The Visual Computer
https://doi.org/10.1007/s00371-020-01840-6

ORIGINAL ARTICLE

# Character motion in function space

Innfarn Yoo[1] · Marek Fišer[1] · Kaimo Hu[1] · Bedrich Benes[1]



**Abstract**
We address the problem of animated character motion representation and approximation by introducing a novel form of motion expression in a function space. For a given set of motions, our method extracts a set of orthonormal basis (ONB) functions. Each motion is then expressed as a vector in the ONB space or approximated by a subset of the ONB functions. Inspired by the static PCA, our approach works with the time-varying functions. The set of ONB functions is extracted from the input motions by using functional principal component analysis and it has an optimal coverage of the input motions for the given input set. We show the applications of the novel compact representation by providing a motion distance metric, motion synthesis algorithm, and a motion level of detail. Not only we can represent a motion by using the ONB; a new motion can be synthesized by optimizing connectivity of reconstructed motion functions, or by interpolating motion vectors. The quality of the approximation of the reconstructed motion can be set by defining a number of ONB functions, and this property is also used to level of detail. Our representation provides compression of the motion. Although we need to store the generated ONB that are unique for each set of input motions, we show that the compression factor of our representation is higher than for commonly used analytic function methods. Moreover, our approach also provides lower distortion rate.

**Keywords** Character motion · Functional principal component analysis · Orthonormal basis functions

## 1 Introduction

Articulated character motion editing, capturing, searching, and synthesizing present important challenges in computer animation. On the one hand, the amount of produced motion data grows rapidly which further exacerbates these challenges. On the other hand, despite the enormous progress in this field, the existing algorithms and methods still have some limitations. Among them the compact motion representation is one underlying common problem. The motion data is usually stored in its raw form as rotations and positions of the joints (or velocities and accelerations) that is space consuming and difficult to process. One of the promising approaches is encoding the motion data by using analytic basis functions (e.g., Fourier, Legendre polynomials, or spherical harmonics). These representations compress the input data, but they may introduce unwanted artifacts such as oscillations and may require high number of basis functions to capture all details. Moreover, synthesizing new motions from those representations may be difficult (Fig. 1).

✉ Bedrich Benes
bbenes@purdue.edu

[1] Purdue University, West Lafayette, USA

A body of previous work addresses the problem of motion synthesis. One class of methods uses motion graphs for representing motion connectivity and synthesizing new motions [21,27,30,39]. Functional analysis has also been applied for encoding and searching motions [8,34,43]. Statistical approaches extract probabilities from motion data with the aim of low-dimensional expression, predicting smoothly connected motions [17,24,28,47], and using physics-based representations to generate new motions by simulation [31, 47]. Although these methods are well-suited for their particular area, they usually require either a large amount of data to represent the motion, or substantial effort for new motion synthesis.

Our work is motivated by recent advances in functional data analysis and modeling in mathematics and statistics [9,11,37,48]. The key observation of our work is that for *a given set* of input motions, we can extract *an optimal set of orthonormal basis functions*. While analytic basis functions have been used for encoding motions, ours extracted ONB functions are tailored for the given set of input motions and they are optimal in the sense that they provide the best coverage for the range of the provided input motions. Each input motion is then simply represented by its coordinates as

Published online: 04 April 2020

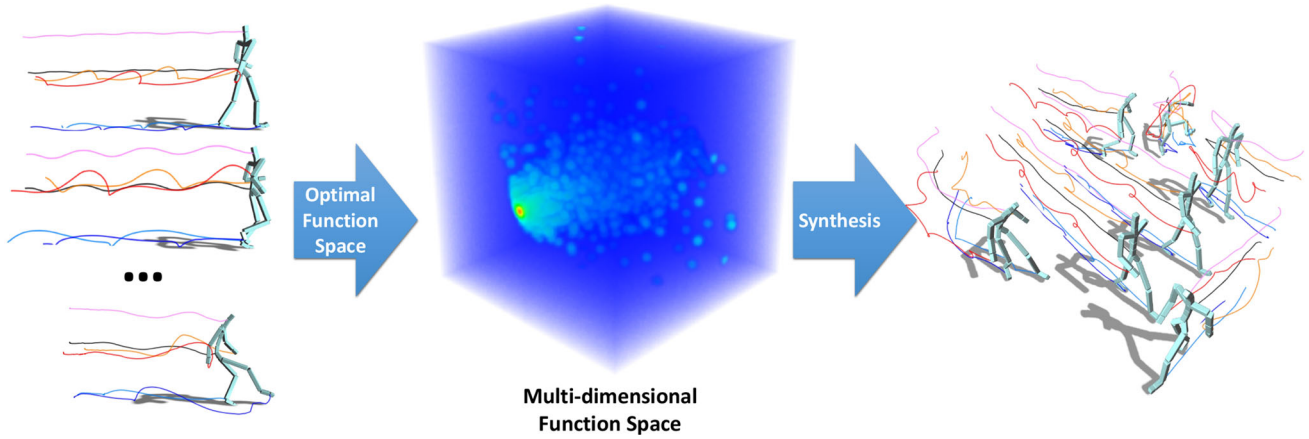



**Fig. 1** A set of raw motion data (left) is used to find orthonormal basis functions by functional principal component analysis (FPCA). Each input motion is encoded as a motion vector in the function space (center). While the novel representation provides compression of the input data, the motion vectors synthesize new motions by interpolating their coordinates (right)

a *motion vector* in the ONB function inner-product space or approximated by using a subset of ONB functions.

The input to our framework is a set of unlabeled raw motion data of articulated characters, from motion capturing databases, hand-made animation, or results of physics-based simulation. In the first step we extract the ONB functions for the input set by using functional principal component analysis (FPCA) and represent each input motion as a motion vector in the ONB space just by its coordinates. This compresses the input data and converts it into a compact representation. The vector representation of motions allows for continuous (linear, or higher order) interpolation in the inner-product space formed by the ONB functions. We can synthesize a new motion simply by selecting two points in the ONB space, and by interpolating between a successions of the closest points between the two motions. However, for some motions, a simple interpolation might not be suitable because they are dissimilar or far from each other. For this case, we have adopted the connectivity optimization from [21] to work for motion vectors in the ONB. In addition, this representation allows for the measure of distance between motions and it provides a compression of motion data. Although we need to store the generated ONB that are unique for input motions, the compression factor is higher than the commonly used analytic function methods. We claim the following contributions:

1. a novel representation of motions by extracting optimal orthonormal basis functions,
2. compact representation of motions by encoding motions as motion vectors into function space,
3. scalable motion reconstruction so that can be used for level of detail (LOD), and
4. motion synthesis via interpolation and partial connectivity optimization in function domain.

## 2 Previous work

Here we discuss the literature related to motion analysis, synthesis, clustering, and dimensionality reduction.

### 2.1 Function analysis of motion

The idea of representing motion in some other domain is rather old. For example, the Fourier transform has been used frequently in signal processing. Unuma et al. [43] apply the Fourier transform to analyze, extract, and synthesize motions by comparing existing motion data. Ormoneit et al. [34] detect cyclic behavior of human motions using function analysis and extract functional principal components in the Fourier domain to remove high frequency components. Similarly, Chao et al. [8] use spherical harmonic (SH) basis for compressing and encoding motion trajectories. They also retrieved similar motions by encoding and comparing user's trajectory sketches. Coffey et al. [9] used PCA to analyze human motion data, but they did not provide a way of synthesizing new ones. Just recently, Du et al. [11] used scaled FPCA to adapt different types of motion for character animation in gaming.

Our method does not use analytic orthonormal basis (ONB) functions, but we extract ad hoc ONB functions from existing motion data. Our extracted ONB functions are guaranteed to be optimal in a sense that we can determine the error threshold and minimize the error based on the number of ONB functions.

### 2.2 Motion graphs

Kovar et al. [21] provide a new distance metric of keyframes and introduce a graph structure, called *motion graphs*, for motion keyframes' connectivity. Synthesizing a new motion can be done by following a path in the graphs. Their work extends in many directions. Lai et al [23] use motion graphs for small crowds by simulation and constraints. Heck and Gleicher [14] find suitable transitions of their parameterized motion space using sampling methods. Reitsma and Pollard [38] provide a task-based quality measure, such as different types of motions, navigation ability, and embedding additional data. Searching optimal interpolation path of motion graphs is researched in Safonova and Hodgins [39]. Beaudoin et al. [6] provide grouping of similar motions and construct a motif graph that allows searching and constructing new motions. Zhao and Safonova [52] improve the connectivity by constructing well-connected motion graphs using interpolated keyframes. Lee et al. [27] show a novel method, called *motion fields*, for interactively controlling locomotion by introducing a new distance measure of motions, which is combined with keyframe similarity and velocity. Recently, Min and Chai [30] combined semantic information with motion graphs to synthesize a new motion from simple sentences.

