Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A New 3D Representation and Compression Algorithm for Non-Rigid Moving Objects using Affine-Octree

Youyou Wang and Guilherme N. DeSouza
Department of Electrical and Computer Engineering
University of Missouri
Columbia, MO, 65211 USA

***Abstract*—This paper presents a new 3D representation for non-rigid objects using motion vectors between two consecutive frames. Our method relies on an Octree to recursively partition the object into smaller parts for which a small number of motion parameters can accurately represent that portion of the object. The partitioning continues as long as the respective motion parameters are insufficiently accurate to describe the object. Unlike other Octree methods, our method employs an affine transformation for the motion description part, which greatly reduces the storage. 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 with other methods like tri-linear interpolation or Principle Component Analysis (PCA) based algorithm, the Affine-Octree method is easy to compute and highly compact. As the results demonstrate, our method has a better performance in terms of compression ratio and PSNR, while it remains simple.**

***Index Terms*—3D Motion representation, Non-rigid objects, Octree, Affine transformation, Animation Compression**

I. Introduction

When it comes to representing 3D data – whether for efficient storage, transmission, rendering, etc. – two basic categories of methods can be defined: time-independent methods and time-dependent methods.

In the first and most traditional category [1], time-independent, the 3D object can be compressed based on its geometric properties alone. In those cases, triangular meshes, surface normals, edges, wavelet coefficients, and other features [2], [3] of the object are analyzed within a single time instant, or frame.

In the second case, or time-dependent methods, such as in [4] and more recently in [5], [6], [7], the basic idea is to represent the motion of the 3D object, or the difference between consecutive frames, rather than the object itself. In order to efficiently achieve that, one must identify the parts of the object that are fixed from the parts that are moving, and describe only the latter. This raises two major problems: 1) how to partition these two portions of the object; and 2) how to describe the moving portions. Although much research has already been done in this area [5], [8], [9], [6], [7], these approaches still suffer from: 1) inaccurate motion transformations [9]; 2) the need for extra space to store the partitioning [6]; 3) the need for apriori information on the entire sequence [7]; etc.

In this paper, we address the above problems by proposing a fixed 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 II, we will describe some related works in terms of compression. Section III contains the details about our approach, while in Section IV we compare our results to other works and show the advantage of using our representation for compression.

II. Related Work

One of the first methods proposed for time-dependent 3D data compression can be found in [4]. From this paper, a new standard of time-dependent compression was established: represent the motion among successive frames by a small number of parameters, and to choose a coding technique to efficiently represent/store the data. Although some papers do not follow exactly this standard – like the wavelet-based approach in [10], [11] – most time-dependent methods can be generally divided into two steps: a) partitioning of complex objects into smaller and simpler object. b) description of the motion of these simplified portions of the object.

With regard to the partitioning method, we find systems using regular spatial partitioning, where vertices are divided according to their spatial location. One such example is the Octree [5], [8], [9], [12]. In this case, the space is recursively divided into 8 equal portions until some termination criteria stops the process.

On the other side of the coin, we find systems employing irregular partitioning. In [6], 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 [13], the clustering of the vertices was based on a method

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

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 know beforehand in order to calculate the subspaces. The same can be said about other PCA-based methods [14], [7] – whether using irregular partitioning or not. Finally, in [15], 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 [16]. Actually, there is a third kind of systems where a spatial partitioning is not at all explicit. That is, in [17], [18] 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, the partitioning is generic enough so that the same motion descriptor can be used by other methods. In [5], [8], [9], all systems used a tri-linear interpolation, a regular partitioning (octree) and eight motion vectors attached to the corners of the cell. Another system, [6], used irregular partitioning and 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[14], [7] and locally[13], [15], but in either case, the entire sequence must be used for the principal component analysis. More recent approaches include the Principal Geodesic Analysis, a variant of the PCA method [19], methods relying on prediction of the motion vectors, [17], and the replica predictor[18].

III. Proposed Approach

A. 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 [20]. This data structure has also been widely used in both time-independent methods, such as [21], [22], as well as time-dependent methods, such as in [5] and later improved in [8], [9], [12].

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 re-scaling (normalizing) all vertices to a size between [0, 1] – that is, the size of the root cube is always regarded as 1 unit.

B. 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 2 and 3 above are repeated for each of the subcubes.
5) Otherwise, it stores the quantized affine transformation as the motion vector for that cube.

