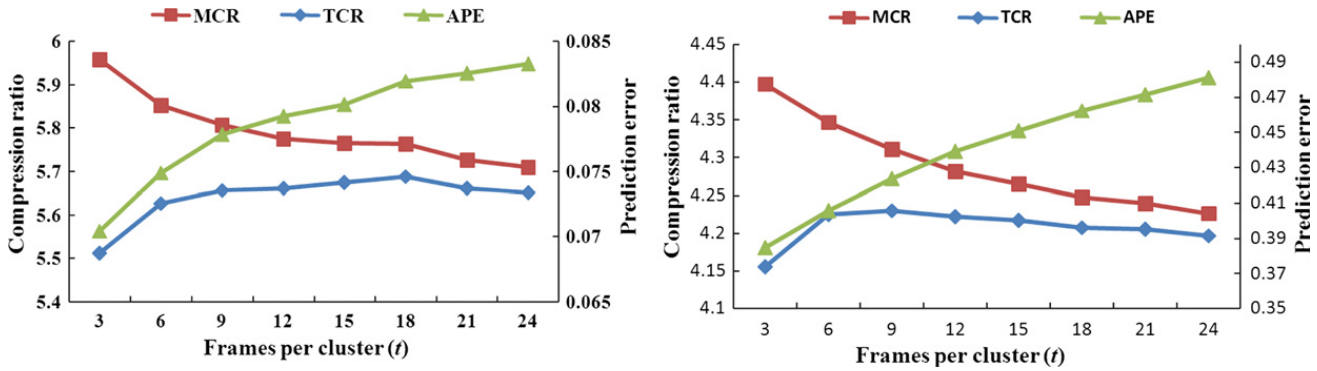


Fig. 9. The compression ratio and the prediction error of two motion clips.

Table 2
Compression ratios of six compression schemes: MPEG-4 LP, MPEG-4 DCT, standard gzip, standard Rar, Non-Alpha PP and our method (k3-non-uniform).

Motion clip	Number of frames	Compression ratio					
		gzip	Rar	DCT	LP	Non-Alpha PP	k3-non-uniform ($t = 15$)
Standing (lean back)	337	3.79	3.85	2.51	2.71	5.22	5.68
Sitting (watching TV)	396	3.01	3.12	2.56	2.84	4.66	5.04
Dancing (break dancing)	641	3.82	3.89	2.58	2.83	5.47	5.83
Jumping (fly kick)	933	2.38	2.43	2.79	2.83	3.78	4.22
Average compression ratio		3.25	3.32	2.61	2.80	4.78	5.19

parents. Our method also performs better than the Non-Alpha PP method by 8.6%. This is because the APP improves the prediction result and produces smaller corrections than the traditional parallelogram predictor.

Table 3 compares the average compression and decompression times (per frame) of the six schemes. We have conducted the experiment on a PC with an Intel E5400 CPU and 2 GB RAM. In general, all six methods are very efficient, with MPEG-4 LP being the most efficient one. Our method is slightly slower than gzip but faster than Rar. The Non-Alpha PP method takes about the same time as our method. This is because the introduction of the α parameter does not really increase the computational complexity of our method. Overall, all six methods can handle real-time compression and decompression of motion capture data without any difficulty.

6. Conclusion and future work

In this paper, we have proposed a novel lossless compression method for motion capture data. We have shown that by using our Alpha Parallelogram Predictor (APP) and by indexing the α values from a lookup table, we can effectively increase the compression ratio and at the same time keep the size of the additional indexing data small. Although the formulation of the predictor is simple, its predictive power is significant for motion capture data. The reason is that through predicting the DOF values of child joints from the DOF values of their immediate neighbors and parents, our APP can exploit both temporal coherency (predicting from immediate neighbors) and spatial coherency (predicting from parents). We have demonstrated that our method outperforms traditional compression tools. The complexity of our algorithm is also low, resulting in similar compression and decompression times as those of the traditional compression tools.

