Efficient Compression and Delivery of Stored Motion Data for Avatar Animation in Resource Constrained Devices

Siddhartha Chattopadhyay Suchendra M. Bhandarkar Kang Li
Department of Computer Science, The University of Georgia
Athens, Georgia 30602-7404, USA
Telephone: +1 (706) 542-3455, Fax: +1 (706) 542-2966

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

ABSTRACT
Animation of Virtual Humans (avatars) is done typically using motion data files that are stored on a client or streaming motion data from a server. Several modern applications require avatar animation in mobile networked virtual environments comprising of power constrained clients such as PDAs, Pocket-PCs and notebook PCs operating in battery mode. These applications call for efficient compression of the motion animation data in order to conserve network bandwidth, and save power at the client side during data reception and motion data reconstruction from the compressed file. In this paper, we have proposed and implemented a novel file format, termed the Quantized Motion Data (QMD) format, which enables significant, though lossy, compression of the motion data. The motion distortion resulting from the reconstructed motion from the QMD file is minimized by intelligent use of the hierarchical structure of the skeletal avatar model. The compression gained by using the QMD files for the motion data is more than twice achieved via standard MPEG-4 compression using a pipeline comprising of quantization, predictive encoding and arithmetic coding. In addition, considerably fewer CPU cycles are needed to reconstruct the motion data from the QMD files compared to motion data compressed using the MPEG-4 standard.

Categories and Subject Descriptors
I.3.7 [Computer Graphics] Three Dimensional Graphics and Realism – Animation.

General Terms
Algorithms, Performance.

Keywords
Human Motion, Avatar Animation, Distributed Virtual Reality.

1. INTRODUCTION
Animation of virtual humans (avatars) has potential applications in the design of human computer interfaces and modeling of virtual environments using power constrained devices [1] [19]. Distributed 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 network 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 humans [7] [8] [16]. A virtual human body model (avatar) 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 comprises of 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 have proposed and implemented a novel motion data file format, termed the Quantized Motion Data (QMD) file format, which is suitable for compression of motion data for bandwidth-and power-constrained clients in a mobile network environment. The QMD files can be used either for locally stored motion data or for streaming motion data from a server. The compression algorithm used to create a QMD file from the motion data uses indexing of the floating point numbers in the original motion data. This enables significant compression, as the indices can be represented using fewer bytes than the original floating point data. Reconstruction of the motion can be obtained by simply using the indices and a corresponding lookup table, which is also included in the QMD file. Using the QMD file, it is possible to recover a good approximation to the original motion data, without having to explicitly decompress the data, thus saving CPU cycles at the client end. Since the creation of indexed data

VRST'05, November 7–9, 2005, Monterey, California, USA.

invariably leads to motion reconstruction error, we have used a novel technique for efficient lookup table construction, by intelligently exploiting the inherent hierarchical structure of the skeletal avatar mode. This results in negligible perceptible error in the reconstructed motion. There is an inevitable tradeoff between the size of the QMD file, network throughput requirements of the distributed application, and the quality of the reconstructed motion. This tradeoff is quantitatively controllable via three quality control parameters which are specified in the QMD file. We have proposed a simple method for determining the optimal combination of the three parameters to maximize the reconstructed motion quality and simultaneously minimize the QMD file size. We have also proposed an application layer protocol which renders the QMD file format useful for real time streaming of the data for distributed virtual reality applications.

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 3D scene structure. Our method uses quantization to achieve data compression in a manner similar to the above paper, but by using a totally different approach. Giacomo et al. [14] present methods for adapting a virtual human's representation and animation stream, and practical details for the integration of these methods into MPEG-4 and MPEG-21 architectures. Aubel et al. [15] explain the 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 de-compression of the compressed motion data. The technique 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 such as PDAs and Pocket PCs in a mobile network environment. In addition, the proposed method incorporates quality control parameters, which lead to fine control of throughput 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 control of the motion files.

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. Next, we describe a technique for converting standard motion data to the QMD file format for efficient compressed representation of the motion data. Next, we describe a simple application layer protocol that can be used to achieve real time animation via streaming. Next, we analyze the network throughput requirements and quality of the reconstructed motion, and compare the results with those obtained using the compression technique defined in the MPEG-4 standard. Finally, we present the conclusions and outline areas for future improvements.

