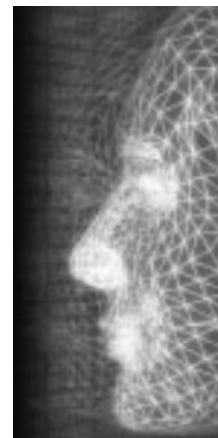


Animating Geometrical Models

Mesh decomposition using motion information from animation sequences

By Tong-Yee Lee*, Ping-Hsien Lin, Shaur-Uei Yan and Chun-Hao Lin



In computer graphics, mesh decomposition is a fundamental problem and it can benefit many applications. In this paper, we propose a novel mesh decomposition algorithm using motion information derived from a given animation sequence. The proposed algorithm first use principal component analysis (PCA) to construct a compact representation of a given animation sequence. Next, from this representation, we derive several motion parameters including motion complexity and similarity. Finally, we decompose a given mesh into sub-meshes using derived motion information and subdivide the triangles along the cutting paths for the smoother borders between the mesh parts. Our experimental results show that this new decomposition scheme can bring the benefit of good compression ratios on animation sequences. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: mesh decomposition; PCA; motion complexity; motion similarity; compression

Introduction

Overview

Mesh decomposition or segmentation is an important problem in computer graphics. Good mesh decomposition schemes can benefit many applications. Decomposition of a given polygonal mesh partitions the mesh into connected subsets of meshes. According to the types of the partitioned subsets, Shamir¹⁶ classifies mesh segmentation techniques into two different categories: patch-type and part-type methods.

Generally, the patch-type methods always generate disk-like patches and the part-type methods attempt to partition the mesh into 'meaningful' components. Patch-type methods are usually used for texture mapping,¹ surface parameterization,² and morphing appli-

cations^{3–7} because geometric property such as planarity and convexity can be well-maintained in the planner patches. In many applications, the requirement of low-distortion after 3D-to-2D surface parameterizations is crucial and therefore the patch-type approach is a naturally better choice than the part-type approach. On the other hand, the part-type approach is more suitable for applications that need to explicitly identify some sub-parts of a model. For example, in shape matching⁸ and modeling by parts,⁹ it is better to decompose the whole object into some feature-salient sub-parts to assist in recognition of object parts or to facilitate the modification of models.

The mesh decomposition can be executed in either a fully automatic or a semi-automatic manner. In this paper, the proposed method is a fully automatic decomposition algorithm and it generates both patch-type and part-type sub-meshes. The major contribution of this paper is a novel mesh decomposition using motion information from a given animation sequence. To the best of our knowledge, this paper might be the first to use the motion information to decompose a mesh model. Most previous work uses the information derived from a static model only to perform mesh decomposition. In this paper, the decomposition method based

*Correspondence to: Tong-Yee Lee, Computer Graphics Group/Visual System Laboratory, Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan. E-mail: tonylee@mail.ncku.edu.tw

Contract/grant sponsor: National Science Council, Taiwan; contract/grant numbers: NSC-93-2213-E-006-026; NSC-93-2213-E-006-060; NSC 94-2213-E-006-005.

on motion information can be potentially beneficial for applications such as compression of animation sequences or generate better level-of-details of models with dynamic motions. We apply the proposed method to the compression of animation sequences. Our experimental results show that this new decomposition can achieve good compression ratios of animation sequences.

Related Work

In the past, many related schemes have been proposed to decompose meshes. Gregory *et al.*³ and Zockler *et al.*⁴ manually decompose the meshes for their metamorphosis applications. This partitioning task can be very tedious and time-consuming. To minimize the need for this manual effort, Shlafman *et al.*⁵ present a k -means based clustering algorithm to partition given models into several meaningful and compatible components. Garland *et al.*¹⁰ propose a face clustering based on the planarity metric to decompose meshes. The created face hierarchy is very useful in many applications such as collision detection and surface simplification. Sander *et al.*² use iterative clustering scheme to decompose a given mesh into several charts for geometry image creation. Their clustering metrics include geometric distance between neighboring faces, and difference between face normal and the chart normal. Mangan and Whitaker¹¹ generalize the watershed method to segment meshes by using either Gaussian curvature or the norm of the covariance of adjacent face normals as the height field at each vertex. Similarly, Razdan and Bae¹² describe a hybrid approach based on curvature information from the meshes for region partitions. Katz and Tal¹³ propose a fuzzy clustering scheme to hierarchically decompose meshes and the final cuts on the region boundaries can be further refined using some minima rule. This method benefits in control-skeleton extraction of meshes for animation and deformation applications. In some applications such as mesh editing, it is necessary to provide some intelligent tools to assist users in manually scissoring meshes. Lee *et al.*¹⁴ and Funkhouser *et al.*⁹ support intelligent tools and allow easy manipulation to intuitively scissor meshes. For mesh compression purpose, Karni and Gotsman¹⁵ partition meshes into sub-meshes to reduce computation cost in mesh compression using spectral analysis. Finally, for more related work, please refer to Shamir's work¹⁶ excellent survey on mesh decomposition.

