Hindawi Publishing Corporation
International Journal of Digital Multimedia Broadcasting
Volume 2010, Article ID 924091, 11 pages
doi:10.1155/2010/924091

Research Article

A Compact Representation for 3D Animation Using Octrees and Affine Transformations

Youyou Wang and Guilherme N. DeSouza

Department of Electrical and Computer Engineering, University of Missouri, Columbia, MO 65211, USA

Correspondence should be addressed to Guilherme N. DeSouza, desouzag@missouri.edu

Received 2 May 2009; Revised 30 September 2009; Accepted 8 December 2009

Academic Editor: Georgios Triantafyllidis

This paper presents a new and compact 3D representation for nonrigid objects using the motion vectors between two consecutive frames. Our method relies on an Octree to recursively partition the object into smaller parts. Each part is then assigned a small number of motion parameters that can accurately represent that portion of the object. Finally, an adaptive thresholding, a singular value decomposition for dealing with singularities, and a quantization and arithmetic coding further enhance our proposed method by increasing the compression while maintaining very good signal-noise ratio. Compared to other methods that use tri-linear interpolation, Principle Component Analysis (PCA), or non-rigid partitioning (e.g., FAMC) our algorithm combines the best attributes in most of them. For example, it can be carried out on a frame-to-frame basis, rather than over long sequences, but it is also much easier to compute. In fact, we demonstrate a computation complexity of $\Theta(n^2)$ for our method, while some of these methods can reach complexities of $O(n^3)$ and worse. Finally, as the result section demonstrates, the proposed improvements do not sacrifice performance since our method has a better or at least very similar performance in terms of compression ratio and PSNR.

1. Introduction

As the technology for graphics processing advances, so does the details in the 3D models used for animation. So, despite these advances, when storing, transmitting, or rendering such models, the need for fast and compact representations is the same as it was a few years ago. In that sense, two categories of systems can be found in the literature: the time-independent methods and the time-dependent methods.

The first and most traditional category is the time-independent, and it was first introduced in [1]. In that case, a 3D object is represented using its geometric properties at that moment. That is, 3D points [2, 3], triangular meshes, surface normals, edge orientations [4–7], wavelet coefficients [8, 9], and other features of the object are analyzed within a single time instant, or frame. These methods have advantages in terms of the quality of the model, but they are of course not very compact. On the other hand, time-dependent methods exploit the temporal relationship of these same types of features in order to increase compression. In one of the first systems to be reported [10], but also in more recent ones [11–15], the basic idea is to encode the motion of the 3D object. That is, the difference between consecutive frames, rather than the properties of the object in the frames. In order to do that efficiently, the system must identify the parts of the object that are stationary from the parts that are moving, and describe only the latter.

This type of approach raises two major problems: (1) how to partition the two portions of the object—that is, moving and stationary portions; (2) how to describe the moving portions so they represent as perfectly as possible the non-rigid motion. Although much research has been done in this area [11–13, 16, 17], these approaches still suffer from the following: (1) inaccurate motion transformations [17], (2) the need for extra space to store the partitioning [12], (3) the need for prior information on the entire sequence [13–15], and so forth.

In this paper, we address the above problems by proposing a rigid partitioning of the 3D space combined with an affine transformation for motion capture. Some of the major

advantages of our method are its computational efficiency, the compactness of the motion, and the space representation.

In Section 2, we will talk about some of the related work in compression of animated sequences. Here, we distinguish animation from real 3D data in particular for the fact that animation can rely on feature correspondence between frames. Section 3 contains the details of our approach and in Section 4, we compare our method against other celebrated methods in the literature. We show the advantages of using our representation for compression, but we also point out where it can be improved.

2. Related Work

One of the first methods proposed for time-dependent 3D data compression can be found in [10]. From this paper, a new paradigm in time-dependent compression was established: represent successive frames by a small number of motion parameters and select a coding scheme to efficiently store the data. Even though many systems today offer completely different approaches to capture and parametrize this motion—from affine transformations [14, 15, 18], to principal components [13, 19, 20], and to wavelets that analyze the parametric coherence in the sequence [21, 22]—most time-dependent methods generally perform two basic steps: (1) partitioning of complex objects into smaller and simpler components; (2) description of the motion of these same components.

With regard to the partitioning method, we find systems using regular or rigid spatial partitioning, where vertices are divided according to their spatial location. One such example is the Octree [11, 16, 17, 23]. In this case, the space is recursively divided into 8 equal portions until some termination criteria stop the process.

On the other side of the coin, we find systems employing irregular partitioning. In [12], for example, the system employed the Iterative Closest Points algorithm (ICP) and assumed that the underlying motion between two consecutive frames followed a rigid transformation. The rigid transformation returned from the ICP was used to reconstruct the frame, while small and irregular portions of the object with small reconstruction error were clustered together to form a single rigid component. In [24], the clustering of the vertices was based on a method similar to a k-means, while the distance between clusters was defined as the Euclidean distance on the subspace defined by a principal component analysis (PCA). That means that the entire sequence had to be known beforehand in order to calculate the subspaces. The same can be said about other PCA-based methods [13, 19]—whether using irregular partitioning or not. Finally, in [20], several local coordinate frames were assigned to the object at the center of each cluster, and the vertices were assigned to clusters depending on their movements between consecutive frames. If the type of objects is restricted to, for example, the human body, the body parts can be clustered using their trajectories, as it was done in [25]. Actually, there is a third kind of systems where a spatial partitioning is not at all explicit. That is, in [26, 27], for example, vertices are grouped despite their spatial location, but rather based on their motion vectors.

After a partitioning is obtained, the next step is to find an efficient encoding for the motion. In that sense, some partitioning methods impose constraints on how the motion can be described. In other cases, however, the partitioning is generic enough that different motion descriptors can be used. In [11, 16, 17], all versions of the system employed a tri-linear interpolation, a regular partitioning (octree), and eight motion vectors attached to the corners of the cell. In other cases [12], the authors proposed an irregular partitioning with an affine transformation between clusters as the motion descriptor. Finally, as we mentioned earlier, PCA-based methods can achieve a good compression by storing only the principal components of the motion vectors, that is, a smaller dimension than the original one. That can be done both globally [13, 19] or locally [20, 24], but in either case, the entire sequence must be available for the calculation of the principal components. More recent approaches include the Principal Geodesic Analysis, a variant of the PCA method [28]; methods relying on prediction of the motion vectors, [26]; the replica predictor [27].

Another method relying on irregular partitioning is the recent Frame-Animated Mesh Compression (FAMC) [14, 15, 18], which was standardized within the MPEG as part of MPEG-4 AFX Amendment 2 [29]. In this case, the partitioning is irregular, but fixed. That is, the partitioning does not follow a rigid structure as in an octree, but it must be decided at the very first frame based on all frames in the sequence. While this allows for a more compact representation, it still requires prior knowledge of the entire sequence.

