Model Dependent Power Aware Compression Algorithms for MPEG-4 Virtual Human Animation in Mobile Computers

Siddhartha Chattopadhyay, Suchendra M. Bhandarkar, Kang Li
Department of Computer Science,
The University of Georgia, Athens, GA – USA

siddh@cs.uga.edu suchi@cs.uga.edu kangli@cs.uga.edu

ABSTRACT

MPEG-4 Body Animation Parameters (BAPs) are used for animation of MPEG-4 compliant virtual human-like characters. Distributed virtual reality applications and networked games on mobile computers require access to locally stored or streamed compressed BAP data. Existing MPEG-4 BAP compression techniques are inefficient for streaming, or storing, BAP data on mobile computers, because (a) MPEG-4 compressed BAP data entails a significant number of CPU cycles, hence significant, unacceptable power consumption, for the purpose of decompression, (b) the lossy MPEG-4 technique of frame dropping to reduce network throughput during streaming leads to unacceptable animation degradation; and (c) lossy MPEG-4 compression does not exploit structural information of the avatar model. In this article, we propose two novel algorithms for lossy compression of BAP data, termed as BAP-indexing and BAP-sparsing. We demonstrate how an efficient combination of the two algorithms results in a lower network bandwidth requirement and reduced power for data decompression at the client end when compared to MPEG-4 compression. The algorithm exploits the structural information in the avatar model, thus maintaining visually acceptable quality of the resulting animation upon decompression. Consequently, the hybrid algorithm for BAP data compression is ideal for streaming of motion animation data to power- and network- constrained mobile computers.

Keywords: *(I.3.7) Three Dimensional Graphics and Realism – Animation (E.4.a) Data Compaction and Compression*

INTRODUCTION

Animation of human-like virtual characters has potential applications in the design of human computer interfaces, computer games 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 entails minimal CPU cycles on the client side for data reception and data decompression, is needed. 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] as depicted in **Figure 1**.

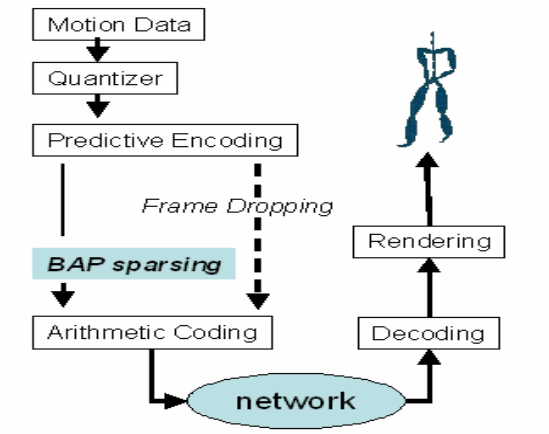

Figure 1: The standard compression and decompression pipeline for MPEG-4 BAPs, enhanced with a novel alternative stage termed as *BAP sparsing*.

A major disadvantage of the existing MPEG-4 compression standard is that decompression of the data at the client end entails extra CPU cycles, consequently consuming extra power in the client device. Lossy adaptation techniques used by the MPEG-4 standard for real time compression of video data in low-bandwidth situations, such as frame dropping, lack intelligent exploitation of the structural information available in the avatar model, and hence result in unacceptable degradation in the resulting animation quality. It is desirable to have a compression method which reduces the network throughput requirement significantly, and also requires minimum computation, hence power, at the client side to reconstruct the motion data from the compressed data stream.

In this paper, we propose and combine two novel compression algorithms for MPEG-4 BAP data which (a) intelligently exploit the structural hierarchy of the virtual human avatar to achieve efficient compression which, though lossy, results in reconstructed motion of good quality, (b) use indexing techniques for compression of BAP data, resulting in significant reduction of power consumption required for decompression, and (c) provide quality control parameters for tight control of reconstructed motion quality and compression ratio. One of our proposed compression algorithms, termed BAP-Sparsing [23], creates a sparse representation of the original BAP data. This results in improved compression of the BAP data after using the MPEG-4 compression pipeline. The second proposed algorithm, BAP-Indexing [22], creates byte-size indices for representation of the BAP data. This results in the compression of the BAP data, since the indices can be represented using fewer bits, compared to the original floating point representation of the BAPs. The resulting compression ratio is significantly superior to that obtained using the MPEG-4 compression standard [22]. We propose and implement a combination of the two algorithms mentioned above, resulting in a sparse, quantized representation of the original BAP data, with a corresponding lookup table to enable reconstruction of the original data. The resulting hybrid algorithm yields compression ratios significantly better than those obtained using the MPEG-4 standard, and requires significantly

less client-side power measured in terms of CPU cycles and Joules of energy needed to decompress the data.