Animation Parameterization Using Principal Components

Alexa and Muller¹⁷ apply principal components analysis (PCA) to an animation sequence and then utilize a small number of principal animation components to reconstruct the whole animation sequence. This PCA-based representation decouples the animation and geometry information by factorizing an affine mapping $T(t_i)$ for i th key-frame B_i with the first frame, and using Equation (1) to reconstruct the frame at time t :

$$A(t) = T^{-1}(t) \times \sum_k \hat{a}_k(t) \times \hat{B}_k \quad (1)$$

where $\hat{a}_k(t)$ s are the importance factors for the principal components and \hat{B}_k s are the principal component bases. In Equation (1), \hat{B}_0 represents the main structure of the geometry and $\hat{B}_1, \hat{B}_2, \hat{B}_3,$ and so on represent all the geometric deviations of \hat{B}_0 with decreasing importance. This technique supports progressive animation compression with high compression ratios.

Motion Analysis

The representation of principal components can decouple the animation from the underlying geometry. In this section, we use this representation to define some factors for evaluating the motion characteristic of a vertex or among the neighboring vertices. In this paper, the motion information on a vertex is selected as the basis for mesh partitions. Therefore, we use a vertex-based symbolism to rewrite the representation of principal components as follows. Assume the animation has M key-frames, the model has N vertices, and the user chooses the first L principal components to reconstruct the animation sequence. For the information of a vertex i in the k th frame, we rewrite its principal components representation using the first L principal components as:

$$\hat{v}_{ik} = T_k^{-1} \left(\left(\sum_{j=1}^L \langle V'_k - \bar{V}', E_j \rangle e_{ij} \right) + \bar{v}'_i \right) \quad (2)$$

where T_k is the affine transformation that maps the k th frame V_k to the first frame, V'_k is the transformed k th key-frame, \bar{V}' is the transformed average shape, E_j is the j th eigenvector, e_{ij} is the information of vertex i in E_j , and \bar{v}'_i is the transformed average information of vertex i .

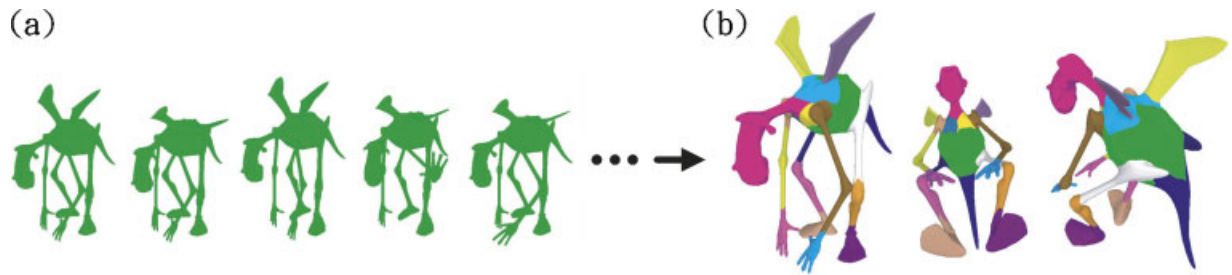


Figure 1. (a) An animation sequence from a given polygonal mesh and (b) different views of decomposed sub-meshes for a given model using the proposed method.

Note that the indices i, j, k do not always mean the same as in Equation (2). They should be read carefully in the following content.

Motion Complexity

The representation of animations using principal components treats a whole object as the subject for PCA. Therefore, these principal component bases are not individualized for each vertex. This representation reconstructs all the vertices by using the same number of bases. However, the motion characteristics of vertices on an object may be quite different. Among the first L user-chosen principal component bases, there may be some bases that are unnecessary for certain vertices. We

attempt to reduce this redundancy. First, we define the motion complexity c_i as the number of necessary eigenvectors for properly describing the motion of vertex i . We use Equation (3) to judge whether the j th base is needed for vertex i .

$$\left[\max_{k=1}^M (\langle V'_k - \bar{V}', E_j \rangle) - \min_{k=1}^M (\langle V'_k - \bar{V}', E_j \rangle) \right] \times \|e_{ij}\| \quad (3)$$

where $\|\cdot\|$ means vector norm. If the value of Equation (3) is below a user-specified threshold δ_c , the j th base is not needed. Moreover, since the eigen-analysis is performed on the whole object, the information of a vertex in different eigenvectors may not be orthogonal to each other. However, they may be highly correlated. When computing the motion complexity, we should treat

