Available online at www.sciencedirect.com
ScienceDirect
Image and Vision Computing 26 (2008) 1012–1026
image AND vision COMPUTING
www.elsevier.com/locate/imavis

Impact of vertex clustering on registration-based 3D dynamic mesh coding

Subramanian Ramanathan [a], Ashraf A. Kassim [a,*], Tiow-Seng Tan [b]

[a] Department of Electrical and Computer Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260, Singapore
[b] School of Computing, National University of Singapore, Computing 1, Singapore 117590, Singapore

Received 19 February 2006; received in revised form 16 October 2007; accepted 8 November 2007

Abstract

3D dynamic meshes are associated with voluminous data and need to be encoded for efficient storage and transmission. We study the impact of vertex clustering on registration-based dynamic mesh coding, where compact mesh motion representation is achieved by computing correspondences for the mesh segments from the temporal reference to obtain high compression performance. Clustering algorithms segment the mesh into smaller pieces and the compression performance is directly related to how effectively these pieces can describe the mesh motion. In this paper, we demonstrate that the use of efficient vertex clustering schemes in the compression framework can bring about a 10% improvement in compression performance.
© 2008 Published by Elsevier B.V.

Keywords: 3D dynamic meshes; Data compression; Computer animation

1. Introduction

Recent advances in computer graphics have accelerated the access and use of 3D mesh models in research and industry. Many interactive applications like collaborative CAD/CAM, interactive video games, and medical visualization support access of remote 3D data. Polygonal and triangular meshes are the de-facto standard for exchanging and viewing 3D models. Some of the attributes that describe a triangular mesh are the *geometry* that defines the position of vertices in the mesh, *connectivity* that describes the association between each triangle and its sustaining vertices, the *surface color*, *surface normal* and *texture*. Also, 3D meshes can be either static or dynamic (animations). 3D animation appears in two forms: rigid-body and soft-body motion. While the whole mesh moves as one entity in rigid-body motion, soft-body motion does not impose any restrictions on the movement of mesh vertices. Each mesh point can move in a separate trajectory, which produces smooth and realistic motion. In typical animations, both the position and connectivity of the points in the 3D mesh may change over time. However, for mesh sequences whose connectivity remains constant over time, the animation is characterized by changes in mesh geometry. Therefore, they can also be termed dynamic geometry sequences. We will restrict our discussions to the compression of dynamic geometry in this paper. Some dynamic geometry sequences are shown in Fig. 1.

3D mesh models representing complex objects typically demand a lot of storage space and rendering time for visualization. Also, transmission of 3D meshes is an onerous task that is severely limited by network bandwidth. Therefore, 3D mesh compression has generated much recent interest among researchers. Extensive work has been done on exploiting spatial coherency for compression of static meshes [6–9,21,28,33,34]. A majority of these static mesh compression techniques address efficient encoding of the mesh connectivity while mesh geometry coding is a supplementary to the connectivity coding scheme. Lengyel [20] pioneered the work on animation compression by repre-

* Corresponding author.
E-mail address: eleashra@nus.edu.sg (A.A. Kassim).

0262-8856/$ - see front matter © 2008 Published by Elsevier B.V.
doi:10.1016/j.imavis.2007.11.005

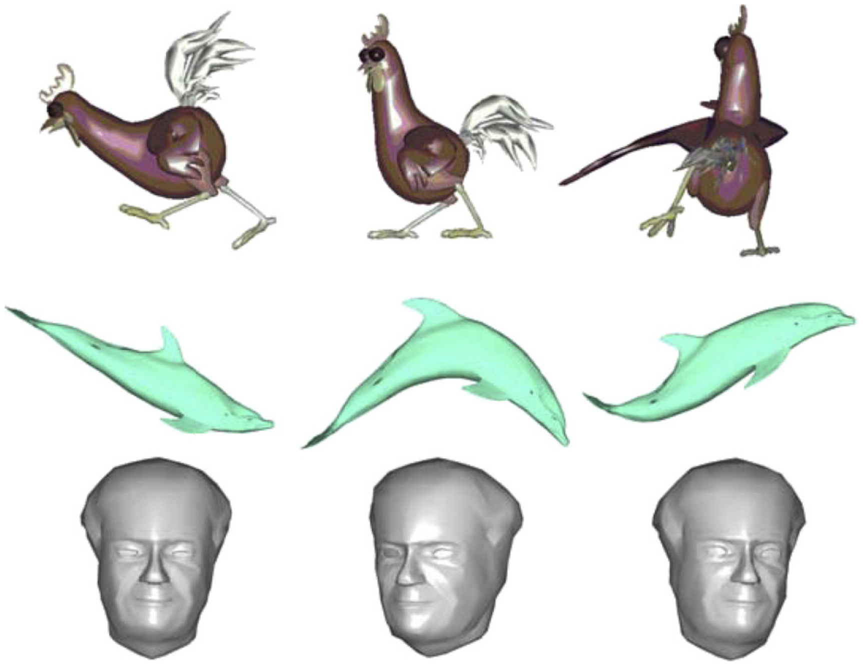

Fig. 1. Sample frames from 3D animation sequences *Chicken* (3029 vertices, 5664 triangles, 400 frames), Dolphin (6179 vertices, 12,337 triangles, 101 frames) and Face (757 vertices, 1468 triangles, 952 frames).

senting mesh motion using a few parameters. He proposed segmentation of the mesh into smaller sub-meshes whose motion can be described as rigid-body motion using affine transforms. The residual errors are encoded using spatial prediction. Ibarria and Rossignac [14] proposed a coding scheme for dynamic meshes with fixed connectivity by using space–time predictors to capture correlations in both spaces.

Another interesting work on dynamic geometry compression is that of Yang et al. [37] where each vertex is given a motion vector obtained from the neighborhood of the vertex. The vertex neighborhood is defined as the set of all vertices within a threshold distance around the vertex and since the motion vectors errors are correlated, they are also predicted using a rate-distortion optimization technique to yield better compression. A connectivity-based prediction technique was proposed by Stefanoski and Ostermann in [32], using a non-linear spatio–temporal predictor with angle preserving properties for encoding 3D dynamic mesh geometry. Another prediction-based compression algorithm using Differential Pulse Code Modulation (DPCM) was proposed by Muller et al. in [25].

Ahn et al. [1] suggested motion compensated compression of 3D animations by segmenting them into blocks for which motion vectors are calculated. Varakliotis et al. [35] proposed encoding with RTP packetization and recommended the insertion of *I* (Intra) frames, which can be decoded without any temporal reference, to tackle the degradation in animation smoothness on account of noisy channels. Gupta et al. [10] proposed a dynamic geometry compression scheme where the mesh is partitioned and the displacement of the vertices falling in each partition is computed using Iterative Closest Point (ICP) based registration. The mesh motion is described completely using a few affine parameters and residual errors and this algorithm achieves high compression performance for 3D animations.

Of late, multi-resolution mesh representation for bandwidth limited streaming applications has gained in importance. Representation of static meshes with various levels of detail is dealt with in [26,7]. Shamir et al. [30] suggested a multi-resolution representation for time-dependent meshes whose geometry and connectivity change with time. Another notable work is that of Alexa and Muller who discussed a compact representation of animations using PCA in [2] where each mesh in the animation sequence is projected on a basis of n PCA eigenvectors. The animation may be reconstructed using k eigenvectors where $k << n$. Higher the k, greater the level of detail. Other examples of wavelet-based multi-resolution encoding schemes are that of Guskov and Khodakovsky [11], who exploited the parametric coherence in mesh sequences and Payan and Antonini [27], who employed the lifting scheme to exploit temporal redundancy in dynamic geometry.

Karni and Gotsman proposed a compression scheme that employs a combination of Principal Component Analysis (PCA) and Linear Predictive Coding (LPC) in [16]. Recently, localized PCA-based techniques for compression have yielded good compression performance. Sattler et al. [29] proposed animation compression using Clustered PCA (CPCA) where the mesh is first segmented into meaningful components based on vertex motion analysis and PCA is then applied on each of these components. This compression scheme outperforms both pure PCA-based and PCA + LPC approaches while achieving better animation reconstruction. Another Localized PCA Analysis (LPCA)-based animation compression scheme was proposed by Amjoun and Straßer in [3]. On clustering the mesh using local similarity properties, a local coordinate system is defined for each cluster with respect to which the cluster motion is encoded using PCA. Experimental results indicate that the LPCA coder achieves better than CPCA-based compression.

1.1. Motivation

We discuss the contribution of vertex clustering to dynamic mesh coding. Efficient determination of intermesh motion regions is the key to compressing dynamic geometry. In fact, the idea is similar to that of MPEG video coding [19] where motion vectors are used to express differences between consecutive video frames. In MPEG compression, the motion parameters are obtained by dividing the image into equal-sized blocks (8×8 or 16×16) and estimating the motion of each of these blocks. Similarly, the 3D mesh also needs to be segmented into smaller pieces for efficient motion detection. However, segmentation of 3D meshes is not a trivial problem since they can model arbitrary shapes. Given that 3D meshes are non-planar, the manner in which they are segmented into pieces has a significant role to play in the animation compression framework. Our experiments confirm the critical impact of vertex clustering on the compression performance. This is a crucial difference between dynamic geometry coding and video compression, where the segmentation procedure is essentially a pre-processing step to motion estimation.