2. MATRIX REPRESENTATION OF BAPs

The BAPs are represented as an n x m dimensional matrix $\mathbf{X}$, where n is a multiple of the frame rate (expressed as frames per second or fps) 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 of a particular joint in the skeletal avatar to achieve the desired pose during the corresponding time interval. Successive rows of $\mathbf{X}$ depict incremental changes in the pose of the avatar in small time steps, thus animating the avatar.

In our application, we have used a 62-dimensional avatar, with a frame rate of 33 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. As a first step in the compression process, the matrix $\mathbf{X}$ is represented by a difference matrix, $\mathbf{d}_{n-1 \times m}$, 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 $\mathbf{X}$.

$$\mathbf{I}_{1,j} = \mathbf{X}_{1,j} \qquad j = 1, 2, \ldots, m$$
$$\mathbf{d}_{i,j} = \mathbf{X}_{i+1,j} - \mathbf{X}_{i,j} \qquad i = 1 \ldots n\text{-}1;\ j = 1 \ldots m$$

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 will assume that $\mathbf{d}$ has n rows.

3. CONVERSION OF d TO QMD FILE FORMAT

Before describing in detail the conversion of the motion matrix $\mathbf{d}$ to the QMD file format, we first outline the intuition behind the proposed technique. The motion difference matrix $\mathbf{d}$ 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 of the corresponding motion matrix $\mathbf{d}$ 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 $\mathbf{d}$ to a finite number of buckets. Each bucket, in turn, is associated with a floating point number which best describes the collection of floating point numbers within the bucket. The basic concept underlying the QMD file format is to be able to index some (perhaps all) of the floating point numbers within the original motion matrix $\mathbf{d}$, 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 that the corresponding floating point number.

In the next three subsections, we describe in detail the process of indexing the motion data in the matrix $\mathbf{d}$, and the creation of the lookup table. The indices and the lookup table constitute the core of the QMD file. In the fourth subsection, we describe the QMD file format, which implements the proposed quantization technique. 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. Later, we will analyze other indices of sizes 9 and higher number of bits.

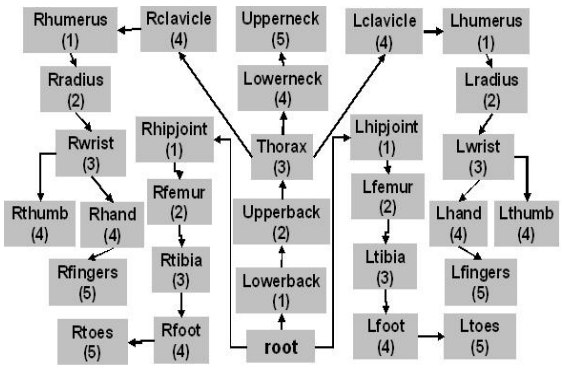

Figure 1: An example of the hierarchical structure of a human skeletal model consisting of 31 nodes, with a total of 62 degrees of freedom of motion (rotational and translational). For convenience, the root node is drawn at the bottom.

3.1 Indexing of motion data d

We have used the equal frequency distribution technique to assign the $n \bullet m$ floating point numbers in **d** to buckets numbered from 0 to 255. This is done as follows:

Step 1: Collect all the data in matrix **d** into a single 1-D array **A** of size $n \bullet m$, and sort the array **A** in ascending order. Multiply all the numbers in **A** by the *resolution quantization term (RQT)*, M (here, M = 10000). Round the numbers 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 creating 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 floating point numbers.

At the end of this step, we get a set of 256 buckets. **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 Lookup table for the index matrix $\mathbf{d}_{\mathbf{index}}$

The next step is to create a lookup table $\mathbf{T}_{\mathbf{lookup}}$ with 256 buckets, numbered 0 to 255, for the indices created in $\mathbf{d}_{\mathbf{index}}$. This lookup table is used to map each of the indices to a corresponding floating point number such that a suitable approximation to the original motion matrix **d** can be recovered.

The creation of the 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 floating point data after discretization invariably results in motion distortion. A simple method to recover the floating point 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 the construction of a superior lookup table $\mathbf{T}_{\mathbf{lookup}}$, which, in turn, results in reduced error in the motion reconstructed 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 (**Figure 1**). For each m-dimensional pose vector, each dimension, or column in the motion matrix **d**, is associated with a level l_i. 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 floating point numbers assigned to a bucket. The lookup value for the j^{th} entry in table $\mathbf{T}_{\mathbf{lookup}}$ is given by:

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

where k is a constant. Empirical observations have revealed that as k 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 $k \rightarrow \infty$, 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. A simple schematic diagram depicting the quantization of the original motion matrix **X** is given in **Figure 2**.

