Model-Based Power Aware Compression Algorithms for MPEG-4 Virtual Human Animation in Mobile Environments

Siddhartha Chattopadhyay, Suchendra M. Bhandarkar, *Member, IEEE*, and Kang Li

***Abstract*—MPEG-4 body animation parameters (BAP) 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: 1) MPEG-4 compressed BAP data entails a significant number of CPU cycles, hence significant, unacceptable power consumption, for the purpose of decompression; 2) the lossy MPEG-4 technique of frame dropping to reduce network throughput during streaming leads to unacceptable animation degradation; and 3) lossy MPEG-4 compression does not exploit structural information in the virtual human 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 virtual human 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.**

***Index Terms*—Animation, mobile communication, multimedia communication, virtual reality.**

I. 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 such as laptop computers in battery mode, pocket PCs and PDAs [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]–[6], [24]. In order to standardize avatar animation, MPEG-4 has proposed H-Anim standards to represent avatars [7], [8], [16]. An avatar is animated using a stream of body animation parameters (BAPs) encoded for low-bitrate transmission [11], using the MPEG-4 compression pipeline (Fig. 1), in dedicated interactive communications and broadcast environments [9], [17], [18], [24]. The BAPs control the various independent degrees of freedom (DOF) in the skeletal avatar model to produce an animation of the body parts [10].

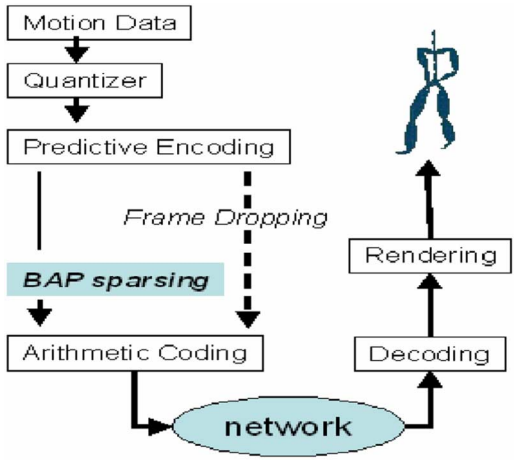


Fig. 1. Standard compression and decompression pipeline for MPEG-4 BAPs, enhanced with a novel alternative stage termed as *BAP sparsing*.

The two major issues in MPEG-4 BAP-based animation in mobile devices are 1) limited bandwidth available for streaming BAP data and 2) limited power available to process the animation data. Network overload, while streaming, can be reduced by using smart techniques such as dead reckoning [24], and BAP quantization and grouping [11]. Although these techniques reduce the network load significantly, decompression of the data at the client end entails extra CPU cycles. To reduce the power consumption resulting from the CPU cycles needed for decompression, raw data may be used. However, the direct use of raw data is undesirable as the network may become too overloaded. Hence, it is desirable to have a compression method which reduces the network load significantly, and also requires minimum computation, hence power consumption, at the client side to reconstruct the motion data from the compressed data stream. In addition, the hierarchical structure of the skeletal avatar model needs to be judiciously exploited in the case of lossy compression such that any undesired reduction in motion animation quality is minimized.

In this paper, we propose and combine two novel compression algorithms for MPEG-4 BAP data which 1) intelligently

Manuscript received October 16, 2005; revised March 22, 2006. The associate editor coordinating the review of this paper and approving it for publication was Dr. Jin Li.

The authors with the University of Georgia, Athens, GA 30602 USA (e-mail: siddh@cs.uga.edu; suchi@cs.uga.edu; kangli@cs.uga.edu).

Color versions of Figs. 1 and 5 are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2006.886326

exploit the structural hierarchy of the virtual human model to achieve efficient compression which, though lossy, results in reconstructed motion of good quality; 2) use indexing techniques for compression of BAP data, resulting in significant reduction in power consumption required for decompression; and 3) provide quality control parameters for tighter control of the reconstructed motion quality and compression ratio. One of our proposed compression algorithms termed as BAP-Sparsing [23], creates a sparsed representation of the original BAP data. This results in improved compression of the BAP data upon using the MPEG-4 compression pipeline (Fig. 1). The other proposed algorithm, BAP-Indexing [22], creates byte-size indices for representation of the BAP data. BAP-Indexing 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 under similar conditions 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. The resulting hybrid algorithm yields significantly better compression ratio compared to those obtained using the MPEG-4 standard, and requires significantly less client-side power measured in terms of CPU cycles and energy (measured in mJoules) needed to receive and decompress the data. Standard MPEG-4 techniques such as grouping can be used atop the proposed technique to further reduce the bit rate, as can be done in the case of MPEG-4 based BAP compression.

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 efficacy of this format by exploiting the three-dimensional (3-D) scene structure. Our method uses quantization to achieve data compression in a manner somewhat similar to the above paper, but incorporates 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 impostors to improve the display rate of animated characters by acting solely on the geometric and rendering information.