Compared to motion graphs, our method represents motions as *motion vectors* in an ONB function space. Our method allows for compact motion representation and provides a distance metric preservation. Also, an interpolation between any two motions can be done by a simple vector interpolation. We also generalize the optimization from motion graphs for the ONB space.

### 2.3 Motion clustering

Alon et al. [2] use multiple hidden Markov models (HMMs) combined with probability, and Kovar et al. [21] generate graph structures of motion data by calculating geometric distances of motion keyframes. The motion graphs were later extended in many ways, such as by using different parametrization Kovar and Gleicher [14,20], they were combined with clustering Beaudoin et al. [6], their connectivity was improved Zhao and Safonova [52], optimal search was suggested Safonova and Hodgins [39], and their evaluations was introduced in Reitsma and Pollard [38]. Keogh et al. [19] solved an important drawback of Dynamic Time Warping (DTW) by allowing users to search similar motions without global and uniform scaling. Barbič et al. [5] show three different approaches based on PCA and Gaussian mixture model (GMM) for automatic segmentation of motion captured data. Forbes and Fiume [12] introduce a pose distance metric and a search algorithm using weighted PCA. Zhou et al. [54] applied an unsupervised learning method for clustering temporal human motion data, and create a partition of segments. Temporal segmentation of human motion is studied by Vögele et al. [44]. However, our method provides naturally defined distance metric in function space that allows for an easy calculation of similarity of motions and clustering.

### 2.4 Motion dimensionality reduction and search

Gall et al. [13] combine global optimization using Gaussian Process (GP), filtering, and local optimization to reconstruct 3D human motion. Mordatch et al. [31] developed a method that can perform user-specified tasks by using GP and learning motions in reduced low-dimensional space. Zhou and De la Torre [53] extend the DTW method by introducing generalized time warping (GTW) that overcomes DTW drawbacks for human motion data. Lau et al. [24] analyze and learn from motion data using dynamic Bayesian network (DBN) and synthesized new variations of motions. Ikemoto et al. [17] exploit generalizations of GP for motion data so that it allows users to edit motion easily. Arikan suggested a motion compression method that is based on clustered principal component analysis (CPCA) in Arikan [3]. Liu and McMillan [29] applied segmentation and PCA for motion, and compressed the motion data. In addition, Tournier et al. [42] provided a novel principal geodesic analysis (PGA), and achieved high compression ratio. A method that uses templates to annotate mocap data has been introduced by Müller and Röder [33] and a direct mocap data annotation has been developed by Müller et al. [32]. Additionally, motion factorization that also allows completion of missing data or noise reduction has been presented by Akhter et al. [1]. Recently, Hou et al. [15] have proposed a tailored transform coding for compression of motion capture data.

Shin and Lee [40] introduced a system that allows for low-dimensional representation and extraction of human motion, its interactive editing, and mapping back to the character. Similarly, Yoo et al. [49] used bilateral surfaces as a representation for lower-dimensional character retiming.

Several works attempt to search directly in mocap data, for example the Motion Explorer system of [7] or the GPU-based approach of Bernard et al. [51] that also allows for quick editing and composition of character motion.

Although the PCA-based method or dimensionality reduction methods studied in many directions, there are several differences between our method and the previous approaches. First, our method does not provide mocap data search. Further, our method provides several additional properties such as motion distance metric, level of detail, and fast reconstruction. Moreover, we interpret motions as a set of continuous functions so that it provides mathematically well-defined distance in function space. Also, our method does not perform

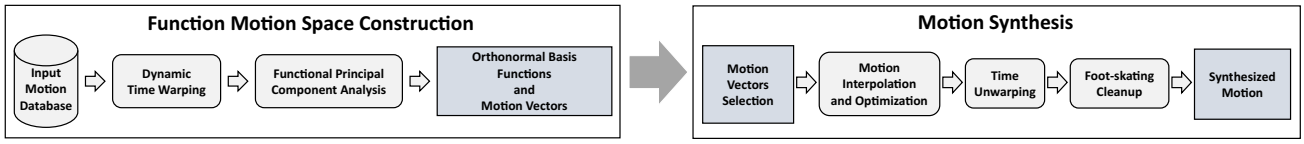



**Fig. 2** An input set of motions is analyzed and an orthonormal basis is found by using functional principal component analysis. The input motions are then encoded as a set of motion vectors in the ONB forming a function motion space. Novel motion can be synthesized by interpolating through existing motion vectors or by optimization directly in the ONB

dimensionality reduction, and it allows for an easy motion synthesis by motion vector interpolation or optimization.

## 3 Overview

Figure 2 shows an overview of our method that consists of two parts: (1) function motion space construction and (2) motion synthesis. The input is a set of input motions. During the first step, we extract orthonormal basis functions (ONB) and represent (approximate if we do not use all ONB) each input motion as a *motion vector* that form a *function motion space*. In the second phase, the motion vectors are used to synthesize new motions.

The input character motion data stores positions and rotations of joints and the data can originate from motion capturing, physics-based animation, manual creation, or similar. In the first step we generate the ONB by using functional principal component analysis (FPCA) for all motions. Then, we obtain the coordinates of each input motion in the ONB space. We call the ONB encoded motions *motion vectors*, because they are represented only by their coordinates in the corresponding ONB, and we call the set of encoded motions in the ONB the *function motion space*. The resulting ONB and motion vectors are smaller than the input data providing a compressed and compact representation of the motions. Moreover, the ONB representation allows for an easy motion synthesis for each pair of vectors by simply interpolating their coordinates. It is important to note that the ONB form a space with a distance metric. We can therefore measure the distance of two motions.

During the motion synthesis, the user defines the start and the end of the motion by selecting two points in the function motion space. The new motion can be generated by interpolation, a process that is suitable for two closely positioned motions. If the points are far from each other, we automatically traverse the space and find the shortest path between the closest motion vectors, effectively combining the animations together from the closest possible candidates. Using a subset of the OBN or points that are too far can result in the combination of two motions that is not visually plausible and introduce, e.g., foot skating. In this case, we apply motion optimization from [21] that has been modified and adapted to work directly in the ONB.

## 4 Orthonormal basis functions and motion vectors extraction

The key idea of our approach is that an animated character motion $\mathbf{m}(t)$ (see Sect. 5.3) can be represented as a vector with an optimal number of orthonormal basis function. Although the idea of representing motion data by using basis function has been already used in computer graphics, previous approaches use given *fixed* (analytic) basis functions such as Fourier (e.g., Unuma et al. [43]) or Spherical Harmonics (e.g., Chao et al. [8]). Those functions attempt to cover all possible motions by a set of a priori given analytic basis functions. Not only this representation is not optimal for a given input set, but also the analytic basis functions may need a large number of coefficients to reduce oscillation of reconstructed curves or to capture fine details.

We use basis functions that are extracted from (a group of) input motions. We use functional principal component analysis (FPCA) to extract our basis that covers the important motions in decreasing order. It also provides the best coverage of the space by the set of functions (see [37,48] for details of FPCA). In this section, we introduce the orthonormal function basis extraction and show how it is applied to motion encoding.

The *function representation* $\widetilde{f}(t)$ is an approximation of a function $f(t)$ and is expressed as

$$\widetilde{f}(t) = \mu(t) + \sum_{i=0}^{n} c_i b_i(t), \tag{1}$$

where $\mu(t)$ is the mean function representing the average of the analyzed functions, $B = \{b_0(t), b_1(t), \ldots, b_n(t)\}$ is the ONB, and $(c_0, c_1, \ldots, c_n)$ are the coordinates of $\widetilde{f}(t)$ in the inner-product function space.