Clustering techniques group mesh vertices into a specified number of sets, where the grouping may be performed in one of the following ways. *Topology-based clustering* techniques partition the mesh based on vertex adjacency. They are more popularly termed “Graph Partitioning Techniques” and many algorithms have been developed to solve the graph partitioning problem over the years [13,18]. In this mode of clustering, vertices are clustered with their connected neighbors as given by the mesh connectivity and no knowledge of mesh geometry is required. *Geometry-based clustering* involves grouping of vertices based on their positional closeness and is independent of the mesh connectivity. Lloyd’s algorithm [24] is an example of geometry-based clustering where vertex neighbors are computed explicitly in the absence of connectivity information. The mesh vertices are clustered such that the mean distance between the cluster center and the cluster vertices is minimum. A third set of techniques perform *Semantic Mesh Decomposition* to segment the mesh into *meaningful components*. These techniques exploit both topology and geometry features to generate components that represent distinctive features of the 3D polygonal mesh. Unlike in geometry or topology-based clustering, where the number of clusters is user specified, most of the semantic mesh decomposition algorithms [23,17,22] automatically determine the number of vertex clusters based on homogeneity of the mesh regions.

Most dynamic geometry compression algorithms use topology-based clustering for segmenting meshes. Lengyel’s [20] algorithm uses a greedy vertex clustering approach based on the triangulation of the original mesh. Prediction-based geometry compression algorithms [37,32] define vertex neighborhoods for prediction based on mesh connectivity. Ahn et al. [1] segment the mesh by converting the triangular mesh structure into a linear triangle strip form. The triangle strip is divided into blocks such that each block has same number of vertices. Gupta et al. [10] use the multilevel k-way graph partitioning technique [13] that generates clusters of approximately equal sizes. Since the mesh connectivity remains constant for dynamic geometry, topology-based clustering needs to be performed only for the first mesh in the sequence (I mesh) and the clusters remain fixed thereafter for the entire sequence.

We find that mesh segmentation based on the *fixed* mesh topology is unsuitable for compressing mesh sequences with *changing* mesh geometry. Efficient detection of mesh pieces that have moved over time is possible only when the components generated upon clustering roughly represent these pieces. When the mesh undergoes arbitrary deformation, the vertex clusters undergoing coherent motion will be different at different times. Clearly, it is impossible for a given set of clusters generated using topology-based partitioning to represent the coherent motion regions at all times. Clustering the mesh on the basis of positional proximity instead of graph adjacency is more suited for encoding dynamic geometry sequences. Alternatively, a fixed set of clusters will effectively describe the *piecewise affine* mesh motion in animations only if they correspond to the distinctive mesh components that can undergo independent motion. Our results confirm that geometry-based clustering and semantic mesh decomposition techniques produce better compression performance than topology-based clustering. The increased compression performance obtained by PCA-based animation compression algorithms [3,29] which segment the mesh into coherent pieces based on local motion characteristics corroborate our experimental findings. We demonstrate that the compression performance can improve by as much as 10% by employing semantic mesh decomposition and geometry-based clustering instead of topology-based clustering.

1.2. Organization

The organization of this paper is as follows. We present an overview of some of the clustering schemes in the next section. These clustering algorithms are evaluated when integrated into the dynamic geometry compression algorithm [10] described in Section 3. We present the compression results to underline the importance of clustering and to compare the different clustering schemes in Section 4.

We conclude with directions for possible future work in Section 5.

2. Overview of vertex clustering techniques

The problem of determining the mesh motion is simplified by segmenting the mesh into smaller components. Mesh partitioning is a necessary pre-processing step for discovering the mesh regions that have moved with respect to the temporal reference and estimating their motion. A brief description of various vertex clustering techniques is presented in this section.

2.1. Multilevel k-way graph partitioning

In MPEG video compression, the image is divided into smaller, equal-sized blocks for efficient motion prediction. Likewise, decomposition of the mesh into equal-sized segments can be achieved using the *topology-based* multilevel k-way graph partitioning algorithm [13]. Given the mesh geometry V, the function of the graph partitioning algorithm is to divide the mesh into k subsets, $V_1, V_2, \ldots, V_k$ such that

$$V_i \cap V_j = \phi \quad \text{for } i \neq j$$
$$|V_i| = n/k$$
$$\bigcup_{i=1\ldots k} V_i = V$$

where $|V_i|$ denotes the cardinality of the ith cluster and $\bigcup V_i$ denotes the union of the k clusters.

Let $G(V, E)$ represent a graph containing vertex set V and edge set E. The graph partitioning algorithm first coarsens the original graph $G^0 = G(V^0, E^0)$ into a series of coarse graphs $G^i = G(V^i, E^i)$, such that the number of vertices at G^i is approximately half the number of vertices at $G^{(i-1)}$ i.e. $|V_i| \approx \frac{1}{2}|V_{i-1}|$. The graph is coarsened by performing a series of edge contractions. A maximal set of edges, no two of which are incident on the same vertex, are first determined and these edges are contracted. This coarsening procedure maps each vertex in the fine graph G^{i-1} to a unique vertex in the coarse graph G^i and therefore, graph topology is preserved. The coarsening terminates when the original graph has been coarsened to $G^m = G(V^m, E^m)$ where $|V_m|$ is typically a small number. The coarsening phase is useful as it is easier to find a good partitioning for the coarse graph than the original.

The coarsest graph G^m is now partitioned using a spectral partitioner [12]. One, two or three eigenvectors of the Laplacian matrix of the graph are used to partition into two, four or eight sets, respectively. The partitions obtained for the coarsest graph are propagated back to the finer graphs by projecting the k partitions onto $G^{m-1}, G^{m-2}, \ldots, G^0$. The projected partitioning onto G^{i-1} is occasionally refined using local refinement heuristics based on the Kerninghan–Lin (KL) algorithm [18]. Vertices are incrementally swapped among the partitions to reduce the number of *cut edges* connecting vertices in different partitions. The mesh is finally divided through recursive bisection into k sets each containing about $|V_0|/k$ vertices. The clusters generated by the k-way graph partitioning algorithm for various meshes are shown in Fig. 2.

Overall, multilevel k-way partitioning performs better than competing inertial or spectral bisection approaches [31] in terms of execution time and the partition quality (based on the number of *cut edges*). However, the vertex clusters obtained by minimization of the number of cut edges are ineffective for determining the mesh motion. This is evident from Fig. 2 where vertices belonging to distinct mesh regions are clustered together (parts of the nose, forehead and cheeks in the *face*; pelvis and thigh regions for *blade*) while vertices corresponding to the same region fall in different clusters (mouth region of the *face* and the claws for the *chicken*). Clearly, the clustered vertices will not undergo homogeneous motion. Also, when the same clusters are used for the entire sequence, detecting the coherent motion regions becomes difficult and consequently, the compression performance is affected as observed from our experimental results.

2.2. Lloyd's k-means clustering

The Lloyd's k-means algorithm [24] is a popularly used *geometry-based* clustering technique. Given a set of n data points $\{x_i\}$ in d-dimensional space and the required number of clusters k, the problem is to determine a set of k centers $\{c_j\}$ such that the mean squared distance of each point to its nearest center, termed the average distortion D, is minimum.

The algorithm works as follows. The initial k cluster centers are chosen at random and the data points $\{x_i\}$ are partitioned into k clusters by assigning each point to the cluster containing the closest c_i. The set of data points to which c_i is the nearest center is known as the neighborhood of c_i and is denoted by $V(c_i)$. Once the initial centers and their neighborhoods have been determined, the algorithm proceeds by moving the c_i's to the centroid of their clusters and recomputing $V(c_i)$ for each of the c_i's. This process iterates until convergence is achieved or the mean distortion D achieves a local minimum. A summary of the Lloyd's algorithm is presented below.

Step 1: Initialize $\{c_j\}$ by selecting the c_j's at random.
Step 2: Determine the neighborhood $V(c_j)$ for each of the c_j's by assigning the x_i's to their closest center.

$$V(c_j) = \{x_i : d(x_i, c_j) \leqslant d(x_i, c_k), \quad \text{for all } k \neq j\}$$

Step 3: Move each of the c_j's to the centroid of $V(c_j)$.

$$c_j = \frac{1}{|V(c_j)|}\sum_i (x_i), x_i \in V(c_j)$$

Step 4: Repeat Step 2 and Step 3 until mean distortion D is minimum i.e.