3.3 Motion matrix decomposition for motion sequences of long durations

In the previous two subsections, we have discussed in detail the creation of the index matrix $\mathbf{d}_{\mathbf{index}}$ from the motion matrix **d**, and the creation of the corresponding lookup table $\mathbf{T}_{\mathbf{lookup}}$. The

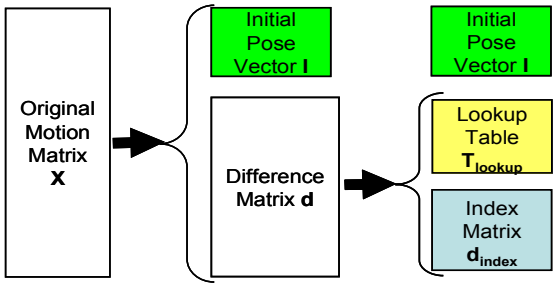

Figure 2: The compression steps. The first step is converting the original motion matrix **X** to an initial pose vector **I**, and a difference matrix **d**. The difference matrix **d** is then converted to an index matrix $\mathbf{d}_{\mathbf{index}}$ and the corresponding lookup table $\mathbf{T}_{\mathbf{lookup}}$.

proposed motion indexing technique yields good results for most types of avatar motions, for time durations of around a minute or less. For a motion data sequence spanning several minutes, it is difficult to quantize the motion matrix in a satisfactory manner. This is because the number of floating point numbers in a motion matrix increases with each additional frame or pose. This leads to the assignment of a large number of floating point numbers to each bucket. As a result, the single floating point number representing the collection of floating point numbers assigned to a bucket is less likely to a good representation of the 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 by a 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. Each of the smaller matrices of size n_d x m in turn is discretized using the above method resulting in separate lookup tables, $\mathbf{T_{lookup}}^1$, …, $\mathbf{T_{lookup}}^L$ and the corresponding index matrices $\mathbf{d_{index}}^1$, …, $\mathbf{d_{index}}^L$. 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**.

3.4 The QMD file format

In the previous three subsections, we discussed in detail the various steps involved in the quantization of the motion matrix **d**. It is important to devise an efficient file format which integrates the resulting compressed data obtained via motion quantization. The proposed Quantized Motion Data (QMD) file format contains the following information: the length of the motion sequences, the dimensionality m of the pose vector, the initial pose vectors **I** for each motion sequence, the lookup table $\mathbf{T_{lookup}}$, and the matrix of indices $\mathbf{d_{index}}$. The numbers of bits b used to encode the indices is specified in the header. The lookup table contains 2^b floating point numbers.

The QMD file format also contains a quality control parameter which allows the animator the flexibility to choose a combination of the original data and the quantized data. The quality control parameter termed the Fixed Degree of Freedom (**FDF**), $0 \leq \mathbf{FDF} \leq m$ ($m = 62$ in our case), defines the number of initial columns of the motion matrix **d** which are **not** to be included in the quantization step. Thus, a totally quantized motion is represented by **FDF** = 0, whereas the original motion matrix is represented by **FDF** = 62.

For motions of durations which are difficult to quantize satisfactorily using a single lookup table, it is essential to decompose the motion matrix into smaller segments. The original motion matrix with n rows (frames) is decomposed into several smaller matrices, each with n_d rows. Each such motion matrix segment requires a separate header containing all the motion file information mentioned above. Successive segments are represented as blocks, each block containing the header and the segment motion data. The additional overhead of a lookup table for each segment is much less compared to the main body of the segment motion data. The representation of a typical motion segment in a QMD file is depicted in **Figure 3**.