The error $E(t)$ of the approximation is

$$E(t) = \|f - \widetilde{f}\| = \left( \int_{t_a}^{t_b} |f(s) - \widetilde{f}(s)|^2 \mathrm{d}s \right)^{1/2}. \tag{2}$$

*Distance of two vectors in ONB space* Let us assume that two functions, $f_1(t)$ and $f_2(t)$, are approximated by orthonormal basis functions, $b_1(t), b_2(t), \ldots, b_n(t)$, so that the coefficients are $c_{11}, c_{12}, \ldots, c_{1n}$ for $\widetilde{f}_1(t)$ and $c_{21}, c_{22}, \ldots, c_{2n}$ for $\widetilde{f}_2(t)$. The mean function of the two

functions are $\mu(t)$, so that the approximated functions, $\widetilde{f}_1(t) = \mu(t) + \sum_{i=1}^{n} c_{1i} b_i(t)$ and $\widetilde{f}_2(t) = \mu(t) + \sum_{i=1}^{n} c_{2i} b_i(t)$.

The squared distance between the two functions is

$$D(\widetilde{f}_1(t), \widetilde{f}_2(t))^2 = \int_{t_a}^{t_b} \left(\widetilde{f}_1(s) - \widetilde{f}_2(s)\right)^2 \mathrm{d}s = \int_{t_a}^{t_b} \left(\sum_{i=1}^{n} (c_{1i} - c_{2i}) b_i(s)\right)^2 \mathrm{d}s$$

Since $b_1(t), b_2(t), \ldots, b_n(t)$ are orthonormal basis functions, any two basis functions, $b_i(t)$ and $b_j(t)$ satisfies $\langle b_i, b_j \rangle = \delta_{ij}$, where $\delta_{ij}$ is kronecker delta function. Thus,

$$\begin{aligned}
&\int_{t_a}^{t_b} \left((c_{11} - c_{21}) b_1(s) + \cdots + (c_{1n} - c_{2n}) b_n(s)\right)^2 \mathrm{d}s \\
&= \int_{t_a}^{t_b} (c_{11} - c_{21})^2 b_1(s)^2 \mathrm{d}s + \cdots \\
&\quad + \int_{t_a}^{t_b} (c_{1n} - c_{2n})^2 b_n(s)^2 \mathrm{d}s \\
&= \sum_{i=1}^{n} \int_{t_a}^{t_b} (c_{1i} - c_{2i})^2 b_i(s)^2 \mathrm{d}s = \sum_{i=1}^{n} (c_{1i} \\
&\quad - c_{2i})^2 \int_{t_a}^{t_b} b_i(s)^2 \mathrm{d}s \\
&= \sum_{i=1}^{n} (c_{1i} - c_{2i})^2 \langle b_i, b_i \rangle = \sum_{i=1}^{n} (c_{1i} - c_{2i})^2
\end{aligned}$$

The distance between two approximated functions is just distance of their coefficients. In particular, having two ONB functions (motion vectors)

$$\widetilde{f}_1(t) = \mu(t) + \sum_{i=1}^{n} c_{1i} b_i(t) \quad \widetilde{f}_2(t) = \mu(t) + \sum_{i=1}^{n} c_{2i} b_i(t),$$

the distance $D(\widetilde{f}_1(t), \widetilde{f}_2(t))$ between them in the ONB is calculated as the distance between two functions in the given inner-product space:

$$\begin{aligned}
D(\widetilde{f}_1(t), \widetilde{f}_2(t)) &= \left(\int_{t_a}^{t_b} \left(\widetilde{f}_1(s) - \widetilde{f}_2(s)\right)^2 \mathrm{d}s\right)^{1/2} \\
&= \left(\sum_{i=1}^{n} (c_{1i} - c_{2i})^2 \langle b_i, b_i \rangle\right)^{1/2} \\
&= \left(\sum_{i=1}^{n} (c_{1i} - c_{2i})^2\right)^{1/2}. \qquad (3)
\end{aligned}$$

### 4.1 Orthonormal basis extraction using FPCA

The input to the ONB extraction is a set of $K$ input functions $f_k(t)$, $k = 1, 2, \ldots, K$. The $f_k(t)$ are time-aligned components of the motion (for example the $y$-coordinate of the quaternion of rotation). We discretize functions, $f_i(t_j)$ to $Y_{ij}$ with equally spaced time steps

$$Y_{ij} = f_i(t_j) + \epsilon_{ij}, \qquad (4)$$

where $\epsilon_{ij}$ is the measurement error per data point (e.g., the error caused by motion capture).

The output of the ONB extraction is the set of ONB functions $b_i(t)$ and the mean component

$$\mu(t) = \frac{1}{K} \sum_{k=1}^{K} f_k(t), \quad \hat{\mu}(t_{ij}) = \frac{1}{K} \sum_{i=1}^{K} Y_{ij}. \qquad (5)$$

One of the important methods to extract orthonormal basis in spatial domain is principal component analysis (PCA) which maximizes space coverage for a given number of basis function. Similarly, functional principal component analysis (FPCA) [37] extracts ONB functions that approximate the given set of functions. The FPCA extract eigenfunctions that have maximal coverage of $f_k(t)$, and they are orthonormal to other eigenfunctions in decreasing order of importance.

The FPCA uses the covariance function $v(s, t)$

$$v(s, t) = \frac{1}{n} \sum_{k=1}^{n} f_k(s) f_k(t).$$

To find the basis $b_i(t)$, we find Fredholm function eigenequation that satisfies

$$\begin{aligned}
&\int v(s, t) b_i(t) dt = \rho b_i(s) \\
&\text{subject to } \langle b_i, b_j \rangle = \delta_{ij}, \qquad (6)
\end{aligned}$$

where $\delta_{ij}$ is the Kronecker delta, $\rho$ is an eigenvalue of the principal component, the orthonormal basis $b_i(t)$ is an eigenfunction, and the function inner-product $\langle f, g \rangle$ of two functions $f(t)$ and $g(t)$ is

$$\langle f_i, f_j \rangle = \int_{t_a}^{t_b} f_i(s) f_j(s) \mathrm{d}s, \qquad (7)$$

where $t_0 \leq t_a < t_b \leq t_m$. The raw covariances are calculated as

$$v_i(t_{ij}, t_{il}) = (Y_{ij} - \hat{\mu}(t_{ij}))(Y_{ij} - \hat{\mu}(t_{il})), i \neq j$$

and the estimation $v(s,t)$ is

$$\tilde{v}(s,t) = \frac{1}{n}\sum_{i=1}^{n} v_i(s,t) = \sum_{\lambda_k>0} \hat{\lambda}_k \hat{b_k}(s)\hat{b_k}(t) \tag{8}$$

where $\hat{\lambda}$ is the estimated eigenvector, and $\hat{b_k}$ is the estimated eigenfunction. Since $E(e_{ij}) = 0$ and the variance of error is

$$\mathrm{Var}(e) = \sigma^2 I,$$

the approximation of $\sigma^2$ can be estimated by

$$\hat{\sigma}^2 = \frac{2}{\tau}\int_{\tau} (\hat{V}(t) - \tilde{v}(t,t))\mathrm{d}t, \tag{9}$$

where $\hat{V}(t)$ is smoothed diagonal elements of $\tilde{v_i}$. The eigenfunctions $b_k$ can be obtained by

$$b_k = \hat{\lambda}_k \hat{b_k} \hat{\Sigma}_{Y_i}^{-1}(Y_i - \hat{\mu}),$$

where $\hat{\Sigma}_{Y_i} = \tilde{v} + \hat{\sigma}^2 I$.

Each eigenfunction $b_i(t)$ provides some coverage of the input and the algorithm is executed until the average length of the residuals of the input is under user-defined percentage. This indirectly controls the actual number of the ONB functions.

### 4.2 Coordinates in the ONB function space

Having extracted the ONB $b_i(t)$ from $f_k(t)$, we can represent each input function $f_k(t)$ in this space as $\tilde{f_k}(t)$ with its coordinates $(c_0, c_1, \ldots, c_n)$ [see Eq. (1)]. The coordinates $(c_0, c_1, \ldots, c_n)$ are found by

$$c_j = \langle f_i, b_j \rangle = \int_{t_a}^{t_b} f_i(s) b_j(s)\mathrm{d}s, \tag{10}$$

where $b_1(t), b_2(t), \ldots, b_n(t)$ are the ONB functions. The coordinates $(c_0, c_1, \ldots, c_n)$ are the coefficients that are the best approximation in the space formed by the given ONB functions.

The ONB functions form a Hilbert space that has vector space characteristics such as distance measure and triangle inequality. A motion can be represented as a linear combination of coefficients (coordinates) with ONB functions. Transition between two motions is achieved by interpolating two motion vectors and reconstructing the result back to the original motion space. In addition, because of the nature of principal component analysis, the order of orthonormal basis is also the order of importance of the orthonormal axis.