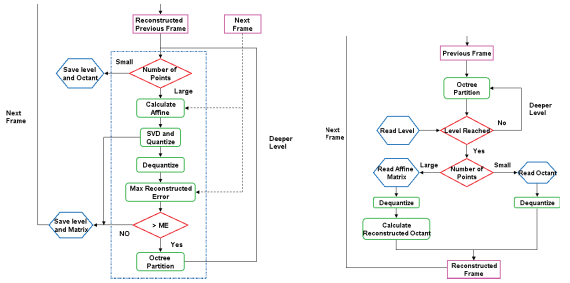
The steps above are highlighted by the blue box in Figure 1a.

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 the steps above 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 1b. 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.

C. Computation of the Affine Transformation

One of the main steps in our approach is the calculation of the motion vectors between two consecutive frames. Since the correspondence between vertices from two different frames is known, the motion vectors can be approximated using an affine



(a) (b)

Figure 1: Flowchart of the Algorithm: (a) Encoder and (b) Decoder

transformation A whose reconstruction error can be expressed as:

$$E = \sum_{i=1}^{N} \|A * \overrightarrow{p_i} - \overrightarrow{q_i}\|^2 \tag{1}$$

where N is the total number of vertices in the cube, and $\overrightarrow{p_i}$ is a 4 by 1 homogeneous vector with the coordinate of vertex i in the previous frame. Similarly, $\overrightarrow{q_i}$ 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,

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

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 *i*th 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:

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

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

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

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 pseudo-inverse. That is:

$$A_j F_{j(f-1)} = F_{jf} \tag{2}$$
$$A_j F_{j(f-1)} F_{j(f-1)}^T = F_{jf} F_{j(f-1)}^T$$
$$A_j = F_{jf} F_{j(f-1)}^T \cdot \left(F_{j(f-1)} F_{j(f-1)}^T \right)^{-1}$$

The matrix A_j is a 4 by 4 matrix with $[0001]$ as the last row. Since each pair of points gives us three equations, N must be equal or larger than 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 are transmitted instead.

D. 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:

$$A' = \left\lfloor 2^k \left(\frac{A - a_{min}}{a_{max} - a_{min}} \right) \right\rfloor \tag{3}$$

where k is the quantization step. Also, in order to be able to compare our method with the method developed in [5], 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 dimension of the root cube to $[0..1]$. That guarantees that the elements of A will only be large in the case of a singularity – e.g. 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 (2); and 2) we fix the reconstruction error to 5%. That is, when approximating the pseudo inverse by its SVD, we use only the eigenvalues corresponding to the first 95% of the principal components.

E. Termination Criteria

In Section III-B, 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:

	Vertices	Triangles	Frames	Size(Bytes)
Dance	7061	14118	190	16099080
Chicken	3030	5664	400	14544000
Cow	2904	5804	193	6725664

Table I: Properties of the benchmark sequences used for testing

$$ME < \max_{i=1,N_j} \left(\left| \overrightarrow{\hat{q}_{d_{ji}}} - \overrightarrow{q_{d_{ji}}} \right| \right) \tag{4}$$

where $\overrightarrow{\hat{q}_{d_{ji}}}$ 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 IV 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 III-C, if that number is 4 or less, we store the coordinates of the vertices directly instead of the motion vectors (affine).

F. Quality of reconstructed sequences

There are a lot of ways of finding the quality of reconstructed sequences. However, in order to have a comparison with [5], [9], we applied the Peak Signal-Noise Ratio(PSNR) defined the same as in[5]

$$\begin{aligned} PSNR &= 10 log_{10} \tfrac{d_{max}}{AVGMSE} \\ AVGMSE &= \tfrac{1}{M} \sum_{i=1}^{M} MSE_i \end{aligned} \tag{5}$$

where $d_{max}$is the size of the largest bounding box. MSE is defined as $MSE_i = (|v_i' - v_i|)^2$.

IV. Results and Analysis

We tested our algorithm using different animation sequences, with objects of different rigidity and number of vertices. The details on each test sequence is presented in Table I. We also performed a comparison with Zhang, Owen and Yu' s algorithms in [5] and [9], as well as Amjoun and StraBer's algorithm in [15].