A judicious combination of the parameters **FDF**, b and n_d is observed to result in very low reconstruction error for the underlying motion data. We propose a simple exhaustive search method for computing these parameters in **Section 5.4.**

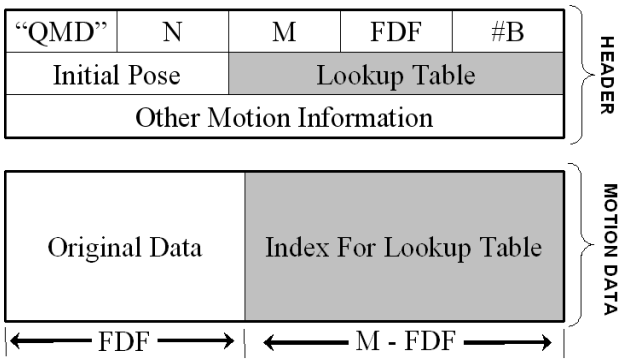

Figure 3: The QMD file format for compressed representation of a segment of motion data in total motion matrix **d**. $N = n_d$ and M correspond to the number or rows and number of columns of motion matrix fragment. **FDF** is the fixed degree of freedom, and #B is the number of bits b used to represent an index value. The Initial Pose, "QMD" and Other Motion Information are present only in the header of the first segment.

4. REAL TIME STREAMING OF MOTION DATA

For motion data that can be stored on the server in the form of motion matrix **X**, we propose a simple application layer protocol to stream the data to resource-constrained clients. The original motion data matrix **X** is converted to an initial pose vector **I**, index matrix $\mathbf{d_{index}}$ and corresponding lookup table $\mathbf{T_{lookup}}$, which are all integrated within a single QMD file. The protocol required to deliver this data across the network for real time distributed applications is straightforward. Delivery of the QMD file to a client is effectively equivalent to delivering the initial pose vector **I** and the lookup table $\mathbf{T_{lookup}}$, followed by a combination of the partial **d** and $\mathbf{d_{index}}$ data. The client uses the partial floating point data in **d** and the indices in $\mathbf{d_{index}}$ to look up the corresponding floating point number in the lookup table $\mathbf{T_{lookup}}$. The rendered animation is a good approximation to the original motion data. A schematic diagram depicting the protocol for streaming of motion data from the server to a client is given in **Figure 4**.

For larger motion matrices, the previously described technique of decomposing the motion matrix into smaller motion matrices (**Section 3.3**) is used to deliver the motion data in the form of distinct smaller groups of lookup tables and motion matrix indices. At the client end, explicit decompression of the data is not needed; instead, the client just uses the indices in $\mathbf{d_{index}}$ to get the corresponding floating point number from $\mathbf{T_{lookup}}$, and generates the pose parameters to render the avatar animation.

5. ANALYSIS OF THROUGHPUT AND QUALITY VS *FDF*, *b* AND n_d

In this section, we analyze the throughput requirement when the motion data is represented as a QMD file. We also analyze the quality of the reconstructed motion when the QMD file format is used for compressed motion data representation. We present our analysis in three subsections. In the first subsection, we analyze the required throughput for different types of motion as a function of quality control parameters **FDF (**Fixed degrees of Freedom) and the number of bits used for each index, *b***.** In the second subsection, we analyze the quality of the reconstructed motion from the QMD file as a function of **FDF** and *b.* In the final

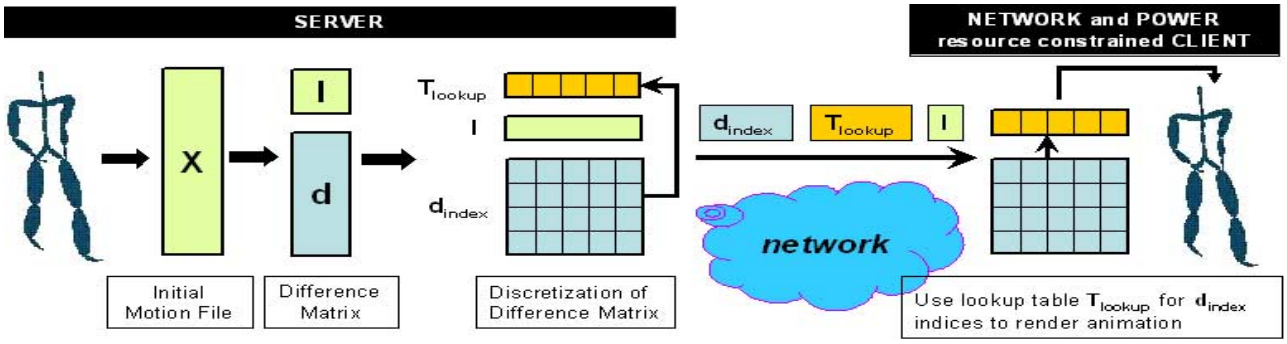

Figure 4: The complete process of motion data delivery over the network. For real time streaming, the QMD file is delivered over the network, which is equivalent to delivering the vector **I**, followed by $\mathbf{T_{lookup}}$ and finally $\mathbf{d_{index}}$ to the client side. The initial pose of the avatar is rendered using the initial pose vector **I**. The client uses the indices of $\mathbf{d_{index}}$ to look up the floating point values from $\mathbf{T_{lookup}}$ (all this information is present in the QMD file) to render the rest of the animation.

subsection, we propose a simple method of determining the three quality control parameters, **FDF**, b and n_d.

