Compression by indexing: an improvement over MPEG-4 body animation parameter compression

Siddhartha Chattopadhyay Suchendra M. Bhandarkar Kang Li

Department of Computer Science, The University of Georgia
Athens, Georgia 30602-7404, USA

ABSTRACT

Body Animation Parameters (BAPs) are used to animate MPEG-4 compliant virtual human-like characters. In order to stream BAPs in real time interactive environments, the BAPs are compressed for low bitrate representation using a standard MPEG-4 compression pipeline. However, the standard MPEG-4 compression is inefficient for streaming to power-constrained devices, since the streamed data requires extra power in terms of CPU cycles for decompression. In this paper, we have proposed and implemented an indexing technique for a BAP data stream, resulting in a compressed representation of the motion data. The resulting compressed representation of the BAPs is superior to the MPEG-4-based BAP compression in terms of both, required network throughput and power consumption at the client end to receive the compressed data stream and extract the original BAP data from the compressed representation. Although the resulting motion after decompression at the client end is lossy, the motion distortion is minimized by intelligent use of the hierarchical structure of the skeletal avatar model. Consequently, the proposed indexing method is ideal for streaming of motion data to power- and network- constrained devices such as PDAs, Pocket PCs and Laptop PCs operating in battery mode and other cellular devices in a mobile network environment.

Keywords
Human Motion, Virtual Human Animation, MPEG-4 BAP compression

1. INTRODUCTION

Animation of human-like virtual characters has potential applications in the design of human computer interfaces and modeling of virtual environments using power-constrained devices [1] [19]. Distributed virtual human (avatar) animation is used in many applications that depict human models interacting with networked virtual environments [2]. Distributed virtual environments (DVEs) either require exchange of motion files between hosts to simulate the avatar motion, or use locally stored motion data [3] [4] [5] [6]. Efficient use of locally stored motion data, or data obtained via streaming from a server, is essential for power-constrained clients, such as pocket PCs, PDAs and laptop PCs operating in battery mode in a mobile networked environment. Efficient compression of the motion data, that yields a significant compression ratio, and also requires minimal CPU cycles on the client side for reception and decompression, is needed. For this purpose, MPEG-4 has proposed H-Anim standards to represent virtual human-like characters [7] [8] [16]. A virtual human body model is animated using a stream of body animation parameters (BAPs) encoded for low-bitrate transmission in dedicated interactive communications and broadcast environments [9] [17] [18]. The BAPs control the various independent degrees of freedom in the skeletal avatar model to produce an animation of the body parts. The MPEG-4 standard has defined a standard compression pipeline for efficient compression of the BAPs [10] [11].

A major disadvantage of the existing MPEG-4 compression standard is that decompression of the data at the client end, for real-time applications, requires extra CPU cycles, consequently consuming extra power in the client device. Hence, it is desirable to have a compression method which reduces the network throughput requirement significantly, and requires minimum computation at the client side to reconstruct the motion data from the compressed data. This is important in order to conserve both, the network bandwidth and power consumption in the client device.

In this paper, we propose and implement an indexing technique to compress the BAP-based motion data. The indexing of the BAP data results in compression of the data, since the indices can be represented with fewer bits compared to the original representation of the BAPs. The original data is retrieved from the indices by using a lookup table. Since such indexing schemes invariably lead to motion distortion, the lookup tables are created by intelligent use of the hierarchical structure of the skeletal model of the virtual character being animated. A lookup table thus

constructed leads to considerable improvement in the reconstructed motion quality. It also leads to considerable reduction in the power consumed by the wireless network interface card (WNIC) to receive the compressed data stream and in the CPU cycles required to recover motion data from the compressed representation, compared to MPEG-4 based compression. Our results show that the proposed compression technique for motion data is superior to MPEG-4 compression of motion data, both in terms of compression ratio and the amount of power required to receive and un-compress the data. These qualities render the proposed compression scheme ideal for low bandwidth networks and low power devices which require streaming motion data from a remote server to render character animation.

There exist quantization methods for efficient use and distribution of avatar motion data over the network. Endo et al. [12] propose quantization of the motion type, rather than the motion data itself. Hijiri et al. [13] describe a new data packet format which allows flexible scalability of the transmission rate, and a data compression method, termed as SHCM, which maximizes the features of this format by exploiting the 3D scene structure. Our method uses quantization to achieve data compression in a manner somewhat similar to the above paper, but by using the hierarchical structure of the skeletal model intelligently. Giacomo et al. [14] present methods for adapting a virtual human's representation and the resulting animation stream, and practical details for the integration of these methods into MPEG-2 and MPEG-4 architectures. Aubel et al. [15] present a technique of using imposters to improve the display rate of animated characters by acting solely on the geometric and rendering information.

The techniques mentioned above do not describe any direct impact on power consumption on the client device where the animation is being rendered. Also, there is not sufficient quantitative analysis of the quality of the rendered motion after decompression of the compressed motion data. The proposed method in this paper not only allows for low-bandwidth transfer of motion data using motion data quantization, but is also suitable for data reception and data reconstruction on power-constrained devices. In addition, the proposed method incorporates quality control parameters, which lead to fine control of network throughput requirement and quality of the reconstructed motion. Quantitative analysis of the quality of reconstructed motion supports our claim that the proposed method is useful for quality controlled compression of BAP-based motion data.

In the sections to follow, we first describe the body animation parameter (BAP) representation of the motion data, which is part of the MPEG-4 H-Anim standard. In Section **3**, we discuss in detail the indexing technique for BAP data. In Section **4**, we compare the BAP indexing technique with existing MPEG-4 BAP compression. In Section **5**, we present a detailed analysis of the network throughput requirement and quality of the reconstructed motion obtained using indexing. In Section **6**, we present experimental results obtained for various types of motion examples. Finally, we present the conclusions and directions for future work in Section **7**.

2. MATRIX REPRESENTATION OF BAP DATA

The BAPs are represented by an n x m dimensional matrix $\mathbf{X}$, where n is a multiple of fps (frames per second) and m is the number of degrees of freedom for the avatar (the maximum value of $m = 296$ as defined in MPEG-4 standard). Each row of the matrix represents a pose of the avatar for a small time step. Each column corresponds to either the displacement of the model from a fixed origin, or the *Euler angle* of rotation needed to achieve the desired pose. Successive rows of $\mathbf{X}$ depict incremental changes in the pose of the avatar in small time steps, thus animating the avatar. Thus, each row of $\mathbf{X}$ corresponds to a frame of animation.