In this work, we have focused on developing a method that stores and transmits data in a streaming order. On the one hand, in some applications, random accessing the motion data is necessary. This can be achieved by first picking some frames as the key-frames, and then compressing the group of frames between two consecutive key-frames separately. A curve simplification algorithm [20] can be used here to pick the key-frames of a motion sequence. The first and the last frames also should be picked as the key-frames. After this process, since the first frame of each group of frames has to be stored in the original format instead of the difference format, there will be some negative effects on the compression ratios. Some adaptation and improvement on the algorithm will be demanded. On the other hand, batch transmission of motion capture data is necessary for sharing large databases over the internet. As inspired by Chen and Wu [7] and Lin et al. [21], we may

Table 3
Compression and decompression times of the six compression schemes: MPEG-4 LP, MPEG-4 DCT, gzip, Rar, Non-Alpha PP and our method (k3-non-uniform).

Computation time (per frame)	gzip	Rar	DCT	LP	Non-Alpha PP	k3-non-uniform ($t = 15$)
Compression time (ms)	0.187	0.211	0.256	0.153	0.191	0.193
Decompression time (ms)	0.182	0.212	0.245	0.150	0.187	0.188

cluster similar motions in the database and conduct the prediction among each cluster, in order to obtain small prediction errors and be able to share the α lookup tables. However, due to the introduction of the clustering process, the processing time will increase significantly. The adoption of some time-efficient clustering methods, such as the time-efficient *k*-means clustering method proposed by Chiang et al. [8], will be necessary. As a future work, we would like to explore these two issues.

Acknowledgements

We are very grateful to the anonymous reviewers for the constructive comments and suggestions on our paper. The work described in this paper was partially supported by a Key Project of National High-tech Research and Development Program of China (Grant No.: 2009AA062704), two Fundamental Research Funds for the Central Universities (DC120101073, DC120101077), the Doctoral Research Fund of DLNU (0710-110005), and two SRG Grants from City University of Hong Kong (Project Numbers: 7002768 and 7002664).