As it is shown in Figure 2, we plot the "Compression Ratio" versus the "PSNR", as calculated by eq(5), for each of the three methods. The detail results for this comparision is also shown in Tables II, IV, and III.

As for the items shown in these tabes, the percentage after each sequence name indicates the error threshold ME defined in the previous section; "Matrices" indicates the total number of affine transformation matrices we have to use for the entire sequence; and "Size" is the number of bytes of the compressed data.

As these results demonstrate, our compression ratios are much higher than those for the other methods assuming the same PSNR. For example, check Chicken(10%) and

	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[5]			Paper[9]
	Size	Ratio	PSNR	Size	Ratio
Chicken(1%)	N/A	N/A	N/A	N/A	N/A
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 II: Results Comparison of "Chicken" sequence

	Ours			
	Matrices	Size	Ratio	PSNR
Dance(1%)	32103	489644	32.88:1	24.08
Dance(2%)	20314	365378	37.18:1	21.53
Dance(3%)	15217	283377	56.81:1	19.95
Dance(5%)	10360	167687	96:1	18.06
	Paper[5]			Paper[9]
	Size	Ratio	PSNR	Size	Ratio
Dance(1%)	N/A	N/A	N/A	878270	18:1
Dance(2%)	N/A	N/A	N/A	490227	33:1
Dance(3%)	N/A	N/A	N/A	344985	47:1
Dance(5%)	N/A	N/A	N/A	205464	78:1

Table III: Results Comparison of "Cow" Sequence

	Ours			
	Matrices	Size	Ratio	PSNR
Cow(1%)	43335	589656	11.4:1	26.35
Cow(5%)	15873	233673	28.78:1	19.68
Cow(10%)	10310	163772	41.07:1	16.65
Cow(15%)	6796	115296	58.33:1	15.27
	Paper[5]			Paper[9]
	Size	Ratio	PSNR	Size	Ratio
Cow(1%)	N/A	N/A	N/A	N/A	N/A
Cow(5%)	N/A	N/A	N/A	424603	16:1
Cow(10%)	N/A	N/A	N/A	202942	32:1
Cow(15%)	N/A	N/A	N/A	140999	48:1

Table IV: Results Comparison of "Cow" Sequence

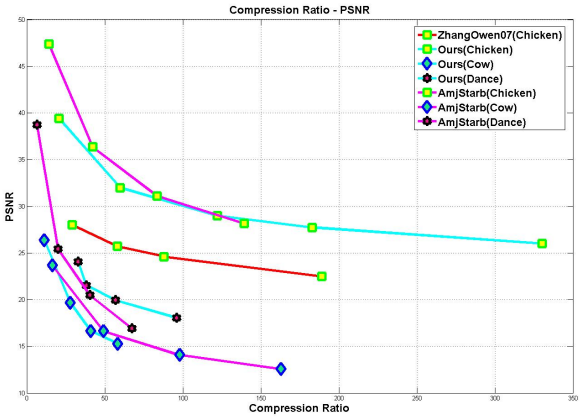


Figure 2: Overall Comparison

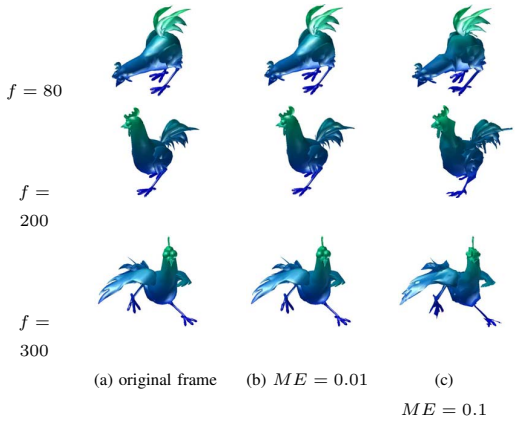


Figure 3: Reconstructed frames for the "Chicken" sequence

	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

Table V: Results of the PCA algorithm for the benchmark sequences for different bases

Chicken(15%). Since the author did not provide their calculation for PSNR in [9], the data for some of the sequences was not available. However, the author did not make any changes between [5] and [9] that could have affected the PSNR. Therefore, we can assume that the PSNR would be the same for [5] and [9].

Finally, for the algorithm in [15], we implemented their approach using the same parameters reported in their paper. Table V summarizes the results for four different number of principal components (bases): 3, 5, 10 and 30.