We have used a 62-dimensional avatar, with a frame rate of 33 frames per second (fps). This means that, for a 10 second motion sequence, the motion matrix $\mathbf{X}$ is a 330 x 62 array of floating point numbers. The first 3 columns of $\mathbf{X}$ represent the absolute displacement of the avatar from a fixed origin in the 3-D virtual world. The next three columns represent the absolute orientation of the avatar with respect to the virtual world coordinates. The remaining columns are the various joint rotation *Euler* angles needed to render a particular pose of the avatar.

As a first step in the compression process, the matrix $\mathbf{X}$ is represented as a difference matrix, $\mathbf{d}_{n\text{-}1 \times m}$, and the initial pose vector $\mathbf{I}$, where $\mathbf{I}$ is assigned the first row of $\mathbf{X}$, and the rows of $\mathbf{d}$ are the differences between successive rows of $\mathbf{X}$.

$$\mathbf{I}_j = \mathbf{X}_{1,j} \qquad j = 1, 2, \ldots, m$$

$$\mathbf{d}_{i,j} = \mathbf{X}_{i+1,j} - \mathbf{X}_{i,j} \qquad i = 1 \ldots n\text{-}1;\ j = 1 \ldots m$$

The difference matrix **d,** subsequently termed the motion matrix**,** can be interpreted as successive small angular changes needed by the avatar for each of its degrees of freedom in order to realize the desired animation. Without loss of generality, we assume that $\mathbf{d}$ has n rows.

3. INDEXING OF BAP DATA

Before explaining the indexing technique in detail, we first outline the basic intuition behind it. The motion difference matrix **d** represents small successive changes in the joint angles over small intervals of time. For approximately periodic and regular motions such as walking, jogging and running, a collection of all the $n \bullet m$ floating point numbers within the corresponding motion matrix **d** exhibit a tendency to form a finite number of clusters. Taking a cue from this observation, we can assign the $n \bullet m$ floating point numbers in **d** to a finite number of buckets. Each bucket, in turn, is associated with a representative number which best describes the collection of the numbers within the bucket. The basic concept underlying the proposed indexing technique is to be able to index some (perhaps all) of the numbers within the original motion matrix **d**, and generate a corresponding lookup table for the indices. This compression method results in significant data reduction, as each index value can be represented using fewer bytes than the corresponding floating point number. The indexing technique proposed in this paper holds good for the animation of any hierarchical character, including cartoons and other human-like characters.

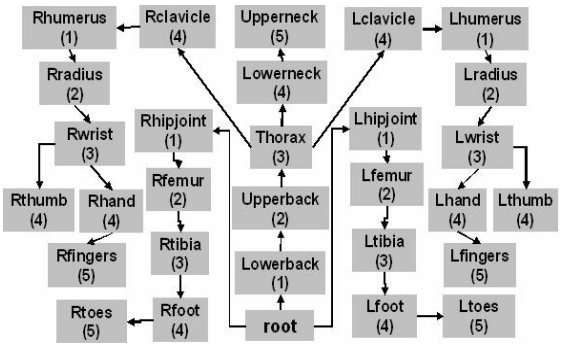

Figure 1: An example of the hierarchical structure of a human skeletal model consisting of 31 nodes, with a total of 62 degrees of freedom of motion (rotational and translational). For convenience, the root node is drawn at the bottom.

In the next three subsections, we describe in detail the process of indexing the motion data in the matrix **d**, and the creation of the lookup table. For the purpose of explanation, we assume that each index is represented by a single byte, or 8-bits; i.e., the values of the indices lie between 0 and 255. The lookup table uses 2^8 buckets. Later, we analyze indices of size other than 8 bits.

3.1 Indexing of motion data d

In order to have efficient indexing, we have used an equal frequency distribution technique to assign the $n \bullet m$ numbers in **d** to buckets numbered from 0 to 255. This is done as follows:

Step 1: All the data in matrix **d** is collected in a single 1-D array **A** of size $n \bullet m$. The array **A** is sorted in ascending order. All the numbers in **A** are multiplied by the *resolution quantization term (RQT)*, M. The RQT depends on the number of significant digits used to represent the floating point number. For example, if the required accuracy of the floating point numbers is a maximum of 4 digits, $RQT = 10000$. The numbers are rounded to represent integers in the range $[A_{min} \bullet M, A_{max} \bullet M]$.

Step 2: The integers in the range $[A_{min} \bullet M, A_{max} \bullet M]$ are divided into buckets numbered from 0 to 255. It is desirable to allocate each of the 256 buckets an equal share of $n \bullet m$ numbers in **A**. This implies that each bucket should have $freq = (A_{max} \bullet M - A_{min} \bullet M)/256$ numbers allocated to it. This is done by computing the histogram of the integers in **A**, and dividing the histogram into 256 vertical strips such that each strip has the same area, $freq$. After all the numbers in **A** have been allocated a bucket numbered from 0 to 255, the numbers in **A** are divided by the RQT to recover the original numbers.

At the end of this step, we get a set of 256 buckets denoted by **bucket(j)** for $j = 0$ to 255, such that each floating point entry in the motion data matrix **d** is contained in exactly one of the 256 buckets. An index matrix $\mathbf{d}_{\mathbf{index}}$ is used to store the bucket number for the corresponding entry in the matrix **d**.

3.2 Creating lookup table for the index matrix $\mathbf{d}_{\mathbf{index}}$

The lookup table is used to map each of the indices to a corresponding representative number such that an approximation to the original motion matrix **d** can be recovered. The creation of an appropriate lookup table for recovery of the original motion data matrix **d**, from the index matrix $\mathbf{d}_{\mathbf{index}}$, is critical, since recovery of the original data after discretization invariably results in motion distortion.