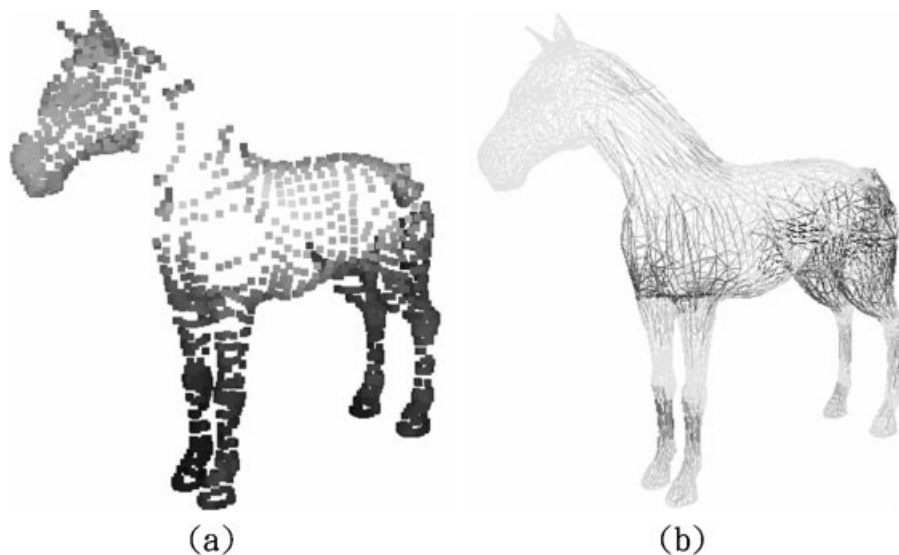


Figure 2. (a) Motion complexity visualization. A vertex with the higher gray-value indicates the lower motion complexity on this vertex. (b) Motion similarity visualization. An edge with the higher gray-value implies the higher motion similarity with two adjacent vertices sharing this edge.

highly correlated eigen-information as the same or similar contribution by only one base. We use a threshold λ_c and the absolute correlation defined in Equation (4) to do this judgment for the j th and k th bases.

$$\left| \frac{\langle e_{ij}, e_{ik} \rangle}{\|e_{ij}\| \|e_{ik}\|} \right| \quad (4)$$

where $|\cdot|$ means absolute value. If the value of Equation (4) is above λ_c , we decrease motion complexity c_i by one. In this manner, we refine the value of motion complexity. After the motion complexity on each vertex is defined, we smooth out the value c_i of each vertex by averaging it with those of their one-ring neighbors. Figure 2(a) visualizes the motion complexity of a running horse (see Figure 6(a)). We can see that the horse's head has lower motion complexity than those of four feet.

Motion Similarity

Next, to judge the motion consistency between neighboring vertices, we define another parameter called motion similarity s_{ij} for two neighboring vertices i and j as:

$$s_{ij} = \frac{1}{L} \sum_{k=1}^L \frac{\langle e_{ij}, e_{ik} \rangle}{\|e_{ij}\| \|e_{ik}\|} \times \frac{4(c_i + \varepsilon)(c_j + \varepsilon)}{(c_i + c_j + 2\varepsilon)^2} \quad (5)$$

In Equation (5), the first term evaluates the correlation of the information of vertex i and vertex j in the L eigenvectors. The second term prefers the neighboring vertices with less discrepancy in their motion complexity, c_i and c_j . Note that we add a small positive value ε in Equation (5). It is not only for preventing from division by zero, but also keeps the correct similarity for vertices with a zero motion complexity vertex. Figure 2(b) visualizes the motion similarity of the running horse. We can see those edges near the knees are of lower motion similarity. This phenomenon agrees with our expectations for the running horse in Figure 6(a).

Mesh Decomposition Using Motion Analysis

Most mesh decomposition methods are based on the analysis of geometric property. However, we observe that, in an animation sequence, the vertices on a patch of a properly decomposed model tend to move more

consistently than the vertices on other patches. Moreover, in this situation, the features of the motion information in animations are usually more obvious than those of geometry.

Decomposition by Region Growing

After motion analysis, we simply use a local-greedy region growing approach to group the vertices. The criterion that determines if a vertex can be added to an existing group is motion similarity obtained in Equation (5). The algorithm starts with a seed and grows a sub-mesh (or patch) incrementally as follows. When the region growing processes a new vertex, we check whether its motion similarity with its neighbors that were already grouped into the current patch is higher than a threshold. If the answer is yes, this vertex will be included. After the region growing has been finished, the decomposed model may look fragmentary (i.e., over-segmented) due to small regions such as isolated vertices/edges/faces that cannot be grouped to any neighboring group. Then, we perform a merging procedure to group it into the neighboring patches according to their motion similarity with less restricted tolerance. In addition, if the size of an isolated element is too small, i.e., below a threshold, this isolated element is merged into the nearest group to reduce over-segmentation. Figure 3 shows the decomposition of the running horse after region growing. Since the above procedure is a vertex-based method, the decomposed

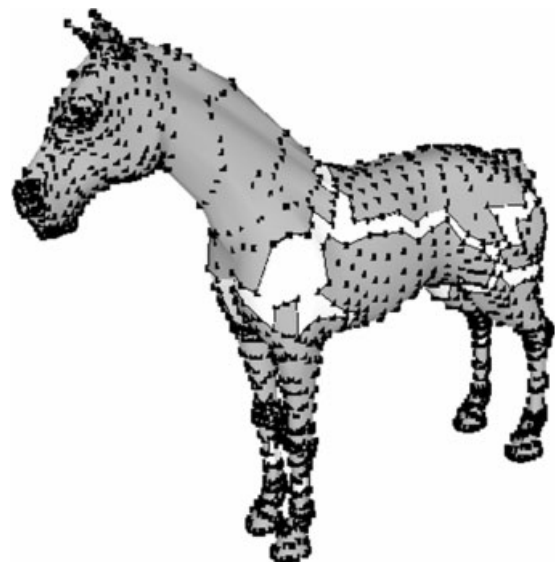


Figure 3. The decomposition of the running horse after region growing and some gaps exist among partitioned patches.