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 via intelligent exploitation of the hierarchical structure of the human skeletal model. Giacomo et al. [14] present methods for adapting a virtual human's representation and the resulting animation stream, and provide practical details for the integration of these methods into MPEG-4 and MPEG-21 architectures. Aubel et al. [15] present a technique for 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 of the client device on which 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 algorithm proposed 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 predefined quality control parameters, which allow for fine control of the 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 indeed useful for quality controlled compression of BAP-based motion data.

2. MATRIX REPERSENTATION OF BAP DATA

The BAPs are represented by an *n* x *m* dimensional matrix **X**, where *n* is a multiple of the video sampling rate or frame rate expressed as 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 of the matrix 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 **X** depict incremental changes in the pose of the avatar in small time steps, thus animating the avatar. Consequently, each row of **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 **X** is a 330 x 62 array of floating point numbers. The first 3 columns of **X** represent the absolute displacement of the avatar from a fixed origin in the 3D virtual world. The next three columns represent the absolute orientation of the avatar with respect to the virtual world coordinates. The remaining 56 columns correspond to the angles made by the degrees of freedoms associated with the various joints in the human avatar model.

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

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

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

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 **d** has *n* rows.

3. INDEXING OF THE 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 bits than that the corresponding floating point number. Note that the indexing technique proposed in this paper holds good for the animation of any hierarchical character, including cartoons and other human-like characters.

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 thus uses 2^8 buckets. Later, we analyze indices of sizes other than 8 bits.

3.1 Indexing of motion data d

In order to ensure efficient indexing, we have used the standard equal frequency distribution technique to uniformly 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 into 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 $\mathbf{d_{index}}$ is used to store the bucket number for the corresponding entry in the matrix **d**.

3.2 Lookup table for the index matrix $\mathbf{d_{index}}$

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

A straight-forward 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 $\mathbf{T_{lookup}}$, 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 2**). 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*.

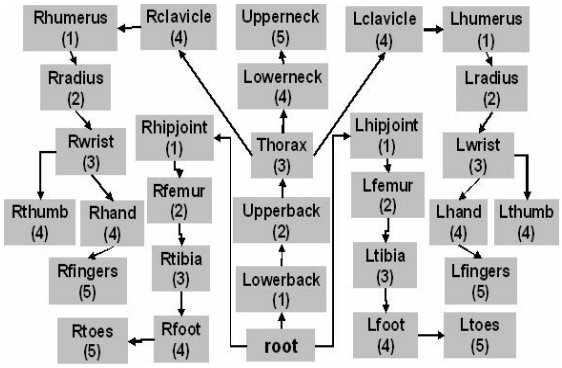

Figure 2: An example of the hierarchical structure of a human skeletal model consisting of 31 nodes, with a total of 62 degrees of freedom for the motion (rotational and translational). For convenience, the root node is drawn at the bottom.

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 $\mathbf{T_{lookup}}$ is given by:

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

where η is a constant. Empirical observations have revealed that as η increases, the $\mathbf{T_{lookup}}$ 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 for 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 $\mathbf{T_{lookup}}$. The proposed motion indexing technique yields reconstructed motion of high visual quality for most types of avatar motions that span 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 in **d** 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.

There is a tradeoff between the size of the motion data that is indexed and the quality of the reconstructed motion. The longer the sequence, the lower the reconstructed motion quality after indexing. To overcome this problem, we decompose larger motion matrices into smaller

motion matrices. The original difference matrix **d** is represented by a succession of smaller matrices $\mathbf{d_1}$, $\mathbf{d_2}$, …, $\mathbf{d}_L$ each consisting of an equal number of frames, denoted by n_d; i.e., each fragmented motion matrix has n_d rows. Each of the smaller matrices of size n_d x m in turn is discretized using the above method resulting in separate lookup tables, $\mathbf{T_{lookup}}^{1}$, …, $\mathbf{T_{lookup}}^{L}$. This simple technique yields a good approximation to the motion data of any arbitrary duration. The value of n_d is determined such that it minimizes both the file size and the resulting reconstruction error.

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 index the remainder of the columns. This is especially true for the first 6 columns of **d**, 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 consequently result in contradictory physical appearances such as the avatar's feet not touching the ground while walking, etc.