A simple method to recover the number associated with a bucket is to compute the simple average of all the floating point numbers assigned to the bucket. However, this invariably leads to poor approximation of the original

motion matrix **d**. We have found that intelligent exploitation of the hierarchical structure of the skeletal avatar model can lead to better construction of the lookup table $\mathbf{T}_{\mathbf{lookup}}$, which in turn results in reduced error in the reconstructed motion using the lookup table. The detailed steps for creating the lookup table are as follows.

Step 1: The avatar is represented by a hierarchical skeletal model. For each *m*-dimensional pose vector, each dimension, or column in the motion matrix **d**, is assigned a level l_i (**Figure 1**). The level l_i signifies the importance of the degree of freedom associated with a particular joint, in the overall displacement of the model joints. A joint *i*, at level $l_i = 1$, when given a small angular displacement, affects the model more in terms of the overall displacement, than a joint *j* at level $l_j = 2, 3, 4, 5$ *or 6*.

Step 2: After assigning level values to the various joints of the avatar model, these joint level values are used to compute a weighted sum of the numbers belonging to a bucket. The j^{th} lookup value in lookup table $\mathbf{T}_{\mathbf{lookup}}$ is given by:

$$\mathrm{T}^{\eta}{}_{\mathrm{lookup}}[\mathrm{j}] = \frac{\sum_{i \in bucket(j)} \frac{1}{(l_i)^{\eta}} A[i]}{\sum_{i \in bucket(j)} \frac{1}{(l_i)^{\eta}}}, \quad j = 0, 1 \ldots 255 \qquad \textbf{(1)}$$

where η is a constant. Empirical observations have revealed that as η increases, the $\mathbf{T}_{\mathbf{lookup}}$ values result in a better approximation to the data, resulting in reduced displacement error. This is due to the fact that the numbers associated with *level = 1* affect the displacements in the body the most. Hence, emphasizing the numbers within a bucket with *level = 1* leads to better approximation of the motion data. As $\eta \rightarrow \infty$, all the weighting terms in equation **(1)** tend to zero, except the terms with *level =1*. Hence, while calculating the weighted sum of the numbers in a bucket, we consider only those numbers with *level = 1 (**selective averaging**)*, and compute a simple mean of these numbers. If none of the entries in a bucket have *level =1*, we use the next smallest level to compute the weighted sum.

3.3 Motion matrix decomposition for motion sequences of long durations

In the previous two subsections, we have discussed in detail the indexing of matrix **d**, and the creation of the corresponding lookup table $\mathbf{T}_{\mathbf{lookup}}$. The proposed motion indexing technique yields high quality reconstructed motion for most types of avatar motions, for small time durations (depending on the type of motion, this can range from a few seconds to a minute). However, for a motion data sequence spanning several minutes, it is difficult to quantize the motion matrix in a satisfactory manner. This is due to the assignment of a large number of the original floating point numbers to each bucket for motions of long time duration. As a result, the single number representing the collection of floating point numbers assigned to a bucket is less likely to be a good representation of the floating point numbers assigned to the bucket. Consequently, the selective averaging method leads to poor approximation of the original motion matrix for long animation sequences.

To overcome this difficulty, we decompose larger motion matrices into smaller motion matrices. The difference matrix **d** is represented as succession of smaller matrices $\mathbf{d}_1$, $\mathbf{d}_2$,…, $\mathbf{d}_L$ consisting of an equal number of frames, denoted by n_d; i.e. each fragmented motion matrix has n_d rows (**Figure 2**). Each of the smaller matrices of size n_d x *m* in turn is discretized using the above method resulting in separate lookup tables, $\mathbf{T}_{\mathbf{lookup}}{}^{\mathbf{1}}$, …, $\mathbf{T}_{\mathbf{lookup}}{}^{\mathbf{L}}$. This simple technique yields a good approximation to the motion data of any arbitrary duration. The computation of n_d is done such that it minimizes both the file size and the resulting reconstruction error. We describe the techniques used to compute n_d in Section **5**.

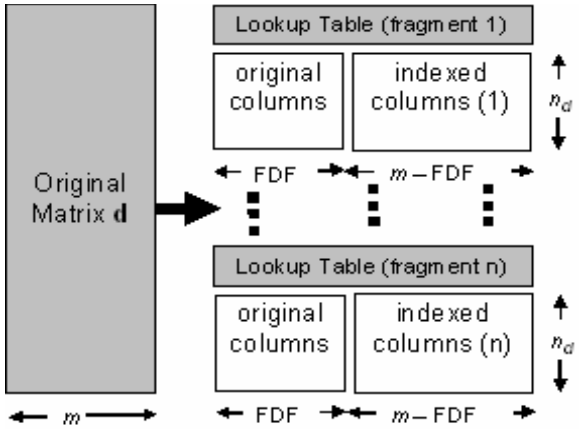

Figure 2: The indexing steps. The motion matrix **d** is decomposed into equal fragments of length n_d. The first **FDF** columns are untouched; the next *m* – **FDF** columns are indexed. Each fragment in indexed separately, and each fragment has its own lookup table.

3.4. Combining the original and indexed data

It is often desirable to preserve the original data within some columns of the original motion matrix **d**, and

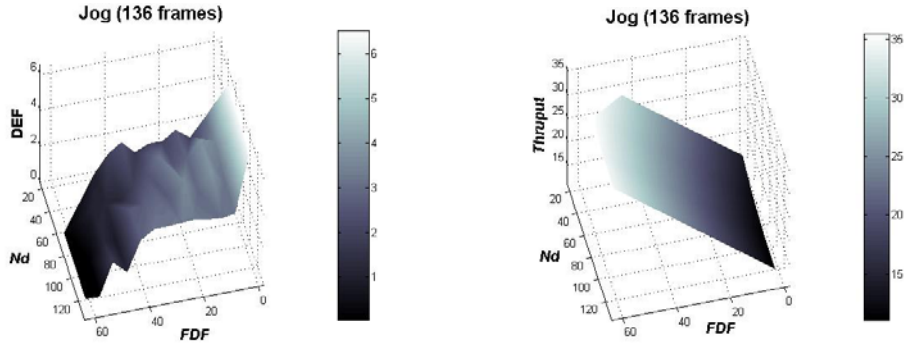

Figure 3: (*Left*) The surface plot for the displacement error per frame (**DEF**) for $30 \leq n_d \leq 120$ rows and $1 \leq$ **FDF** ≤ 62 columns. (*Right*) The surface plot for the obtained throughput for $30 \leq n_d \leq 120$ and $1 \leq$ **FDF** ≤ 62. Both the plots have index-bit $b = 8$.