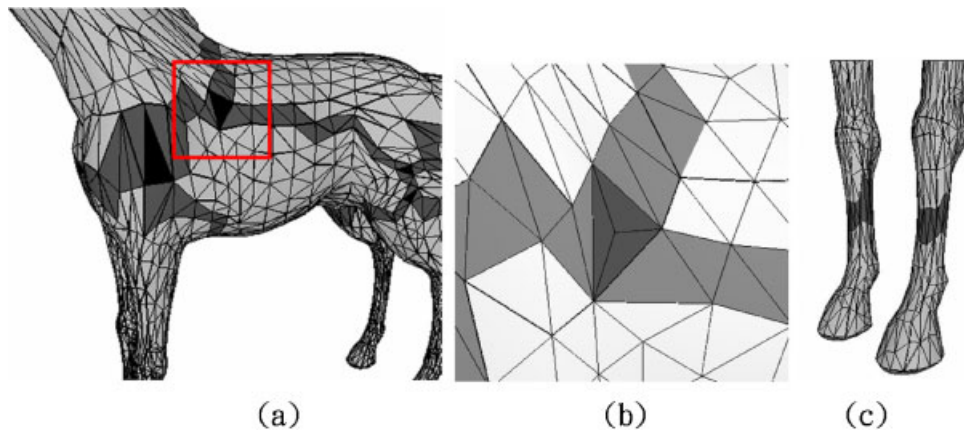


Figure 4. (a) The boundary gaps form a triangle strip network. (b) Re-triangulation of a corner triangle. This square area is the enlargement of the area enclosed by the red lines in (a). (c) The boundary gaps do not contain corner triangles.

result may have no hard boundary,¹² i.e., a triangle can not belong to any one neighboring patch. This is due to the three vertices of this isolated triangle can be part of three different patches. In Figure 3, the gaps between neighboring patches illustrate the ‘no-hard-boundary’ problem. Next, we will find a smooth boundary for each pair of neighboring patches.

Boundary Smoothing

First, we find out all the corner triangles whose vertices connect three different patches on the model (see the darkest triangles in Figure 4(a)). Then, we add a new vertex at the centroid of each corner triangle and re-triangulate it as shown in Figure 4(b).

We would like to find a smooth border between the centroids of two adjacent corner triangles. For this purpose, we first find the triangle strips between two adjacent corner triangles. Then, we grow the area along the two sides of the triangle strips by extra few layers constrained by a distance threshold. See an example in the region enclosed by the thick black lines in Figure 5(a). Next, this enclosed region is then flattened onto a 2D circle using the L^2 stretch criterion,¹⁸ as shown in Figure 5(b). Finally, we regularly resample this 2D circle (see Figure 5(c)) and use barycentric coordinate to map these samples back to the 3D model. We compute the normal for each sample, and find a smooth boundary path P passing through these samples by a minimization procedure guided by Equation (6).

$$\min \left(\sum_P \frac{|v_i - v_j|}{(1 + n_i \cdot n_j)} \right) \quad (6)$$

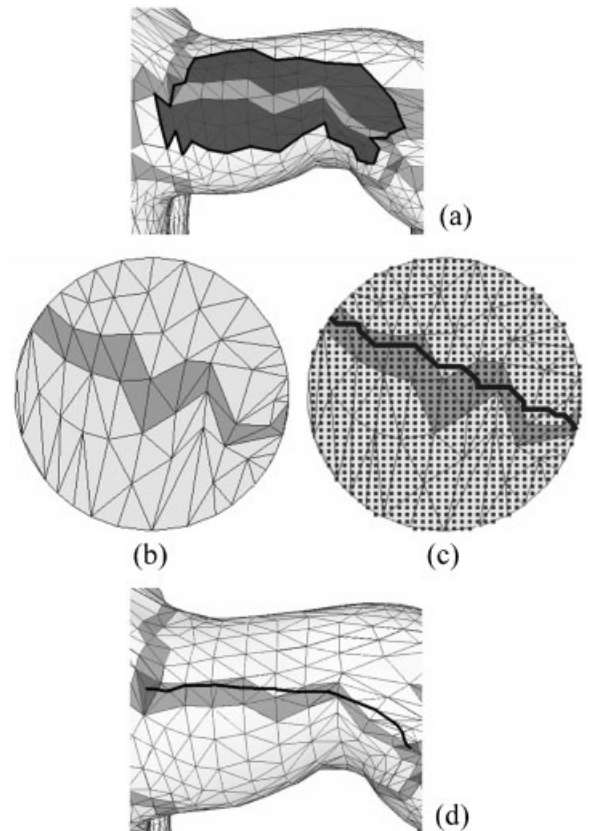


Figure 5. (a) The concerned region for determining the boundary of two adjacent corner triangles after growing extra few layers. (b) The concerned region in (a) is mapped to a 2D circle. (c) The final boundary path on 2D circle. (d) The final boundary path on 3D model.

where v_i and v_j are the positions of two neighboring sample i and j on P , and n_i and n_j are their normals. Figure 5(c) and (d) show the final boundary path between centroids of two adjacent corner triangles on 2D circle and on 3D model, respectively.