3. Proposed Approach

In many applications involving virtual reality and computer graphics, such as for human-robot interaction, real-time processing is a requirement that cannot be undermined. With that application in mind, we devised a fast and compact representation for 3D animation using rigid partitioning of the space and an affine transformation. As we will show in Section 4, our claim is that such combination can produce results comparable to or better than other methods in terms of compression, while still preserving a good signal-to-noise ratio.

3.1. Octree Structure. Our approach starts with the use of octrees for the partitioning of 3D objects. Octrees have been used in computer vision and computer graphics for many years [30]. This data structure has also been widely used in both time-independent methods, such as [2, 3], as well as time-dependent methods, such as in [11] and later improved in [16, 17, 23].

In our case, the partitioning using octree is similar to that in other time-dependent methods, however, the decision as to when partition and the termination criteria are different, making our method unique. That is, with octrees, the 3D space containing the object vertices is recursively divided into

8 subspaces, also known as the octants, cells, or cubes. In this paper, we will use the terms cube, cell, node, and octant interchangeably.

The partitioning starts with the application of an affine transformation and the calculation of an error measurement based on the motion vectors. If this error is too high, the cell is subdivided and the process repeats for each subcell. As for the termination criteria, we propose an adaptive thresholding of the reconstruction error followed by a singular value decomposition and quantization using arithmetic coding to further increase the compactness of the representation. All this process is simplified by rescaling (normalizing) all vertices to a size between [0, 1]—that is, the size of the root cube is always regarded as 1 unit.

3.2. Algorithm. Our algorithm consists of an encoding of the motion vector of the current frame with respect to the previous one. That is, the algorithm perform the following steps.

(1) First, it applies a tightly bounded cube around all vertices in the previous frame.

(2) Next, it calculates the affine transformation matrix between all vertices in the bounding cube and the corresponding vertices from the current frame.

(3) It checks for singularities of the affine and then it quantizes and dequantize the resulting affine matrix. This step is required in order to produce the reconstructed current frame and to calculate the error between the reconstructed and actual current frames.

(4) If the error in the previous step is too large, the algorithm partitions the bounding cube into eight smaller subcubes and the steps (5) and (7) above are repeated for each of the subcubes.

(5) Otherwise, it stores the quantized affine transformation as the motion vector for that cube.

The steps above are highlighted by the blue box in Figure 1(a).

Once a representation for the current frame is completed, the algorithm proceeds to the next frame. That is, it now uses the reconstructed current frame as the "previous" frame and the next frame as the "current" frame and steps are repeated until the last frame in the sequence is encoded. The idea is that only the positions of the vertices for the first frame are recorded and transmitted to the other side—in the case of 3D video streaming, for example, when frames are generated on one machine and rendered on another machine. After the first frame is transmitted, only motion vectors related to each cube of the octree are transmitted to the receiving end. In practice, in order to achieve better signal-to-noise ratios, intra frames could be inserted after an arbitrary number of frames to *reset* the error. However, in this paper we are interested in maximum compression only, and therefore, we will not offer any further discussion on how or when to insert intra frames.

The "dual" of the *encoding* algorithm described above is the *decoding* algorithm, and it is presented in Figure 1(b). Since this algorithm consists of the same (dual) parts of the steps of the encoder, we will leave to the reader to explore the details of the decoder.

3.3. Computation of the Affine Transformation. One of the main steps in our approach is the calculation of a single motion vector that will describe the movement of all the vertices in the cube with respect to two consecutive frames. Since the correspondence between vertices from two different frames is known, this motion vector can be approximated using an affine transformation A whose reconstruction error can be expressed as

$$E = \sum_{i=1}^{N} \left\| A * \vec{p}_i - \vec{q}_i \right\|^2, \tag{1}$$

where N is the total number of vertices in the cube, and $\vec{p}_i$ is a 4 by 1 homogeneous vector with the coordinate of vertex i in the previous frame. Similarly, $\vec{q}_i$ is the homogeneous coordinates of the corresponding vertex in the current frame. In other words, the affine transformation A is the actual motion vector between the vertices of a cube in the previous frame, and the corresponding vertices of the current frame.

Considering the entire structure of the octree, the total reconstruction error is the sum of all the errors at the leaf nodes of the tree. That is

$$\mathbf{E} = E_1 + E_2 + \cdots + E_M = \sum_{j=1}^{M} \left(\sum_{i=1}^{N_j} \left\| A_j * \overrightarrow{p_{d_{ji}}} - \overrightarrow{q_{d_{ji}}} \right\|^2 \right), \tag{2}$$

where M is the number of leaf nodes, N_j is the number of vertices in the jth leaf node, and d_{ji} is the index of the ith vertex in that same leaf node.

In vector form, the homogeneous coordinates of the points in the leaf node j, at the previous frame $f-1$, are given by

$$F_{j(f-1)} = \begin{bmatrix} p_{1d_{j1}} & p_{1d_{j2}} & \cdots & p_{1d_{jN_j}} \\ p_{2d_{j1}} & p_{2d_{j2}} & \cdots & p_{2d_{jN_j}} \\ p_{3d_{j1}} & p_{3d_{j2}} & \cdots & p_{3d_{jN_j}} \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \tag{3}$$

and the corresponding coordinates at the current frame f are given by

$$F_{jf} = \begin{bmatrix} q_{1d_{j1}} & q_{1d_{j2}} & \cdots & q_{1d_{jN_j}} \\ q_{2d_{j1}} & q_{2d_{j2}} & \cdots & q_{2d_{jN_j}} \\ q_{3d_{j1}} & q_{3d_{j2}} & \cdots & q_{3d_{jN_j}} \\ 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{4}$$