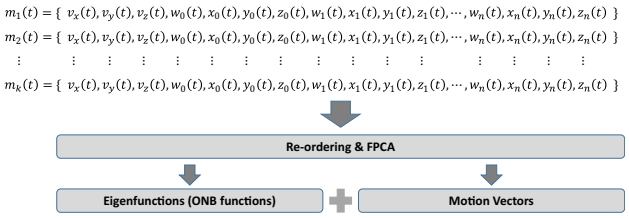



**Fig. 3** Per component ONB is extracted for different components of the input motion data

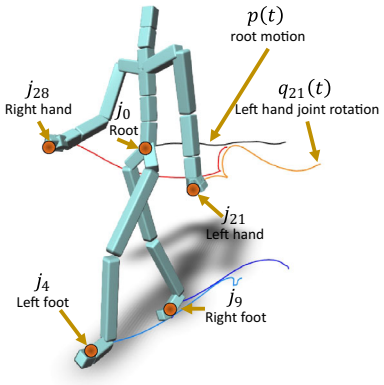



**Fig. 4** Skeleton and joint curves labeling

## 5 Character motion represented as orthonormal basis functions

We have shown how a function can be represented by its coordinates in a function ONB space Eq. (1). Moreover, we assumed there is a set of input functions $f_k(t)$. From this input we extracted the ONB $b_i(t)$ and each motion $f_k(t)$ is then represented (approximated) as $\tilde{f_k}(t)$ by its coordinates $c_i$. In this section, we show how a character motion can be represented by using ONB function representation.

### 5.1 Skeleton and motion representation

The input of our framework is an animated character, and we use notation from Lee and Shin [25]. The articulated character is a skeleton hierarchy structure (Fig. 4) represented as a directed graph $X = (J, E)$, where $J = \{j_0, j_1, \ldots, j_{|J|}\}$ are the joints (we use 31 joints in our experiments) and $E = \{e_0, e_1, \ldots, e_{|E|}\}$ is a set of joint-index pairs. The root node of the hierarchy is denoted by $j_0$ and corresponds to the pelvis of the character (Fig. 3).

The articulated character motion is represented as a set of translations of the root $j_0$ and the rotations of each joint over time span $t_0, t_1, \ldots, t_m$ where $m+1$ is the number of the input motion poses. Although the input is a set of discrete poses, we consider it a continuous function. The velocity of the root is denoted by $\mathbf{v}(t)$ and the rotation of each joint is $q_j(t)$. The rotations of the joints are quaternions. The character motion is a set

$$\mathbf{m}(t) = \{\mathbf{v}(t), \mathbf{q}_1(t), \ldots, \mathbf{q}_n(t)\}, \tag{11}$$

where $\mathbf{v}$ is the velocity vector of root position (without initial yaw rotation) and $\mathbf{q}_j(t)$ is the rotation of joint $j$ at the time $t$ in the local coordinate system of the skeleton. The world coordinates of the joint are calculated by recursively traversing the skeleton from the root $j_0$ and concatenating the corresponding rotations and translations (Fig. 4).

### 5.2 Dynamic time warping

Although two input motions are similar, they may have different speed (timescale). To solve the issue, we first calculate dynamic time warping (DTW) input motions before extracting motion vectors and ONB functions. We adopt the distance function between two keyframes by following [27]

$$d(m, m') = \sqrt{\begin{aligned} &\beta_{\text{root}} \|v_{\text{root}} - v'_{\text{root}}\|^2 \; + \\ &\beta_0 \|q_0(\hat{u}) - q'_0(\hat{u})\|^2 \; + \\ &\Sigma_{i=1}^{n} \beta_i \|p_i(\hat{u}) - p'_i(\hat{u})\|^2 \; + \\ &\Sigma_{i=1}^{n} \beta_i \|(q_i p_i)(\hat{u}) - (q'_i p'_i)(\hat{u})\|^2 \; , \end{aligned}} \tag{12}$$

where $p$ is a positional unit quaternion, $q$ is a unit quaternion of a joint's velocity, $v$ is velocity vector (see Eq. 13), $\beta_i$ is a weight of a joint, and $p(\hat{u})$ and $q(\hat{u})$ mean rotation of arbitrary vector, $\hat{u}$. We use the same weights for $\beta_i$ as in [27]. In particular, we set the weight of the hip $\beta_0 = 0.5$ and others $\beta_i, i = 1, \ldots, k$ are set to the length of the corresponding bone lengths. The hip joint is the root of skeleton hierarchy, and it is important for overall movement of the skeleton, so it has higher weight.

The velocity $v$ of a pose can be calculated as

$$\begin{aligned} v &= x' \ominus x = (v_{\text{root}}, q_0, q_1, \ldots, q_n) \\ &= (x'_{\text{root}} - x_{\text{root}}, p'_0 p_0^{-1}, p'_1 p_1^{-1}, \ldots, p'_n p_n^{-1}). \end{aligned} \tag{13}$$

Based on the above keyframe distance, a DTW texture is calculated for each pair of motions by accumulating minimum distance as shown in Fig. 6. The time warping (time pairs from one to another) follows the minimum distances in the given DTW texture. The DTW improve the quality of FPCA and reduce the number of basis functions at the same time (see Table 1).

### 5.3 Motion as ONB

Let us recall that the motion Eq. (11) has velocity of the root $\mathbf{v}(t)$ and motion of each joint $q_j(t)$. The *components* of $\mathbf{v}(t) = (\mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t))$ and $q_j(t) = \big(w_j(t), x_j(t), y_j(t), z_j(t)\big)$ are used in the function analysis (Sect. 4) as 1D functions. Let's denote $f(t)$ and $g(t)$ as 1D functions corresponding to any pair of the above-described components of motion. In the following text, we will not use the parameter $t$ whenever it is clear from the context.

**Table 1** Error and variance of FPCA result before and after dynamic time warping (DTW) (with precision of 0.9999)

| | Avg error | Var. | # of Basis Func |
|---|---|---|---|
| Before DTW | 0.003176 | 0.000079 | 10 |
| After DTW | 0.002338 | 0.000088 | 6 |

The DTW reduces the error and the number of basis functions

The ONB extraction needs a set of input functions. We can construct the ONB for all motions by taking all components and running the algorithm from Sect. 4. Let us recall that the ONB generation is executed until 0.999999 of variance is covered that also defines the number of basis functions. Without loss of generality we group motions as shown in Fig. 3. For example, all components of root velocity $\mathbf{v}$, and joints' quaternions $q_i$, are merged and then ONB functions are extracted.

To calculate the distance between two motion vectors, we account for the importance of joints in motion distance calculation by associated weights of each joint, and we use the approach of Tang et al. [41] who defined the joint weights. Let us have two motion vectors $v_1 = \{c_{11}, c_{12}, \ldots, c_{1m}\}$ and $v_2 = \{c_{21}, c_{22}, \ldots, c_{2m}\}$. We use a modified distance equation that accounts for the above-mentioned
weighting:

$$\widetilde{D}(\widetilde{f}_1(t), \widetilde{f}_2(t)) \approx \left( \sum_{i=1}^{n} w_i^2 (c_{1i} - c_{2i})^2 \right)^{1/2}, \tag{14}$$

where $w_i$ is the weights of joints. Figure 5 shows examples of closely matched two motion sequences.

We experimented with three different ways of combining functions for processing FPCA: (a) per component, (b) per joint and per component, and (c) all functions together. As intuitively expected, collecting functions per joint and per component provides best quality (i.e., lowest error and lowest variance). However, the overall number of basis functions was too high and it will lower the compression ratio. As a result, we combine all functions together and run FPCA to extract ONB functions. It provides comparable error, but much smaller number of basis functions as shown in Table 2.

We measured different sizes of ONBs in various configurations. Our experimentations show that there is no significant difference if the motions are clustered together in different ways, although a better insight could be obtained by a careful evaluation.