In cases, as shown in Figure 4(c), the boundary gaps do not contain corner triangles. In this situation, the boundary gap itself is a loop and each triangle in this gap is shared by two regions. This loop automatically partitions the mesh into two parts. Therefore, The proposed scheme can create two kinds of sub-meshes: patch-type and part-type regions according to Shamir's work¹⁶ classification. For the smoother borders, we also apply the similar smoothing procedure described above to this loop.

Results and Applications

Experimental Results

In this section, we demonstrate the usefulness of the proposed algorithm in the compression of animation sequences. The first example in Figure 1 is a part of Wally animation from Discrete Character Studio,

300 frames of the Wally's geometry that consists of 8880 vertices. This animation sequence requires an uncompressed size of $300 \text{ frames} \times 8880 \text{ vertices} \times 6 \text{ dimensions (vertex and normal)} \times 4 \text{ bytes} = 63\,936\,000$ bytes. In the accompanying video of this animation, we can observe the following motion features: (1) left hand is swinging, (2) feet and right hand are moving linearly, (3) head and tail are turning left and right, and (4) wings are swinging upwards and downwards. As a result in Figure 1(b), most decomposed regions follow the observed behaviors in this sequence. The second example in Figure 6(a) is a part of running horse animation, 96 frames of the horse's geometry that consists of 2982 vertices. This animation sequence requires an uncompressed size of 6 870 528 bytes. In this animation sequence, we also observe the following motion features: (1) head is moving gently, (2) tail is swinging upwards and downwards, and (3) four feet are running and knees are bending. Figure 6(a) (right) demonstrates the decomposed results rendered with two different views. In this example, the horse body is partitioned into three regions illustrated by yellow, orange, and white colors. This result is due to the orange region is near to four feet that have more dynamic motion.

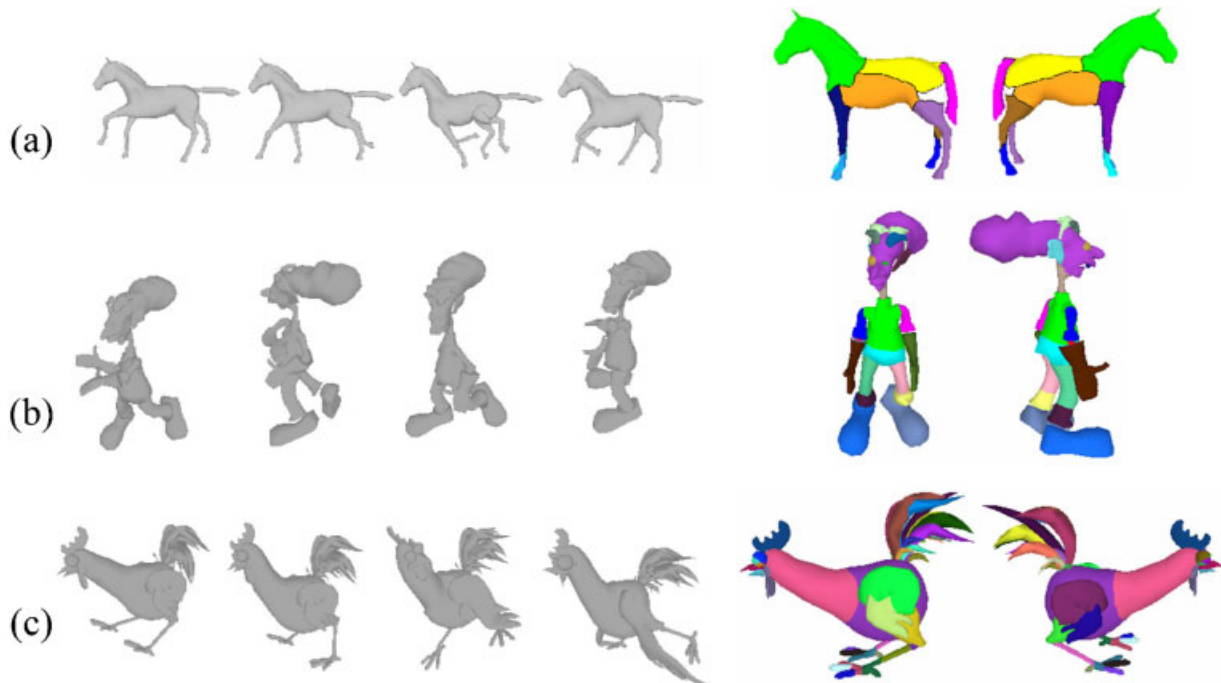


Figure 6. The decomposition examples using the proposed method. Left: A part of animation sequence. Right: Two different views of decomposed results.