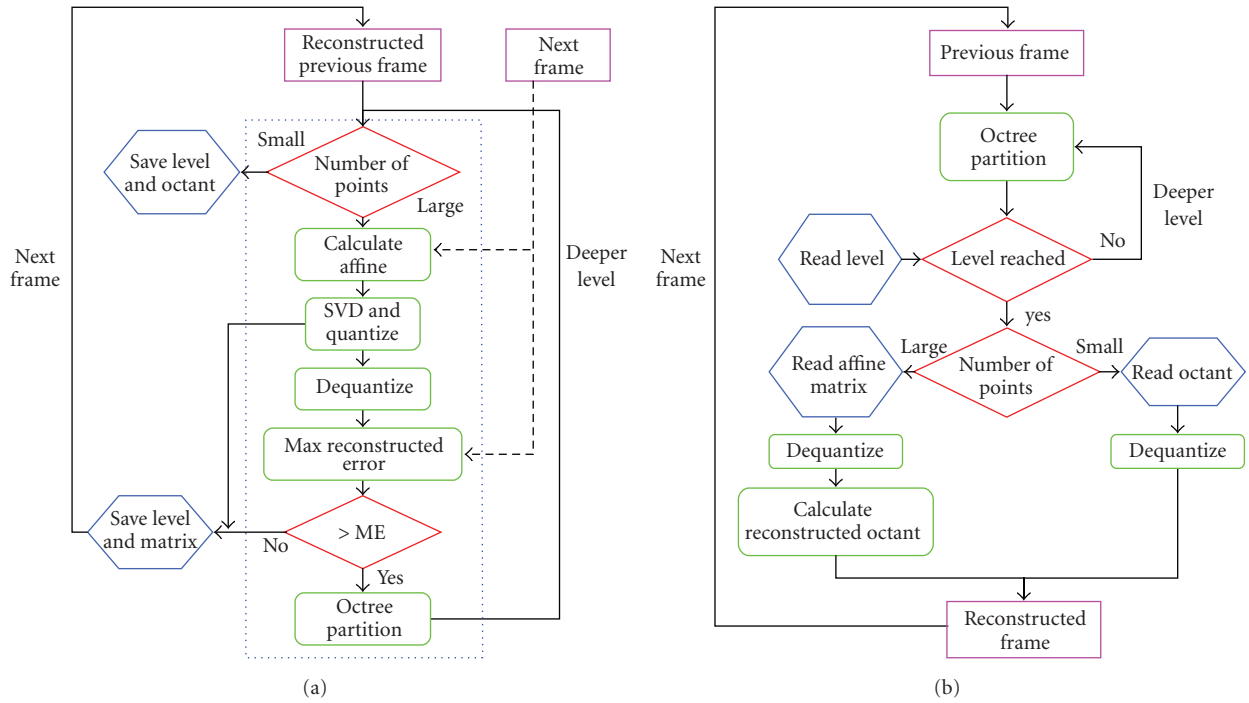


Figure 1: Flowchart of the algorithm: (a) Encoder, and (b) Decoder.

```
Algorithm(node)
   Affine(node)                    = 4 × n²
   Quantization(Affine)            = c_onst
   Reconstruction Error(node)      = 16 × n
   MaxError(vertices):             = 3 × n
   If Max Error less then Treshold = c_onst
      Stops
   Else
      Partition(node):             = 8 ∗ Algorithm(node/8)
   End
End
```

Algorithm 1: Complexity of the proposed algorithm.

The affine A_j that minimizes the error E_j, that is, minimizes $A * F_{j(f-1)} = F_{jf}$ in the least square sense is given by a right pseudoinverse. That is

$$A_j F_{j(f-1)} = F_{jf}, \tag{5}$$

$$\begin{aligned} A_j F_{j(f-1)} F_{j(f-1)}^T &= F_{jf} F_{j(f-1)}^T, \\ A_j &= F_{jf} F_{j(f-1)}^T \cdot \left(F_{j(f-1)} F_{j(f-1)}^T\right)^{-1}. \end{aligned} \tag{6}$$

The matrix A_j is a 4 by 4 matrix with [0 0 0 1] as the last row. Since each pair of corresponding points provides three constraints, to solve for the unknowns in the system of equations above N must be $\geqslant 4$. Also, since the transformation between $F_{j(f-1)}$ and F_{jf} is not a perfect transformation, the calculated A_j leads to a reconstruction error $|A_j F_{j1} - F_{j2}| > 0$. If N is smaller than 4, no affine is calculated and the position of the vertices in that cube is transmitted instead.

3.4. Quantization and Singular Nodes. Each element of the affine transformation matrix is stored using integers, which affects the precision, but increases the compactness of the representation. To compensate for this loss of precision, a frame f is encoded with respect to the reconstructed frame, rather than the actual frame $f - 1$. By doing so, the quantization error in the latter frame is corrected by the motion estimation for the current one. Therefore, quantization errors only affect the frame, but do not propagate throughout the whole sequence.

The quantized affine transformation matrix A' derived from the original affine transformation matrix A by

$$A' = \left\lfloor 2^k \left(\frac{A - a_{\min}}{a_{\max} - a_{\min}} \right) \right\rfloor, \tag{7}$$

where k is the quantization step. Also, in order to be able to compare our method with the method developed in [11], we set the same linear quantization method with a step of 16 bits. Ideally, $a_{\min}$ and $a_{\max}$ would be the minimum and maximum elements among all affine matrices. However, that would require the prior calculation of the motion vectors for the entire sequence. Instead, we use a predefined value for both $a_{\min}$ and $a_{\max}$. This arbitrary choice is possible because, as we explained earlier, we normalize the dimensions of the root cube to $i = 1 \cdots N_j$. That guarantees that the elements

Table 1: Properties of the benchmark sequences used for testing.

	Vertices	Triangles	Frames	Size (bytes)
Dance	7061	14118	190	16099080
Chicken	3030	5664	400	14544000
Cow	2904	5804	193	6725664
Snake	9179	18354	134	14759832
Ball	5552	11100	100	6662400

Table 2: Results of the Comparison for the "Chicken" sequence.

	Ours			
	Matrices	Size	Ratio	PSNR
Chicken (1%)	52694	709139	20.51 : 1	39.43
Chicken (5%)	15120	242625	59.94 : 1	32
Chicken (10%)	6822	119231	121.98 : 1	29
Chicken (15%)	4184	79548	182.83 : 1	27.74
Chicken (25%)	2025	44023	330.37 : 1	26

	Paper[11]			Paper [17]	
	Size	Ratio	PSNR	Size	Ratio
Chicken (5%)	6481000	22.5:1	28	490227	29 : 1
Chicken (10%)	318000	45.7:1	25.7	344985	58 : 1
Chicken (15%)	175000	83:1	24.6	205464	88 : 1
Chicken (25%)	76700	189:1	22.5	N/A	N/A

Table 3: Results of the Comparison for the "Dance" sequence.

	Ours				Paper [17]	
	Matrices	Size	Ratio	PSNR	Size	Ratio
Dance (1%)	32103	489644	32.88 : 1	24.08	878270	18 : 1
Dance (2%)	20314	365378	37.18 : 1	21.53	490227	33 : 1
Dance (3%)	15217	283377	56.81 : 1	19.95	344985	47 : 1
Dance (5%)	10360	167687	96 : 1	18.06	205464	78 : 1

Table 4: Results of the Comparison for the "Cow" Sequence.