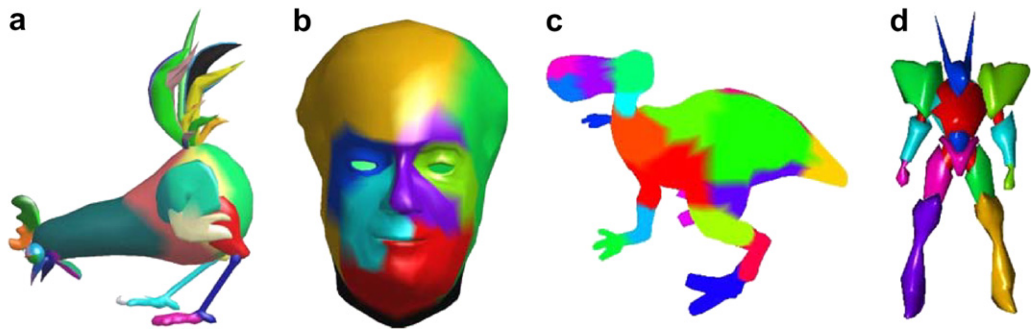

Fig. 2. Clusters generated by topology-based k-way partitioning algorithm for (a) *Chicken* – 32 partitions; (b) *Face* – 8 partitions; (c) *Dinopet* – 32 partitions; and (d) *Blade* – 8 partitions.

$$D = \frac{1}{k}\sum_{j=1}^{k}\frac{1}{|V(c_j)|}\sum_{x\in V(c_j)}(x_i - c_j)^2 = D_{\min}$$

Fig. 3 illustrates the working of the Lloyd's algorithm. In the context of mesh partitioning, the clusters themselves are more important than the cluster centers. For k-means clustering, it can be proved that the local minimum distortion measure would correspond to a "centroidal Voronoi" configuration [15], where each data point is closer to its cluster center than any other cluster center. The partitions move closer to this configuration at every step until convergence, and the final clusters would correspond to the local energy minima, even when the initial centers are badly chosen. However, slightly different initial partitionings do not produce the same set of clusters. Also, while the final partitioning is definitely better than the initial partitioning, it need not correspond to the global minimum. Nevertheless, this is not a significant problem for our application since data repartitioning may be performed later, as explained in the next section. Since clustering is performed independent of the mesh connectivity, vertex neighbors have to be computed explicitly. For 3D meshes, computing nearest neighbors is not a trivial problem. We use the Lloyd's implementation in [15], where the nearest neighbor queries are answered using a kd-tree (k-dimensional tree) data structure. A kd-tree is built for the data points and as the data points do not change throughout the cluster computation process, the kd-tree needs to be computed only once. The clusters at every step are determined by computing the nearest center for each of the nodes in the tree.

Since clusters are determined based on the vertex positions, the cluster configurations will vary for different meshes in the animation sequence (Fig. 4). Clustering based on vertex proximity produces better quality partitions whose vertices are more likely to undergo homogeneous motion. The cluster sizes are variable and the mesh can be segmented into arbitrary number of clusters. A noticeable improvement in compression performance is observed when geometry-based clustering is employed instead of topology-based partitioning for high-motion sequences. However, since the vertex clusters do not correspond to the distinctive mesh components, the general performance of geometry-based clustering is inferior to that of semantic mesh decomposition for dynamic mesh coding.

2.3. 3D mesh segmentation using spectral clustering

Semantic decomposition of a polygonal mesh into *meaningful components* is useful in mesh editing and morphing applications [23,17,22]. The mesh components represent the distinctive regions of the object consistent with human perception, which defines boundaries along concavities of the surface. They can be used to establish shape correspondence, and in most cases, also correspond to regions capable of undergoing independent motion (Fig. 5). Recently, Liu and Zhang proposed a spectral clustering approach to mesh decomposition in [23]. A brief description of their algorithm is presented below.

To segment a 3D mesh with n faces along the edges, the $n \times n$ affinity matrix W is initially constructed for the dual

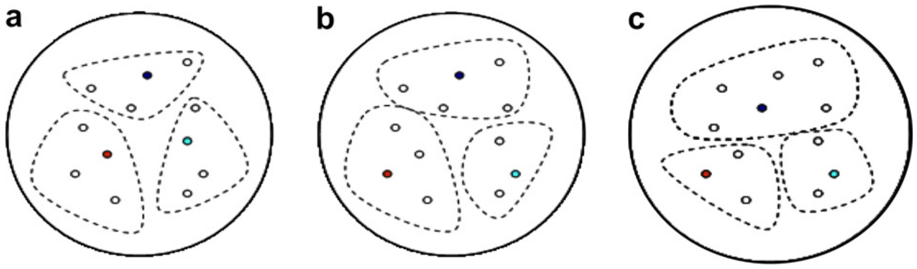

Fig. 3. Illustration of Lloyd's clustering for $k = 3$. (a) The initial cluster centers in red, blue and green and their computed neighborhoods. (b) Centers are moved to the centroid of the cluster and the data points are re-assigned to the nearest centers. (c) Final clusters.

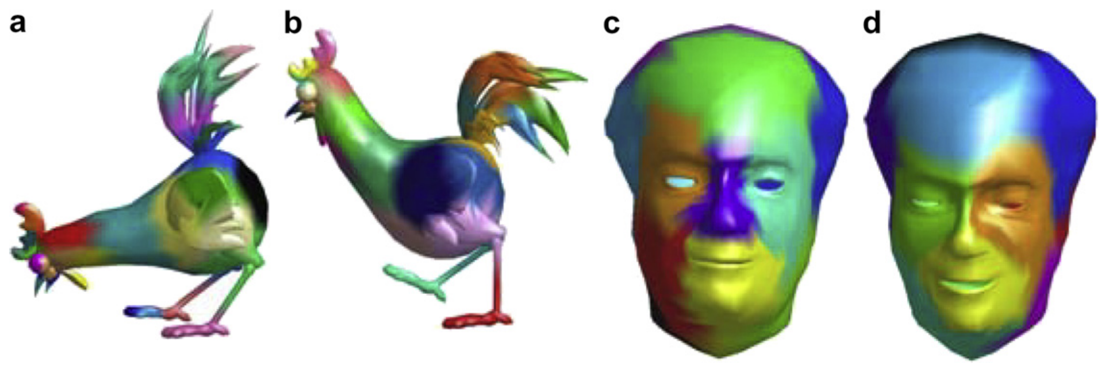

Fig. 4. Segmentation using Lloyd's clustering for frames (a) 70 and (b) 120 of the *Chicken* animation (maximum cluster size = 100); frames (c) 0 and (d) 505 of the *Face* animation (maximum cluster size = 75).

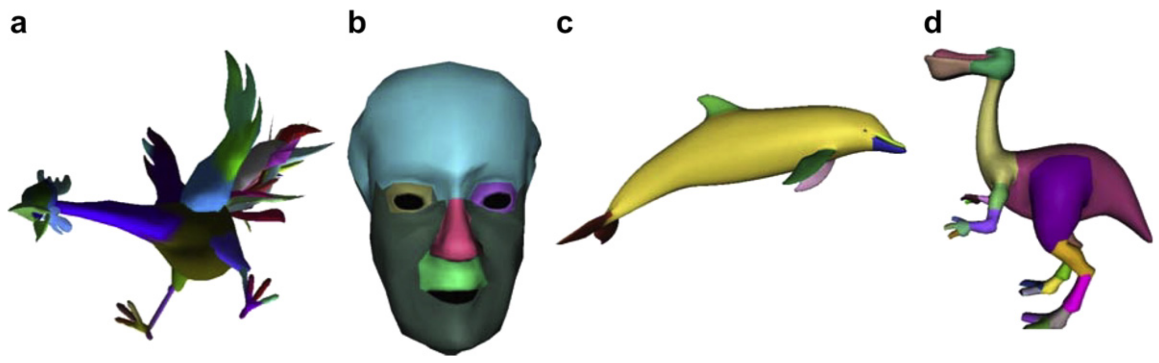

Fig. 5. Segmentation of (a) *Chicken* (59 components), (b) *Face* (6 components), (c) *Dolphin* (7 components) and (d) *Dinopet* (29 components) meshes through spectral clustering. A number of mesh segments e.g. fins of the dolphin, limbs of the dinosaur can undergo independent motion.

of the mesh graph to group faces closer to each other. Each vertex in the dual graph corresponds to a mesh face and two vertices are connected if and only if the corresponding mesh faces are adjacent to each other. For grouping of faces, the pairwise face distance measure used in [17] is used to define the affinity matrix. The distance measure between mesh faces f_i and f_j is defined as the shortest path between their dual vertices given by

$$\begin{aligned}\mathrm{Dist}(i,j) &= \mathrm{weight}(\mathrm{dual}(f_i),\mathrm{dual}(f_j))\\ &= \delta\frac{\mathrm{Geod}(f_i,f_j)}{\mathrm{avg}(\mathrm{Geod})}+(1-\delta)\frac{\mathrm{Ang\_Dist}(\alpha_{ij})}{\mathrm{avg}(\mathrm{Ang\_Dist})}\end{aligned}$$

Here, $\mathrm{Geod}(f_i,f_j)$ is the geodesic distance between f_i and f_j while the angular distance is defined as

$$\mathrm{Ang\_Dist}(\alpha_{ij}) = \eta(1-\cos\alpha_{ij})$$

where α_{ij} is the angle between the normals for adjacent faces f_i and f_j. Since the angular distance plays a more important role for visually meaningful segmentation, δ is set to a value close to zero. Also, a smaller value of η favors concavities and therefore it is set in the range $0.1 \leqslant \eta \leqslant 0.2$. On obtaining the pairwise face distances, the affinity matrix is defined by the Gaussian kernel

$$W(i,j) = e^{\frac{-\mathrm{Dist}(i,j)}{2\sigma^2}}$$

It can be easily seen that $0 < W(i,j) < 1$ and takes larger values for faces closer to each other. A suitable value for the width of the Gaussian, σ, is empirically set to $\frac{1}{n^2}\sum_{1\leqslant i,j\leqslant n}\mathrm{Dist}(i,j)$.