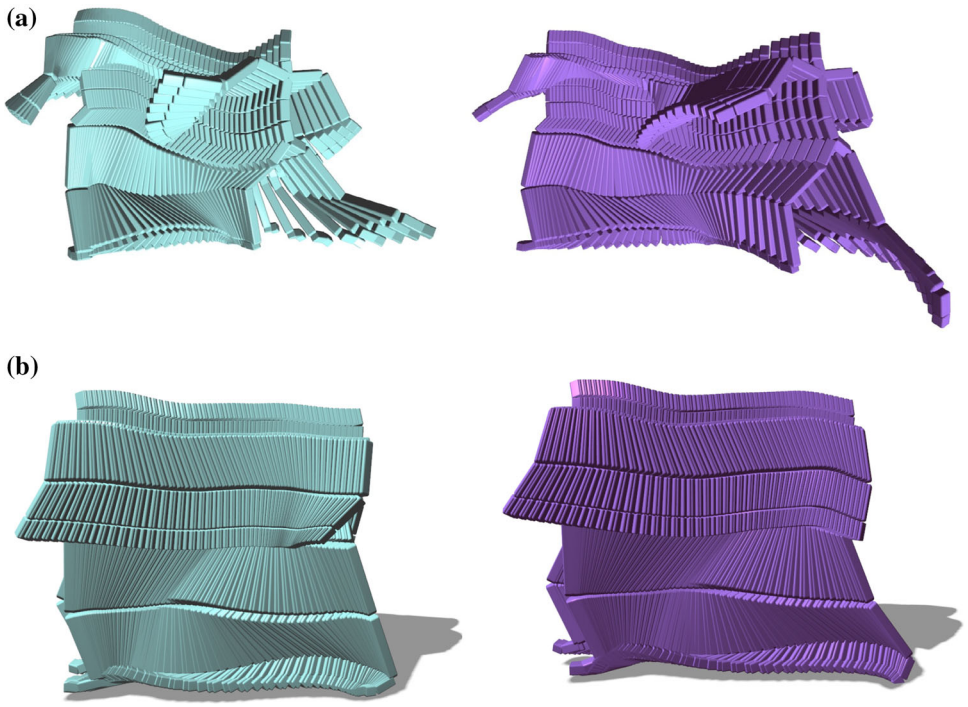



**Fig. 5** Closely matched motions that were calculated by our approximated motion distance (Eq. 14). **a** Two soccer kicking motions (0.022) and **b** different walking motions (0.007)

**Table 2** Average error, variance, and the required number of basis functions for different configuration

| | Avg error | Variance | # of basis |
|---|---|---|---|
| Per component | 0.002525 | 0.000093 | 114 |
| Per joint Component | 0.002140 | 0.000051 | 2032 |
| All together | 0.002255 | 0.000084 | 23 |

Comparison of three different ways of FPCA processing: (1) per component (velocities $x$, $y$, $z$ and quaternion $w$, $x$, $y$, $z$), (2) per joint and per component (velocity $x$, $y$, $z$, and per joint quaternion $w$, $x$, $y$, $z$), and (3) all together

## 6 Motion synthesis using ONB

One advantage of the ONB representation is the intrinsic compression. Another advantage is the ease of novel motion synthesis (Fig. 6).

### 6.1 Motion interpolation

Simple motion synthesis can be achieved by interpolating two or more motion vectors, and then reconstructing their spatial functions by using Eq. (1). This corresponds exactly to a time step interpolation of the original motions in the time domain, but it is achieved in a very compact way simply as an interpolation of the coordinates of motion vectors.

Let us recall that the distance of motion vectors is calculated by using Eq. (3). When $n$ motions are close enough, the interpolation and reconstruction create smooth motion transition between them. In order to provide smooth interpolation of input motions, we calculate $k$ nearest neighbors for each motion, and then provide the option to interpolate them. If the points are close enough so that they belong to $k$ nearest neighbors, we apply Bézier interpolation that is suitable for short motion clip. In order to create longer sequences of motion, we need to connect the motions by finding a partial connectivity of motion clips. An example in Fig. 7 shows the interpolation of one component of the motion vector.

### 6.2 Partial similarity by optimization

Some vectors can be too far to create a perceptually good result. We compensate for this problem by optimization between the time sliding. We optimize the objective function Eq. (15) that attempts to find scaling and sliding of time $u$ and $v$ from given interval $[t_c, t_d]$ of two motions

$$\begin{aligned} &\arg\min_{u,v} \sum_{j=1}^{N} w_j^2 \int_{t_c}^{t_d} \left(\widetilde{f}_1(us+v) - \widetilde{f}_2(s)\right)^2 \mathrm{d}s \\ &= \arg\min_{u,v} \sum_{j=1}^{N} w_j^2 \int_{t_c}^{t_d} \Big(M(s) + A(s)\Big)^2 \mathrm{d}s, \end{aligned} \tag{15}$$

where $N$ is the number of functions, $M(s) = \mu(us+v) - \mu(s)$, and $A(s) = \sum_{i=1}^{n} c_{1i} b_i(us+v) - c_{2i} b_i(s)$. The resulting parameters $u$ and $v$ are the scaling and sliding of time between two motions. This connectivity is similar to motion

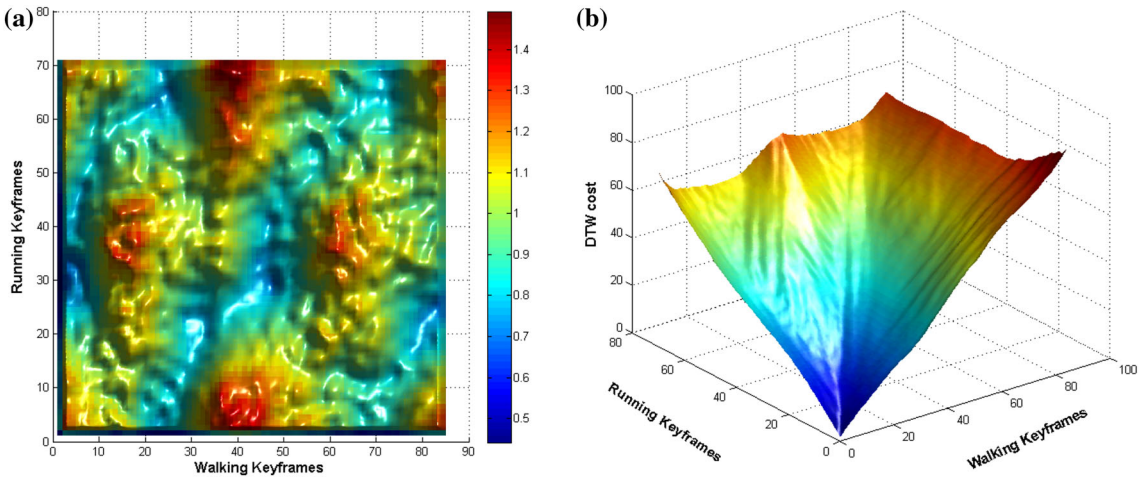



**Fig. 6** A keyframe distance table is calculated by using Eq. 12. Then, minimum cost connectivity (time pairs) is calculated by finding local minimum lines from the DTW texture

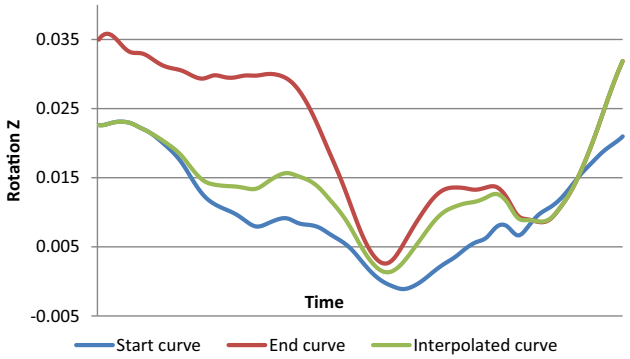



**Fig. 7** Interpolation of one component of the motion vector

graphs [21]. However, our approach is finding similar connectivity in inner-product space, not keyframe distance in spatial domain.

### 6.3 Foot-skating cleanup

The synthesized motions may contain foot-skating artifacts. We resolve this problem by detecting the footplants and then smoothly adjusting the nearby root and knee joint positions on their adjacent keyframes, similar to [16,22].

The footplants are automatically detected from the trained keyframes [16]. In the training process, the motion that contains the keyframe with the farthest distance to the labeled keyframes is selected for manual footplants marking. This iterative process terminates when the satisfied results are achieved. In the detecting process, we calculate the footplant values for each keyframe by averaging the values of its $k$ nearest neighbors in the trained database.

Once the footplants are detected, we set the positions of the consecutive footplants as their average values, and then smoothly relocate the root positions of every keyframe in the sequence, such that their legs are reachable to the positions. To avoid the pop-up problems that may raise on the boundary of footplant sequences, we linearly interpolate the root and foot positions of the keyframes laying between the footplants sequences. Additionally, the height of the root for each keyframe is adjusted smoothly to make sure its feet do not penetrate the ground. Finally, we apply the inverse kinematics on all the keyframes in the synthesized motion (Fig. 8).