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

The QMD file format includes both the quantized and original floating point data obtained by varying the quality control parameters **FDF** (Fixed Degree of Freedom), b (number of bits used to represent each index) and n_d (number of frames in each segment of a long motion sequence). The upper and lower bounds for these integer parameters are: $0 \le \mathbf{FDF} \le m$, $b = 8, 9,..,16$ and $n_d \le n$, where the motion matrix **d** is an n x m matrix. 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 $\mathbf{T_{lookup}}$. The data section contains the actual data from the matrices **d** and $\mathbf{d_{index}}$. The first header contains the initial pose vector, **I**, which takes $4 \bullet m$ bytes.

The header file for each segment (**Figure 3**) takes 3 integers to define n_d, m and **FDF**, a single byte for b ($b < 255$), and $4 \bullet 2^b$ bytes for the lookup table. Hence, the total number of bytes taken by the header for a motion segment is

$$segment\_header = 4 \bullet 2^b + 13 \qquad \textbf{(2)}$$

The actual motion data is represented by a matrix consisting of floating point numbers and indices that can be represented by a maximum of 2 bytes. The first **FDF** columns ($0 < \mathbf{FDF} \le m$) are floating point numbers taken directly from the motion matrix **d**, and the next m – **FDF** columns are indices for the lookup table, such that each index takes a maximum of b bits, or a maximum of $\lceil b/8 \rceil$ bytes. Hence, the total number of bytes needed for the motion data segment is given by:

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

For motion data spanning n frames, the number of blocks, or motion segments, each consisting of n_d frames, is $\lceil n/n_d \rceil$. The last block may contain less than n_d frames. Hence, the maximum size of a QMD file, in bytes, is given by:

$$Bytes\_QMD = \lceil n/n_d \rceil \bullet (segment\_header + segment\_motion) + m \qquad \textbf{(4)}$$

It is obvious from equations **(2)**, **(3)** and **(4)** that the QMD 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. **Figure 5** shows the linear behavior of the file size *bytes_QMD* as a function of **FDF**, with the values for b = 8, 9, 10, 11 corresponding to lookup table of sizes 256, 512, 1024 and 2048 respectively, and with $n_d = n$. In Section **5.4**, we present a method to determine optimal values of the parameters **FDF**, b and n_d to maximize the quality of the reconstructed motion, and minimize the throughput requirement.

5.2 Motion quality as function of *FDF* and *b*

From the last section, it can be observed that keeping b as low as 8 bits and **FDF** = 0 results in the minimum QMD file size for representing the corresponding underlying motion. However, reducing the number of bits leads to increased motion reconstruction error; similarly, reducing **FDF** to zero also results in increased motion reconstruction error. A plot of motion reconstruction error (displacement error per frame after the motion is recovered from the QMD file) for some typical motion examples is given in **Figure 6**. As expected, keeping b to its lowest value 8, results in the maximum motion reconstruction error. Also, the motion reconstruction error decreases with increase in the quality control parameter **FDF**, as more of the original data is preserved.

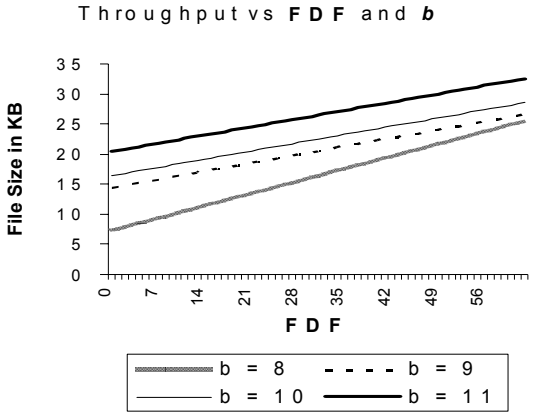

Figure 5: QMD File Size as a function of fixed degree of freedom (**FDF**) and values of b as 8, 9, 10 and 11. The number of frames n = 100 and m = 62.