$W(i,j)$ encodes the likelihood of faces i and j belonging to the same patch. The normalization of the affinity matrix is performed as $N = D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$ where D is the diagonal matrix whose ith diagonal element is the sum of the ith row of W, the vertex degree at node i. N possesses desirable properties in the context of spectral clustering [36] and $N_{ij} = \frac{W_{ij}}{\sqrt{D_{ii}D_{jj}}}$. Let V be $n \times k$ matrix formed using the k leading eigenvectors of N. Then, the $n \times k$ matrix $Q = VV^{\mathrm{T}}$ represents the most energy preserving projection of N to rank k. By normalizing the rows of V to unit length, we obtain $\widehat{V}$, whose rows $\hat{v}_1 \ldots \hat{v}_n$ (of dimension k) represent the embedding of the W_{ij}s onto the k-dimensional unit sphere centered at origin. $\widehat{Q} = \widehat{V}\widehat{V}^{\mathrm{T}}$ is known as the association matrix whose elements $\widehat{Q}_{ij} = \hat{v}_i\hat{v}_j^{\mathrm{T}} = \cos\theta_{ij}$ are the cosine of the angle between unit vectors $\hat{v}_i$ and $\hat{v}_j$. As N is projected to successively lower rank k, the sum of squared angle cosines $\sum_{i,j}(\cos\theta_{ij})^2$ is strictly increasing [5]. Point pairs likely to be clustered together will move towards each other as k decreases, while other pairs will move further apart. Therefore, clustering points in k-dimensional space is easier than clustering the original data and is accomplished by performing k-means clustering on the rows of $\widehat{V}$.

The spectral clustering algorithm performs a semantic decomposition of the mesh with the generated components corresponding to the salient features as shown in Fig. 5. The algorithm tends to segment the mesh in a hierarchical fashion on varying the number of eigenvectors chosen for V. The computation of the shortest distance face pairs and the affinity matrix W are of complexity $O(n^2\log(n))$ and $O(n^2)$, respectively, but this computation time is greatly reduced in the implementation described in [38]. Also, a recursive 2-way spectral cut procedure used in [38] overcomes the problem of choosing the optimal k for clustering and produces better quality partitions. Since the components generated upon mesh decomposition correspond to the salient features of the object, the same set of vertex clusters can be used for performing motion estimation for the entire animation sequence. Also, as the components can describe the piecewise affine mesh motion effectively, it is possible to encode the animation more efficiently. Our experiments confirm that the compression obtained through semantic mesh decomposition is generally much higher than that of k-way partitioning or Lloyd's clustering for dynamic geometry compression.

3. ICP based 3D dynamic geometry compression

The vertex clustering algorithms described in the previous section simplify the problem of determining the coherent motion regions by segmenting the mesh into pieces that can possibly undergo affine motion. In order to determine the actual regions that have moved, motion prediction needs to be performed. In MPEG video coding [19], motion estimation chiefly contributes to data compression. For 3D dynamic geometry, the inter-mesh motion is typically small. The mesh motion can be completely described using a few affine transformations and residual errors and this compact representation leads to compression. An efficient dynamic geometry compression algorithm [10] that performs systematic motion estimation is discussed in this section. In addition, we also look at associated performance measures used for comparing various compression schemes.

3.1. Algorithm description

A partitioning based dynamic geometry compression scheme that estimates the mesh motion through systematic motion segmentation is described in [10]. The algorithm uses the multilevel k-way graph partitioning algorithm for initially segmenting the mesh. For each of the pieces in the current mesh, the corresponding piece in the temporal reference is detected using the Iterative Closest Point (ICP) algorithm. Using the results of ICP based registration, the motion segmentation module segments the mesh vertices into distinct sets based on their motion characteristics. Finally, the motion of all mesh vertices is expressed using affine transforms and a few residual errors. The key features of the dynamic geometry compression algorithm are as follows.

3.1.1. ICP based motion segmentation

The Iterative Closest Point (ICP) technique for registration of two 3D surfaces was introduced by Besl and McKay [4] to evaluate the rigid-body transformation required for registering the two surfaces. Also, point correspondences are computed such that for every point in the first set, there is a *closest* point in the second with respect to a predefined distance measure. Since soft-body animation may be described using piecewise affine motion of the mesh, ICP can be applied upon the mesh segments generated using mesh decomposition to discover the coherent motion regions and the corresponding motion parameters.

Let $V(t)$ represent the geometry of the current mesh composed of the set of n vertices $v_j(t), j = 1, \ldots, n$ at time t. If $(x_j(t), y_j(t), z_j(t))$ denote the coordinates of $v_j(t)$, the position matrix $V(t)$ is given by

$$V(t) = \begin{pmatrix} x_1(t) & x_2(t)\cdots x_n(t) \\ y_1(t) & y_2(t)\cdots y_n(t) \\ z_1(t) & z_2(t)\cdots z_n(t) \end{pmatrix}$$

If the mesh has been divided into k segments and $V^i(t)$ represents the set of vertices falling in the ith segment, $i = 1 \ldots k$, $V(t) = [V^1(t)\ V^2(t) \ldots V^k(t)]$. The jth vertex in the ith segment is denoted using $v^i_j(t)$ and the reconstructed position of v_j at $(t-1)$ using $\bar{v}_j(t-1)$. The segments $V^i(t)$ form the inputs to the ICP based motion prediction module. ICP outputs the closest segment $\overline{V}^i(t-1)$ in the reconstructed reference mesh $\overline{V}(t-1)$ for each $V^i(t)$ in $V(t)$. Now, the motion of the cluster V^i can be described using the affine transformation given by the least square solution

$$A^i = V^i(t) \times (\overline{V}^i(t-1))^-$$

where $(\overline{V}^i(t-1))^-$ represents the pseudo-inverse of $\overline{V}^i(t-1)$. The estimated position of the vertices in V^i at t is computed as

$$\bar{v}_j(t) = A^i \times \bar{v}^i_j(t-1)$$

and the accuracy with which A^i can represent the vertex position $v^i_j(t)$ is given by the reconstruction error

$$e^i_j = \|v^i_j(t) - A^i \times \bar{v}^i_j(t-1)\|$$

The coherent motion regions (that have undergone consistent motion with respect to the temporal reference $V(t-1)$) are determined using the mean reconstruction error

$$\epsilon^i = \frac{1}{|V^i|}\sum_j e^i_j$$

where $|V^i|$ denotes the number of vertices in V^i. All the motion regions that correspond to an error less than a threshold τ are deemed as regions with acceptable motion error. τ is usually set to $0.25v$, where v is the average motion between $V(t)$ and $\overline{V}(t-1)$ in our experiments. The motion segmentation algorithm divides the mesh vertices into three sets.

- *First set (Type 1)* – consisting of clusters of vertices, such that the motion of each of the clusters may be described accurately using the associated affine transform A^i. The reconstruction error for the vertices falling in this set is less than τ.
- *Second set (Type 2)* – consisting of clusters of vertices, such that each cluster has an affine transformation matrix A^i associated with it. In addition, residual errors also need to be encoded for accurately representing the vertex positions. The reconstruction error for the vertices falling in this set using the affine transform alone is less than 20 τ in our experiments.
- *Third set (Type 3)* – consisting of vertices whose motion cannot be described effectively using affine transforms. These are encoded using DPCM-based techniques.

The ability of ICP based motion segmentation to divide the vertices in the mesh into distinct sets helps achieve better compression performance compared to other dynamic geometry compression algorithms. This is because the motion of *Type 1* and *Type 2* vertices, which constitute over 70% of the total, can be described using a few affine transformations and residual errors. Also, the residual errors can be adaptively coded using variable number of bits for different groups of vertices in order to maintain the animation smoothness.

3.1.2. Repartitioning and vertex regrouping

For some regions, the A^is computed using the correspondences output by ICP are associated with very high reconstruction errors. These regions are re-clustered and ICP based registration is repeated on these new partitions. Also, in order to encode the inter-mesh motion using as few parameters as possible, the A^is that accurately describe vertex motion are applied to vertices in the neighboring clusters as well. Finally, A^i is associated with the entire set of vertices for which it provides a good motion estimate. Our overall compression scheme is illustrated in Fig. 6.

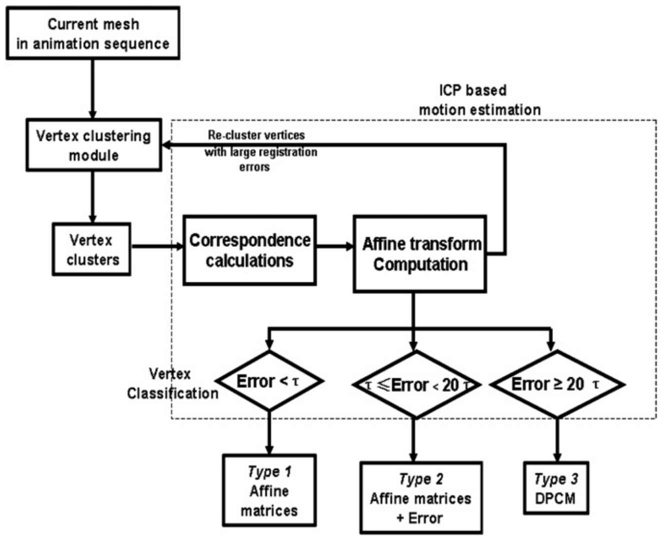
Current mesh in animation sequence
ICP based motion estimation
Vertex clustering module
Re-cluster vertices with large registration errors
Vertex clusters
Correspondence calculations
Affine transform Computation
Vertex Classification
Error < τ
τ ≤ Error < 20 τ
Error ≥ 20 τ
Type 1 Affine matrices
Type 2 Affine matrices + Error
Type 3 DPCM

Fig. 6. Overview of partitioning based dynamic geometry compression scheme.