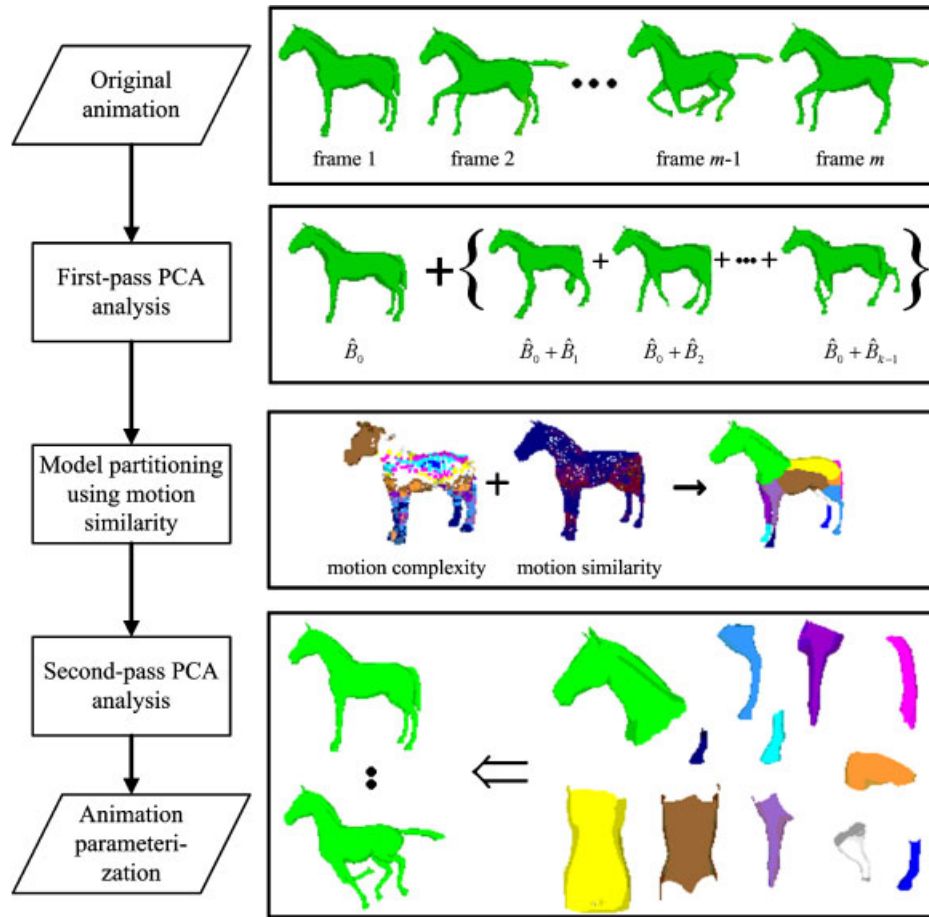


Figure 7. A flowchart of the animation compression scenario using two-pass principal component analysis (PCA).

Table 1 shows experimental configurations and compression ratios for these above two examples. In both examples, we use 20 PCA bases to reconstruct the animation sequence and we also show their PCA reconstructed errors. The horse and Wally are decomposed into 12 parts and 19 parts, respectively. After decomposition, we apply the second pass PCA to each individual part and then to reconstruct each part. As a result, on the average, we only need 10.1 and 7.0 PCA bases for

each part in horse and Wally cases, respectively. Experiments show that we can achieve less PCA errors as well as the higher compression ratios. Additionally, after the second pass PCA, we also evaluate the reconstruction error using distortion measure¹⁹ defined below.

$$\text{distortion} = 100 \times \frac{\|A - \tilde{A}\|}{\|A - \text{Average}(A)\|} \quad (7)$$

model size (byte)	method	size (byte)	L (patches)	PCA error	compression ratio	distortion %
63,963,000	PCA	4,499,520	20	0.00073	14.20952	0.21707
	2nd-PCA	2,211,888	7.0 (avg.) (19)	0.000105 (avg.)	28.90562	0.08311
6,870,528	PCA	1,510,608	20	0.03069	4.54819	0.86210
	2nd-PCA	882,168	10.1 (avg.) (12)	0.00915 (avg.)	7.78823	0.59927

Table I. Experimental configurations and compression ratios.

where A represents vertex and the normal information in the original animation sequence, \tilde{A} is reconstructed counterpart using PCA and A_{avg} is the average counterpart in the original sequence. Both cases show less distortion measurement once we apply 2nd-PCA to reconstruct animation sequences. Alexa and Muller¹⁷ propose a single pass PCA approach to compress an animation sequence. Based on our results, we have the following claim. Once we decompose models using 1st PCA motion information and then apply 2nd PCA to each individual part, we can potentially achieve better compression ratios than those of a single pass or called 1st PCA in this paper to the animation sequence.