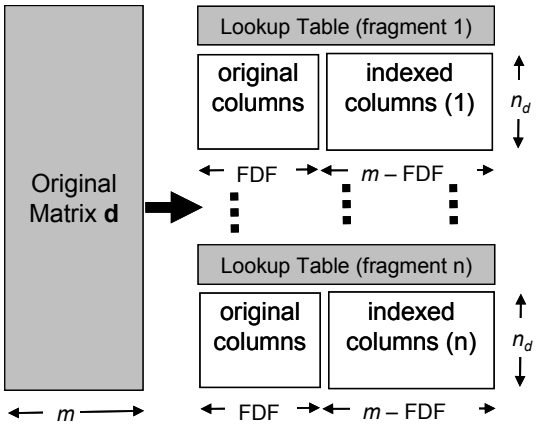

Figure 3: The indexing steps. The first step is converting the original motion matrix **X** to an initial pose vector **I**, and a difference matrix **d**. The motion matrix **d** is decomposed into equal fragments of length n_d rows each. The first **FDF** columns are untouched; the next m – **FDF** columns are indexed. Each fragment in indexed separately, and each fragment has a lookup table.

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 total indexing of the 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 3**.

4. SPARSING THE INDEXED REPRESENTATION OF THE BAP DATA

Once the difference matrix **d** is indexed, we now use the second algorithm, *BAP sparsing*, to remove many indices from the indices-matrix. This leads to further reduction in the amount of data required to represent the lossy animation. The basic intuition underlying *BAP sparsing* is to allow the joint corresponding to the BAP parameter to be *frozen*, for a fraction of a second, and then *released* all of a sudden. However, once released, the joint corresponding to the BAP parameter will have greater cumulative displacement to make up for the lost displacements when frozen. If performed judiciously, the *freeze-release* operation should be imperceptible to the human eye.

```
for k ← 1 to m
   S_k = d_1,k

for i ← 2 to n
   for k ← 1 to m
      S_k ← S_k + G_i-1, k
      if ((|S_k - m_k |/ S_k) > T_k)
         B_i, k = S_k
         S_k  = X_i, k
      else
         B_i, k = 0

for i ← 2 to n
   for k ← 1 to m
      d_i-1,k = B_i, k
```

Figure 4: The algorithm for *BAP Sparsing*, assuming the P-Frames are in a matrix format $\mathbf{X}_{n \times m}$. $\mathbf{G}_{i,k} = \mathbf{X}_{i,k} - \mathbf{X}_{i-1,k}$. **B** is a temporary matrix, whose values are finally assigned to **d**.

4.1. Implementation of the Sparsing algorithm

To implement this idea, for each column of the motion matrix **d**, we consider each floating point number corresponding to the BAP index, and drop (reduce to zero) consecutive indices along a column, and accumulate the floating point values corresponding to the dropped indices until the cumulative value exceeds a predefined *threshold*. At this point, we replace the current index by the corresponding index of the cumulative value, and reset (to zero) the variable corresponding to the cumulative value. The process is continued till all the rows in the column are either reduced to zero or replaced by a corresponding index for the cumulative value. In the remainder of this section, we detail the sparsing algorithm and the computation of the threshold value.

Without loss of generality, it is assumed that the BAP P-frame encoding error data is represented by an *n* x *m* matrix $\mathbf{X}_{n \times m}$, where *n* is the number of frames of the animation, and *m* is the number of BAP parameters to be used for defining a pose of the model (the maximum value for *m* = 296 as defined in the MPEG-4 standard). Initially, the mean $\mathbf{m_i}$ and standard deviation $\mathbf{s}_i$ of each column (parameter) *i* are computed. For each instance of each BAP parameter *i* that passes through the compression pipeline, a variable $\mathbf{S_i}$ is used to accumulate the difference of predictive encoding errors, in successive frames. The current encoding error is set to zero (*sparsing stage*), until the normalized cumulative encoding error ($|\mathbf{S_k} - \mathbf{m_k}| / \mathbf{s_k}$) exceeds a predefined threshold value $\mathbf{T_i}$. The value of the threshold $\mathbf{T_i}$ is determined using information derived from the hierarchical skeletal model of the human, for which the BAPs are being used. The error sparsing algorithm is given in **Figure 4**. A detailed discussion on the determination of the threshold value $\mathbf{T_i}$ is presented in the next section.

The centered and normalized value of the cumulative predictive encoding ($|\mathbf{S_k} - \mathbf{m_k}| / \mathbf{s_k}$) is used to determine the value of the threshold $\mathbf{T_i}$ in a manner irrespective of the mean and spread of the data in the columns of **X.** This stage renders the algorithm more general, enabling it to span a wide range of motions, and also enables the setting of a default value of the control parameter.

4.2. Determination of the Threshold $\mathbf{T_i}$

We compute the threshold value for each BAP parameter by exploiting the hierarchical structure of the underlying human skeletal model. The basic observation that we have utilized is the fact that the threshold value for a particular BAP parameter should depend on the position of the corresponding body joint in the hierarchical human skeletal model. For example, consider the hierarchical human skeletal model given in **Figure 2**. Any joint higher in the hierarchy is allowed less angular error compared to joints lower down in the hierarchy. For example, an

angular error at *Rhipjoint* displaces the joint *Rfoot* to a greater extent than the same angular error at the joint *Rtibia*, which is lower down in the hierarchy.