The known techniques mentioned above do not describe any direct impact on the 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 upon decompression of the compressed motion data. The algorithm proposed in this paper not only allows low-bitrate encoding of motion data, but is also suitable for data reception and data reconstruction on power-constrained devices. Henceforth, we will term the proposed algorithm as *Sparse-indexing* in the remainder of the paper.

II. Matrix Representation of BAP Data

The BAPs are represented by an $n \times m$ dimensional matrix $\mathbf{X}$, where n is a multiple of the video sampling rate or frame rate expressed as frames per second (fps) and m is the number of DOF for the avatar (the maximum value of $m = 296$ as defined in the 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.

As a first step in the compression process, the matrix $\mathbf{X}$ is alternatively represented as a difference matrix, $\mathbf{d}_{\mathrm{n-1 \times m}}$, 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 $\mathbf{X}$

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

The difference matrix $\mathbf{d}$ is subsequently termed the motion matrix. In the following two sections, we describe the two techniques, *BAP-Indexing* and *BAP-Sparsing*, in detail.

III. BAP-Indexing: Indexing of BAP Data

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 $\mathbf{d}$, 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 bits than that the corresponding floating-point number.

A. Indexing Motion Matrix $\boldsymbol{d}$

In order to ensure efficient indexing, we have used the standard equal frequency distribution technique to uniformly assign the $n \cdot m$ numbers in $\mathbf{d}$ to buckets numbered from 0 to 255. This is done as follows.

Step 1) The floating-point numbers in matrix $\mathbf{d}$ are collected into a single 1-D array $\mathbf{A}$ of size $n \cdot m$. The array $\mathbf{A}$ is sorted in ascending order. All the numbers in $\mathbf{A}$ 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. The numbers are rounded to represent integers in the range $[A_{\min} \cdot M, A_{\max} \cdot M]$.

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

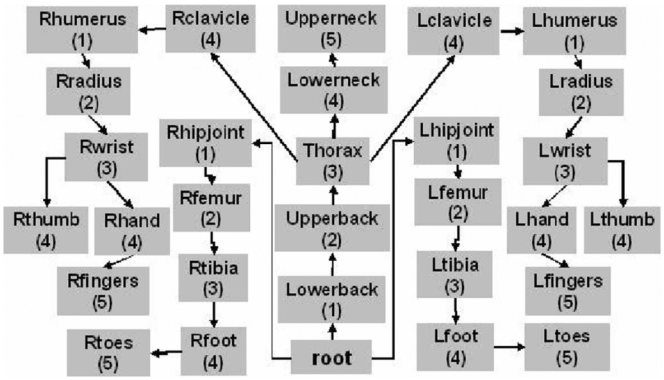


Fig. 2. Hierarchical structure of a human skeletal model consisting of 31 nodes, with a total of 62 DOF of motion (rotational and translational). For convenience, the root node is drawn at the bottom.

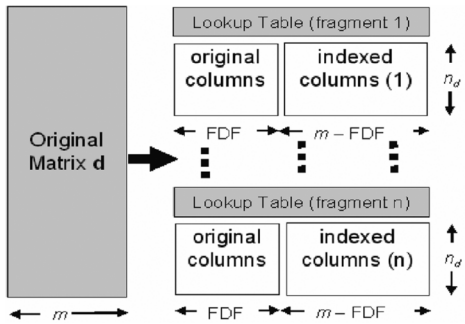


Fig. 3. Motion matrix $\mathbf{d}$ is decomposed into equal fragments of length n_d each. The first **FDF** rows are untouched; the next m—**FDF** columns are indexed. Each fragment is indexed separately, with its own lookup table.

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 $\mathbf{d}$ is contained in exactly one of the 256 buckets. An index matrix $\mathbf{d}_{\text{index}}$ is used to store the bucket number for the corresponding entry in the matrix $\mathbf{d}$.

B. Lookup Table for $\mathbf{d_{index}}$

The creation of an appropriate lookup table for the recovery of the original motion data matrix $\mathbf{d}$ from the index matrix $\mathbf{d_{index}}$ is critical, since recovery of the original data after discretization invariably results in motion distortion. A straightforward method to recover the number associated with a bucket is to compute the arithmetic mean of all the floating-point numbers assigned to the bucket. However, this invariably leads to poor approximation of the original motion matrix $\mathbf{d}$. We have intelligently exploited the hierarchical structure of the skeletal avatar model to construct the lookup table $\mathbf{T_{lookup}}$ as follows.

Step 1) The avatar is represented by a hierarchical skeletal model. For each m-dimensional pose vector (or row in $\mathbf{d}$), each dimension, or column in $\mathbf{d}$, is assigned a level l_i (Fig. 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.