	Ours				Paper [17]	
	Matrices	Size	Ratio	PSNR	Size	Ratio
Cow (1%)	43335	589656	11.4 : 1	26.35	N/A	N/A
Cow (5%)	15873	233673	28.78 : 1	19.68	424603	16 : 1
Cow (10%)	10310	163772	41.07 : 1	16.65	202942	32 : 1
Cow (15%)	6796	115296	58.33 : 1	15.27	140999	48 : 1

Table 5: Results for the "Snake" and "Ball" sequences.

	Ours			
	Matrices	Size	Ratio	PSNR
Snake (1%)	40590	616136	23.96 : 1	29.67
Snake (2%)	27066	410012	36 : 1	26.98
Snake (3%)	21588	327996	45 : 1	25.38
Snake (5%)	15167	231451	63.77 : 1	23.34
Ball (2–5%)	99	143	46590 : 1	43.83

Table 6: Results for the PCA algorithm using different bases.

	3 bases		5 bases	
	Ratio	PSNR	Ratio	PSNR
Dance (sd = 14)	67.5	16.9	40.5	20.5
Cow (sd = 6)	163	12.6	98.1	14.1
Chicken (sd = 16)	139.3	28.2	83.6	31.1
	10 bases		30 bases	
	Ratio	PSNR	Ratio	PSNR
Dance (sd = 14)	20.2	25.4	6.7	38.7
Cow (sd = 6)	49.2	16.64	16.5	23.7
Chicken (sd = 16)	42.2	36.4	14.2	47.4

of A will only be large in the case of a singularity, for example, points are too close to each other. In that case, two things happen: (1) we apply a singular value decomposition (SVD) to solve for A in (5); (2) we fix the reconstruction error to 5%. That is, when approximating the pseudoinverse by its SVD, we use only the eigenvalues corresponding to the first 95% of the principal components.

3.5. Termination Criteria. In Section 3.2, we explained how the algorithm stops at step (4). However, there are actually two criteria for such termination.

The first criterion to stop the partitioning of the octree comes from the reconstruction error. That is, the maximum reconstruction error allowed for any single vertex is defined by

$$\mathrm{ME} < \max_{i=1\cdots N_j} \left(\left| \overrightarrow{\widehat{q}_{d_{ji}}} - \overrightarrow{q_{d_{ji}}} \right| \right), \tag{8}$$

where $\overrightarrow{\widehat{q}_{d_{ji}}}$ and $\overrightarrow{q_{d_{ji}}}$ are the original and reconstructed vertices of the jth node.

In other words, if the reconstruction error of any single vertices exceeds ME, the node is partitioned into eight subcubes. Otherwise, the algorithm stops. In Section 4 we explain the choices of this threshold.

The second criterion to stop the partitioning is the number of vertices inside a cell. As we explained in Section 3.3, if that number is 4 or less, we store the coordinates of the vertices directly instead of the motion vectors (affine).

3.6. Algorithm Complexity. The complexity of our algorithm derives mainly from the pseudo inverse used during the calculation of the affine transformation. That is, for a $n \times 4$ matrix $F_{j(f-1)}$, the complexity of calculation the pseudo inverse is $O(4 \times n^2)$. Here, we will use $O(f(\cdot))$ as the *asymptotic upper bound* of $f(\cdot)$; $\Theta(f(\cdot))$ the *asymptotic tight bound* of $f(\cdot)$; $\Omega(f(\cdot))$ as the *asymptotic lower bound* of $f(\cdot)$.

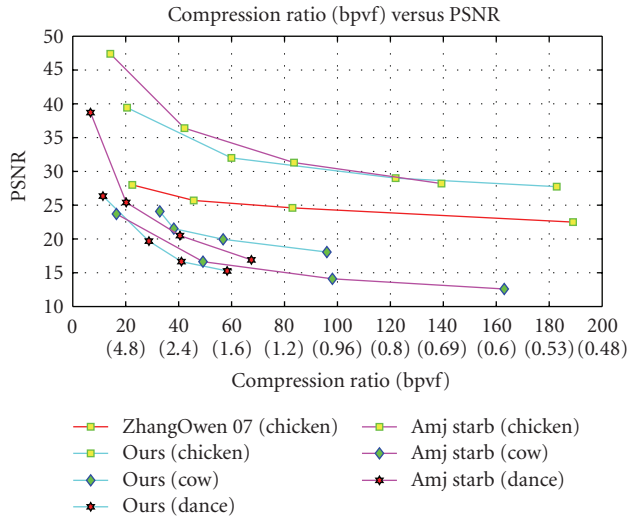


Figure 2: Comparison between our proposed method and ZhangOwen's and AmjStarb's algorithms.

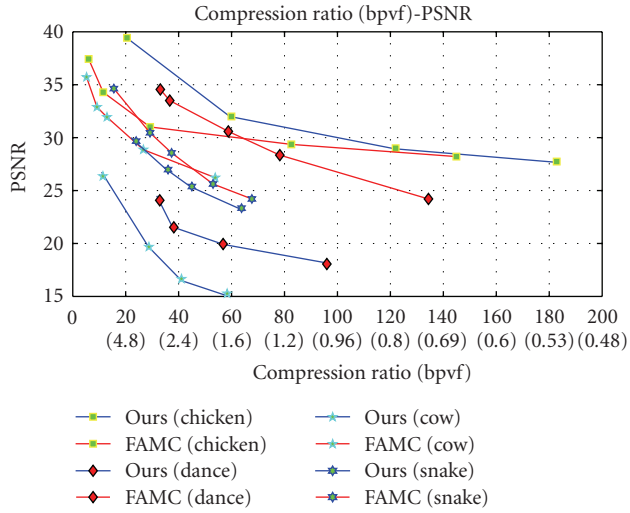


Figure 3: Comparison between the FAMC + DCT [14] and our proposed method.

Hence, when we consider each step of the proposed algorithm in Algorithm 1, we arrive the following recursive equation:

$$T(n) = c_1, \quad \text{for } n = 4,$$
$$T(n) = 4n^2 + c_2 + 19n + n + 8 * T\left(\frac{n}{8}\right), \quad \text{for } n > 4, \tag{9}$$

which can be further simplified as in:

$$T(n) = \Theta(1), \quad \text{for } n = 4$$
$$T(n) = 8 * T\left(\frac{n}{8}\right) + f(n), \quad \text{for } n > 4, \tag{10}$$

where $f(n) = 4n^2 + 20n + c_2$. This equation can be solved using the third case of the Master Theorem [31], where $f(n) = \Omega(n^{1+\epsilon})$, $\epsilon = 1$. That is, $f(n)$ is polynomially larger than n^2. Therefore, the solution for the recursion is $T(n) = \Theta(n^2)$.