We have represented the hierarchical significance of the various joints in the human body by a level number, as shown in the parentheses in **Figure 2**. The threshold $\mathbf{T_i}$ for a joint *i*, represented by column *i* in matrix $\mathbf{X}_{n \times m}$, should be such that the joints higher in the hierarchy, with a smaller level value, should be allowed *less angular error* due to sparsing of the predictor error, when compared to joints with a larger level value. This ensures that the *displacement error* induced by the sparsing operation is small. For a column *i* of $\mathbf{X}_{n \times m}$, with level L_i, the corresponding threshold $\mathbf{T_i}$ is given by

$$\mathbf{T_i} = \boldsymbol{K} L_i \qquad \textbf{(2)}$$

where ***K*** is the another quality control parameter (QCP). Typically, the value of ***K*** is chosen such that the maximum threshold value is less than 1.

The two compression algorithms lead to the formulation of four quality control parameters, including *K* mentioned above. In the next section, we present a detailed analysis of the impact of these quality control parameters on the compression ratio and quality of animation.

5. EFFECT OF THE QUALITY CONTROL PARAMETERS ON COMPRESSION RATIO AND ANIMATION QUALITY

The two algorithms combined, as mentioned above, have four associated quality control parameters (QCPs); three indexing parameters: **FDF** (Fixed degree of freedom), *b* (number of bits for an index), and n_d (the number of rows in each block of the segmented motion matrix), and one sparsing parameter *K* (a constant that determines the threshold $\mathbf{T_i}$). The bounds for the QCPs are as follows:

$1 \leq \mathbf{FDF} \leq m$ (where m = number of columns in motion matrix d)

$8 \le b \le 32$ (assuming byte boundary for the indices)

$n_d^{min} \le n_d \le n$

$0 < K < \infty$

In order to analyze the effect of the QCPs, it is essential to determine the size of the motion file as a function of the above four QCPs and other motion data parameters.

5.1. Compressed File Size as functions of FDF, b, n_d and K

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 be easily extended to double-precision floating point numbers (8 bytes). We assume that the integer and floating point numbers are each 4 bytes long. In order to determine the compression ratio, it is essential to compare the file sizes (in bytes), of the original motion data file and the sparsed-index file. Assuming that the entries in the matrix **d** are floating point numbers (4 bytes each), and the initial vector **I** has m floating point numbers, the original file size is given by

$$\text{original\_file\_size} = 4m(n\text{-}1) + 4m = 4mn$$

where m = number of columns in **d** (or the total number of degrees of freedom) and n = number of rows in the original motion matrix **X** (number of frames of avatar poses in the animation sequence).

5.1.1. File size assuming no sparsing ($K = 0$)

Assuming no sparsing (i.e. sparsing parameter $K = 0$), the file size is obtained simply by expressing the number of bytes as a function of the three indexing QCPs (Quality Control Parameters). 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 $\mathbf{T}_{\mathbf{lookup}}$. We assume that a part of the matrix **d** has been indexed and is termed as $\mathbf{d}_{\mathbf{index}}$. 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 3**) 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 given by:

$$\text{segment\_header} = 4 \bullet 2^b + c_1 \quad (3)$$

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 ($0 \leq \mathbf{FDF} \leq m$) 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:

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

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

$$\mathbf{T}(\mathbf{FDF}, b, n_d) = \lceil n/n_d \rceil \bullet (\text{segment\_ header} + \text{segment\_motion}) + 4m \quad (5)$$

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 $\mathbf{T_{min}}$ with sparsing parameter $K = 0$ is obtained for **FDF** = 0, $n_d = n$ and $b = 8$ (assuming a byte boundary for the indexed data). The maximum file size, $\mathbf{T_{max}}$, 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, where each entry is a 4-byte floating point number). For the parameter values mentioned for $\mathbf{T_{min}}$, the minimum file size is (including the lookup table and initial pose vector) is given by:

$$\mathbf{T_{min}} = (n\text{-}1)m + 4m + c_2 \qquad (6)$$

where c_2 is the number of bytes 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 given by:

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

Consider a 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 these 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 corresponding to 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, such that streaming this file in a low-bandwidth situation is possible. For a desired fraction 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:

$$(n\text{-}1)m + 4m + 1024 \leq \mathbf{T}(\mathbf{FDF}, b, n_d) \leq f \bullet 4mn \qquad (8)$$

where f (henceforth termed *minimum-compression-ratio*) is the fraction of the maximum file size of the original motion data. The parameter values that satisfy equation (**8**) 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 fix $b = 8$, 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 33KB for the original file. Thus, MPEG-4 compression results in a file size which is 87% of the original file size. Thus, we fix *minimum-compression-ratio* $f = 87\%$ for the jogging example in equation (8), and exhaustively find all the values for parameters **FDF** and n_d (recall that we have already fixed $b = 8$, and for now $K = 0$) which satisfy equation (8). This subset of the original set of possible parameter values is termed the set of 'valid' points in the parameter space for the jogging motion data.