## 7 Implementation and results

Our system is implemented in C++ and uses OpenGL and GLSL to visualize results. All results were generated on an Intel® Xeon® E5-1650 CPU, running at 3.20 GHz with 16 GB of memory, and rendered with an NVidia 970GTX GPU. All analysis and synthesis computations were performed on a single CPU thread. Initially, we used a FPCA library (PACE package) that is implemented in MATLAB. However, it requires five hours to analyze 41 motions. To improve the performance, we reimplement FPCA code in C++ and by using CUDA. Our new implementation provides significantly faster performance than MATLAB PACE package, the achieved speedup is 10× for 210 curves and 225× for 6510 curves. Once the ONB has been generated, the motion synthesis and decoding are interactive.

### 7.1 FPCA CUDA implementation

We use Eigen math library to represent matrices and vectors, ALGLIB for spline smoothing, Armadillo for fast singular value decomposition (SVD), and fast Moore–Penrose pseudoinverse is implemented by following Courrieu's method [10].

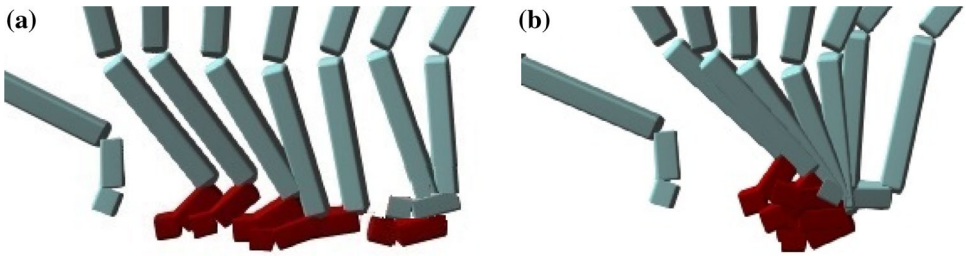



**Fig. 8** The partial keyframe sequences before (**a**) and after foot-skating cleanup (**b**). The adjacent keyframes containing no footplants are also smoothly adjusted to avoid pop-up problems

**Table 3** Comparison of FPCA implementations

| Implementation | Test case | |
|---|---|---|
| | 210 curves | 6510 curves |
| Average error (a) | | |
| PACE | 0.00326511 | 0.00145804 |
| Ours | 0.00326484 | 0.00146001 |
| Max error (b) | | |
| PACE | 0.06804195 | 0.20938692 |
| Ours | 0.06802791 | 0.20937758 |
| Processing time (s) (c) | | |
| PACE | 24.2851400 | 784.976200 |
| Ours | 2.27931000 | 3.47349000 |

While the MATLAB implementation is general, we did not require all the functionality in our code. We only consider special case which sampling points are regular. In addition, we speed up FPCA processing by applying CUDA for large-scale vector dot product in local weighted least square (LWLS) estimation, and removing cross-validation of residuals. Table 3 shows the comparisons of MATLAB PACE package and our implementation. The CUDA implementation will be available on our Web site.

## 7.2 Evaluation

We compare our method against two other approaches that use analytic basis functions: Fourier series and Legendre polynomials. The advantage of the two approaches is that they do not need to store their analytic basis functions because they are expressed as equations. However, they generally need more coefficients to represent the function with the similar error and the reconstruction artifacts are usually high-frequency oscillations that are unwanted in motion data. Our method is less sensitive to these errors.

### 7.2.1 Reconstruction comparison

We have used 41 motions and, in the first step, we have generated the ONB representation covering 0.999 of the variance and measured the error of the approximation. In the next step we encoded the same motion set by using Fourier series and Legendre polynomials while enforcing the same error as for the ONB. The results are displayed in Fig. 9 where the Fourier series is in green, Legendre polynomials blue, original motion curve black, and our method in red. The approximation by using analytic functions introduces unwanted oscillations as can be seen in the inset showing a detailed span of 30–120 frames in Fig. 9d and in the accompanying video. This is due to the fact that the high-order basis functions have high frequencies that would require more coefficients to capture. In contrast, our basis functions adapt to the data and the resulting reconstructed curve is smoother. At the same time, while Fourier representation needed 476 basis function and Legendre polynomials 293, our method needed only 96 basis functions to approximate the motion with the same error (see Table 4).

Another advantage of our method is the control over the error of the approximation. In our approach, we do not need to specify the absolute error values. We specify how much of the original information should be preserved in the reconstructed curves and run the corresponding ONB basis extraction. In all experiments, we set the error value to 0.1% (99.9% quality).

### 7.2.2 Dimension comparison

We compared the space needed for an accurate representation of all motion curves per each component. We encoded all component curves by using as few basis functions as possible while making sure that 90th percentile error is below a given error value. The comparison of the generated number of basis vectors for our method, Fourier, and Legendre is shown in Table 4. Overall our method outperforms the other methods (96:476 with Fourier and 96:293 to Legendre).

### 7.2.3 Compression

Our ONB function space representation of the character motion provides compression of the input data. Although the basis functions are generated for each set of motions and they need to be stored in order to reconstruct the motion, they outperformed Fourier and Legendre approximation in our experiments as shown in Table 4.

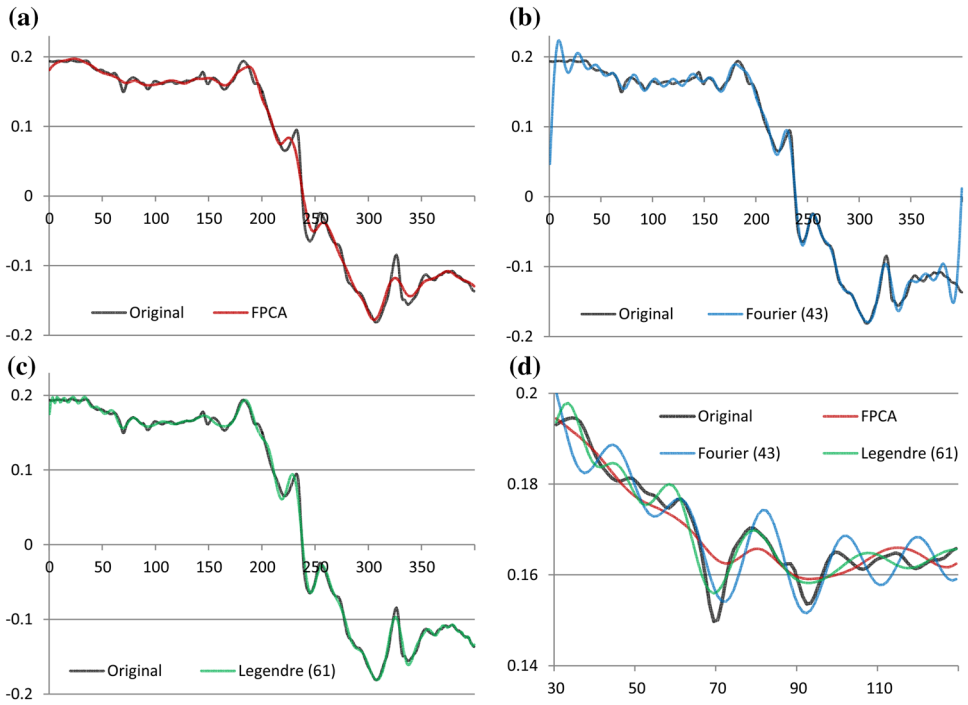



**Fig. 9** Comparison of the original motion to our method (**a**), Fourier (**b**), and Legendre (**c**). Detail of 30–110 s shows the analytic basis functions have higher oscillations (**b**) when encoded with the same error as our method

**Table 4** Comparison of the required number of basis functions

| 90th percentile | Number of basis functions | | |
|---|---|---|---|
| Comps. | Our method | Fourier | Legendre |
| Total | 96 | 476 | 293 |

Number of basis function for a given error

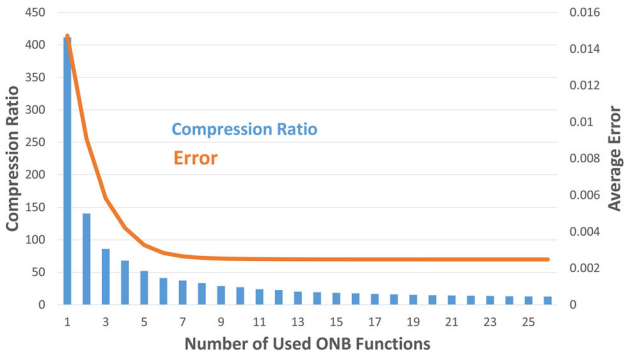



**Fig. 10** The compression factor depends on the number of basis functions. We removed motion vector elements with the absolute values smaller than 1e−7 and calculated the compression factors

We have encoded 321 different motions (2605 short sequences, each of 0.8–1.2 seconds length at 120 Hz) that represented a skeleton with 31 joints. The size of the raw input data is 227 MB. The compression ratios depends on the number of basis functions. In motion vectors, we do not store the vector elements with the absolute values less than 1e−7. For the first 5 basis functions (mean function and the basis functions), the compression ratios were 50× (see Fig. 10), for Fourier 9×, and for Legendre polynomials 10×. The effect of the size of the ONB will further diminish if more motions would be encoded and more motion vectors would be present.