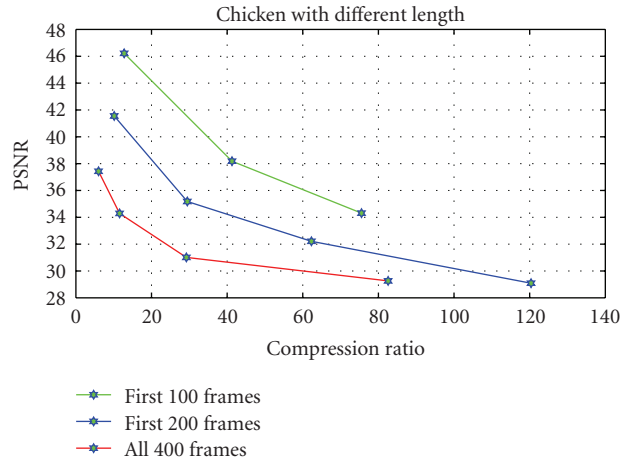


Figure 4: Decrease in compression ratio as a function of the number of frames used in the FAMC + DCT algorithm—for any given (fixed) PSNR.

3.7. Internal Representation of the Data. The final data consists of the following items. The first frame of the sequence with all N uncompressed 3D coordinates of the vertices using single float-point precision: $N \times 3 \times 4$ bytes. The m_f quantized affine transformation matrices corresponding to the m_f leaf nodes found in the octree for each frame f in the sequence of F frames: $\sum_{f=1}^{F} m_f \times 3 \times 4 \times 2$. If a leaf node contains less than four vertices, the coordinates of these vertices are used instead. Finally, one byte for level of each leaf node in the octree. To further increase the compression ratio, we apply an arithmetic encoding to this data.

4. Results

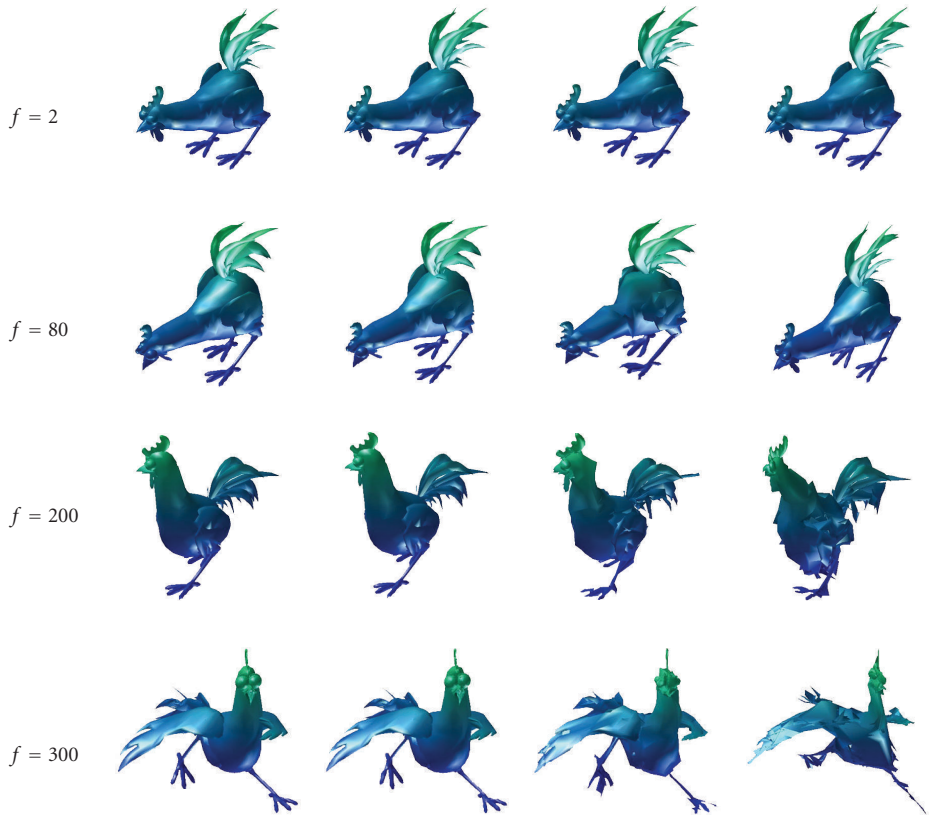
We compared our algorithm against three other methods in the literature. We will refer to these as the ZhangOwen's algorithm [11, 17]; the AmjStarb's algorithm [20]; the FAMC + DCT algorithm [14, 15]. As we already explained in Section 2, the first method uses an octree with a trilinear interpolation, while the second one uses a PCA-based approach, and the third uses an irregular partitioning also followed by an affine transformation.

4.1. Quantitative Measurement. In order to compare our method against these other approaches, we calculated the *Peak Signal-Noise Ratio*, or PSNR, as defined in [11]

$$\text{PSNR} = 10 \log_{10} \frac{d_{\max}}{\text{AVGMSE}},$$
$$\text{AVGMSE} = \frac{1}{M} \sum_{i=1}^{M} \text{MSE}_i, \tag{11}$$

(a) Original frame (b) Our method: PSNR = 32 dB (c) FAMC: PSNR = 34 dB (d) FAMC: PSNR = 30 dB

Figure 5: Illustration of the visual perception for differences of 2 dB in frame reconstruction.



(a) Original frame (b) ME = 0.01 (c) ME = 0.1 (d) ME = 0.25

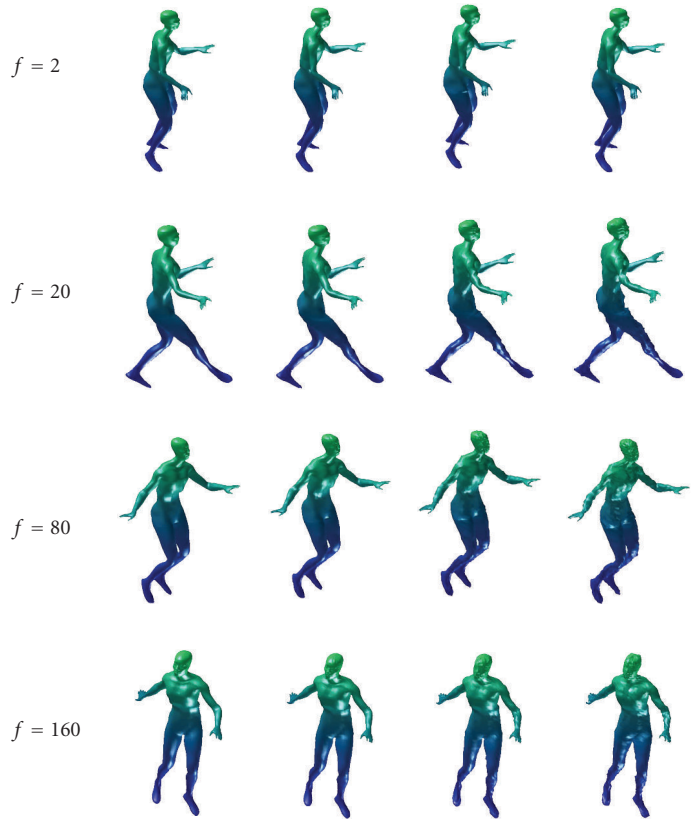
Figure 6: Reconstructed frames for the "Chicken" sequence.

where $d_{\max}$ is the size of the largest bounding box among the different frames. Also, MSE is the mean-square error of the distance between reconstructed and actual vertices, that is, $\text{MSE}_i = |\vec{\hat{q}}_i - \vec{q}_i|^2$.