V. Conclusion

We proposed an affine-based motion representation with adaptive threshold and quantization using an octree structure. Our experimental results indicated that the proposed algorithm is superior to other octree-based methods, and could achieve similar performance when compared to PCA-based methods, but with a much smaller computation complexity.

Both the PSNR and the compression ratios 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 the future, we plan to solve this problem by building a pseudo correspondence between frames using the Iterative Closet Points algorithm.

References

[1] M.Dearing, "Geometry compression," *ACM AIGGRAPH'95*, 1995.

[2] A.Szymczak, "Optimized edgebreaker encoding for large and regular triangles meshes," *IEEE Proceedings of Data Compression*, p. 472, 2002.

[3] T. Lewiner, "Efficient edgebreaker for surfaces of arbitrary topology," *IEEE Proceedings of Computer Graphics and Image Processing*, pp. 218–225, 2004.

[4] J. E. Lengyel, "Compression of time-dependent geometry," *ACM Symposium Interactive 3D Graphics*, pp. 89–95, 1999.

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

[6] 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, pp. 116–130, 2003.

[7] P.-F. Lee, C.-K. Kao, B.-S. Jong, and Y.-W. Lin, "3d animation compression using the affine transformation matrix and pca," *IEICE Transacations on Infromation and Systems*, pp. 1073–1084, 2007.

[8] J.Zhang and J.Xu, "Optimizing octree motion representation for 3d animation," *ACM Session:Graphices and Real-time Systems*, pp. 50–55, Mar. 2006.

[9] J.Zhang, J.Xu, and H.Yu, "Octree-based 3d animation compression with motion vector sharing," *IEEE International Comference on Information Technology*, pp. 202–207, 2007.

[10] I.Guskov and A.Khodakovsky, "Wavelet compression of parameterically coherent mesh sequences," *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 136–146, 2004.
[11] F.Payan and M.Antonini, "Wavelet-based compression of 3d mesh sequences," *Proceedings of IEEE ACIDCA-ICMI*, 2005.
[12] K.Muller, A.Smolic, M.Kautzner, and T.Wiegand, "Rate-distrotion optimization in dynamic mesh compression," *IEEE International Conference of Image Processing*, pp. 533–536, Oct. 2006.
[13] M.Sattler, R.Sarletter, and R.Klein, "Simple and efficient compression of animation sequences," *ACM Siggraph Symposium on Computer Animation*, pp. 209–217, 2005.
[14] M.Aleca and W.Muller, "Representaing animations by principal components," *Computer Graphics Forum*, vol. 19, no. 3, pp. 411–418, 2000.
[15] R.Amjoun and W.StraBer, "Efficient compression of 3d dynamic mesh sequences," *Journal of the WSCG*, no. 1-3, pp. 99–107, 2007.
[16] Q.Gu, J.Peng, and Z.Deng, "Compression of human motion capture data using motion pattern indexing," *Computer Graphics Forum*, vol. 28(1), pp. 1–12, 2009.
[17] J. Yang, C. Kim, and S. Lee, "Compression of 3d triangle meshe sequences based on vertex-wise motion vector prediction," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 12, no. 12, pp. 1178–1184, 2002.
[18] L.Ibarria and J.Rossignac, "Dynapack:space-time compression of the 3d animations of triangle meshes with fiexed connectivity," *SCA '03: proceedings of the 2003 ACM SIGGRAPH/Eurographics Sysmposium on Computer animation*, pp. 126–135, 2003.
[19] M. Tournier, L. Reveret, X. Wu, N. Courty, and E. Arnaud, "Motion compression using principal geodesics analysis," in *ACM Siggraph/Eurographics Symposium on Computer Animation, SCA (Poster)*, July 2008. [Online]. Available: http://perception.inrialpes.fr/Publications/2008/TRWCA08
[20] C. Jackins and S.L.Tanimoto, "Oct-tree and their use in representing three dimensinal objects," *Computer Grapics and Image Processing*, vol. 14, p. 29, 1980.
[21] Y.Huang, J.Peng, and M.Gopi, "Octree-based progressive geometry coding of point clouds," *Eurographics Symposium on Point-based Graphics*, pp. 103–110, 2006.
[22] R.Schnabel and R.Klein, "Octree-based point cloud compression," *Eurographics Symposium on Point-based Graphics*, pp. 111–120, 2006.