index the remainder of the columns. This is especially true for the first 6 columns, which represent the absolute displacement and orientation of the avatar with respect to a fixed origin. Indexing these columns may lead to undesirable motion distortion, and lead to contradictory physical appearances such as feet not touching the ground while walking, etc. A combination of the original and indexed columns can be achieved by not indexing the first **FDF** (Fixed Degrees of Freedom) columns, and indexing the remainder of the m – **FDF** columns. Intuitively it is obvious that **FDF** = 0 implies indexing of the entire motion matrix, whereas **FDF** = m implies the original matrix itself. A simple schematic diagram depicting the indexing of the original motion matrix **d** is given in **Figure 2**.

4. INDEXED BAP COMPRESSION VS MPEG-4 BAP COMPRESSION

In the previous section, we introduced three quality control parameters (QCPs): bit per index b, segment-size n_d and fixed degrees of freedom **FDF**. By varying the three QCPs, a tradeoff can be obtained between the number of bytes for representing the motion data (subsequently called the file size or network throughput requirement, **T**) and the quality of reconstructed motion. It is possible to get different values of the file size and reconstructed motion quality by varying the three QCPs.

For example, consider a jogging motion consisting of 136 frames. By assigning 8-bits or one byte to each index, the file size (**T**) and the displacement error per frame, **DEF** (refer to Section **5.2**) are plotted as functions of **FDF** and b, and the corresponding surface plots are shown in **Figure 3**. The smallest file size obtained is less than one-third the size of the original file, though resulting in the maximum value of **DEF**. For the same jogging motion, MPEG-4 based compression, comprising of predictive encoding and arithmetic coding, compressed the file to about 87% of the original size. In order to have a fair comparison with the MPEG-4 standard, we exhaustively searched the space of all the possible values for parameters n_d and **FDF** (with $b = 8$) such that the file size is 87% of the original, and the error, **DEF**, is a minimum. We found the minimum value of **DEF** to be 1.1893, which yields a slightly lossy reconstructed motion which is hardly perceptible to the human eye. In fact, a higher **DEF** value can be allowed, thereby reducing the throughput significantly to a minimum of around 33% of the original, as compared to only 87% of the original obtained using MPEG-4 based compression.

In addition to the significant reduction in network throughput requirement (filesize), the proposed indexing technique does not require any additional CPU cycles for decoding the compressed data. MPEG-4 compressed files, on the other hand, require a computationally expensive decoding procedure to recover the original motion data. The proposed indexing method, though superior both in terms of the network throughput requirement and significant power savings at the client end for data reception and decoding, is a lossy form of compression. However, videos of the reconstructed motion reveal that the degradation is visually imperceptible, when the selective averaging technique is used to create the lookup tables for the indices. For example, the jogging video sequence, when encoded using a standard MPEG-4-based compression technique (predictive encoding followed by arithmetic coding), yields a 29 KB of compressed representation from the original 33 KB data. Decoding the 29 KB of compressed data takes 0.109375 seconds on a 2.2 GHz Pentium processor, which implies that the number of CPU cycles used was around 0.2 Gcycles.

In contrast, when the proposed indexing technique is used, no CPU cycles are spent on decoding; instead, a simple table lookup suffices to reconstruct the original data.

In addition to the power consumed on account of the CPU cycles used in data decoding, power is also consumed by the WNIC during reception of the streamed data. The power consumption during data reception depends entirely on the number of bytes received and the size of the packets. Since the proposed indexing technique results in much less compressed data compared to MPEG-4-based compression, the power consumed by the WNIC during data reception is much lower in the case of the proposed technique compared to MPEG-4-based compression. A detailed description of the power usage by the network card and the number of CPU cycles used for decompression of MPEG-4 compressed BAP data is provided in Section **6**.

In general, every motion type can be compressed with greater efficiency (in terms of both compression ratio and reduction of power consumption during data reception and decoding) by the proposed indexing technique when compared to MPEG-4 based compression. However, one has to keep in mind that the success of the proposed technique depends entirely on the proper selection of the three quality control parameters: bits per index (b), fixed degree of freedom (**FDF**) and segmentation size (n_d). The determination of these parameters, and their impact on the quality of reconstructed motion and resulting network throughput requirement, is presented in details in the next section. In Section **6**, we will provide results for many different types of motion to show that the proposed technique can be used for a wide range of motion types.

5. ANALYSIS OF THROUGHPUT AND MOTION QUALITY

In the subsections to follow, we will first discuss the file size (or equivalently, the network throughput requirement) of the indexed BAP stream as a function of the three aforementioned quality control parameters. Next, we present a quantitative discussion of the effect of the three quality control parameters on the quality of motion. We present a detailed analysis of the performance of the proposed indexing technique using the jogging video sequence example mentioned previously. However, note that the analysis is applicable to other motion examples as well.

5.1 Throughput requirement as function of *FDF, b and* n_d

In practice, the BAPs in motion matrix **d** are represented by floating point numbers that are 4 bytes each. We present our network throughput analysis based on this assumption, which can easily be extended to double precision floating point numbers (8 bytes). We assume that the integer and floating point numbers are each 4 bytes long. In the case of a motion data file that has been decomposed into smaller segments, each segment contains a header and a data section. The header contains all the relevant parameters and the lookup table $\mathbf{T}_{\mathbf{lookup}}$. The data section contains the actual data from the matrices **d** and $\mathbf{d}_{\mathbf{index}}$. The motion matrix **d** is preceded by a header containing the initial pose vector, **I**, which takes $4 \bullet m$ bytes.