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 file size and reconstruction error. This is discussed in detail in **Subsection 5.2**.

5.1.2. File size obtained after combining Sparsing with Indexing ($K > 0$)

In order to sparse the indexed representation of the motion matrix **d**, it is essential to determine a good value for the sparsing parameter K such that the resulting motion quality is not distorted. A basic heuristic is to keep the normalized threshold value $\mathbf{T_i} \leq 1$ for each column (joint degree of freedom). This is possible by keeping $K \leq 0.2$, since the maximum level value can be 5. Assigning K an empirically selected value less than 0.2, say $K = 0.1$, the amount of sparsing

resulting in the indexed matrix depends on the type of motion. We will give the results obtained for $K = 0.1$ for various motion examples to demonstrate how sparsing helps to reduce the file size. The total file size is now given by **T**(**FDF**, b, n_d, K), which is some percentage of **T**(**FDF**, b, n_d).

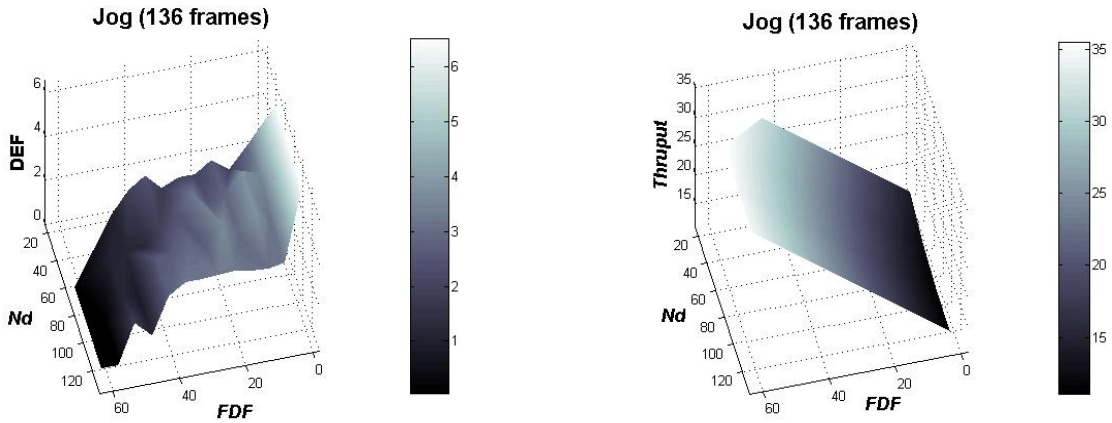

Figure 5: (**Left**) The surface plot for the displacement error per frame (**DEF**) for $30 \leq n_d \leq 120$ rows and $1 \leq$ **FDF** ≤ 62 columns. (**Right**) The surface plot for the obtained throughput for $30 \leq n_d \leq 120$ and $1 \leq$ **FDF** ≤ 62. Both the plots have index-bit $b = 8$, and sparsing parameter $K = 0.1$.

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

To quantify the reconstructed motion quality, we use the metric **DEF** (Displacement Error per Frame). For each of the m joints in the human model, the displacement error for that joint is defined as the frame-by-frame sum of all the differences between the actual joint position in the original motion data and the joint position in the reconstructed motion data. Let $\mathbf{C}_{nx3m}$ be the original coordinates of the joints of the human model, and $\mathbf{C'}_{nx3m}$ be the coordinates of the joints of the human model after reconstruction of the motion from the indexed matrix. We define the error function, Δ:I $\rightarrow$ R such that

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

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 error at each joint position. The **DEF** is defined as the sum of the errors for all the joints, normalized by the total number of rows (i.e. frames) in the motion data matrix.