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 jth lookup value in lookup table $\mathbf{T_{lookup}}$ is given by

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

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 $\mathrm{level} = 1$ affect the displacements in the body the most. Hence, emphasizing the numbers within a bucket with $\mathrm{level} = 1$ leads to better approximation of the motion data. As $\eta \to \infty$, all the weighting terms in (1) tend to zero, except for the terms with $\mathrm{level} = 1$. Hence, when computing the weighted sum of the numbers in a bucket, we consider only those numbers with $\mathrm{level} = 1$ (*selective averaging*), and compute a simple mean of these numbers. If none of the entries in a bucket have $\mathrm{level} = 1$, we use the next smallest level to compute the weighted sum.

C. Motion Matrix Decomposition for Long Motion Sequences

There is an inherent tradeoff between the size of the motion data that is indexed and the quality of the reconstructed motion. The longer the motion sequence, the lower the reconstructed motion quality after indexing. To overcome this problem, the original difference matrix $\mathbf{d}$ is fragmented into a succession of smaller matrices $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_{\mathrm{L}}$. Each fragmented motion matrix has n_d rows. Each of the fragmented matrices of size $n_d \times m$ in turn is discretized using the above method resulting in separate lookup tables, $\mathbf{T^1_{lookup}}, \ldots, \mathbf{T^L_{lookup}}$. This simple technique yields a good approximation to the motion data of any arbitrary duration.

D. Combining the Original and Indexed Data

It is often desirable to preserve the original data contained within some of the columns of the original motion matrix $\mathbf{d}$, and index the remainder of the columns. This is especially true for the first six columns of $\mathbf{d}$, 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. A combination of the original and indexed columns can be achieved by not indexing the first fixed DOF (**FDF**) columns, and indexing the remainder of the m – **FDF** columns. Fig. 3 depicts the indexing of the original motion matrix $\mathbf{d}$.