The header file for each segment (**Figure 2**) takes $4 \bullet 2^b$ bytes for the lookup table, and a few extra c_1 bytes to store the numbers n_d and b. Hence, the total number of bytes taken by the header for a motion segment is

$$\text{segment\_header} = 4 \bullet 2^b + c_1 \qquad \textbf{(2)}$$

The actual motion data is represented by a hybrid matrix consisting of floating point numbers and indices that can be represented by a maximum of 4 bytes. The first **FDF** columns ($0 \le \mathbf{FDF} \le m$) are floating point numbers taken directly from the motion matrix **d**, and the next $m - \mathbf{FDF}$ columns are indices for the lookup table, such that each index takes a maximum of b bits ($\lceil b/8 \rceil$ bytes). Hence, the total number of bytes needed for the motion data segment is given by:

$$\text{segment\_motion} = n_d \bullet (4 \bullet \mathbf{FDF} + \lceil b/8 \rceil \bullet (m - \mathbf{FDF})) \qquad \textbf{(3)}$$

For motion data spanning n frames, the number of blocks, or motion segments, each consisting of n_d frames, is $\lceil n/n_d \rceil$. Note that the last block may contain fewer than n_d frames. Hence, the total number of bytes, $\mathbf{T}(\mathbf{FDF}, b, n_d)$ of the semi-indexed matrix is given by:

$$\mathbf{T}(\mathbf{FDF}, b, n_d) = \lceil n/n_d \rceil \bullet (\text{segment\_ header} + \text{segment\_motion}) + 4m \qquad \textbf{(4)}$$

It is obvious from equations (2), (3) and (4) that the file size increases exponentially with the number of bits b used to represent each index, but grows linearly with the value of **FDF** and n_d.

The minimum file size $\mathbf{T}_{\mathbf{min}}$ is obtained for $\mathbf{FDF} = 0$, $n_d = n$ and $b = 8$ (assuming byte boundary for the data). The maximum file size, $\mathbf{T}_{\mathbf{max}}$, should not exceed the original file size; i.e. $4mn$ (since the original matrix has n-1 rows and m columns, and one row for the initial pose, and each entry is a 4-byte floating point number). For the parameter values mentioned for $\mathbf{T}_{\mathbf{min}}$, the minimum file size is (including table lookup and initial pose vector)

$$\mathbf{T}_{\mathbf{min}} = (n\text{-}1)m + 4m + c_2 \qquad \textbf{(5)}$$

where c_2 is the number of bytes needed to store 256 buckets, each containing a 4-byte floating point number; i.e. c_2 = 1024. Hence, the bounds for the file size **T** for any motion sequence with n frames and m degrees of freedom are:

$$(n\text{-}1)m + 4m + 1024 \le \mathbf{T}(\mathbf{FDF}, b, n_d) \le 4mn \qquad \mathbf{(6)}$$

Consider the jogging motion consisting of 136 frames (n = 136, 135 rows in **d** and one row for the initial pose). The frames are obtained at a rate of 33 frames per second (fps). From equation **(6)**, the obtained bounds for **T**(**FDF**, b, n_d) are [9642, 33728]. From the bounds, it is clear that it is possible to obtain a compressed, indexed representation of the jogging motion that is less than one third the size of original motion data. Indeed, it seems that choosing the parameters to yield the minimum file size should be the optimal thing to do. However, the drawback is that the resulting reconstructed motion for the minimum file size is often of poor quality.

It is often desirable to have the compressed data be a certain fraction of the original size of the motion data. For a desired fraction f of the original motion data, the corresponding quality control parameters can be obtained by exhaustively searching the space of all possible parameter values such that the following constraint holds:

$$(n\text{-}1)m + 4m + 1024 \le \mathbf{T}(\mathbf{FDF}, b, n_d) \le f \bullet 4mn \qquad \mathbf{(7)}$$

The parameter values that satisfy equation (**7**) can be termed as 'valid' points in the parameter space. Empirical studies reveal that for b > 8, the size of the file explodes exponentially, thus violating the upper bound even for f = 1 (100%). Hence, we keep b = 8, which means that a motion segment will have a 256-bucket lookup table, and that each index is represented by a single byte.

The MPEG-4 compressed file size for the jogging motion data is 29 KB, whereas the filesize is 33 KB for the original file. Thus, MPEG-4 compression results in a file size which is 87% of the original. Thus, we keep f = 87% for the jogging example in equation **(7)**, and exhaustively find all the values for parameters **FDF** and n_d (recall that we have already fixed b = 8) which satisfy equation **(7)**. This subset of the original set of possible parameter values is termed the set of 'valid' points in the parameter space. The next step is to find the optimal values of **FDF** and n_d from the set of 'valid' points in the parameter space which minimize both, the compressed filesize and reconstruction error. This is discussed in detail in the next subsection.

5.2. Quality of reconstructed motion as function of *FDF*, n_d and b

To quantify the reconstructed motion quality, we use the metric **DEF** (Displacement Error per Frame). For each of the p joints in the human model, the displacement error for that is defined as the frame-by-frame sum of all the difference between the actual joint position in the original motion data and the joint position in the reconstructed motion data. Let $\mathbf{C}_{nx3p}$ be the original coordinates of the joints of the human model, and $\mathbf{C'}_{nx3p}$ be the coordinates of the joints of the human model after reconstruction of motion from the indexed matrix. We define the error function, Δ:I → R such that

$$\Delta(\mathrm{i}) = \sum_{j=1}^{n} \left\| P_{ji} - P'_{ji} \right\|, \; i = 1, 2, ..., m \qquad \mathbf{(8)}$$

where $P_{ji} = (C_{j,3i-2}, C_{j,3i-1}, C_{j,3i})$ and $P'_{ji} = (C'_{j,3i-2}, C'_{j,3i-1}, C'_{j,3i})$, $i = 1,2, .., m$, and $\|.\|$ is the Euclidean norm defined as $\|(x, y, z)\| = \sqrt{(x^2 + y^2 + z^2)}$. This error metric represents the resulting error for each joint position. The **DEF** is defined as the sum of the errors for all the joints, normalized by dividing by the total number of rows (i.e. frames).