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

Although the **DEF** value does not represent the motion quality in an absolute sense, comparing the **DEF** values for various quality control parameters provides an adequate measure of relative motion quality. The surface plots of **T**(**FDF**, b, n_d, K) and **DEF**(**FDF**, b, n_d, K), for the jogging motion example, for the various possible values of **FDF** and n_d, while fixing b = 8 and K = 0.1, are presented in **Figure 5**; 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, K), and **T**(**FDF**, b, n_d, K), are the optimal parameters. Since we have already fixed b = 8 and K = 0.1, we now need to determine the optimal values for the quality control parameters **FDF** and n_d. These are the values which minimize the following figure of merit function:

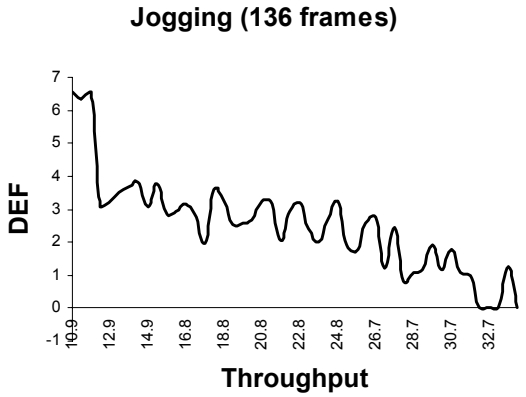

Figure 6: Plot of Displacement Error per Frame (**DEF**) vs. throughput **T** (in Kbytes) obtained by varying *f*. Note that **DEF** dips to zero at **T** = 32 KB, which is the throughput for the original data.

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

A simple exhaustive search reveals that **FDF** = 41 and n_d = 60 (with b = 8 and K = 0.1 as fixed values) minimizes the figure of merit function **M** for the jogging example. However, while the MPEG-4 compression of BAP data is lossless, the indexing method is lossy. Visually, the degradation in the reconstructed motion is imperceptible. An immediate observation is that the *minimum-compression-ratio f* is relatively quite large (87%); theoretically, *f* can take much lower values, which, though, will yield a higher **DEF** value. Indeed, it is true that the **DEF** does not give a direct quantification of the motion quality. 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 with 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

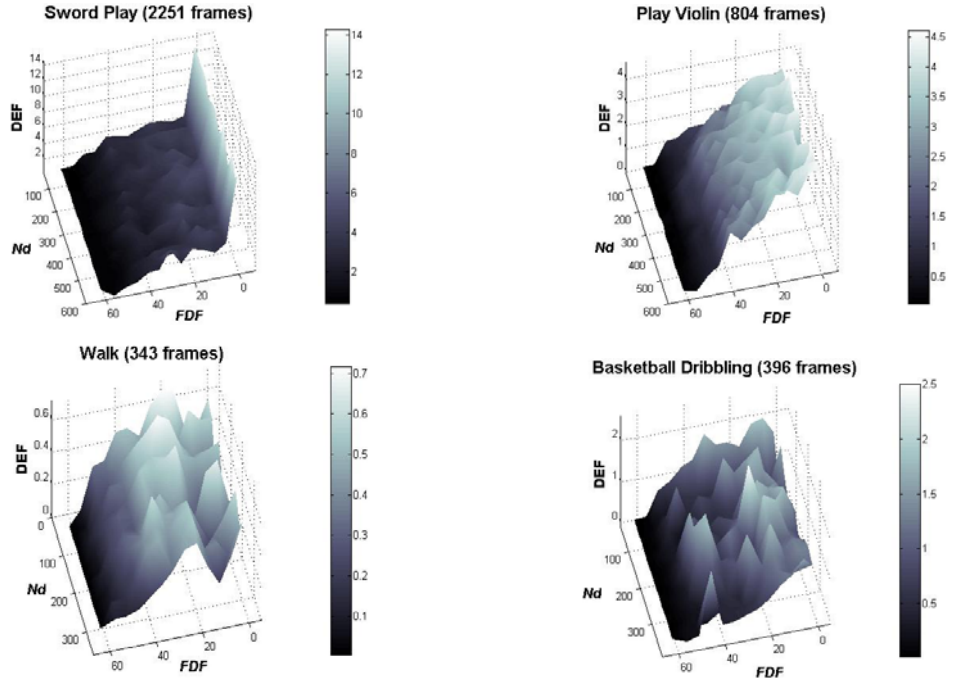

Figure 7: **DEF** plots as a function of n_d (number of rows per segment of motion) and **FDF** (no. of columns not indexed) for more motion examples, keeping parameters K = 0.1 and b = 8 as constant. The most general trend is that **DEF** increases with decrease of **FDF**

in **Figure 6**. For a maximum acceptable value of **DEF**, the corresponding throughput **T** can be obtained from the plot.

6. EXPERIMENTAL RESULTS FOR DIFFERENT TYPES OF MOTIONS

In order to test the usefulness of the proposed sparsed-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 such as dancing. We have considered $b = 8$ (number of bits used to encode an index) and $K = 0.1$ (the sparsing coefficient) for all of 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 various motion examples are presented in **Figure 7**. The throughput vs **DEF** plots for these motion examples display characteristics similar to those of the corresponding plot for the jogging example in **Figure 6**, and hence have not been shown. The **DEF** surface plots in **Figure 7** 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 which have not been fixed (i.e. **FDF** and n_d) 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** surface supports our use of the exhaustive search method, as the surface does not seem to exhibit any known distribution or function.