3.2. Performance metrics

This section briefly describes the indices used for evaluating various compression algorithms – Signal to Noise Ratio (SNR) and compression performance. The SNR and PSNR (Peak Signal to Noise Ratio) are considered objective measures for comparing the quality of the compressed data against the original data. We use the following definition for the per-frame Peak Signal to Noise Ratio (PSNR) proposed in [35] for evaluating the reconstruction quality of the encoding scheme.

$$\mathrm{PSNR} = -10\log_{10}\mathrm{PMSE}$$

where PMSE is the Peak Mean Square Error per vertex given by

$$\mathrm{PMSE} = \frac{\frac{1}{N_n}\sum_{j=1}^{n}\frac{1}{3}\sum_{i=x,y,z}(v_{ji}(t) - \bar{v}_{ji}(t))^2}{R^2}$$

where R is the maximum inter-mesh displacement for the entire animation sequence, N_n and n denote the number of vertices that have moved between two consecutive meshes and the total number of mesh vertices, respectively. The PSNR provides a quantitative measure of the animation smoothness.

Another performance metric used for comparing various compression algorithms is the compression ratio which is defined as the size of the original data to the encoded data. The coding framework produces two types of meshes – *I* and *P*. *I* (Intra) meshes are encoded using static mesh compression techniques and complete information can be obtained by decoding the *I* mesh without any reference. *P* (Predicted) meshes contain only the differences from the temporally previous *I* or *P* mesh and chiefly contribute to compression. The difference needs to be added to the reference mesh data in order to obtain the complete *P* mesh information. Though the compression algorithm is inherently lossy, it is still acceptable if the reconstructed animation is “perceptually lossless”, as there is a trade-off between compression and quality. For *P* meshes, the following information needs to be encoded:

- Affine matrices and vertices associated with each affine transform matrix.
- Error values associated with the vertex positions for *Type 2* and *Type 3* vertices.

To compactly encode the above information, the following procedure is used:

- Vertices whose motion can be described using affine transforms are given the symbol P. Every P vertex is associated with a patch index to denote the associated affine transform. When the patch index of the vertex has not changed from the reference, a symbol N is used to signify no change and the previous patch index is used.
- *Type 2* vertices are represented using the symbol E to denote error information.
- *Type 3* vertices encoded using DPCM techniques are assigned the symbol D.
- As affine transforms associated with the vertex clusters exhibit a high spatio–temporal correlation, the differences between the affine matrices are quantized and encoded.
- The number of bits used to encode error data for *Type 2* and *Type 3* vertices is determined by the PSNR. More bits are added to encode error for vertices whose PSNR is below a certain threshold.
- Vertex symbols, affine matrices and residual errors are encoded using Arithmetic coding.

The per-frame compression ratio (CR) is calculated as follows:

$$\mathrm{CR} = \frac{(\text{Bits for raw data})}{(\text{Encoded vertex data bits} + \text{Affine transform bits} + \text{Error bits})}$$

The process of reconstructing mesh geometry from P meshes is outlined in Fig. 7. When there is large inter-mesh motion, the same vertex may be associated with a number of affine transforms and many vertices may require error information to be encoded, which results in considerably lower compression ratios. Therefore, such meshes are encoded as I meshes for which only the spatial coherence in mesh geometry is exploited. The Edgebreaker algorithm [28] is used for encoding I meshes. Insertion of I meshes helps maintain animation smoothness with a marginal reduction in compression. Also, periodic transmission of I meshes is necessary while transmitting data over noisy channels and for enabling random access to the animation sequence.

A number of encoding schemes also use (a) Distortion Factor to evaluate reconstruction quality, and (b) Bits per Vertex per-Frame for compression performance comparison. The Distortion Factor, d_a (also called KG_{error}), is defined as

$$d_a = 100\frac{\|B - \widehat{B}\|}{\|B - C(B)\|}$$

where B is a $3V \times F$ matrix representing the geometry of the V vertices in the F frames of the original animation, $\widehat{B}$ represents the reconstructed animation geometry and $C(B)$ contains the average vertex positions for the animation. The compression performance defined using encoded Bits per Vertex per-Frame (bpvf) is related to the compression ratio as given by the following equation.

$$\mathrm{bpvf} = \frac{\text{Bits for encoding each vertex}}{F} = \frac{\text{Raw data bits(total)}}{\left(\frac{\text{Compressed bits(total)}}{FV}\right)} = \frac{96}{\text{Avg. compression ratio}}$$

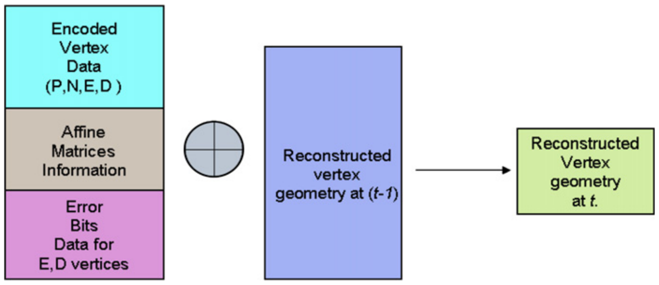
Encoded Vertex Data (P,N,E,D)
Affine Matrices Information
Error Bits Data for E,D vertices
Reconstructed vertex geometry at (t-1)
Reconstructed Vertex geometry at t.

Fig. 7. Reconstruction of mesh geometry from P meshes.

4. Results and discussion

In this section, we study the impact of vertex clustering on dynamic geometry compression by comparing the aforementioned performance metrics for the various clustering schemes.

4.1. Test animations and clustering statistics

Four animation sequences, namely, *Chicken, Face, Cow* and *Dance* were used for comparing the clustering schemes. The *Chicken* animation contains 400 frames with each mesh in the sequence consisting of 3029 vertices and 5664 triangles. The animation is highly non-linear with the motion becoming extremely rapid after frame 260. The *Face* sequence contains a realistic animation of a talking human face in various poses and exhibiting various facial expressions as well. There are 952 frames in the sequence with 757 vertices and 1468 triangles per mesh. The *Cow* (2904 vertices, 5804 triangles, 204 frames) animation is also a high-motion sequence while the *Dance* sequence (7061 vertices, 14,118 triangles, 201 frames) depicts a person performing various dance movements. Due to the similar nature of motion throughout the *Dance* animation, we used only the first 100 meshes for our experiments.

Evidently, spectral decomposition takes the maximum time for vertex clustering among the algorithms discussed in this paper. As against k-way clustering, based on recursive bisection followed by local refinement, and kd-tree based Lloyd's clustering (complexity $O(n\log n)$), spectral decomposition runs in $O(p\log(n))$ time, where n denotes number of mesh faces and p is the required number of recursive mesh cuts. The timing statistics for the various clustering algorithms on a 3.6 GHz, 1 GB RAM PC, are presented in Table 1. It is to be noted here that only the I mesh is processed by the k-way and spectral clustering

Table 1
Clustering statistics showing number of vertex clusters and the time for mesh segmentation taken by the different vertex clustering algorithms for our test animations

Animation	Clustering algorithm	No. of clusters	Clustering time (s)
Chicken	*k*-way	32	0.02
	Lloyd's	31	0.1
	Spectral	59	2.9
Cow	*k*-way	32	0.05
	Lloyd's	30	0.1
	Spectral	14	1.4
Dance	*k*-way	64	0.2
	Lloyd's	71	0.25
	Spectral	7	2.1

schemes to produce a single, representative set of clusters, while Lloyd's clustering is performed on every mesh in the animation sequence since the clusters vary with the mesh geometry (and therefore, the mean per-mesh clustering time is presented in the table).

4.2. Experimental results

The results of ICP based dynamic geometry compression [10] using *k*-way partitioning for the *Chicken* animation are shown in Fig. 8. An average compression of 45 is obtained for the animation using *k*-way partitioning with about 100 vertices per cluster. One can observe that the Lloyd's clustering (Fig. 4) and spectral decomposition (Fig. 5) schemes are able to segment the chicken's neck from its torso more effectively compared to *k*-way topology partitioning (Fig. 2). The motion in frames 40–60 and 200–230 (marked as **A** and **B**, respectively, in Fig. 8) is mainly localized around the neck of the chicken as seen in Fig. 9.

For *P* frames, the per-frame compression is directly proportional to the number of *Type 1* and *Type 2* vertices and inversely proportional to the number of *Type 3* vertices. The number of *Type 1* vertices, in turn, is determined by how well the affine transforms computed using ICP based registration can represent the piecewise motion of the mesh. There exists a direct relationship between the number of *Type 1* vertices registered using ICP and the accuracy with which the initial clusters input to ICP can represent the independent mesh regions. To illustrate this point, we encoded frame sequences 40–60 and 200–230 using Lloyd's clustering (100 vertices per cluster) and spectral mesh decomposition (59 mesh components).

Table 2 shows that the number *Type 1* vertices registered using ICP are much higher when Lloyd's and spectral clustering are used instead to *k*-way graph partitioning for both frame sequences. It is evident that the increase in the number of *Type 1* vertices owing to better quality of input clusters has a direct impact on the compression performance. For frames 40–60, all mesh vertices are encoded as either *Type 1* or *Type 2*. However, for frames 200–230, the motion is such that a number of vertices need to be encoded using DPCM techniques. For this frame sequence, while the number of *Type 1* vertices registered using Lloyd's and spectral clustering are about the same, the number of *Type 2* vertices are more for spectral clustering compared to Lloyd's resulting in improved compression performance.