$$DEF(FDF, b, n_d) = \frac{\sum_{i=1}^{m} \Delta(i)}{n} \qquad \mathbf{(9)}$$

Although the **DEF** value does not represent the motion quality in an absolute sense, comparing the **DEF** values for various quality control parameters indicates whether the quality is better or worse in a relative sense. The surface plots of **T**(**FDF**, b, n_d) and **DEF**(**FDF**, b, n_d), for the jogging motion example, for the various possible values of **FDF** and n_d, while keeping b = 8, is presented in **Figure 3**; the lower the **DEF**, the better the motion quality. Hence, the set of parameters amongst the valid parameters which minimize both **DEF**(**FDF**, b, n_d) and **T**(**FDF**, b, n_d) are the optimal parameters. Since we have already fixed b = 8, the optimal parameter values for **FDF** and n_d are those which minimize the following figure of merit function:

$$\mathbf{M(FDF,}\ n_d\mathbf{) = T(FDF}, 8, n_d\mathbf{)} \bullet \mathbf{DEF(FDF}, 8, n_d\mathbf{)} \qquad \mathbf{(10)}$$

A simple exhaustive search reveals that **FDF** = 41 and n_d = 60 (with b = 8) minimizes the figure of merit function **M**, yielding a final file size of 27.74 KB and **DEF** = 1.1893. While MPEG-4 compression of BAP data is lossless, the indexing method yields a **DEF** value of 1.1893 per frame; the corresponding value of f is 87%. Depending on the

application, f can be made as low as possible, in order to achieve great reduction in the network throughput requirement, with a tradeoff against the motion quality.

The functions **T** and **DEF** are inversely proportional to each other. A relationship plot can be used to select the desired throughput requirement **T** based on an acceptable value of **DEF**. A graphical plot depicting the relationship between **T** and **DEF** for the jogging example is given in **Figure 4**. For a maximum acceptable value of **DEF**, the corresponding throughput can be obtained from the plot.

6. EXPERIMENTAL RESULTS FOR DIFFERENT TYPES OF MOTIONS

In order to test the usefulness of the proposed indexing technique, we have experimented with various types of motions. The motion types we have selected range from periodic motions (jogging, walking, jumping, long jump etc) to extremely ill-correlated and complex motions like dancing. We have considered $b = 8$ for all our analysis. These motion examples have been created using motion capture data from real human actors. The motion data files are obtained from mocap.cs.cmu.edu. The plots of the displacement error per frame (**DEF**) for two representative motion examples are presented in **Figure 5**. The throughput values display the characteristics similar to that of the throughput value for the jogging example in **Figure 3**, and hence have not been shown. The **DEF** surface plots in **Figure 5** show that the **DEF** values are more affected by the value of **FDF**, compared to n_d. Though the optimal values of the quality control parameters can be obtained computationally by exhaustive search, the surface plots give an insight into the behavior of the **DEF** as a function of the parameters. The complexity of the **DEF** surfaces in **Figure 5** supports our exhaustive search method, as the surface does not seem to exhibit any known distribution or function.

6.1. Comparison with MPEG-4

In order to provide a fair comparison with MPEG-4, we first encode the motion data using MPEG-4 arithmetic coding based compression. We compute the ratio of the obtained file size to the original motion file size, and use this ratio as f in equation **(8)** to compute an upper bound on the network throughput requirement (file size) for the motion example. Next, we exhaustively enumerate all the possible values of **FDF** and n_d which satisfy the above constraint. We select the parameter values **FDF** and n_d which yield the minimum reconstruction error (**DEF**). The final parameters for the indexed motion file are chosen to be these parameter values. The results of the experiment are presented in **Table 1**. As evident from the table, the proposed indexing method, when tuned to yield the same throughput requirement as MPEG-4 compression,

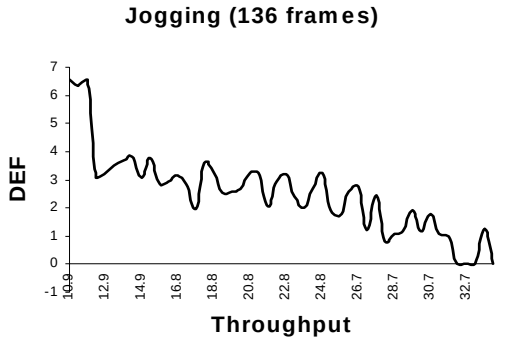

Figure 4: Plot of Displacement Error per Frame (**DEF**) vs. throughput **T** obtained by varying f

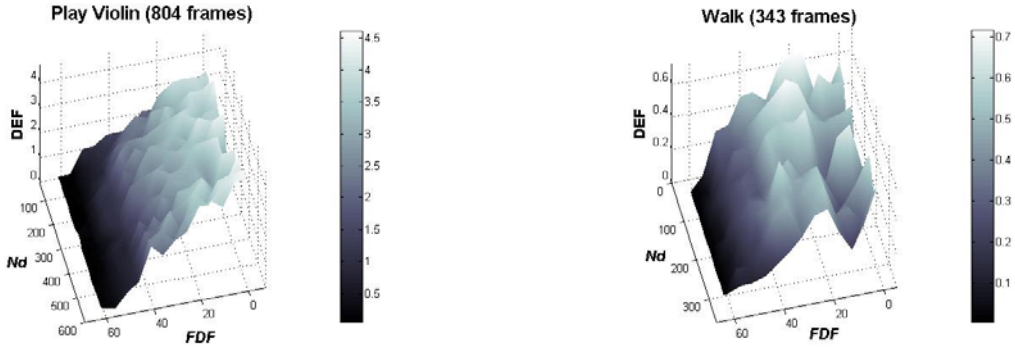

Figure 5: DEF plots of "Play Violin" and "Walk" motion capture data, as a function of n_d rows and **FDF** columns. The most general trend is that **DEF** increases with decrease in **FDF**

motion type	no of frames	org. size (Kbytes)	MPEG-4 Comp-ressed (Kbytes)	G-cycles decoding MPEG-4	ratio f	Indexed Comp-ressed (Kbytes)	DEF of indexed Comp-ressed	FDF	b	nd
jog	136	33	28	0.65	85%	27.74	1.19	41	8	60
run-leap	251	61	50	0.93	82%	47.5	0.29	41	8	180
stride	298	72	59	1.23	82%	47.29	1.08	31	8	240
walk	343	83	73	1.34	88%	69.13	0.15	46	8	330
jump	351	85	72	1.35	85%	62.47	0.14	36	8	90
basketball	396	96	84	1.44	88%	79.48	0.23	46	8	360
forward-jump	415	101	86	1.56	86%	85.22	0.33	46	8	120
dance	434	106	92	1.68	87%	86.9	11.16	46	8	420
Play violin	804	195	162	2.89	83%	159.17	0.78	46	8	420
sword play	2251	545	465	8.49	85%	450.93	1.45	46	8	210