4.2. Benchmark Dataset. We applied the four algorithms—ZhangOwen's algorithm, AmjStarb's, FAMC + DCT, and our method—to a total of five different benchmark sequences. Each of these sequences contains objects with different rigidity and number of vertices. Table 1 summarizes the properties of these sequences.

4.3. Quantitative Results. First, we compared our approach against ZhangOwen's and AmjStarb's algorithms in terms of Compression Ratio versus PSNR. Figure 2 shows the relationship between these two performance measures for three of the benchmark sequences. For detailed numerical data, the reader should refer to Tables 2, 3, 4, and 5. The points of the curves in Figure 2 for our method correspond to the different percentages in parenthesis in the tables, and they were obtained by controlling the choices of ME in (8). As for ZhangOwen's and AmjStarb's algorithms, the controlling parameters were, respectively, a similar maximum error threshold and the number of principal components (i.e., the number of bases)—as we explain later in this section.

As Figure 2 shows, our approach consistently outperforms ZhangOwen's algorithm, both in terms of compression ratio and PSNR. For AmjStarb's algorithm, our approach



(a) Original frame (b) ME = 0.01 (c) ME = 0.03 (d) ME = 0.05

Figure 7: Reconstructed frames for the "Dance" sequence.

performs quite similarly, however, we must point out once again that a PCA-based approach requires prior knowledge of the entire sequence in order to achieve good results, while our method can be carried out on a frame-to-frame basis.

As the tables above imply, the compression ratio is highly dependent on the animation sequence. For example, for the "Ball" and "Chicken" sequences, which present a simpler motion, the compression ratio is also higher. In especial, for the "Ball" sequence, the entire set of 100 frames required only 99 matrices. That means that one single affine matrix was sufficient for each of these frames—no subnode was required under the root node of the octree.

Also, the value used for ME does not completely determine the PSNR, although they are highly correlated. For example, the Chicken sequence with an ME = 5%, our method achieved a PSNR of 32, while for the Cow sequence an ME = 1% turned out to reach a PSNR of only 26.35. We assume that this is because of the complexity of the motion, which still is the dominating factor in the final result.

Since the author did not provide the calculation for PSNR in [17], the data for some of the sequences was not available. However, the author did not make any changes between [11] and [17] that could have affected the PSNR. Therefore, we can assume that the PSNR would be the same for both methods in [11, 17].

Finally, for the PCA-based algorithm in [20], it is important to mention that we implemented their approach using the same parameters reported in their paper. Table 6 summarizes the results for four different numbers of principal components or bases, that is, 3, 5, 10, and 30.

After the tests above, we also compared our method to a state-of-the-art algorithm: the FAMC + DCT [14, 15]. As we mentioned earlier, the FAMC is the standard method for the MPEG-4 AFX Amendment 2 [29] and it uses an irregular partitioning based on the topology of the mesh. That is, clusters of vertices are created by successively incorporating neighboring vertices that maintain the error in reconstruction from the affine transformation within a certain limit. This analysis of the error is performed over long stretches of the sequence, while a fixed partitioning, decided at the time of the first frame, is employed throughout the sequence. In Figure 3, we present a comparison between our method and the FAMC + DCT for the first four benchmark sequences in Table 1. As the figure shows, our method outperforms the FAMC + DCT for the Chicken sequence; it presents a performance similar to the FAMC + DCT for the Snake sequence; it is outperformed by the FAMC + DCT for the other two sequences (Dance and Cow). The reason for that behavior comes mainly from the rigid versus nonrigid characteristics of the partitioning in both methods.

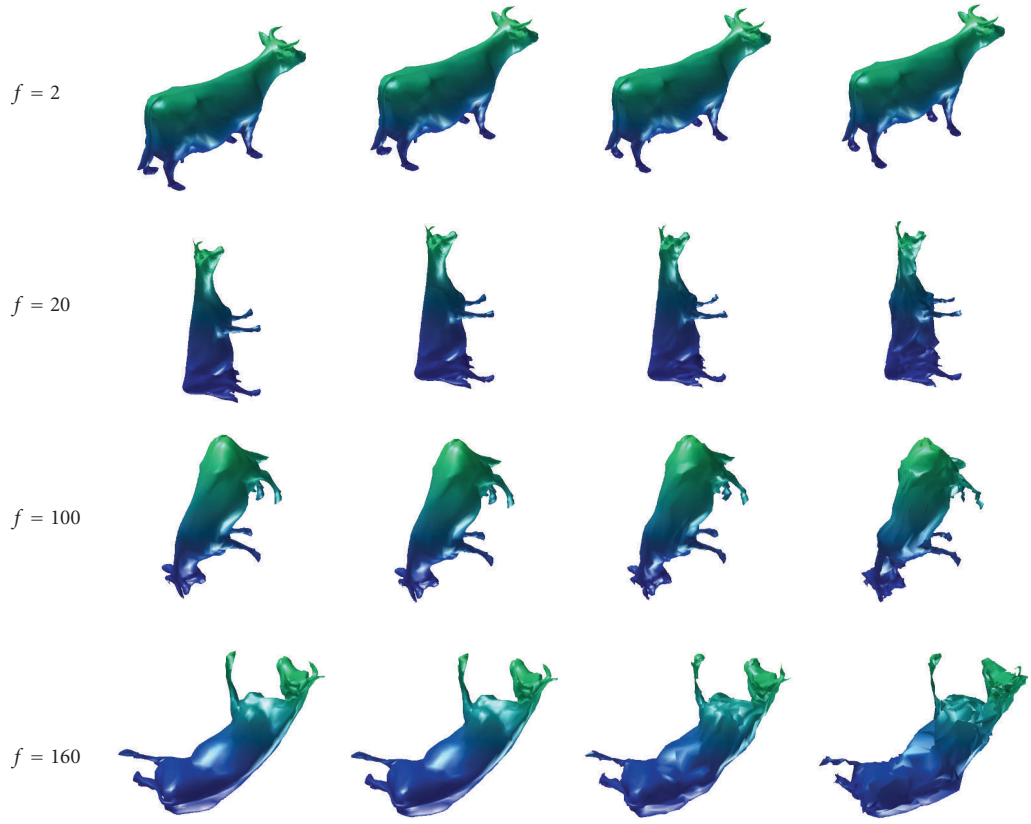


Figure 8: Reconstructed frames for the "Cow" sequence.