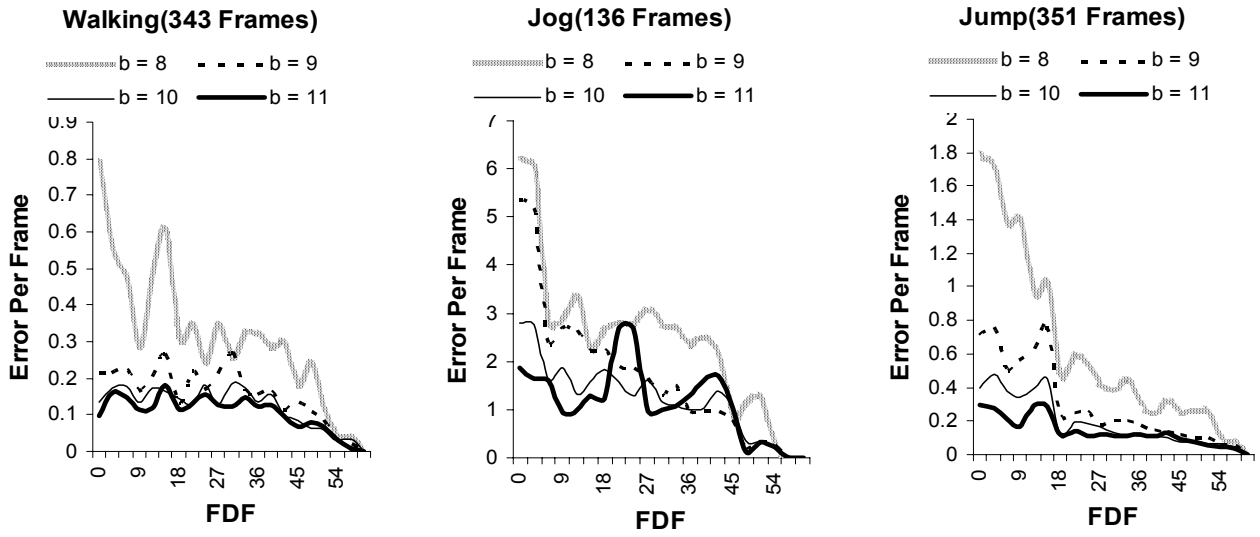

Figure 6: Comparison of Reconstruction error obtained as a function of the **FDF** for varying number of bits *b* used to represent an index (8, 9, 10 and 11) corresponding to lookup tables of sizes 256, 512, 1024 and 2048 respectively.

A pertinent question that immediately arises is: how does one quantify the quality of the recovered motion? Indeed, this is a fundamental problem with the current quantitative measures of animation quality. We have found little existing research that gives a good quantification of the quality of motion. The metric used in this paper, the displacement error per frame (**DEF**), gives an indication of the relative quality of the recovered motion when compared with the **DEF** of the same motion reconstructed using different QMD file parameters. Note that the **DEF** also gives a relative measure of motion complexity between different types of motions.

5.3 Optimal values for FDF, *b* and n_d

The compression ratio and quality of reconstructed motion are determined and controlled by the three parameters, **FDF**, *b* and n_d. Since the parameters are discrete, an optimal solution can be determined by an exhaustive search of all the parameter combinations, resulting in an O($\mathbf{n^3}$) search algorithm. The objective function or figure of merit for the optimization (minimization in our case) is given by:

$$figure\_of\_merit(FDF, b, n_d) = bytes\_QMD(FDF, b, n_d)\,DEF(FDF, b, n_d) \quad \textbf{(5)}$$

where *bytes_QMD* is the size of the file and **DEF** is the displacement error per frame. For an optimum combination of low file size and high quality of motion (low **DEF**), the value of *figure_of_merit* is very low. The simplest method for minimizing *figure_of_merit* is to perform an exhaustive search for the values of **FDF**, *b* and n_d that result in a minimum value of the objective function in **equation (5)**. An exhaustive search is possible since the number of possible values for each parameter is finite. An exhaustive search leads to the selection of optimal values for **FDF**, *b* and n_d, resulting in low file size and high quality of reconstructed motion. We have used an exhaustive search method to determine the optimal values for 12 different types of motion examples. The experimental results and a comparison with standard MPEG-4 based compression are presented in the next section.