In Figure 6(b) and (c) show a part of another two interesting animation sequences called *Dr. X* (300 frames, 4935 vertices and 20 components) and *Chicken Crossing* (400 frames, 3030 vertices, and 41 components). In these two examples, the original geometries were already decomposed into distinct components as they were created using animation software. Actually, not all available animation models were originally created as a single component such as horse and Wally. Alexa and Muller¹⁷ apply a PCA to the whole geometry of *Chicken Crossing* example and obtain a good compression ratio. Instead, we apply a PCA to each distinct component. In Table 2, as we expect, the compression ratio can be improved as each component is individually processed by a PCA. Therefore, these better results are still met with our claim. Moreover, we also attempt to further decompose an individual component into sub-components using the proposed algorithm if the size of a component is larger than a threshold. For example, the wings of the *Chicken* geometry are further decomposed into two parts. Additionally, the neck is separated from the body and claws are segmented from feet in this *Chicken Crossing* example, as these components have more dynamic motion. Then, we further apply a PCA to these newly decomposed components.

We term these cases are under 3rd PCA execution in contrast to the whole geometry executed by the 1st PCA like Alexa and Muller¹⁷ work. Both 2nd and 3rd PCAs approaches perform much better than the 1st PCA approach. However, as indicated in Table 2, the 3rd PCA approach only slightly improves compression ratios over the 2nd PCA method. This performance behavior is not surprising, because the animator decomposed models on purpose into several components for the requirement of animation motions. Parts with similar motion characteristics tend to be grouped in the same components. Therefore, the further PCA, i.e., 3rd PCA, is not always required if the models were already well partitioned.

Finally, for the page limit, we only show detailed decomposition information for a running horse example in Table 3. Twelve individual partitioned components are illustrated and the number of PCA base for each component and the reconstructed PCA errors are given.

Compression of Animation Sequence and Analysis

The flowchart of the proposed algorithm applied to the compression of an animation sequence can be depicted in Figure 7. In contrast to work,¹⁷ we term this compression approach as ‘animation compression using two-pass principal component analysis (PCA)’. Note that, if the mesh decomposition is only for the animation compression, we can omit the boundary smoothing step in the proposed method. Since the boundary smoothing will generate some new vertices that would decrease the compression ratio and a smoothing boundary is not needed for compression purpose. The original, uncompressed animation storage complexity is: NM , where N is the number of vertices and M is the number of key-frames. After using a single pass principal



model size (byte)	method	size (byte)	L (patches)	PCA error	compression ratio	distortion %
 35,532,000	PCA	12,082,440	100	0.99384	2.94080	2.64434
	2nd-PCA	8,734,992	39.85 (avg.) (20)	0.31035 (avg.)	4.06778	2.13774
	3rd-PCA	6,823,335	36.44 (avg.) (27)	0.10103 (avg.)	5.20742	2.08109
 29,088,000	PCA	3,788,720	50	0.16669	7.67753	2.34897
	2nd-PCA	2,432,104	17.00 (avg.) (41)	0.04033 (avg.)	11.96001	0.95843
	3rd-PCA	2,294,535	15.03 (avg.) (53)	0.00982 (avg.)	12.67708	0.81056

Table 2. Experimental configurations and compression ratios.













running horse								
patch	#PCA bases	PCA error	patch	#PCA bases	PCA error	patch	#PCA bases	PCA error
	7	0.00551		15	0.01863		3	0.00013
	9	0.00851		10	0.01178		11	0.01051
	18	0.01624		6	0.00407		8	0.00682
	9	0.00614		20	0.01573		5	0.00572

Table 3. Further decomposition information for a running horse example.

component representation like work,¹⁷ the storage complexity reduces to: $NK + MK + N$, where K is the number of principal components needed to reconstruct the animation, and usually $K \ll N$ and $K \ll M$. For the presented two-pass PCA approach, the storage complexity is

$$\left(\sum_i^L n_i \times k_i + k_i \times M \right) + N, \quad \sum_i n_i = N \quad (8)$$

where L is the number of decomposed components, n_i and k_i are the number of vertices in the i th component and the number of PCA bases for the i th component. In the animation sequence, if n_i is large and its corresponding k_i is small, i.e., less motion, the two-pass PCA approach can perform much better. Finally, we should note that we need to blend vertex information for display if a vertex is shared by different decomposed components.

Conclusions and Future Work

In this paper, we propose a novel mesh decomposition algorithm using motion information from an animation sequence. Experiments are demonstrated to successfully partition models into sub-meshes. Results fit well motion characteristics inherent in a given animation sequence. It is evident that our mesh decomposition technique relies on proper animations. This means if a joint is not swinging in the animation sequences, it will not be segmented at or near this joint. But in our test data, we found that our mesh decomposition technique is not very sensitive to the animations. If a joint has swinging slightly among the animation, it will be decomposed properly near the joint. This