Our method cannot be directly compared to other methods, such as [3,29,42], since our method is not used only for motion compressing, but it shares ONB functions for further processing. In addition, the compression ratio of our method can vary depending on the number of used basis functions as shown in Fig. 10.

Table 5 provides comparison of compression ratio and distortion rate (%) based on the provided result from Liu and McMillan [29] and Tournier et al. [42]. The same motion clips were used for the comparison. The distortion rate is calculated by Eq. (16) which was defined by Karni and Gotsman in [18].

$$d = 100\frac{\|A - \tilde{A}\|}{\|A - E(A)\|}, \tag{16}$$

where $A$ and $\tilde{A}$ are the $3m \times n$ matrices that consist of absolute markers' position of original motion and the decompressed motion respectively, $m$ is the number of markers, $n$ is the number of keyframes, and $E(A)$ is the mean of marker positions with respect to time. For reconstructing a frame, our method only requires a few calculations by following Eq. (1) so that it can reconstruct frames on the fly.

**Table 5** The comparison between our method and other approaches

| Method/motion | 09/06 | 13/29 | 15/04 | 17/08 | 17/10 | 85/12 | 86/02 | 86/08 |
|---|---|---|---|---|---|---|---|---|
| Compression ratio (a) | | | | | | | | |
| Liu and McMillan [29] | N/A | 1:55 | N/A | N/A | N/A | 1:18 | 1:53 | 1:56 |
| Tournier et al. [42] | 1:18 | N/A | 1:69 | 1:182 | 1:61 | 1:97 | N/A | N/A |
| Ours (motion vector only) | 1:19 | 1:33 | 1:63 | 1:34 | 1:33 | 1:34 | 1:34 | 1:29 |
| Distortion rate (%) (b) | | | | | | | | |
| Liu and McMillan [29] | N/A | 5.1 | N/A | N/A | N/A | 7.1 | 5.1 | 5.4 |
| Tournier et al. [42] | 0.36 | N/A | 1.55 | 0.049 | 0.49 | 0.56 | N/A | N/A |
| Ours | 1.11 | 0.30 | 0.21 | 0.39 | 0.23 | 0.34 | 0.38 | 0.40 |

Note that we only used vector size for calculating compression ratio, because our method the ONB functions are shared for all motions. In this table, lower than 1e−5 values are not saved, and 6 basis functions were used

## 8 Conclusion

We have introduced a novel compact representation of motion for character animation. Our method is inspired by analytic basis methods, such as Fourier and Legendre polynomials, but instead of using analytic representation the orthonormal basis (ONB) is extracted automatically by using functional principal analysis (FPCA) for each input set of motions. The ONB is unique for each input set and because of the FPCA the basis are ordered by their importance it provides optimal coverage of the input space. Our method not only provides better compression of the raw input data than the analytic basis approximations, and it also allows for an easy motion synthesis. Each motion from the input set is represented as a motion vector and motion is performed by simply interpolating motion vector coordinates and connecting partially similar motions. We also provide optimization in the ONB for more complex motions.

There are several limitations and avenues for future work. One limitation is that the FPCA processes only 1D functions. Theoretically, it would be possible to apply the FPCA directly to $n$-dimensional character animation and the per component optimizations would not be necessary. Moreover, FPCA assumes that the input functions are smooth, and also internally smooth the resulting ONB functions. As a side effect, this could hide oscillations. Another limitation is that the FPCA is always lossy due to numerical errors in the computation.

This paper is an invited extended version of a conference paper Yoo et al. [50]. Since this paper has been submitted, a number of new papers addressing related issues have been published. In particular, there is a body of new work dealing with deep learning that has been, for example, used to learn and extend human motion synthesis Lee et al. [26], deep learning has been combined with adversarial networks to generate and control human motion by Wang et al. [46], and related work has addressed generation of motion signatures for character motion (e.g., Aristidou et al. [4], Wang et al. [45]). An important body of work also deals with learning for efficient motion control, see for example Peng et al. [35,36]. This opens a potential for future work, in which the FPCA could be learned directly from motion data or used to generate motions by adversarial networks.

**Acknowledgements** This research was funded in part by National Science Foundation Grants #10001387, Functional Proceduralization of 3D Geometric Models and #10001364, Multimodal Affective Pedagogical Agents for Different Types of Learners.

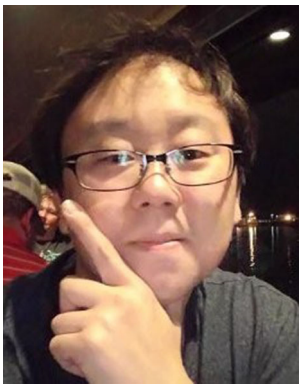
## References

1. Akhter, I., Simon, T., Khan, S., Matthews, I., Sheikh, Y.: Bilinear spatiotemporal basis models. ACM Trans. Gr. **31**(2), 1–12 (2012)
2. Alon, J., Sclaroff, S., Kollios, G., Pavlovic, V.: Discovering clusters in motion time-series data. In: Proceedings of 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, pp. I-375–I-381 (2003)
3. Arikan, O.: Compression of motion capture databases. ACM Trans. Gr. **25**(3), 890–897 (2006)
4. Aristidou, A., Cohen-Or, D., Hodgins, J.K., Chrysanthou, Y., Shamir, A.: Deep motifs and motion signatures. ACM Trans. Gr. **37**(6), 1–13 (2018)
5. Barbič, J., Safonova, A., Pan, J.-Y., Faloutsos, C., Hodgins, J. K., Pollard, N. S.: Segmenting motion capture data into distinct behaviors. In: Proceedings of Graphics Interface 2004, GI '04, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Canadian Human-Computer Communications Society, pp. 185–194 (2004)
6. Beaudoin, P., Coros, S., van de Panne, M., Poulin, P.: Motion-motif graphs. In: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '08, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 117–126 (2008)
7. Bernard, J., Wilhelm, N., Krüger, B., May, T., Schreck, T., Kohlhammer, J.: Motionexplorer: exploratory search in human motion capture data based on hierarchical aggregation. IEEE Trans. Vis. Comput. Gr. **19**(12), 2257–2266 (2013)
8. Chao, M.-W., Lin, C.-H., Assa, J., Lee, T.-Y.: Human motion retrieval from hand-drawn sketch. IEEE Trans. Vis. Comput. Gr. **18**(5), 729–740 (2012)
9. Coffey, N., Harrison, A., Donoghue, O., Hayes, K.: Common functional principal components analysis: a new approach to analyzing human movement data. Hum. Mov. Sci. **30**(6), 1144–1166 (2011)