That is, while the octree can lead to unnecessary partitioning of neighboring vertices with the same motion vectors simply because of the proximity of another group of vertices with a different motion, in a non-rigid partitioning as in the FAMC, vertices are grouped independently of the existence of this second group of vertices. That difference between the two approaches usually leads to a more efficient partitioning of the vertices by the FAMC. However, that same efficiency is compromised by long sequences—as it was the case for the Chicken, which has the largest number of frames in our tests (400)—and by radical motions of the vertices—as it was the case for the Snake, whose vertices move much more freely than in the other sequences. In other words, since the size and the number of clusters in the FAMC depend on those two factors (the number of frames and the nonrigidity of the motion of the vertices), the larger they become, the more likely it is for the FAMC to require more clusters with fewer vertices in each cluster, and consequently to present a reduction in compression ratio. This claim is supported by Figure 4, where we depict the performance of the FAMC + DCT method for the Chicken sequence as a function of the number of frames used in the sequence. As the figure indicates, if we fix the PSNR, the compression ratio decreases as we add more frames to the sequence. It is important to notice however that the compression ratio used in those curves were obtained without taking into account the need to add a key frame at the end of each smaller sequence. This added reference frame is usually very large and even if a static compression is performed as suggested in [14], the cost of constantly including such key frames would lower even further the compression ratio. In order to perform the comparison in Figure 3, we chose to use all 400 frames in the Chicken sequence for the FAMC + DCT algorithm.

4.4. Qualitative Results. We also perform a qualitative analysis of our algorithm for different parameters. However, before we present these results, we would like to illustrate what means visually for two algorithms to present a difference of 2 dB in PSNR. In that regard, Figure 5(a) presents one of the original frames in the Chicken sequence and the corresponding reconstructed frames achieved by our method and by the FAMC with different PSNRs. Our algorithm, in Figure 5(b), obtained a PSNR of 32 dB, while the FAMC in Figures 5(c) and 5(d) presented PSNRs of 30 dB and 34 dB, respectively. As the reader will notice, a change of 2 dB between (b) and (c) does not seem as pronounced as the same 2 dB difference between (b) and (d). This result is again to illustrate that the same 2 dB can appear quite differently to humans, and that some times the absolute number alone

is not enough to determine which reconstructed frame looks "better."

Finally, Figures 6, 7, and 8 depict various reconstructed frames for each of the three sequences: Chicken, Dance and Cow; using different values of ME.

5. Conclusion and the Future Work

We proposed an affine-based motion representation with adaptive threshold and quantization that uses a rigid octree structure for partitioning. Our experimental results indicated that the proposed algorithm is, in terms of compression ratio, superior to other octree-based methods, while it achieved similar performance when compared to PCA-based methods or the FAMC + DCT algorithm. However, our method has a much smaller computation complexity and it does not require prior knowledge of the sequence. Instead, it can be carried out on a frame-to-frame basis.

We have demonstrated that the computation complexity of our algorithm for one frame is $\Theta(n^2)$, where n is the number of vertices in the frame. Once again, this is a major advantage of our method when compared to other algorithms such as ZhangOwen's algorithm [11, 17] and the AmjStarb's algorithm [20]. Moreover, while state-of-the-art methods such as the FAMC + DCT can be improved by means of a PCA technique (as described in [18]) and outperform the compression ratio of our method in all tested sequences, the price tag to pay in that case is a complexity greater than $\Theta(n^3)$.

Both the PSNR and the compression ratios for our method were very high, and a choice of ME = 0.01 provided an excellent compromise between these two performance measurements.

One serious limitation of most time-dependent methods, including our method, is the requirement for correspondence between vertices in different frames. This prevents this method from being applied to real 3D data—cloud of points. In order to solve this problem, we propose to build a pseudocorrespondence between frames using the Iterative Closet Points algorithm [32, 33]. Another option would be the use of Robust Points Matching (RPM) [34–36] however, we anticipate that this technique would lead to a much more complex algorithm. A combination of ICP and RPM was proposed [37], which have a much better performance, but still the run time is a concern, since the RPM is $O(n^3)$ and ICP is $O(n \cdot \log)$. In the future, we intend to attack this problem of pseudocorrespondences.

Acknowledgments

The authors would like to thank Andrew Glassner for giving them access to Chicken data; Aljoscha Smolic, Nikolce Stefanoski, and Kivanc Kose for sharing the Chicken data; Hector Briceno for sharing the Cow, Ball, Snake, and Dance data. They would also like to thank Jinghua Zhang for explanations provided regarding their papers.

References

[1] M. Dearing, "Geometry compression," in *Proceedings of the 22nd Annual Conference on Computer Graphics (SIGGRAPH '95)*, pp. 13–20, Los Angeles, Calif, USA, August 1995.

[2] Y. Huang, J. Peng, and M. Gopi, "Octree-based progressive geometry coding of point clouds," in *Proceedings of Eurographics Symposium on Point-Based Graphics*, pp. 103–110, 2006.

[3] R. Schnabel and R. Klein, "Octree-based point cloud compression," in *Proceedings of Eurographics Symposium on Pointbased Graphics*, pp. 111–120, 2006.

[4] C. Touma and C. Gotsman, "Triangle mesh compression," in *Proceedings of Graphics Interface Conference*, pp. 26–34, Canadian Human-Computer Communications Society, Vancouver, Canada, June 1998.

[5] J. Rossignac, "3D compression made simple: edgebreaker with zip and warp on a corner-table," in *Proceedings of IEEE Conference on Shape Modelling and Applications*, pp. 278–283, 2001.

[6] A. Szymczak, "Optimized edgebreaker encoding for large and regular triangles meshes," in *Proceedings of Data Compression Conference (DCC '02)*, p. 472, Snao Bird, Utah, USA, 2002.

[7] T. Lewiner, H. Lopes, J. Rossignac, and A. W. Vieira, "Efficient Edgebreaker for surfaces of arbitrary topology," in *Proceedings of the 17th Brazilian Symposium of Computer Graphic and Image Processing and 2nd Ibero-American Symposium on Computer Graphics*, pp. 218–225, Curitiba, Brazil, October 2004.