The latter part of the *Chicken* animation (frames 261–399 marked as C in Fig. 8) is characterized by extensive motion. In these frames, while the computed affine transforms can register a large number of vertices within the error threshold τ, reconstruction errors are associated with a large number of vertices and the reconstructed animation is noisy. Additional bits need to be encoded to improve animation quality at the expense of compression performance for these frames as seen from Fig. 8(b). PSNR calculations are used to measure and improve the animation smoothness for this set of frames. For each frame in the animation, we measure the PSNR for *Type 1*, *Type 2*, *Type 3* vertices and the entire reconstructed frame for analysis. The minimum PSNR required for ensuring smooth animation reconstruction varies for different sequences and depends on the nature of the mesh motion. A minimum PSNR of 35 db is required for the *Chicken* sequence with $R = 0.7662$ (while a PSNR threshold of 20 db is sufficient for the low-motion *Face* sequence with $R = 0.0673$). The frame PSNR can be improved by (i) allocating extra bits for encoding *Type 2* and *Type 3* vertices (ii) using a smaller error threshold τ to register *Type 1* vertices and (iii) transmission of *I* meshes. Examples of (i) and (ii) are shown in Fig. 10.

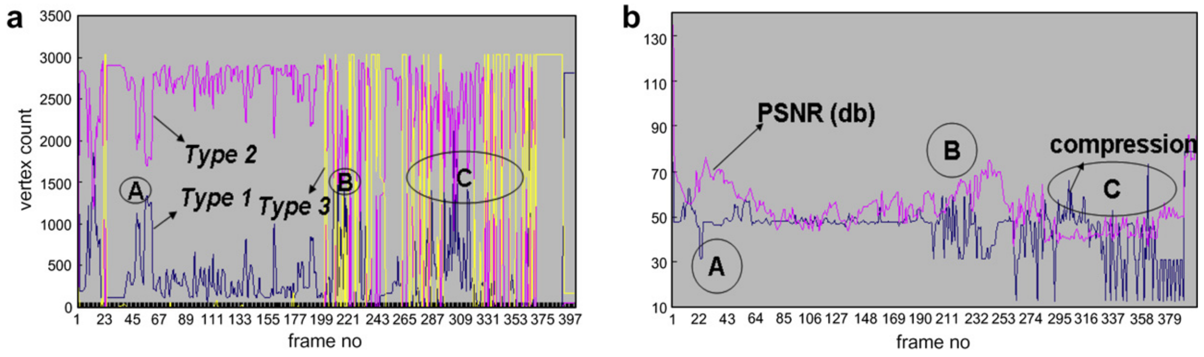

Fig. 8. (a) Variation in the number of *Type 1*, *Type 2* and *Type 3* vertices using *k*-way partitioning and (b) Per-frame PSNR (in pink) and compression ratio (in blue) for the *Chicken* animation.

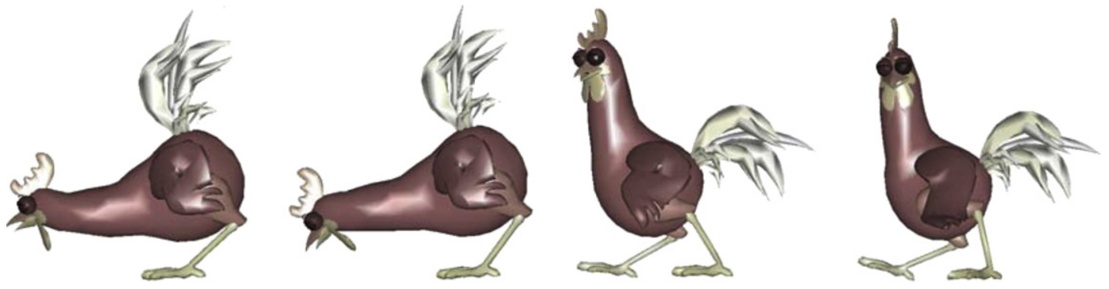
Fig. 9. Frames 48, 56, 203 and 221 of the *Chicken* animation.

Table 2
Impact of vertex clustering on compression performance for frame sequences (a) 40–60, (b) 200–230 and (c) 261–399 of the *Chicken* animation

Frame Nos.	Clustering mode	*Type 1* count (avg)	*Type 2* count (avg)	CR (avg)	PSNR (avg)
40–60	*k*-way	733	2296	52	66.1
	Lloyd's	746	2283	52.2	66.5
	Spectral	839.6	2189.4	52.8	67
200–230	*k*-way	470	1787	45.6	63.7
	Lloyd's	604	1632	47.2	63.9
	Spectral	603	1925	48.3	64.6
261–399	*k*-way	264	1406	38.3	51.1
	Lloyd's	357	1299	39.9	50.8
	Spectral	296	1407	39.1	51.1

The number of *Type 1* vertices and compression ratios increase with improvement in quality of input clusters.

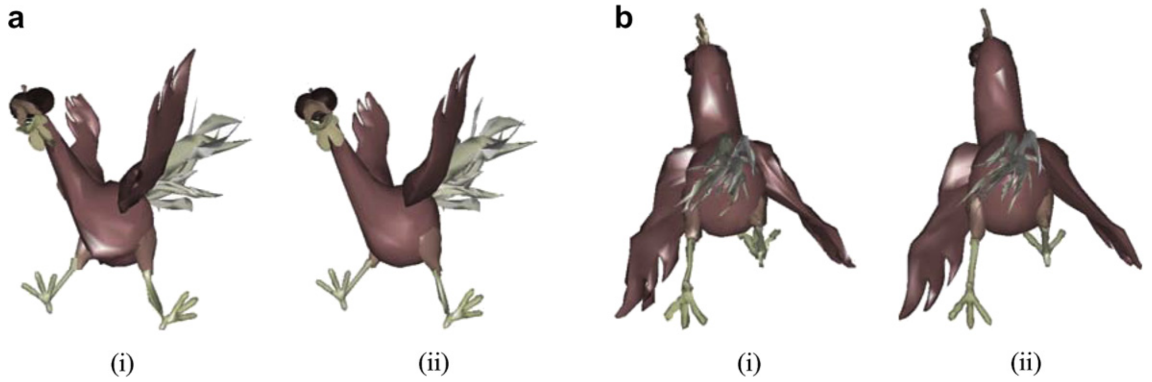

Fig. 10. (a) Reconstructed frame 286 with (i) PSNR = 25.7 db (CR = 46.6) and (ii) PSNR = 43.2 db using $\tau = v/11$ and 6 bits for error encoding (CR = 46.3). (b) Reconstructed frame 320 with (i) PSNR = 31.8 db (CR = 49.5) and (ii) PSNR = 39.7 db using $\tau = v/7$ and 5 bits for error encoding (CR = 49.3).

For frames 261–399, Lloyd's clustering performs better than spectral clustering and provides the best compression performance (Table 2). This is because the motion in these frames is concentrated around the chicken's wings which is not well segmented by spectral clustering. The rapidness in motion also necessitates a number of meshes in the animation to be coded as *I* meshes which correspond to the minima in the compression curve. The performance of various dynamic geometry coding schemes for the *Chicken* animation is presented in Table 3 and we find that the use of a potent clustering mechanism can bring about a 3.8% improvement in compression.

For the *Face* sequence, the inter-frame motion is very low and very few vertices are registered with $\tau = v/4$ for many frames. The compression results for the different clustering schemes for $\tau = v/4, v/3$ and $v/2$ are shown in Table 4. Clearly, spectral clustering produces higher compression performance than Lloyd's or *k*-way partitioning. The ability of the spectral clustering algorithm to accurately segment the various face regions (Fig. 5) enables the ICP module to register a maximum number of *Type 1* vertices. This leads to a major improvement in the compression performance even when the encoded mesh is small in size. The compression obtained using spectral clustering is 9.7%, 15% and 7.8% higher than *k*-way partitioning for τ equal to $v/4$, $v/3$ *and* $v/2$, respectively. However, the compression obtained using Lloyd's and *k*-way partitioning is very similar inspite of Lloyd's clustering producing better

Table 3
Performance of various dynamic geometry compression algorithms for the *Chicken* animation (uncompressed file size = 13.9 MB)

Compression algorithm	CR	d_a
Motion compensated compression [1]	10	
Motion vector prediction-based compression [37]	18.3	
Time-dependent geometry compression [20]	27	
PCA representation [2]	39.8	
Connectivity-guided connectivity compression [32]	33	0.13
Clustered PCA Analysis [29]	34.3	0.076
Partitioning based compression [10]	45.3	0.11
[10] with spectral	46.6	0.12
[10] with Lloyd's	46.7	0.12
Local PCA Analysis [3]	64	0.057

Table 4
Compression performance of the different clustering schemes algorithms at various values of τ for the *Face* animation

ICP threshold (τ)	Clustering scheme	*Type 1* count	CR (avg)	PSNR (avg)
$v/4$	*k*-way	182	41.9	44.3
	Lloyd's	165	41.2	44.5
	Spectral	268	45.9	43.8
$v/3$	*k*-way	372	53.3	39.9
	Lloyd's	367	52.9	40
	Spectral	526.8	61.3	39.2
$v/2$	*k*-way	670	69.5	34.4
	Lloyd's	675	69.2	33.9
	Spectral	710	74.9	34.9

quality clusters. This is possibly because a marginal improvement in cluster quality cannot significantly improve the number of *Type 1* vertices for the low-motion, small-sized *Face* mesh sequence. A PSNR threshold of 20 db is sufficient to smoothly reconstruct the animation. The SNR measure proposed in [10] does not work well for the *Face* animation and very low SNR values are obtained even when the reconstructed animation is smooth. Some of the reconstructed frames in the *Face* animation are shown in Fig. 11.