10. Courrieu, P.: Fast computation of moore-penrose inverse matrices. In: CoRR. arXiv:0804.4809 (2008)
11. Du, H., Hosseini, S., Manns, M., Herrmann, E., Fischer, K.: Scaled functional principal component analysis for human motion synthesis. In: Proceedings of the 9th International Conference on Motion in Games, MIG '16, New York, NY, USA, ACM, pp. 139–144 (2016)
12. Forbes, K., Fiume, E.: An efficient search algorithm for motion data using weighted PCA. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '05, New York, NY, USA, ACM, pp. 67–76 (2005)
13. Gall, J., Rosenhahn, B., Brox, T., Seidel, H.-P.: Optimization and filtering for human motion capture. Int. J. Comput. Vis. **87**(1–2), 75–92 (2010)
14. Heck, R., Gleicher, M.: Parametric motion graphs. In: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, I3D '07, New York, NY, USA, ACM, pp. 129–136 (2007)
15. Hou, J., Chau, L., Magnenat-Thalmann, N., He, Y.: Human motion capture data tailored transform coding. *CoRR*, arXiv:1410.4730 (2014)
16. Ikemoto, L., Arikan, O., Forsyth, D.: Knowing when to put your foot down. In: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, I3D '06, New York, NY, USA, ACM, pp. 49–53 (2006)
17. Ikemoto, L., Arikan, O., Forsyth, D.: Generalizing motion edits with gaussian processes. ACM Trans. Gr. **28**(1), 1:1–1:12 (2009)
18. Karni, Z., Gotsman, C.: Compression of soft-body animation sequences. Comput. Gr. **28**(1), 25–34 (2004)
19. Keogh, E., Palpanas, T., Zordan, V. B., Gunopulos, D., Cardle, M.: Indexing large human-motion databases. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB '04, VLDB Endowment, pp. 780–791 (2004)
20. Kovar, L., Gleicher, M.: Automated extraction and parameterization of motions in large data sets. ACM Trans. Gr. **23**(3), 559–568 (2004)
21. Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. ACM Trans. Gr. **21**(3), 473–482 (2002a)
22. Kovar, L., Schreiner, J., Gleicher, M.: Footskate cleanup for motion capture editing. In: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '02, New York, NY, USA, ACM, pp. 97–104 (2002b)
23. Lai, Y.-C., Chenney, S., Fan, S.: Group motion graphs. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '05, New York, NY, USA, ACM, pp. 281–290 (2005)
24. Lau, M., Bar-Joseph, Z., Kuffner, J.: Modeling spatial and temporal variation in motion data. ACM Trans. Gr. **28**(5), 171:1–171:10 (2009)
25. Lee, J., Shin, S. Y.: A hierarchical approach to interactive motion editing for human-like figures. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co, pp. 39–48 (1999)
26. Lee, K., Lee, S., Lee, J.: Interactive character animation by learning multi-objective control. ACM Trans. Gr. **37**(6), 1–10 (2018)
27. Lee, Y., Wampler, K., Bernstein, G., Popović, J., Popović, Z.: Motion fields for interactive character locomotion. ACM Trans. Gr. **29**(6), 138:1–138:8 (2010)
28. Levine, S., Wang, J.M., Haraux, A., Popović, Z., Koltun, V.: Continuous character control with low-dimensional embeddings. ACM Trans. Gr. **31**(4), 28:1–28:10 (2012)
29. Liu, G., McMillan, L.: Segment-based human motion compression. In: Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '06, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 127–135 (2006)
30. Min, J., Chai, J.: Motion graphs++: a compact generative model for semantic motion analysis and synthesis. ACM Trans. Gr. **31**(6), 153:1–153:12 (2012)
31. Mordatch, I., de Lasa, M., Hertzmann, A.: Robust physics-based locomotion using low-dimensional planning. ACM Trans. Gr. **29**(4), 71:1–71:8 (2010)
32. Müller, M., Baak, A., Seidel, H.-P.: Efficient and robust annotation of motion capture data. In Proceedings of the: ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '09, New York, NY, Association for Computing Machinery, USA, pp. 17–26 (2009)
33. Müller, M., Röder, T.: Motion templates for automatic classification and retrieval of motion capture data. In: Proceedings of the: ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '06, Goslar. DEU, Eurographics Association, pp. 137–146 (2006)
34. Ormoneit, D., Black, M.J., Hastie, T., Kjellstrom, H.: Representing cyclic human motion using functional analysis. Image Vis. Comput. **23**(14), 1264–1276 (2005)
35. Peng, X.B., Abbeel, P., Levine, S., van de Panne, M.: Deepmimic: example-guided deep reinforcement learning of physics-based character skills. ACM Trans. Gr. **37**(4), 1–14 (2018)
36. Peng, X.B., Berseth, G., Yin, K., Van De Panne, M.: Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Trans. Gr. **36**(4), 1–13 (2017)
37. Ramsay, J., Silverman, B.W.: Functional Data Analysis. Wiley Online Library, New York (2006)
38. Reitsma, P.S.A., Pollard, N.S.: Evaluating motion graphs for character animation. ACM Trans. Gr. (2007). https://doi.org/10.1145/1289603.1289609
39. Safonova, A., Hodgins, J.K.: Construction and optimal search of interpolated motion graphs (2007). https://doi.org/10.1145/1275808.1276510
40. Shin, H.J., Lee, J.: Motion synthesis and editing in low-dimensional spaces: research articles. Comput. Anim. Virtual Worlds **17**(3–4), 219–227 (2006)
41. Tang, J.K.T., Leung, H., Komura, T., Shum, H.P.H.: Emulating human perception of motion similarity. Comput. Anim. Virtual Worlds **19**(3–4), 211–221 (2008)
42. Tournier, M., Wu, X., Courty, N., Arnaud, E., Revéret, L.: Motion compression using principal geodesics analysis. Comput. Gr. Forum **28**(2), 355–364 (2009)
43. Unuma, M., Anjyo, K., Takeuchi, R.: Fourier principles for emotion-based human figure animation. In: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95, ACM, New York, NY, pp. 91–96 (1995)
44. Vögele, A., Krüger, B., Klein, R.: Efficient unsupervised temporal segmentation of human motion. In: 2014 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2014)
45. Wang, H., Ho, E. S., Shum, H. P., Zhu, Z.: Spatio-temporal manifold learning for human motions via long-horizon modeling. In: IEEE Transactions on Visualization and Computer Graphics (2019a)
46. Wang, Z., Chai, J., Xia, S.: Combining recurrent neural networks and adversarial training for human motion synthesis and control. In: IEEE Transactions on Visualization and Computer Graphics (2019b)
47. Wei, X., Min, J., Chai, J.: Physically valid statistical models for human motion generation. ACM Trans. Graph. **30**(3), 19:1–19:10 (2011)
48. Yao, F., Mueller, H.-G., Wang, J.-L.: Functional linear regression analysis for longitudinal data. Ann. Stat. **33**(6), 2873–2903 (2005)
49. Yoo, I., Abdul Massih, M., Ziamtsov, I., Hassan, R., Benes, B.: Motion retiming by using bilateral time control surfaces. Comput. Graph. **47**(C), 59–67 (2015)

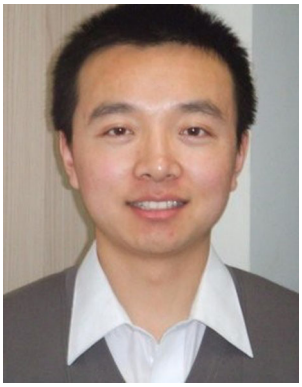
50. Yoo, I., Fišer, M., Hu, K., Benes, B.: Character motion in function space. In: Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, vol. 1, INSTICC, SciTePress, pp. 110–121 (2019)
51. Yoo, I., Vanek, J., Nizovtseva, M., Adamo-Villani, N., Benes, B.: Sketching human character animations by composing sequences from large motion database. Vis. Comput. **30**(2), 213–227 (2014)
52. Zhao, L., Safonova, A.: Achieving good connectivity in motion graphs. Gr. Models, **71**(4), 139– 152 (Special Issue of ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2008) (2009)
53. Zhou, F., De La Torre, F.: Generalized time warping for multi-modal alignment of human motion. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1282–1289 (2012)
54. Zhou, F., De la Torre, F., Hodgins, J.: Hierarchical aligned cluster analysis for temporal clustering of human motion. IEEE Trans. Pattern Anal. Mach. Intell. **35**(3), 582–596 (2013)





**Innfarn Yoo** is a research scientist at Google Inc. He received his Master's degree from Purdue, and got bachelor degree at Konkuk University in South Korea, majoring Mathematics. He also has 5 years of working experiences in game industry. He finished his Ph.D. in computer graphics at Purdue University in the HPCG Lab.

**Marek Fišer** is a Software Engineer in Google Brain Robotics team working on systems that can learn navigation policies with reinforcement learning. This includes creation and integration of simulated environments, designing and training agents using RL, and deploying learnt policies on real robots. Marek has Master's in Computer Graphics from Purdue University.



**Kaimo Hu** received the bachelor's and Ph.D. degrees from Tsinghua University, in 2006 and 2012, respectively. He was a post doctoral research assistant in the Department of Computer Graphics Technology, Pu rdue University. His research interests include computer graphics, shape analysis, geometric processing, and procedural modeling.

**Bedrich Benes** is George McNelly professor of Technology and professor of Computer Science at Purdue University. His area of research is in procedural and inverse procedural modeling and simulation of natural phenomena and he has published over 150 research papers in the field.