IV. BAP-Sparsing: Sparse Representation of Indexed Matrix

Once the difference matrix $\mathbf{d}$ is indexed, we now use the second algorithm, *BAP-Sparsing*, to eliminate several indices

```
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 = d_i, k
      else
         B_i, k = 0

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

Fig. 4. Algorithm for *BAP Sparsing*, for the difference values in matrix $\mathbf{d}_{\mathrm{nxm}}$. $\mathbf{G}_{\mathrm{i,k}} = \mathbf{d}_{\mathrm{i,k}} - \mathbf{d}_{\mathrm{i-1,k}}$. $\mathbf{B}$ is a temporary matrix. The constant m_k is the mean across all rows of column k and $\mathbf{S}_{\mathrm{k}}$ is the standard deviation of column k.

from the index matrix. 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.

A. Implementation of Sparsing Algorithm

To implement BAP-Sparsing, for each column i of the motion matrix $\mathbf{d}$, we consider each floating-point number corresponding to the BAP index, and drop (reduce to zero) consecutive indices along column i, and accumulate the floating-point values corresponding to the dropped indices until the cumulative value exceeds a predefined *threshold* $\mathbf{T_i}$. 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. This process is repeated until all the rows in the column are either reduced to zero or replaced by a corresponding index for the cumulative value (Fig. 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, $|\mathbf{S_k} - \mathbf{m_k}|/s_k$ (see Fig. 4) is used so that the value of the threshold $\mathbf{T_i}$ can be determined irrespective of the mean $(\mathbf{m}_k)$ and spread $(\mathbf{S}_k)$ of the data in the columns of $\mathbf{d}$. 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.

B. Determination of the Threshold T_i

We hypothesize that the threshold value $\mathbf{T_i}$ for a particular BAP parameter should depend on the position of the corresponding body joint in the hierarchical human skeletal model. We have represented the hierarchical significance of the various joints in the human body by a level number, as shown in the parentheses in Fig. 2. The threshold $\mathbf{T_i}$ for a joint i, represented by column i in matrix $\mathbf{d}_{\mathrm{n\times m}}$, 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 prediction 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 $\mathbf{d}$, with level L_i, the corresponding threshold $\mathbf{T_i}$ is given by

$$\mathbf{T_i} = KL_i \tag{2}$$

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

V. Effect of QCP on Compression Ratio and Animation Quality

The two algorithms, *BAP-Indexing* and *BAP-Sparsing* when combined, as mentioned above, have four associated QCPs. The bounds for the QCPs are as follows: $1 \leq \mathbf{FDF} \leq m$ (where m = number of columns in motion matrix $\mathbf{d}$), $8 \leq b \leq 32$ (assuming byte boundary for the indices), $\mathrm{n_d^{min}} \leq n_d \leq n$ and $0 < K < \infty$.

A. Compressed File Size as a Function of $\mathbf{FDF}, b, n_d$ and K

In practice, the BAPs in motion matrix $\mathbf{d}$ are represented by floating-point numbers. Assuming that floating-point numbers take 4 bytes each, the original raw file size for $\mathbf{d}_{\mathrm{nxm}}$ is $4 \cdot m \cdot n$. We first compute the file size assuming no sparsing, and then introduce sparsing to compute the file size as a function of all four QCPs ($\mathbf{FDF}, n_d, b$ and K).

1) File Size Assuming no Sparsing ($K = 0$): Assuming no sparsing (i.e., sparsing parameter $K = 0$), the resulting file size is obtained simply by expressing the number of bytes as a function of the three indexing parameters $\mathbf{FDF}, n_d$ and b. 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 file for each segment (Fig. 3) takes $4 \cdot 2^b$ 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:

$$\mathrm{seg\_head} = 4 \cdot 2^b + c_1 \tag{3}$$

For the actual motion data, the first **FDF** columns ($0 \leq \mathbf{FDF} \leq m$) are floating-point numbers taken directly from the motion matrix $\mathbf{d}$, and the next m – **FDF** columns are indices for the lookup table, where 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

$$\mathrm{seg\_mot} = n_d \cdot (4 \cdot \mathbf{FDF} + \lceil b/8 \rceil \cdot (m - \mathbf{FDF})). \tag{4}$$

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 obtained by combining (3) and (4)

$$\mathbf{T}(\mathbf{FDF}, b, n_d) = \lceil n/n_d \rceil \cdot (\mathrm{seg\_head} + \mathrm{seg\_mot}) + 4m. \tag{5}$$

The minimum file size $\mathbf{T_{min}}$ with sparsing parameter $K = 0$ is obtained by assigning the values $\mathbf{FDF} = 0, n_d = n$ and $b = 8$ (assuming a byte boundary for the indexed data) in (5). The maximum file size, $\mathbf{T}_{\max}$, should not exceed the original file size $4 \cdot m \cdot n$. For the parameter values mentioned above, the minimum file size $\mathbf{T}_{\min}$ (including the lookup table and initial pose vector) is given by

$$\mathbf{T}_{\min} = (n-1)m + 4m + c_2 \tag{6}$$

where c_2 is the number of bytes needed to store 256 buckets, each containing a 4-byte floating-point number; i.e., $c_2 = 1024$.

Since the motion quality decreases with increase in compression ratio, it is desirable to obtain the maximum file size permitted by the network as a fraction of the original file size. 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:

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

where *f* (henceforth termed as the *minimum-compression-ratio*) is the fraction of the maximum file size of the original motion data. The parameter values that satisfy (7) can be termed as *valid* points in the parameter space.

2) File Size With Sparsing and Indexing $(K > 0)$*:* In order to obtain a *sparsed-indexed* representation of the motion matrix $\mathbf{d}$, it is essential to determine a suitable value for the sparsing parameter K such that the resulting motion quality is not too distorted. A basic heuristic is to maintain the normalized threshold value $\mathbf{T}_i \leq 1$ for each column (joint degree of freedom). This is possible by ensuring that $K \leq 0.2$, since the maximum level value can be 5. Having assigned K an empirically selected value less than 0.2, say $K = 0.1$, the degree of sparsing in the resulting indexed matrix depends on the type of motion. We will present experimental results obtained for $K = 0.1$ for various motion examples to demonstrate how sparsing helps to reduce the file size in Section VI. The total file size is now given by $\mathbf{T}(\mathbf{FDF}, b, n_d, K)$, which is a certain function of $\mathbf{T}(\mathbf{FDF}, b, n_d)$.

B. Quality of Reconstructed Motion versus $\mathbf{FDF}, b, n_d$ *and* K

To quantify the reconstructed motion quality, we use the metric displacement error per frame (**DEF**). Let $\mathbf{C}_{nx3m}$ be the original coordinates of the joints of the human model, and $\mathbf{C\prime}_{nx3m}$ be the coordinates of the joints of the human model after reconstruction of the motion from the indexed matrix. We define the error function, $\Delta : \mathrm{I} \rightarrow \mathrm{R}$ as follows:

$$\Delta(\mathrm{i}) = \sum_{j=1}^{n} \|P_{ji} - P'_{ji}\|, \quad i = 1, 2, \ldots, m \tag{8}$$

where $P_{ji} = (C_{j,3i-2}, C_{j,3i-1}, C_{j,3i})$ and $P'_{ji} = (C'_{j,3i-2}, C'_{j,3i-1}, C'_{j,3i}), i = 1, 2, \ldots, m$, and $|\cdot|$ 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.

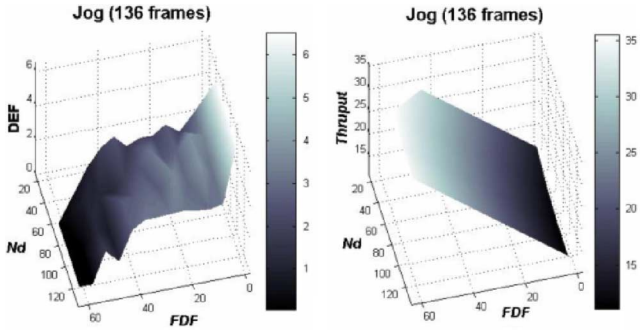


Fig. 5. Left: Displacement error per frame (**DEF**) for $30 \leq n_d \leq 120$ rows and $1 \leq$ **FDF** ≤ 62 columns. Right: Obtained throughput for $30 \leq n_d \leq 120$ rows and $1 \leq$ **FDF** ≤ 62 columns. $(b = 8, K = 0.1)$.

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

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

Although the **DEF** value does not represent the motion quality in an absolute sense, comparing the **DEF** values for various values of the QCPs provides an adequate measure of relative motion quality.

C. Determining Optimal Values of $\mathbf{FDF}, b, n_d$ *and* K

In order to compress the BAP data matrix $\mathbf{d}$ in practice, it is essential to obtain the values for the four QCPs ($\mathbf{FDF}, n_d, b$ and K). Recall that the value of K is fixed at 0.1 as described in subsection 5.1. Empirical studies reveal that for $b > 8$, the size of the file explodes exponentially, thus violating the upper bound in (7) 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 value is represented by a single byte. The surface plots of $\mathbf{T}(\mathbf{FDF}, b, n_d, K)$ and $\mathbf{DEF}(\mathbf{FDF}, b, n_d, K)$, for a *jogging* motion example, for the various possible values of $\mathbf{FDF}$ and n_d, while fixing the other parameters as $b = 8$ and $K = 0.1$, are presented in Fig. 5. As can be observed, the lower the **DEF**, the better the motion quality. Hence, the set of parameters amongst the valid parameters which minimize both $\mathbf{DEF}(\mathbf{FDF}, b, n_d, K)$, and $\mathbf{T}(\mathbf{FDF}, b, n_d, K)$, are the optimal quality control parameters (QCPs). 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:

$$\mathbf{M}(\mathbf{FDF}, b, n_d, K) = \mathbf{T}(\mathbf{FDF}, b, n_d, K) \cdot \mathbf{DEF}(\mathbf{FDF}, b, n_d, K). \tag{10}$$

A simple exhaustive search reveals that $\mathbf{FDF} = 41$ and $n_d = 60$ (with $b = 8$ and $K = 0.1$ as fixed values) minimizes the figure of merit function $\mathbf{M}$ for the *jogging* example. Although the indexing method is inherently lossy, the visual degradation in the reconstructed motion is imperceptible. The functions $\mathbf{T}$ and **DEF** are inversely proportional to each other. A relationship plot can be used to select the desired throughput requirement $\mathbf{T}$ based on an acceptable value of **DEF**.

TABLE I
COMPARISON OF COMPRESSION RATIO OBTAINED VIA SPARSE-INDEXING AND POWER CONSUMPTION IN MJ BY THE WNIC 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 SPARSE-INDEXING. COLUMNS [6] AND [7] GIVE THE ENERGY CONSUMPTION IN MILLIJOULES FOR THE RECEPTION OF STREAMING MPEG-4 AND SPARSE-INDEXING DATA FOR A 128 KBPS NETWORK. COLUMN [8] GIVES THE RATIO OF POWER CONSUMPTION BY THE WNIC USING THE SPARSE-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

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%*

The QCPs can also depend on the display size of the screen on which the animation is being rendered. For small QVGA resolution displays (320×240), the amount of error allowed for the joints of the avatar in each frame can be higher, compared to that in the case of higher resolutions such as 640×480. This screen size information can be used to *weigh* the **DEF** value appropriately (perhaps empirically) in (10) to allow greater quantization (smaller **FDF** and larger n_d), and hence more compact compressed file sizes in the case of lower-resolution displays.