Some partitioned frames of the *Cow* and the *Dance* animations are shown in Fig. 12. A minimum PSNR of 35 db is required to smoothly reconstruct the animation for both sequences. As seen from the figure, the number of mesh clusters are more for *k*-way and Lloyd's compared to spectral clustering. As vertex clustering is performed solely based on proximity for *k*-way and Lloyd's clustering, the cluster sizes affect the compression and SNR performance as observed in [10]. While ICP works well on small-sized clusters, a large number of mesh clusters are associated with increased processing time and reduced compression performance (as more affines need to be encoded). Also, large-sized clusters produce registration errors and consequently, a degradation in compression and SNR performance. We observe that optimal compression and SNR performance is achieved for cluster sizes of 100 and 125 for *k*-way and Lloyd's clustering, respectively.

Fig. 13(a) outlines the compression performance obtained using the various clustering algorithms for the *Cow* sequence. The animation is characterized by high-motion and *I* meshes need to be encoded frequently after frame 100. Lloyd's clustering enables most efficient encoding of the mesh motion as shown in Table 5. The number of *Type 1* vertices is minimum for spectral clustering but it still outperforms *k*-way partitioning as the number of registered *Type 2* vertices are more. The poor performance of spectral compression for the *Cow* and the latter part of the *Chicken* animations underline the limitations of semantic mesh decomposition. This is because the mesh decomposition is purely based on the intrinsic geometric structure of the mesh. While it is difficult to achieve accurate segmentation of the mesh into distinctive components, efficient segmentation of coherent motion regions can only be performed by exploiting the motion cues available from the animation. Overall, about a 4% improvement in compression performance is obtained when Lloyd's clustering is used instead of graph partitioning. On the other hand, spectral clustering performs exceedingly well for the *Dance* animation. As evident from Fig. 13(b), the compression performance obtained using spectral clustering is significantly higher compared to Lloyd's clustering and *k*-way partitioning due to the increased number of ICP registered *Type 1* vertices. The use of spectral clustering improves compression performance by over 10% for the *Dance* animation as shown in Table 5.

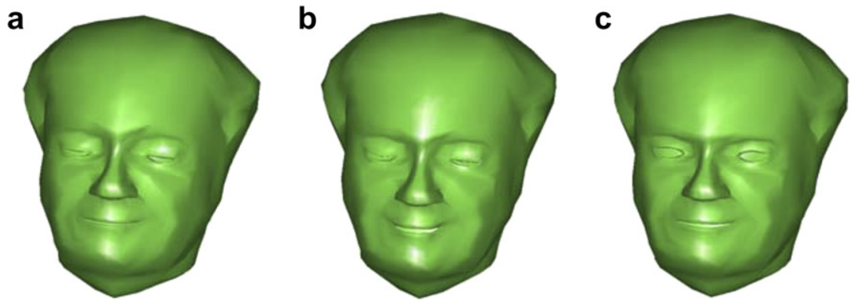

Fig. 11. Reconstructed frames of the *Face* animation. (a) Frame 704 with PSNR = 32 db (CR = 32.5), (b) frame 904 with PSNR = 29.6 db (CR = 43.7) and (c) frame 108 with PSNR = 25.4 db (CR = 71.6).

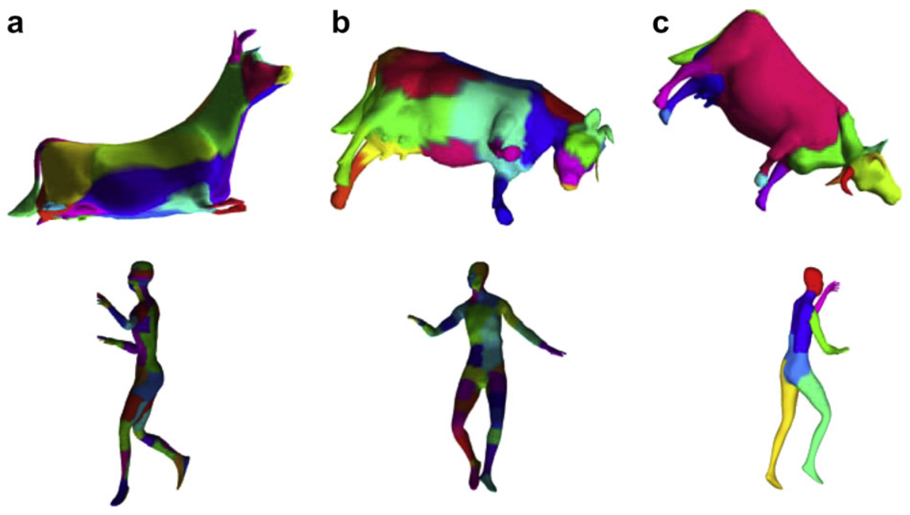

Fig. 12. Frames of the *Cow* and *Dance* animations partitioned using (a) *k*-way (b) Lloyd's and (c) spectral clustering.

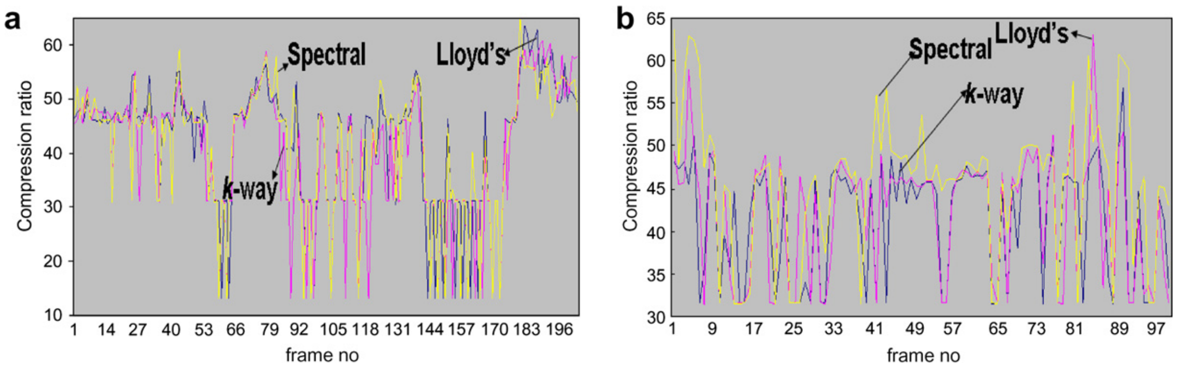

Fig. 13. Compression performance obtained using the different clustering schemes for (a) *Cow* and (b) *Dance* animations.

Table 5
Compression performance of the different clustering schemes algorithms for the *Cow* and *Dance* animations

Animation	Clustering	*Type 1* count	*Type 2* count	CR (avg)	% improvement	PSNR (avg)
Cow	*k*-way	351	1386	40.3	–	43
	Spectral	332.3	1597	41.7	3.5	44.7
	Lloyd's	368	1581	42	4.2	42.5
Dance	*k*-way	312	4038	41.4	–	42.2
	Lloyd's	414	3993	41.9	1.2	42.9
	Spectral	740	4765	45.8	10.6	41.5

4.3. Static vs motion-based mesh segmentation

As seen from the experimental results, the clustering scheme employed to segment the mesh for motion detection greatly affects compression performance. The discussed clustering schemes, namely, *k*-way partitioning, Lloyd's clustering and Spectral mesh decomposition are *static* mesh segmentation techniques that segment the mesh to be encoded into smaller pieces without any temporal considerations. For encoding motion in dynamic mesh sequences, *temporal* cues can also be used to group vertices likely to undergo similar motion. Motion-based segmentation can facilitate identification of those "pieces" that cannot be easily detected using static mesh decomposition. For example, for the human figure in the *Dance* sequence, segmentation of the arms and limbs is achieved by spectral clustering (Fig. 12). Motion cues can be used to achieve further segmentation around articulated joints like the elbow and knee, and clearly, the segmented parts will correspond to the coherent motion regions better.

Recently, two approaches that perform motion-based clustering to efficiently represent motion have been found to achieve high compression performance. Sattler et al. [29] proposed the clustered PCA (CPCA) approach to dynamic geometry compression that can identify the mesh parts undergoing coherent motion over time. The vertex trajectories are clustered using Lloyd's clustering [24] in combination with PCA to segment the coherent mesh parts. Each mesh part is then compressed using PCA on the complete animation as performed in [2]. This method results in higher compression than standard PCA and PCA + LPC approaches while producing lesser distortion. Also, Amjoun and Straßer [3] proposed local PCA-based compression, where the mesh is segmented into clusters based on local motion characteristics and a local coordinate system is defined for each cluster, with respect to which the cluster motion is encoded. Table 6 compares the performance of ICP-based compression using spectral clustering with CPCA- and LPCA-based compression for similar distortion.