6. EXPERIMENTAL RESULTS

In this section, we present our results obtained by constructing QMD files as explained in the previous sections for various motion examples. We have used 12 motion examples that encompass a wide range of avatar motions at a frame rate of 33 fps. The avatar skeletal model is 62-dimensional; i.e. $m = 62$. The results are grouped into two subsections. The first subsection discusses the benefits of un-segmented QMD files. The second subsection discusses the results obtained by optimizing the file using the optimizing technique present in **Section 5.3**.

6.1 QMD files without segmentation

We have observed that creating QMD files without segmentation results in considerable reduction in the compressed file size (and consequently, the network throughput requirement), but, at the same time results in higher reconstruction error. We have compared the sizes of the un-segmented QMD files with compressed files obtained using MPEG-4 predictive encoding-based BAP compression for the 12 motion examples. MPEG-4 specifies a compression pipeline consisting of quantization, predictive encoding and arithmetic coding for bit-level compression. The experimental results are presented in **Table 1**. The empirically determined values for the quality control parameters are **FDF** = 6 and *b* = 8. The motion example with the longest duration that could be converted to an un-segmented QMD file without visible motion distortion is sword play (2251 frames). An interesting observation is the fact that fast motions (jogging, play drums and violin) and very complex motions (dancing) result in a larger displacement error per frame (**DEF**) compared to more regular motions. This is probably because the floating point numbers in the motion matrices of these examples of fast and complex motions appear to be uniformly distributed and do not form clusters, thus reducing the chance of accurate reconstruction of the data. The resulting file size, in most cases, is almost half that of the MPEG-4 compressed file, with acceptable motion quality.

6.2 Optimized QMD files with segmentation

Optimizing the QMD file, using the method presented in **Section 5.3**, leads to reconstructed motion of better quality (with lower **DEF** value) compared to that obtained using the corresponding un-segmented QMD file. However, the optimal QMD file is, in general, larger due to the fact that more number of bytes may be required for indexing, and also because fragmentation of the data leads to higher overhead on account of more lookup tables. A comparison of the sizes of the optimal QMD files and compressed motion data files resulting from MPEG-4 encoding is presented in **Table 2**. The QMD file sizes are comparable, and sometimes greater than the sizes of the corresponding MPEG-4 compressed files. However, the quality of the reconstructed motion from the optimal QMD file is very good with hardly any perceptible difference between the original and the reconstructed motion. The DEF for the segmented QMD file is much less than that for the un-segmented QMD file (compare columns marked [2] in **Table 1** and **Table 2**). This is because the optimal QMD file segments the original motion file, in order to maximize the quality of the reconstructed motion, and also uses a greater number of buckets per motion segment, in general. As explained in the next section, QMD files have a distinct advantage in low-powered devices. Unoptimized and un-segmented QMD files yield reconstructed motion of acceptable quality, and yield significant reduction in the network throughput requirement. Hence, it is left to the animator's discretion to either obtain reconstructed motion of the best quality, or seek a compromise between the network throughput requirement and the reconstructed motion quality.

Results	Number of Frames [1]	DEF [2]	KB of Original Motion Data [3]	KB after MPEG-4 Compression [4]	KB of corresponding QMD file [5]	Compression Ratio: QMD vs. MPEG-4 [6]
jog	136	3	33.2	28	11.9	42.65%
run_Leap	251	0.7	61.0	52	20.9	40.25%
stride	298	1.4	72.4	64	24.6	38.44%
walk	343	0.5	83.3	69	28.1	40.75%
jump	351	1.4	85.3	62	28.7	46.36%
basket Ball	396	0.8	96.1	78	32.3	41.35%
forward jump	415	1.4	100.8	68	33.7	49.62%
dance	434	20	105.4	86	35.2	40.96%
play piano	691	1	167.6	99	55.3	55.86%
play drums	804	3.4	195.0	88	64.1	72.88%
play violin	804	3.2	195.0	98	64.1	65.44%
sword play	2251	4	545.4	390	177.2	45.43%

Table 1: Comparison of MPEG-4 predictive encoding and arithmetic coding based compression vs. compression obtained using the QMD file with unsegmented motion data. The fixed degree of freedom, **FDF** = 6. The reconstructed motion obtained from the QMD file exhibits negligible distortion, after using the proposed intelligent lookup table. The resulting QMD file is, on an average, half the size of the compressed MPEG-4 file.