Table 1: Comparison of MPEG-4 predictive encoding and arithmetic coding based compression vs compression obtained as indexed motion data. The ratio *f* is computed such that the compression ratio obtained by indexing is similar to MPEG-4 based compression. Note the large number of CPU cycles used by MPEG-4 for decompression; BAP indexing does not require any extra CPU cycles for decompression.

results in a slightly lossy reconstructed motion, but needs no CPU cycles for decoding the compressed data. MPEG-4, on the other hand requires a significant number of CPU cycles to decode the motion from the compressed motion data. To compute the CPU cycles needed for decoding, we have used a 2.2 GHz Celeron CPU with 128 KB L2 cache and 512 MB RAM.

6.2. Minimum throughput obtainable by indexing

For applications that can accept a slight visual distortion of the reconstructed motion, indexing can be used judiciously to reduce the throughput requirement significantly, compared to MPEG-4-based compression. **Table 2** gives the minimum throughput obtainable using indexing, and the corresponding displacement error **DEF**. The throughput requirement (compressed file size) is significantly less than that obtained by MPEG-4-based compression. At the same time, no extra CPU cycles are needed for decompression of the data. The resulting distortion in the reconstructed motion is limited, due to the selective averaging technique that has been used to create the lookup tables intelligently.

6.3. Power usage due to data reception at the network card

When data is streamed to a mobile client, the wireless network interface card (WNIC) receiving the data consumes some power. The indexed motion data does not need any CPU cycles to de-compress the data, and also benefits from lower network resource usage. For a motion of time duration time T, data size S and given available bandwidth B, the energy used by the WNIC is given by

$$E_{network} = E_R \bullet S/B + E_S \bullet (T - S/B) \qquad \textbf{(12)}$$

E_R is the energy used by the network card during data reception and E_S is the energy used by the WNIC when it is sleeping and not receiving data. Using equation **(12)**, we computed network card energy utilization for MPEG-4 and the indexed file representation (**Table 3**). We have used energy usage data from [20] [21] to obtain energy usage for data reception in m-joules. Again, the indexed motion data showed significantly less energy consumption by the WNIC when compared to MPEG-4.

From the obtained results reported in the

motion type	no of frames	org. size (Kbytes)	MPEG-4 Comp-ressed	Min Indexed Comp-ressed	indexed/MPEG-4	DEF
jog	136	33	28	10.9	34%	6.35
walk	343	83	73	24	29%	0.40
dance	434	106	92	29.82	29%	24.37
basketball	396	96	84	27.41	29%	0.87
sword play	2251	545	465	147	27%	11.71
run-leap	251	61	52	18.2	35%	0.58
play_violin	804	195	166	53.31	32%	3.52
forward-jump	415	101	86	28.61	33%	1.13
jump	351	85	73	24.55	34%	1.06
stride	298	72	62	21.19	34%	1.49

Table 2: Maximum Compression ratio achieved by indexing, compared to MPEG-4 compression.

motion type [1]	Motion Duration (sec.) [2]	MPEG-4 comp. [3]	Min Indexed Comp. [4]	mJoules MPEG-4 128 Kbps [5]	mJoules indexed 128 Kbps [6]	indexed/ MPEG-4 [7]	mJoules MPEG-4 1 Mbps [8]	mJoules indexed 1 Mbps [9]	indexed/ MPEG-4 [10]	mJoules MPEG-4 4 Mbps [11]	mJoules indexed 4 Mbps [12]	indexed/ MPEG-4 [13]
jog	4.1	28	10.9	2913	1580	*54.2%*	1002	836	*83.4%*	798	756	*94.8%*
walk	10.4	73	24	7534	3712	*49.3%*	2551	2074	*81.3%*	2018	1898	*94.1%*
dance	13.2	92	29.82	9504	4654	*49.0%*	3225	2619	*81.2%*	2552	2401	*94.1%*
basketball	12.0	84	27.41	8676	4262	*49.1%*	2943	2391	*81.3%*	2329	2191	*94.1%*
sword play	68.2	465	147	48344	23540	*48.7%*	16607	13507	*81.3%*	13207	12432	*94.1%*
run-leap	7.6	50	18.2	5246	2766	*52.7%*	1834	1524	*83.1%*	1468	1391	*94.7%*
play_violin	24.4	162	53.31	16948	8471	*50.0%*	5892	4832	*82.0%*	4707	4442	*94.4%*
forward-jump	12.6	86	28.61	8934	4457	*49.9%*	3064	2505	*81.7%*	2436	2296	*94.3%*
jump	10.6	72	24.55	7499	3798	*50.6%*	2585	2122	*82.1%*	2058	1942	*94.4%*
stride	9.0	59	21.19	6200	3251	*52.4%*	2174	1805	*83.0%*	1742	1650	*94.7%*

Table 3: Comparison of power consumption in m-joules by the network card interface. $\boldsymbol{E_S}$ = 177 mJ/s and $\boldsymbol{E_R}$ = 1425 mJ/s. The table shows power consumption for 3 bandwidths; 128 Kbps, 1 Mbps and 4 Mbps.

last three subsections, it is obvious that indexing the motion data using the techniques described in this paper leads to significant reduction in both network throughput requirement and power consumption for the resource constrained client. The reconstructed motion quality is significantly improved by using the selective averaging technique discussed in this paper. Hence, for resource constrained clients that can tolerate slight distortion in motion data, the proposed indexing method offers enormous benefits.