VI. EXPERIMENTAL RESULTS FOR DIFFERENT MOTION TYPES

In order to test the usefulness of the proposed *Sparse-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. The motion examples have been created using motion capture data from real human actors. The motion data files are obtained from the website mocap.cs.cmu.edu.

A. Compatibility of Sparse-Indexing With Existing BAP Encoding Technologies

Sparse-indexing is partially compatible with existing MPEG-4 quantization and grouping techniques [11]. The quantization portion of *Sparse-indexing* has several advantages over MPEG-4 quantization. The former uses a byte to represent each of the derived indices, whereas MPEG-4 quantization may require indices up to 4 bytes long, depending on the precision [11]. MPEG-4 spatial grouping groups together joint types, in order to encode only certain portions of the joints. A similar approach can be easily incorporated in *Sparse-indexing*; i.e., for the 62 DOF avatar, a 62-bit mask (rounded up to 8 bytes) can be used as a mask to tell the decoder which joints are encoded in the BAP data. Another popular technique, called *dead-reckoning* [24] can be used easily in conjunction with *Sparse-indexing*. In *joint-level dead reckoning*, joint angles of the avatar are the only information that is required. Since *Sparse-indexing* is another way of encoding the joint angles, dead-reckoning techniques require trivial modifications to be made compatible with *Sparse-indexing*.

In Section VI-B–D, various comparisons are made with MPEG-4. We have not used grouping for the MPEG-4 encoded file; i.e., the entire 62 DOF of the BAP data are completely encoded in each row/frame. Quantization, followed by arithmetic coding, is used for MPEG-4 encoding. To ensure a fair comparison, the *Sparse-indexing* does not group the data either.

B. Minimum BAP Data File Size Obtained by Sparse-Indexing Compared to MPEG-4

Table I gives the minimum throughput $\mathbf{T}$ obtainable using *Sparse-indexing*, and the corresponding displacement error **DEF**. As expected, the throughput $\mathbf{T}$ is significantly less than that obtained by MPEG-4-based compression.

C. Power Consumption by Sparse-Indexing Compared to MPEG-4

The process of data reception, decoding and rendering for BAP-based animation can be broken into four steps.

- Step 1) The wireless network interface card (WNIC) periodically receives the encoded BAP data and stores it in the WNIC buffer
- Step 2) The CPU periodically transfers the data from the WNIC buffer to MEMORY.
- Step 3) The CPU reads the encoded data from MEMORY, and stores the reconstructed floating-point numbers corresponding to the various joint angles in MEMORY.
- Step 4) The graphics processing unit (GPU) renders avatar animation using reconstructed data.

Power consumption in each of the above four high level steps can be attributed to the CPU (we will ignore the power consumption by the MEMORY) and also by the three main buses; i.e., the buses connecting CPU and MEMORY ($\mathrm{BUS_{cm}}$),

MEMORY and WNIC ($\mathrm{BUS_{mw}}$), and MEMORY and GPU ($\mathrm{BUS_{mg}}$). Power is consumed for each bus clock cycle, and the number of bus clock cycles is directly proportional to the number of bytes transferred over the bus. Hence, in order to estimate power consumption by the buses, it is adequate to quantify the amount of data transferred.

The above four steps consume power as follows. In Step 1, power is consumed by the WNIC while receiving data from the server. In Step 2, power is consumed by the $\mathrm{BUS_{mw}}$ while transferring data from the WNIC buffer to the MEMORY (the power consumed by the CPU during this data transfer is considered relatively small and hence ignored). In Step 3, power is consumed by both the CPU while decoding the data, and the $\mathrm{BUS_{cm}}$ while transferring data between the CPU and the MEMORY. In Step 4, power is consumed data by both $\mathrm{BUS_{mg}}$ (while transferring data from MEMORY to GPU), and the GPU while rendering the avatar for each frame.

Power consumption by $\mathrm{BUS_{mg}}$ is the same for both the MPEG-4 coded BAP data (henceforth termed as M-data) and *Sparse-indexing* coded BAP data (henceforth termed as SI-data), since the amount of data upon decompression is the same in both cases. Decoding M-data requires a substantial number of CPU cycles, compared to decoding of SI-data, since decoding of M-data requires a pipeline of processes such as inverse-arithmetic coding, and inverse quantization, whereas decoding of SI-data entails only an access to the lookup table. Hence, the power consumption by $\mathrm{BUS_{cm}}$ is greater in the case of decoding of M-data compared to SI-data. Finally, the $\mathrm{BUS_{mw}}$ transfers less data in the case of SI-data compared to M-data. This is evident from the results presented in Section VI-B, which shows that the amount of SI-data generated is much less compared to the amount of M-data generated, in comparable settings.