From the table, it is evident that for the bpvf values for ICP-based compression using spectral decomposition are much lower than those for Clustered PCA for comparable values of distortion. While LPCA-based compression performs better than CPCA- or ICP-based coding for the *Chicken* sequence, ICP coding with spectral clustering achieves maximum compression for the *Cow* animation. This could be attributed to the inadequate segmentation achieved by the pure motion-based clustering schemes in [29,3]. While motion-based clustering can produce meaningful segmentation of the mesh into components, e.g. wings and legs of the *Chicken*, more coherent mesh segments are obtained for the *Cow* using spectral decomposition (Fig. 12) compared to pure motion-based clustering. An ideal segmentation scheme for compressing dynamic geometry should involve a first level of segmentation using static mesh segmentation techniques like spectral decomposition which is further refined using motion-based analysis. Future work should involve development of such a segmentation scheme that can maximize compression performance for 3D dynamic geometry.

5. Conclusions

We will now summarize our major observations on the impact of clustering on dynamic mesh coding as follows:

- As demonstrated by our results, use of a potent clustering algorithm alone can bring about a significant improvement in compression performance. Topology-based clustering of the mesh into pieces based on vertex adjacency as given by the connectivity matrix is inefficient for encoding dynamic geometry sequences. Clustering based on mesh geometry or a semantic decomposition of the mesh into components can produce better compression.
- Spectral mesh decomposition, in general, produces the best compression performance and results in higher compression (by as much as 10%) over Lloyd's or *k*-way clustering. The improvement in compression performance is chiefly due to the increased number of ICP registered *Type 1* vertices. ICP tends to register more vertices when the input vertex clusters can approximate the coherent motion regions better.
- Geometry-based Lloyd's clustering performs significantly better than the other clustering schemes when the inter-mesh motion is high (latter part of the *Chicken* and the *Cow* animations). However, the performance of Lloyd's clustering and *k*-way partitioning are very similar when the inter-frame motion is low as in the *Face* and *Dance* sequences.
- When the inter-mesh motion is high, spectral decomposition does not perform as well. This is because automated mesh decomposition is a difficult problem and the generated mesh components are not always accurate. Also, when the motion is rapid, the mesh can be assumed to typically consist of many motion regions. Efficient motion encoding would only be possible if a majority of these motion regions can be detected. Motion cues need to be employed to discover these motion regions.
- Future work should focus on developing an efficient clustering mechanism for animations. The mesh segmentation strategy should use global mesh features (given by the mesh geometry) as well as local motion characteristics to detect the coherent motion regions. We believe that such a mesh segmentation approach can enable very efficient compression of 3D dynamic geometry.

Table 6
Comparison of CPCA- and LPCA-based compression with ICP-based compression using Spectral clustering for the *Chicken* and *Cow* animations

Animation	CPCA-based compression		LPCA-based compression		ICP-based compression	
	bpvf	d_a	bpvf	d_a	bpvf	d_a
Chicken	4.7	0.076	3.5	0.008	2.16	0.12
	2.8	0.139	1.5	0.057	1.72	0.26
Cow	7.4	0.16	6.8	0.128	2.9	0.33
	3.8	0.50	4.1	0.47	2.3	0.47
	2.0	7.4	2.2	1.22	1.9	0.9

Acknowledgements

We are thankful to Rong Liu and Richard Zhang for the discussions we had during the course of this paper. Thanks are due to Sumit Gupta for his valuable comments and suggestions. The *Chicken* animation was created by Andrew Glassner, Tom McClure, Scott Benza and Mark Van Langeveld and is the property of Microsoft Corporation. The *Face* sequence is the property of Visage Technologies. The *Cow* and *Dance* animations were kindly provided by Matthias Müller, ETH Zurich.

References

[1] J.H. Ahn, C.S. Kim, C.C. Kuo, Y.S. Ho, Motion compensated compression of 3d animation models, Electronic Letters 37 (24) (2001) 1445–1446.

[2] M. Alexa, W. Muller, Representing animations by principal components, EUROGRAPHICS 19 (3) (2000) 411–418.

[3] R. Amjoun, W. Straßer, Efficient compression of 3d dynamic mesh sequences, 2007.

[4] P. Besl, N. McKay, A method of registration of 3d shapes, IEEE Transactions on Pattern Analysis and Machine Intelligence 14 (2) (1992) 239–256.

[5] M. Brand, K. Huang, A unifying theorem for spectral embedding and clustering, in: Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, 2003.

[6] M. Chow, Geometry compression for real-time graphics, in: Proceedings of Visualization '97, 1997.

[7] D. Cohen-Or, O. Remez, D. Levin, Progressive compression of arbitrary triangular meshes, in: Proceedings of Visualization '99, 1999.

[8] M. Deering, Geometry compression, in: Proceedings of SIGGRAPH '95, 1995, pp. 13–20.

[9] S. Gumhold, W. Strasser, Real time compression of triangle mesh connectivity, in: Proceedings of SIGGRAPH '98, 1998, pp. 133–140.

[10] S. Gupta, K. Sengupta, A. Kassim, Compression of 3d dynamic geometry data using iterative closest point algorithm, Computer Vision and Image Understanding 87 (2002) 116–130.

[11] I. Guskov, A. Khodakovsky, Wavelet compression of parametrically coherent mesh sequences, in: SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2004, pp. 183–192.

[12] B. Hendrickson, R.W. Leland, An improved spectral graph partitioning algorithm for mapping parallel computations, SIAM Journal on Scientific Computing 16 (2) (1995) 452–469.

[13] B. Hendrickson, R.W. Leland, A multi-level algorithm for partitioning graphs, in: Supercomputing, 1995.

[14] L. Ibarria, J. Rossignac, Dynapack: space–time compression of the 3d animation of triangle meshes with fixed connectivity, in: Proceedings of the ACM SIGGRAPH Symposium on Computer Animation, 1999.

[15] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu, The analysis of a simple k-means clustering algorithm, in: Proceedings of the 16th Annual Symposium on Computational Geometry, 1991, pp. 100–109.

[16] Z. Karni, C. Gotsman, Compression of soft body animation sequences, Computers and Graphics 28 (2004) 25–34.

[17] S. Katz, A. Tal, Hierarchical mesh decomposition using fuzzy clustering and cuts, ACM Transactions on Graphics 22 (3) (2003) 954–961.

[18] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, The Bell System Technical Journal 49 (2) (1970) 291–307.

[19] R. Koenen, Overview of the MPEG-4 standard, Moving Picture Experts Group (2000).

[20] J. Lengyel, Compression of time dependent geometry, in: Symposium on Interactive 3D Graphics, 1999, pp. 89–95.

[21] J. Li, C. Kuo, A dual graph approach to 3d triangular mesh compression, in: Proceedings of the IEEE International Conference on Image Processing, 1998.

[22] X. Li, T. Toon, T. Tan, Z. Huang, Decomposing polygon meshes for interactive applications, in: Proceedings of the Symposium on Interactive 3D Graphics, 2001, pp. 35–42.

[23] R. Liu, H. Zhang, Segmentation of 3d meshes through spectral clustering, in: Proceedings of Pacific Graphics, 2004, pp. 298–305.

[24] S.P. Lloyd, Least squares quantization in pcm, IEEE Transactions on Information Theory 18 (2) (1982) 129–137.

[25] K. Muller, A. Smolic, M. Kautzner, P. Eisert, T. Wiegand, Predictive compression of dynamic 3d meshes, in: ICIP05, 2005, pp. 589–592.

[26] R. Pajarola, J. Rossignac, Compressed progressive meshes, IEEE Transactions on Visualization and Computer Graphics 6 (1) (2000) 79–93.

[27] F. Payan, M. Antonini, Wavelet-based compression of 3d mesh sequences, in: Proceedings of IEEE ACIDCA-ICMI '2005, 2005.

[28] J. Rossignac, Edgebreaker: Connectivity compression for triangle meshes, IEEE Transactions on Visualization and Computer Graphics 5 (1) (1999) 47–61.

[29] M. Sattler, R. Sarlette, R. Klein, Simple and efficient compression of animation sequences, in: SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2005, pp. 209–217.

[30] A. Shamir, C. Bajaj, V. Pascucci, Multi-resolution dynamic meshes with arbitrary deformations, in: Proceedings of the Conference on Visualization '00, 2000, pp. 423–430.

[31] H. Simon, Partitioning of unstructured problems for parallel processing, in: Proceedings of the Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications, Pergammon Press, 1991.

[32] N. Stefanoski, J. Ostermann, Connectivity-guided predictive compression of dynamic 3d meshes, in: ICIP06, 2006, pp. 2973–2976.

[33] G. Taubin, J. Rossignac, Geometric compression through topological surgery, ACM Transactions on Graphics 17 (2) (1998) 84–115.

[34] C. Touma, C. Gotsman, Triangle mesh compression, in: Proceedings of Graphics Interface, 1998, pp. 26–34.

[35] S. Varakliotis, J. Ostermann, V. Hardman, Coding of animated 3d wireframe models for internet streaming applications, in: Proceedings of International Conference on Multimedia and Expo., 2001, pp. 353–356.

[36] Y. Weiss, Segmentation using eigenvectors: a unifying view, in: International Conference on Computer Vision, 1999, pp. 975–982.

[37] J. Yang, C. Kim, S. Lee, Compression of 3-d triangle mesh sequences based on vertex-wise motion vector prediction, IEEE Transactions on Circuits and Systems for Video Technology 12 (12) (2002) 1178–1184.

[38] H. Zhang, R. Liu, Mesh segmentation via recursive and visually salient spectral cuts, in: Proceedings of Vision, Modeling and Visualization, 2005.