References

[1] O. Arikan, Compression of motion capture databases, ACM Transactions on Graphics 25 (3) (2006) 890–897.
[2] P. Beaudoin, P. Poulin, M. van de Panne, Adapting wavelet compression to human motion capture clips, in: Proc. Graphics Interface, Montreal, Canada, 2007, pp. 313–318.
[3] H. Briceño, P. Sander, L. McMillan, S. Gortler, H. Hoppe, Geometry videos: a new representation for 3D animations, in: Proc. ACM Symp. on Computer Animation, San Diego, USA, 2003, pp. 136–146.
[4] M. Burtscher, P. Ratanaworabhan, High throughput compression of double-precision floating-point data, in: Proc. Data Compression Conference, Snowbird, USA, 2007, pp. 293–302.
[5] M. Burtscher, P. Ratanaworabhan, FPC: a high-speed compressor for double-precision floating-point data, IEEE Transactions on Computers 58 (1) (2009) 18–31.
[6] S. Chattopadhyay, S. Bhandarkar, K. Li, Human motion capture data compression by model-based indexing: a power aware approach, IEEE Transactions on Visualization and Computer Graphics 13 (1) (2007) 5–14.
[7] T. Chen, C. Wu, Compression-unimpaired batch-image encryption combining vector quantization and index compression, Information Sciences 180 (9) (2010) 1690–1701.
[8] M. Chiang, C. Tsai, C. Yang, A time-efficient pattern reduction algorithm for *k*-means clustering, Information Sciences 184 (4) (2010) 716–731.
[9] F. Ghido, An efficient algorithm for lossless compression of IEEE float audio, in: Proc. Data Compression Conference, Snowbird, USA, 2004, p. 429.
[10] Q. Gu, J. Peng, Z. Deng, Compression of human motion capture data using motion pattern indexing, Computer Graphics Forum 28 (1) (2009) 1–12.
[11] S. Gupta, K. Sengupta, A. Kassim, Compression of dynamic 3d geometry data using iterative closest point algorithm, Computer Vision and Image Understanding 87 (1–3) (2004) 116–130.
[12] I. Guskov, A. Khodakovsky, Wavelet compression of parametrically coherent mesh sequences, in: Proc. ACM Symp. on Computer Animation, Grenoble, France, 2004, pp. 183–192.
[13] L. Ibarria, P. Lindstrom, J. Rossignac, A. Szymczak, Out-of-core compression and decompression of large *n*-dimensional scalar fields, Computer Graphics Forum 22 (3) (2003) 343–348.
[14] L. Ibarria, J. Rossignac, Dynapack: space-time compression of the 3D animations of triangle meshes with fixed connectivity, in: Proc. ACM Symp. on Computer Animation, San Diego, USA, 2003, pp. 126–135.
[15] M. Isenburg, P. Alliez, Compressing polygon mesh geometry with parallelogram prediction, in: Proc. IEEE Visualization, Boston, USA, 2002, pp. 141–146.
[16] M. Isenburg, P. Lindstrom, J. Snoeyink, Lossless compression of floating-point geometry, Computer-Aided Design 37 (8) (2005) 869–877.
[17] Z. Karni, C. Gotsman, Compression of soft-body animation sequences, Computers and Graphics 28 (1) (2004) 25–34.
[18] L. Kovar, J. Schreiner, M. Gleicher, Footskate cleanup for motion capture editing, in: Proc. ACM Symp. on Computer Animation, Grenoble, France, 2004, pp. 97–104.
[19] J. Lengyel, Compression of time-dependent geometry, in: Proc. ACM Symp. on Interactive 3D graphics, New York, USA, 1999, pp. 89–95.
[20] I. Lim, D. Thalmann, Key-posture extraction out of human motion data by curve simplification, in: Proc. Int'l Conf. on IEEE Engineering in Medicine and Biology Society, Istanbul, Turkey, 2001, pp. 1167–1169.
[21] I. Lin, J. Peng, C. Lin, M. Tsai, Adaptive motion data representation with repeated motion analysis, IEEE Transactions on Visualization and Computer Graphics 17 (4) (2011) 527–538.
[22] P. Lindstrom, M. Isenburg, Fast and efficient compression of floating-point data, IEEE Transactions on Visualization and Computer Graphics 12 (5) (2006) 1245–1250.
[23] G. Liu, L. McMillan, Segment-based human motion compression, in: Proc. ACM Symp. on Computer Animation, Banff, Alberta, Canada, 2006, pp. 127–135.
[24] M. Preda, F. Preteux, MPEG-4 human virtual body animation, in: S. Bourges (Ed.), MPEG-4 Jump-Start, Prentice Hall, New Jersey, 2002.
[25] P. Ratanaworabhan, J. Ke, M. Burtscher, Fast loseless compression of scientific floating point data, in: Proc. IEEE Data Compression Conference, Snowbird, USA, 2006, pp. 133–142.
[26] C. Touma, C. Gotsman, Triangle mesh compression, in: Proc. Graphics Interface, Vancouver, BC, Canada, 1998, pp. 26–34.
[27] M. Tournier, X. Wu, C. Nicolas, A. Élise, R. Lionel, Motion compression using principal geodesic analysis, in: Proc. Eurographics, Munich, Germany, 2009, pp. 355–364.
[28] A. Trott, R. Moorhead, J. McGenley, Wavelets applied to lossless compression and progressive transmission of floating point data in 3-D curvilinear grids, in: Proc. IEEE Visualization, San Francisco, USA, 1996, pp. 355–388.
[29] M. Sattler, R. Sarlette, R. Klein, Simple and efficient compression of animation sequences, in: Proc. ACM Symp. on Computer Animation, Los Angeles, USA, 2005, pp. 209–217.
[30] P. Sloan, J. Hall, J. Hart, J. Snyder, Clustered principal components for precomputed radiance transfer, ACM Transactions on Graphics 22 (3) (2003) 382–391.
[31] B. Usevitch, JPEG2000 extensions for bit plane coding of floating point data, in: Proc. Data Compression Conference, Snowbird, USA, 2003, pp. 451–461.
[32] B. Usevitch, JPEG2000 compatible lossless coding of floating-point data, Journal on Image and Video Processing (1) (2007) 22.
[33] J. Yang, C. Kim, S. Lee, Compression of 3D triangle mesh sequences based on vertex-wise motion vector prediction, IEEE Transactions on Circuits and Systems for Video Technology 12 (12) (2002) 1178–1184.
[34] J. Yang, C. Kim, S. Lee, Semi-regular representation and progressive compression of 3D dynamic mesh sequences, IEEE Transactions on Image Processing 15 (9) (2006) 2531–2544.
[35] J. Zhang, C. Owen, Octree-based animated geometry compression, in: Proc. IEEE Data Compression Conference, Snowbird, USA, 2004, pp. 508–517