Power consumption by the GPU in order to render the frames is same for both M-data and SI-data, since after decompression, the data required to animate the actual avatar model, are the same for both M-data and SI-data. The GPU is a major source of power consumption, along with data decoding and the bus data transfer. The significance of reducing power consumption for data reception, data transfer and data decoding, increases when simpler graphics models are used, such as in mobile devices [25].

In order to ensure a fair comparison of power consumption during decompression of M-data and SI-data, we encode the M-data and SI-data to yield identical data file sizes. To achieve this, we first generate the M-data using MPEG-4 quantization and arithmetic coding (without grouping). We then compute the *minimum-compression-ratio* f of the obtained M-data file size to the original motion file size, and use this *minimum-compression-ratio* f in (7) 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 constraint in (10) (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 II. As evident from the table, decoding of the M-data requires a significant number of CPU cycles. On the other hand, a simple table lookup is required for decoding of the SI-data to obtain the actual data for joint angles, which, logically, entails fewer CPU cycles.

TABLE II
COMPARISON OF MPEG-4 BAP COMPRESSION VERSUS *SPARSE-INDEXING* ($b = 8$ AND $K = 0.1$). THE FILE SIZES ARE IN KILOBYTES (KB). 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

Motion Type	Org size	MPEG-4	G-cycles decoding MPEG-4	Comp. Ratio f	Sparse-indexing	DEF
Jog	33	28	0.65	85%	18.99	1.49
Run Leap	61	50	0.93	82%	28.55	0.45
Stride	72	59	1.23	82%	29.67	1.97
Walk	83	73	1.34	88%	45.91	0.17
Jump	85	72	1.35	85%	41.95	0.25
Basketball	96	84	1.44	88%	53.84	0.42
Forward jump	101	86	1.56	86%	55.71	0.34
Dance	106	92	1.68	87%	58.93	14.09
Play violin	195	162	2.89	83%	104.81	1.23
Sword play	545	465	8.49	85%	315.58	2.49

Finally, an analytical comparison of power consumption for receiving M-data and SI-data at the WNIC is made. For a motion of time duration T, data size S and given available bandwidth B, the energy used by the WNIC is given by

$$E_{\mathrm{network}} = E_R S/B + E_S \cdot (T - S/B) \tag{11}$$

where E_{R} is the energy used by the WNIC during data reception and E_{S} is the energy used by the WNIC when it is sleeping and not receiving data. Using (11), we compute the WNIC energy utilization for reception of M-data and SI-data (Table I). We have used energy usage data from [20], [21] to obtain the energy usage for data reception in millijoules. Again, reception of SI-data resulted in significantly less energy consumption by the WNIC when compared to reception of M-data.

We conclude that BAP data compressed using *Sparse-indexing* leads to less power consumption for decompression, and much smaller compressed file sizes, compared to MPEG-4 compressed BAP data.

VII. CONCLUSIONS AND FUTURE WORK

We have proposed and implemented a novel *Sparse-indexing* technique to compress the BAP data used for MPEG-4 compliant character animation. The proposed *Sparse-indexing* method leads to reduction in both, the throughput requirement for networked applications requiring motion data exchange, and client power consumption for data reception and data decompression. The resulting quality of the reconstructed motion is improved considerably by intelligent exploitation of the hierarchical structure of the skeletal avatar model during the process of creation of optimal lookup tables for reconstruction of the quantized motion. The quality and throughput requirements of the motion data are controlled via four quality control parameters. We have proposed a simple systematic search procedure to obtain the optimum combination of these parameters depending on the required compression ratio.

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.

REFERENCES

[1] M. Gutiérrez, F. Vexo, and D. Thalmann, "Controlling virtual humans using PDAs," in *Proc. 9th Int. Conf. Multimedia Modelling (MMM'03)*, Taiwan, R.O.C., 2003.

[2] T. K. Capin, I. S. Pandzic, and N. Magnenat-Thalmann, *Avatars in Networked Virtual Environments*. New York: Wiley, 1999.

[3] C. Joslin, T. Molet, and N. M. Thalmann, "Advanced real-time collaboration over the internet," in *Proc. ACM Symp. Virtual Reality Software and Technology*, Seoul, Korea, 2000, pp. 25–32.

[4] M. Cavazza, O. Martin, F. Charles, S. J. Mead, and X. Marichal, "Interacting with virtual agents in mixed reality interactive storytelling," in *Proc. Intelligent Virtual Agents*, Kloster Irsee, Germany, 2003.

[5] L. Vacchetti, V. Lepetit, G. Papagiannakis, M. Ponder, and P. Fu, "Stable real-time interaction between virtual humans and real scenes," in *Proc. 3DIM 2003*, Banff, AL, Canada, 2003.

[6] I. Barakonyi, T. Psik, and D. Schmalstieg, "Agents that talk and hit back: Animated agents in augmented reality," in *Proc. 3rd IEEE and ACM Int. Symp. Mixed and Augmented Reality (ISMAR)*, Washington, DC, Nov. 2–5, 2004, pp. 141–150.

[7] *Coding of Audio-Visual Objects, Systems*, ISO/IEC 14496-1:1999, Dec. 1999, Amendment 1.

[8] *Coding of Audio-Visual Objects, Visual*, ISO/IEC 14496-2:1999, Dec. 1999, Amendment 1.

[9] T. K. Capin, E. Petajan, and J. Ostermann, "Very low bitrate coding of virtual human animation in MPEG-4," in *Proc. IEEE Int. Conf. Multimedia and Expo (ICME)*, New York, NY, 2000, vol. 2.

[10] T. K. Capin, E. Petajan, and J. Ostermann, "Efficient modeling of virtual humans in MPEG-4," in *Proc. ICME'2000*, New York, NY, Jul. 2000.