6.1. Minimum throughput obtainable by Sparsed-Indexing compared to MPEG-4

For applications that can accept a slight visual distortion of the reconstructed motion, indexing

motion type [1]	Motion Duration (seconds) [2]	MPEG-4 (K-Bytes) [3]	Indexed (K-Bytes) [4]	Sparsed-Indexed (K-Bytes) [5]	MPEG-4 128 Kbps (mJoules) [6]	Sparsed-Indexed 128 Kbps (mJoules) [7]	Sparsed-Indexed/ MPEG-4 (ratio) [8]	MPEG-4 1 Mbps (mJoules) [9]	Sparsed-Indexed 1 Mbps (mJoules) [10]	Sparsed-Indexed/ MPEG-4 (ratio) [11]	MPEG-4 4 Mbps (mJoules) [12]	Sparsed-Indexed 4 Mbps (mJoules) [13]	Sparsed-Indexed/ MPEG-4 (ratio) [14]
jog	4.1	28	10.9	6.8	2913	1259	*43.2%*	1002	796	*79.4%*	798	746	*93.5%*
walk	10.4	73	24	14.6	7534	2977	*39.5%*	2551	1982	*77.7%*	2018	1875	*92.9%*
dance	13.2	92	29.82	18.1	9504	3742	*39.4%*	3225	2505	*77.7%*	2552	2372	*92.9%*
basketball	12.0	84	27.41	18.0	8676	3528	*40.7%*	2943	2299	*78.1%*	2329	2168	*93.1%*
sword play	68.2	465	147	100.9	48344	19947	*41.3%*	16607	13058	*78.6%*	13207	12320	*93.3%*
run-leap	7.6	50	18.2	12.2	5246	2298	*43.8%*	1834	1465	*79.9%*	1468	1376	*93.7%*
play_violin	24.4	162	53.31	34.8	16948	7030	*41.5%*	5892	4652	*79.0%*	4707	4397	*93.4%*
forward-jump	12.6	86	28.61	18.9	8934	3702	*41.4%*	3064	2410	*78.7%*	2436	2272	*93.3%*
jump	10.6	72	24.55	16.1	7499	3138	*41.9%*	2585	2040	*78.9%*	2058	1922	*93.4%*
stride	9.0	59	21.19	12.8	6200	2598	*41.9%*	2174	1723	*79.3%*	1742	1630	*93.5%*

Table 1: Comparison of compression ratio obtained via sparsed-indexing and power consumption in m-joules by the network card interface for various motion examples. Column [3] gives the compression file size after MPEG-4 arithmetic coding based compression. Column [5] gives the compressed file size obtained after sparsed-indexing. Columns [6] and [7] give the energy consumption in milli-Joules for the reception of streaming MPEG-4 and Sparsed-Indexing data for a 128 Kbps network. Column [8] gives the ratio of power consumption by the network card using the sparsed-indexing, compared to MPEG-4. Columns [9] – [11] give comparisons for network bandwidth of 1 Mbps, and columns [12] – [14] for 4 Mbps. $\boldsymbol{E_S}$ = 177 mJ/s and $\boldsymbol{E_R}$ = 1425 mJ/s. The indexed size obtained is the minimum size possible by indexing. Sparsing is done on the indexed representation.

can be used judiciously to reduce the throughput requirement significantly, compared to MPEG-4-based compression. **Table 1** 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, by the use of the selective averaging technique to create the lookup tables intelligently. **Table 1** also displays the power consumption for the reception of streaming BAP data in the network card for various network bandwidths. A detailed analysis of the power consumption and its computation is presented in **Section 6.3**.

6.2. Power Consumption for decoding by Sparsed-Indexing compared to MPEG-4

In order to provide a fair comparison with MPEG-4 based compression, we first encode the motion data using MPEG-4 arithmetic coding based compression. We compute the *minimum-*