[8] M. Weeks and M. Bayoumi, "3D discrete wavelet transform architectures," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '98)*, vol. 4, pp. 57–60, Monterey, Calif, USA, May-June 1998.

[9] F. F. Rodler, "Wavelet-based 3D compreesion with fast random access for very large volume data," in *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, p. 108, 1999.

[10] J. E. Lengyel, "Compression of time-dependent geometry," in *Proceedings of the Symposium on Interactive 3D Graphics*, pp. 89–95, Atlanta, Ga, USA, April 1999.

[11] J. Zhang and C. B. Owen, "Octree-based animated geometry compression," in *Proceedings of Data Compression Conference (DCC '04)*, pp. 508–517, Snowbird, Utah, USA, March 2004.

[12] S. Gupta, K. Sengupta, and A. A. Kassim, "Compression of dynamic 3D geometry data using iterative closest point algorithm," *Computer Vision and Image Understanding*, vol. 87, no. 1–3, pp. 116–130, 2002.

[13] P.-F. Lee, C.-K. Kao, J.-L. Tseng, B.-S. Jong, and T.-W. Lin, "3D animation compression using affine transformation matrix and principal component analysis," *IEICE Transactions on Information and Systems*, vol. E90-D, no. 7, pp. 1073–1084, 2007.

[14] K. Mamou, T. Zaharia, and F. Prêteux, "Famc: the MPEG-4 standard for animated mesh compression," in *Proceedings of International Conference on Image Processing (ICIP '08)*, pp. 2676–2679, San Diego, Calif, USA, October 2008.

[15] N. Stefanoski and J. Ostermann, "Spatially and temporally scalable compression of animated 3D meshes with MPEG-4/FAMC," in *Proceedings of the 15th International Conference on Image Processing (ICIP'08)*, pp. 2696–2699, San Diego, Calif, USA, October 2008.

[16] J. Zhang and J. Xu, "Optimizing octree motion representation for 3D animation," in *Proceedings of the 44th Annual Southeast Conference*, pp. 50–55, Melbourne, Fla, USA, March 2006.

[17] J. Zhang, J. Xu, and H. Yu, "Octree-based 3D animation compression with motion vector sharing," in *Proceedings of the 4th International Conference on Information Technology-New Generations (ITNG '07)*, pp. 202–207, Las Vegas, Nev, USA, April 2007.

[18] K. Mamou, T. Zaharia, F. Preteux, A. Kamoun, F. Payan, and M. Antonini, "Two optimizations of the MPEG-4 FAMC standard for enhanced compression of animated 3D meshes," in *Proceedings of the 15th IEEE International Conference on Image Processing (ICIP'08)*, pp. 1764–1767, San Diego, Calif, USA, October 2008.

[19] M. Alexa and W. Müller, "Representing animations by principal components," *Computer Graphics Forum*, vol. 19, no. 3, pp. 411–418, 2000.

[20] R. Amjoun and W. Straßer, "Efficient compression of 3D dynamic mesh sequences," *Journal of the WSCG*, vol. 15, no. 1–3, pp. 99–107, 2007.

[21] I. Guskov and A. Khodakovsky, "Wavelet compression of parametrically coherent mesh sequences," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 136–146, Grenoble, France, 2004.

[22] F. Payan and M. Antonini, "Wavelet-based compression of 3D mesh sequences," in *Proceedings of the 2nd IEEE International Conference on Machine Intelligence (ACIDCA-ICMI '05)*, Tozeur, Tunisia, November 2005.

[23] K. Muller, A. Smolic, M. Kautzner, and T. Wiegand, "Rate-distrotion optimization in dynamic mesh compression," in *Proceedings of IEEE International Conference on Image Processing*, pp. 533–536, Atlanta, Ga, USA, October 2006.

[24] M. Sattler, R. Sarletter, and R. Klein, "Simple and efficient compression of animation sequences," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 209–217, Los Angeles, Calif, USA, 2005.

[25] Q. Gu, J. Peng, and Z. Deng, "Compression of human motion capture data using motion pattern indexing," *Computer Graphics Forum*, vol. 28, no. 1, pp. 1–12, 2009.

[26] J.-H. Yang, C.-S. Kim, and S.-U. Lee, "Compression of 3-D triangle mesh sequences based on vertex-wise motion vector prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp. 1178–1184, 2002.

[27] L. Ibarria and J. Rossignac, "Dynapack: space-time compression of the 3D animations of triangle meshes with fiexed connectivity," in *Proceedings of the ACM SIGGRAPH/Eurographics Sysmposium on Computer Animation (SCA '03)*, pp. 126–135, 2003.

[28] M. Tournier, X. Wu, N. Courty, E. Arnaud, and L. Revéret, "Motion compression using principal geodesics analysis," in *Proceedings of ACM Siggraph/Eurographics Symposium on Computer Animation (SCA '08)*, July 2009.

[29] "Iso/iec 14496-16, mpeg-4 animation framework extension (afx) model," http://wwwwww.iso.org//iso/iso_catalogue/_tc/catalogue_detail.htm?csnumber=44097.

[30] C. L. Jackins and S. L. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," *Computer Graphics and Image Processing*, vol. 14, no. 3, pp. 249–270, 1980.

[31] T. H. Cormen, R. L. Rivest, C. E. Leisersonc, and C. Stein, *Introduction to Algorithms*, McGraw-Hill, New York, NY, USA, 2003.

[32] Y. Chen and G. Medioni, "Object modeling by registration of multiple range images," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2724–2729, Sacramento, Calif, USA, April 1991.

[33] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Proceedings of the 3rd International Conference on 3-D Digital Imaging and Modeling*, pp. 145–152, Quebec City, Canada, 2001.

[34] H. Chui, J. Rambo, J. Duncan, R. Schultz, and A. Rangarajan, " Registration of cortical anatomical structures via robust 3D point matching," in *Proceedings of the 16th International Conference Information Processing in Medical Imaging (IPMI '99)*, pp. 168–181, Visegrád, Hungary, June-July 2003.

[35] A. Rangarajan, H. Chui, and E. Mjolsness, "A relationship between spline-based deformable models and weighted graphs in non-rigid matching," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. I897–I904, Kauai, Hawaii, USA, December 2001.

[36] H. Chui and A. Rangarajan, "A new point matching algorithm for non-rigid registration," *Computer Vision and Image Understanding*, vol. 89, no. 2-3, pp. 114–141, 2003.

[37] N. Okada and M. Hebert, "Fast 3D tracking of non-rigid objects," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 3, pp. 3497–3503, Taipei, Taiwan, September 2003.