[11] T. K. Capin and D. Thalmann, "Controlling and efficient coding of MPEG-4 compliant avatars," in *Proc. IWSNHC3DI'99*, Santorini, Greece, 1999.

[12] M. Endo, T. Yasuda, and S. Yokoi, "A distributed multi-user virtual space system," *IEEE Comput. Graph. Applic.*, vol. 23, no. 1, pp. 50–57, 2003.

[13] T. Hijiri, K. Nishitani, T. Cornish, T. Naka, and S. Asahara, "A spatial hierarchical compression method for 3D streaming animation," in *Proc. 5th Symp. Virtual Reality Modeling Language (Web3D-VRML)*, Monterey, CA, 2000, pp. 95–101.

[14] T. Giacomo, C. Joslin, S. Garchery, and N. Magnenat-Thalmann, "Adaptation of facial and body animation for MPEG-based architectures," in *Proc. Int. Conf. Cyberworlds*, 2003, pp. 221–.

[15] A. Aubel, R. Boulic, and D. Thalmann, "Animated impostors for real-time display of numerous virtual humans," in *Proc. 1st Int. Conf. Virtual Worlds (VW-98)* Transl.:Springer, Berlin, Germany, 1998, vol. 1434, pp. 14–28.

[16] M. Preda, A. Salomie, F. Preteux, and G. Lafruit, "Virtual character definition and animation within the MPEG-4 standard," in *3D Modeling and Animation: Synthesis and Analysis Techniques for the Human Body*, M. Strintzis and N. Sarris, Eds. Hershey, PA: IRM Press, 2004, ch. 2, pp. 27–69.

[17] M. Preda and F. Preteux, "MPEG-4 human virtual body animation," in *MPEG-4 Jump-Start*, M. Bourges-Sévenier, Ed. Upper Saddle River, NJ: Prentice-Hall, 2002, ch. 9.

[18] M. Preda and F. Preteux, "Advanced virtual humanoid animation framework based on the MPEG-4 SNHC standard," in *Proc. EUROIMAGE Int. Conf. Augmented, Virtual Enviroments and Three-Dimensional Imaging (ICAV3D'01)*, Mykonos, Greece, 2001, pp. 311–314.

[19] M. Kruppa and A. Krüger, "Concepts for a combined use of Personal Digital Assistants and large remote displays," in *Proc. Simulation and Visualization*, Magdeburg, Germany, 2003, pp. 349–361.

[20] M. Stemm, P. Gauthier, D. Harada, and R. H. Katz, "Reducing power consumption of network interfaces in hand-held devices," in *Proc. 3rd Int. Workshop on Mobile Multimedia Comm.*, Sep. 1996.

[21] J. M. H. Paul, "Mobile Multimedia Systems," Ph.D. dissertation, Univ. of Twente, Enschede, The Netherlands, Feb. 2000.

[22] S. Chattopadhyay, S. M. Bhandarkar, and K. Li, "Compression by indexing: An improvement over MPEG-4 body animation parameter compression," in *Proc. ACM Multimedia Computing and Networking (MMCN'06)*, San Jose, CA, Jan. 2006.

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

[24] T. K. Capin, I. S. Pandzic, N. M. Thalmann, and D. Thalmann, "A dead-reckoning algorithm for virtual human figures," in *Proc. VRAIS'97*, pp. 161–169.

[25] S. Chandra, "Wireless network interface energy consumption implications for popular streaming formats," *Multimedia Syst.*, vol. 9, pp. 185–201, 2003.

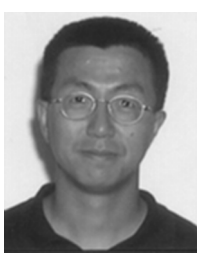
Siddhartha Chattopadhyay received the integrated five years M.S. degree in mathematics and scientific computing from the Indian Institute of Technology, Kanpur, in 2002. He is currently pursuing the Ph.D. degree at the University of Georgia, Athens.

His research interests are in content-based multimedia data encoding for resource constrained devices and computer animation techniques for power-aware devices.

Suchendra M. Bhandarkar (S'82–M'90) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, and the M.S. and Ph.D. degrees in computer engineering from Syracuse University, Syracuse, NY.

He is currently a Professor in the Department of Computer Science, University of Georgia, Athens, where he directs the Visual and Parallel Computing Laboratory (VPCL). He also serves as a Consultant to various organizations in the private and public sector, including government agencies such as NASA, the Department of Defense, the Department of Energy, and the Department of Health and Human Services. He is a coauthor of the book *Object Recognition from Range Images* (New York: Springer, 1992). His research interests include computer vision, pattern recognition, multimedia systems, artificial intelligence and parallel algorithms and architectures for computer vision and multimedia. He has over 100 published research articles in these areas, including over 50 articles in peer-reviewed archival journals.

Dr. Bhandarkar was a Syracuse University Fellow for the academic years 1986–1987 and 1987–1988. He is a member of the AAAI, ACM, and SPIE. He serves on the editorial boards of the *Computer Journal*, the *Journal of Machine Vision and Applications*, and the *Applied Intelligence Journal* and on the program committees of several international conferences and symposia. He serves on the Technical Committee for Pattern Analysis and Machine Intelligence and Multimedia Computing in the IEEE Computer Society and for Robot Vision in the IEEE Robotics and Automation Society. He is a member of the honor societies Phi Kappa Phi and Phi Beta Delta.



Kang Li received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 1995, and the Ph.D. degree in computer science and engineering from Oregon Graduate Institute of Science and Technology, Beaverton, in 2002

He is currently an Assistant Professor in the Computer Science Department, University of Georgia, Athens. His research interests are in the area of computer networks and high-performance anti-abuse systems.