7. CONCLUSIONS AND FUTURE WORK

We have proposed and implemented a novel indexing technique to compress BAP data used for MPEG-4 compliant character animation. The proposed indexing method has advantages both in terms of improved compression ratio, and reconstruction of the motion data via a lookup table included in the compressed file, thus eliminating the need for explicit decompression. This leads to both reduction in the throughput requirement for networked applications requiring motion data exchange, and reduced power consumption for power constrained clients. The resulting quality of the reconstructed motion is improved considerably by intelligent exploitation of the hierarchical structure of the skeletal avatar model in the process of creation of optimal lookup tables for reconstruction of the quantized motion. For motions of long durations, we have proposed a simple method for decomposing the motion data sequence into smaller motion data sequences, thus preserving the quality of motion for arbitrarily large motion files. The quality and throughput requirements of the motion data are controlled via three quality control parameters. We have proposed a simple systematic elimination procedure to obtain the best possible combination of the parameters depending on the required compression ratio.

The proposed technique results in a compressed motion data representation which is superior in terms of compression ratio and power consumption needed for motion reconstruction, compared to the existing MPEG-4 compression technique that uses predictive encoding and arithmetic coding. Although MPEG-4 motion data compression is lossless, the lossy compression resulting from indexing exhibits negligible motion distortion, and is explicitly controllable via the three aforementioned parameters.

A limitation of the proposed technique is that the optimal values of the parameters (**FDF**, b and n_d) are obtained via exhaustive search, which is naive. It may be possible to obtain the optimal values for these parameters more efficiently. Another drawback with any animation research is that there is no perfect quantitative measure for the quality of the reconstructed motion. Finally, the intelligent use of the hierarchical structure of the model yields good results for full body motions of the avatar; for small delicate motions such as movement of the fingers, or for facial animation, the proposed technique offers considerable scope for future improvement.

REFERENCES

1. Gutiérrez, M., Vexo, F. and Thalmann, D., *Controlling Virtual Humans Using PDAs*, Proc. of the 9th International Conference on Multimedia Modelling (MMM'03), Taiwan, 2003.
2. Capin, T.K., Pandzic, I.S. and Magnenat-Thalmann, N., *Avatars in Networked Virtual Environments*, John Wiley and Sons, 1999.
3. Joslin, C., Molet, T. and Thalmann, N. M., *Advanced real-time collaboration over the internet*, Proceedings of the ACM symposium on Virtual reality software and technology, Seoul, Korea, pp: 25 - 32, 2000.
4. Cavazza, M., Martin, O., Charles, F., Mead, S.J. and Marichal, X., *Interacting with Virtual Agents in Mixed Reality Interactive Storytelling*, Proceedings of Intelligent Virtual Agents, Kloster Irsee, Germany, 2003.
5. Vacchetti, L., Lepetit, V., Papagiannakis, G., Ponder, M. and Fu, P., *Stable real-time interaction between virtual humans and real scenes*, 3DIM 2003, Banff, AL, Canada, 2003.
6. Barakonyi, I., Psik, T. and Schmalstieg, D., *Agents That Talk And Hit Back: Animated Agents in Augmented Reality*, Proceedings of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR), Washington D.C Nov 2-5, pp. 141-150, 2004.
7. ISO/IEC 14496-1:1999, *Coding of Audio-Visual Objects, Systems*, Amendment 1, December 1999.
8. ISO/IEC 14496-2:1999, *Coding of Audio-Visual Objects, Visual*, Amendment 1, December 1999.
9. Capin, T., K., Petajan, E. and Ostermann, J., *Very Low Bitrate coding of virtual human animation in MPEG-4*, IEEE International Conference on multimedia and expo (ICME), New York, NY, volume: 2, 2000.
10. Capin T. K., Petajan, E. and Ostermann, J., *Efficient Modeling of Virtual Humans in MPEG-4*, Proc. ICME'2000, New York, NY, July 2000.
11. Capin, T. K. and Thalmann, D., *Controlling and Efficient Coding of MPEG-4 Compliant Avatars,* Proceedings of IWSNHC3DI'99, Santorini, Greece, 1999.
12. Endo, M., Yasuda, T. and Yokoi, S., *A Distributed Multi-user Virtual Space System,* IEEE Computer Graphics and Applications**,** (Vol. 23, No. 1), pp 50 – 57, 2003.
13. Hijiri, T., Nishitani, K., Cornish, T., Naka, T. and Asahara, S., *A spatial hierarchical compression method for 3D streaming animation*, Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML), Monterey, California, USA, pp. 95 – 101, 2000.
14. Giacomo, T., Joslin, C., Garchery, S. and Magnenat-Thalmann, N., *Adaptation of Facial and Body Animation for MPEG-based Architectures*, Proceedings of International Conference on Cyberworlds, pp. 221, 2003.
15. Aubel, A., Boulic, R. and Thalmann, D., *Animated Impostors for Real-Time Display of Numerous Virtual Humans*, Proceedings of 1st Int'l Conf. Virtual Worlds, (VW-98), vol. 1434, Springer, Berlin, pp. 14-28, 1998.
16. Preda, M., Salomie, A., Preteux, F. and Lafruit, G., *Virtual character definition and animation within the MPEG-4 standard*, in M. Strintzis, N. Sarris (Ed.), 3D modeling and animation: Synthesis and analysis techniques for the human body (Chapter 2), IRM Press, Hershey, PA, pp. 27-69, 2004.
17. Preda, M. and Preteux, F., *MPEG-4 Human Virtual Body Animation*, in M. Bourges-Sévenier (Ed.), MPEG-4 Jump-Start (Chapter 9), Prentice Hall, Upper Saddle River, NJ., 2002.
18. Preda, M., Preteux, F., *Advanced virtual humanoid animation framework based on the MPEG-4 SNHC standard*, Proceedings of EUROIMAGE International Conference on Augmented, Virtual Enviroments and Three-Dimensional Imaging (ICAV3D'01), Mykonos, Greece, pp. 311-314, 2001.
19. Kruppa, M. and Krüger, A., *Concepts for a combined use of Personal Digital Assistants and large remote displays*, Proceedings of Simulation and Visualization, Magdeburg, Germany, 2003, pp. 349-361, 2003.
20. Stemm, M., Gauthier, P, Harada, D. and Katz. R. H., *Reducing power consumption of network interfaces in hand-held devices*, Proceedings of 3rd Intl. Workshop on Mobile Multimedia Comm., September 1996.
21. Havinga, P. J. M., *Mobile Multimedia Systems*, PhD thesis, Univ. of Twente, Feb 2000.