Motion Type	No. of frames	Org. size (Kbytes)	MPEG-4 (K-Bytes)	G-cycles decoding MPEG-4	Ratio *f*	Indexed (K-Bytes)	DEF of Indexed	Sparsed-Indexed (K-Bytes)	DEF of Sparsed-Indexed
jog	136	33	28	0.65	85%	27.74	1.19	18.99	1.49
run-leap	251	61	50	0.93	82%	47.5	0.29	28.55	0.45
stride	298	72	59	1.23	82%	47.29	1.08	29.67	1.97
walk	343	83	73	1.34	88%	69.13	0.15	45.91	0.17
jump	351	85	72	1.35	85%	62.47	0.14	41.95	0.25
basketball	396	96	84	1.44	88%	79.48	0.23	53.84	0.42
forward-jump	415	101	86	1.56	86%	85.22	0.33	55.71	0.34
dance	434	106	92	1.68	87%	86.9	11.16	58.93	14.09
play_violin	804	195	162	2.89	83%	159.17	0.78	104.81	1.23
sword play	2251	545	465	8.49	85%	450.93	1.45	315.58	2.49

Table 2: Comparison of MPEG-4 predictive encoding and arithmetic coding based compression vs compression obtained as indexed motion data. The *minimum-compression-ratio f* is computed such that the compression ratio obtained by indexing is similar to MPEG-4 based compression. The sparsing parameter $K = 0.1$.

compression-ratio f of the obtained file size after MPEG-4 compression to the original motion file size, and use this *minimum-compression-ratio 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 (note that $b = 8$ and $K = 0.1$ have been fixed previously). 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 2**. As evident from the table, the indexing method, when tuned to yield the same throughput requirement as MPEG-4 compression, results in a slightly lossy reconstructed motion, but needs no additional 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, and hence significantly more power consumption for decoding compared to the method developed by us. 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.

6.3. Power usage due to data reception at the network card by Sparsed-Indexing compared to MPEG-4

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 duration time T, data size S and given available bandwidth B, the energy used by the WNIC is given by

$$E_{network} = E_R \bullet S/B + E_S \bullet (T - S/B) \qquad (11)$$

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 **(11)**, we computed the network card energy utilization for MPEG-4 and the indexed file representation (**Table 1**). We have used energy usage data from [20] [21] to obtain the 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 based compression.

An interesting observation from **Table 1** is that as more bandwidth is available, the power consumption for reception of low-throughput indexed data approaches that of MPEG-4 compressed data. The reason for this is that the two differ only in the reception time; for the sleep time, the power consumption is the same. For higher bandwidths, the time taken to receive the data is a small fraction of the total time of the motion sequence. For low-power devices in mobile network environments, such as PDAs and Pocket PCs, the available bandwidth is often as low as 128 Kbps. Streaming indexed motion data instead of MPEG-4 compressed data is significantly beneficial in this case.

From the last three subsections, it is obvious that sparsing-indexing the motion data using the techniques described in the 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 sparsed-indexing method offers enormous benefits.

7. CONCLUSIONS AND FUTURE WORK

We have proposed and implemented a novel sparsed-indexing technique to compress BAP data used for MPEG-4 compliant character animation. The proposed sparsed-indexing method has advantages in terms of both 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 reduction in both the throughput requirement for networked applications requiring motion data exchange, and client power consumption for data reception and decompression. 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 motion sequences of long duration, we have proposed a simple method for decomposing the motion data sequence into smaller motion data sequences, thus preserving the quality of motion data for arbitrarily large motion files. The quality and throughput requirements of the motion data are controlled via four quality control parameters. We have proposed a simple systematic elimination procedure to obtain the best possible combination of these 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 and data reception at the network card interface, compared to the existing MPEG-4 based 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 four aforementioned parameters.

A limitation of the proposed technique is that the optimal values of two of the parameters, **FDF** and n_d, are obtained via exhaustive search, and values for the sparseness coefficient K and indexing-bits b are obtained via empirical observations. 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.

8. 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., 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., Marichal, X., "Interacting with Virtual Agents in Mixed Reality Interactive Storytelling", *Proc. 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," *in 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"*, Proc. IWSNHC3DI'99*, Santorini, Greece, 1999.

[12] Endo, M., Yasuda, T., 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", *Proc .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", *Proc. 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", *Proc. 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", *Proc. 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", In *Proc. 3rd Intl. Workshop on Mobile Multimedia Comm.*, September 1996.

[21] Paul J. M. Havinga. "Mobile Multimedia Systems", *PhD thesis*, Univ. of Twente, Feb 2000."

[22] Chattopadhyay, S., Bhandarkar S. M. and Li, K., "Compression by Indexing: An Improvement over MPEG-4 Body Animation Parameter Compression", *Proc. Of ACM Multimedia Computing and Networking (MMCN'06)*, San Jose, California, Jan 2006.

[23] Chattopadhyay, S., Bhandarkar S. M. and Li, K., "BAP Sparsing: A novel approach to MPEG-4 Body Animation Parameter Compression", *Proc. of IEEE Computer Society International Conference on Multimedia Communications Systems (ICMCS'05)*, Montreal, Canada, Aug 2005. pp. 104 - 109.