Results	Number of Frames [1]	FDF	b	nd	DEF [2]	KB of Original Motion Data [3]	KB after MPEG-4 Compression [4]	KB of corresponding QMD file [5]	Compression Ratio: QMD vs. original motion data [6]
jog	136	11	9	30	0.73	33.2	28	29.7	89.50%
run_Leap	251	11	10	120	0.20	61.0	52	48.1	78.76%
stride	298	1	8	120	1.23	72.4	64	22.2	30.65%
walk	343	1	10	330	0.11	83.3	69	50.5	60.58%
jump	351	26	10	300	0.07	85.3	62	68.6	80.46%
basket Ball	396	11	10	210	0.19	96.1	78	64.7	67.32%
forward jump	415	6	10	90	0.20	100.8	68	75.4	74.86%
dance	434	1	8	150	19.06	105.4	86	30.8	29.26%
play piano	691	26	10	60	0.13	167.6	99	167.2	99.74%

Table 2: Optimal QMD file obtained after exhaustive search. The **DEF** is significantly less than that of the QMD file for the unsegmented motion (as shown in Table 1).

7. POWER SAVINGS USING THE QMD FILE FORMAT

QMD files have the option of either saving the network throughput requirement by compromising quality of motion slightly, or getting the best quality motion, but settling for throughput requirement comparable to standard MPEG-4 based compression. QMD files, compressed under optimal conditions, generally yield approximately the same file sizes as compressed motion files using standard MPEG-4 compression. The optimal QMD file gives reconstructed motion of the best quality that is visually indistinguishable from the original motion. In addition, the QMD file representation leads to vast reduction in the number of CPU cycles needed to decode the motion data from the compressed version, as opposed to MPEG-4 based compression, which requires a computationally expensive decoding procedure.

Decompression in the QMD file format is equivalent to performing a table lookup operation using the index values. This is a straightforward operation, and requires no extra CPU cycles in the device which is decoding the data. On the other hand, MPEG-4 decompression consumes significantly more CPU cycles just to recover the data from the compressed file, by first arithmetically decoding the data and then recovering the floating point data from the quantized data. This makes QMD file formats, optimally created for the best quality of reconstructed motion, to be much more advantageous for low powered devices than MPEG-4 compressed motion files.

8. CONCLUSIONS AND FUTURE WORK

We have proposed and implemented a novel file format, termed the Quantized Motion Data (QMD) format, to store avatar motion data for animation. The proposed file format has advantages both in terms of improved compression ratio, and ease of reconstruction of the motion data via a lookup table included in the file, thus eliminating the need for explicit decompression. This leads to both, reduction in throughput requirements 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 large motions files, 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 an $O(n^3)$ algorithm to compute the optimum values of these parameters to obtain minimum reconstruction error for the reconstructed motion. We have also proposed a simple application layer protocol that can be used for efficient real time streaming of the motion data using the QMD file format.

The proposed QMD file format for motion data 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 the QMD file exhibits negligible motion distortion, and is in fact, controllable via the three aforementioned parameters.

A limitation of the proposed technique is that the optimal values of the QMD file 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 quantitative measure for the quality of motion. Finally, the intelligent use of the hierarchical structure of the motion yields good results for full body motions of the avatar; for small delicate motions such as movement of the finger, or for facial animation, the proposed technique offers considerable scope for future improvement.

9. ACKNOWLEDGEMENTS

The data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-019621.

10. REFERENCES

[1] Gutiérrez, M., Vexo, F. and Thalmann, D., Controlling Virtual Humans Using PDAs, *Proc 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, *Proc. 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, *Proc. 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, *Proc. Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, Washington DC, 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, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, New York, NY, Vol. 2, 2000.

[10] Capin T.K., Petajan, E. and Ostermann, J., Efficient Modeling Of Virtual Humans In MPEG-4, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, 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, *Proc. Fifth Symposium on Virtual Reality Modeling Languages (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 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. First Intl. 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 (Eds.), *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. and Preteux, F., Advanced Virtual Humanoid Animation Framework Based On The MPEG-4 SNHC Standard, *Proc. EUROIMAGE International Conf. 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. Simulation and Visualization*, Magdeburg, Germany, pp. 349-361, 2003.