phenomenon is especially obvious in the Wally case: the right hand and two feet of Wally only stretch and swing slightly among the animation, but the decomposition result is good, see Figure 1. The proposed algorithm is, in particular, useful to handle models with dynamic motions. For example, we apply the proposed algorithm to the compression of animation sequences. As a result, the compression ratios for tested animation sequences are better than the method using the single-pass PCA such as work.¹⁷ In future, we plan to apply the proposed method to generate LOD geometry of an animation sequence. In the LOD application, the models are decomposed into sub-meshes according to motion similarity. Then, we can apply LOD simplification technique to each sub-mesh. For example, in the Chicken Crossing example, once each claw is segmented apart, when we individually simplify each claw instead of the whole foot, the shape of each claw can be maintained well in much simpler manner using LOD simplification techniques. In this paper, the major factor considered is motion information in the proposed algorithm. In future, for better decomposition, other geometric metrics such as curvature or concavity and geodesic or angular distance can be included, too. In addition, we also would like to investigate better schemes to compress animation sequences such as including linear prediction coding (LPC) plus PCA like work.¹⁹ For colour images in this paper, please see them at http://couger.csie.ncku.edu.tw/~vr/CAV79/color_figures.htm.

ACKNOWLEDGEMENTS

The *Chicken Crossing* animation sequence is courtesy of Dr. Andrew Glassner at Microsoft Research. The *Wally* and *Dr. X* are courtesy of Discrete Character Studio.

References

1. Levy B, Petitjean S, Ray N, Mallot J. Least squares conformal maps for auto-matic texture atlas generation. In *SIGGRAPH'02 Proceedings*, 2002; pp. 362–371.
2. Sander P, Wood Z, Gortler S, Snyder J, Hoppe H. Multi-chart geometry images. In *Proceedings of Eurographics Symposium on Geometry Processing*, 2003; pp. 146–155.
3. Gregory A, State A, Lin M, Manocha D, Livingston M. Interactive surface decomposition for polyhedral morphing. *The Visual Computer* 1999; **15**(9): 453–470.
4. Zockler M, Stalling D, Hedge H-C. Fast and intuitive generation of geometric shape transitions. *The Visual Computer* 2000; **16**(5): 241–253.
5. Shlafman S, Tal A, Katz S. Metamorphosis of polyhedral surfaces using decom-position. *Eurographics'02 Proceedings* 2002; **21**(3): 219–228.
6. Lee T-Y, Huang P-H. Fast and intuitive metamorphosis of 3d polyhedral models using smcc mesh merging scheme. *IEEE Transactions on Visualization and Computer Graphics* 2003; **9**(1): 85–98.
7. Lin C-H, Lee T-Y. Metamorphosis of 3d polyhedral models using progres-sive connectivity transformations. *IEEE Transactions on Visualization and Computer Graphics* 2005; **11**(1): 2–12.
8. Zuckerberger E, Tal A, Shlafman S. Polyhedral surface decomposition with applications. *Computer and Graphics* 2002; **26**(5): 733–743.
9. Funkhouser T, Kazhdan M, Shilane P, et al. Modeling by example. *SIGGRAPH'04 Proceedings*, 2004; pp. 649–660.
10. Garland M, Willmott A, Heckbert P. Hierarchical face clustering on polygonal surfaces. *Proceedings of ACM Symposium on Interactive 3D Graphics*, 2001; pp. 49–58.
11. Mangan A, Whitaker R. Partitioning 3d surface meshes using watershed segmen-tation. *IEEE Transactions on Visualization and Computer Graphics* 1999; **5**(4): 308–321.
12. Razdan A, Bae M. A hybrid approach to feature segmentation of triangle meshes. *Computer-Aided Design* 2003; **35**: 783–789.
13. Katz S, Tal A. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH'03 Proceedings*, 2003; pp. 954–961.
14. Lee Y, Lee S, Shamir A, Cohen-Or D, Seidel G-P. Intelligent mesh scissoring using 3d snakes. In *The Proceedings of Pacific Conference on Computer Graphics and Applications*, 2004; pp. 279–287.
15. Karni Z, Gotsman C. Spectral compression of mesh geometry. In *SIGGRAPH'00 Proceedings*, 2000; pp. 279–286.
16. Shamir A. A formation of boundary mesh segmentation. In *Proceedings of the second International Symposium on 3DPVT (3D Data Processing, Visualization, and Trans-mission)*, 2004; pp. 82–89.
17. Alexa M, Muller W. Representing animation by principal components. In *Euro-graphics'02*, 2002; pp. 411–418.
18. Sander P, Snyder J, Gortler S, Hoppe H. Texture mapping progressive meshes. *SIGGRAPH'01 Proceedings*, 2001; pp. 409–416.
19. Karni Z, Gotsman C. Compression of soft-body animation sequences. *Computer and Graphics* 2004; **28**: 25–34.

Authors' biographies:



Tong-Yee Lee was born in Tainan county, Taiwan in 1966. He received his B.S. in Computer Engineering from Tatung Institute of Technology in Taipei, Taiwan, in 1988, his M.S. in Computer Engineering from National Taiwan University in 1990, and his Ph.D. in Computer Engineering from Washington State University, Pullman, in May 1995. Now, he is a professor in the Department of Computer Science and Information Engineering at National Cheng-Kung University in Tainan, Taiwan. He serves as a guest associate editor for *IEEE Transactions on Information Technology* in Biomedicine from 2000 to 2005. His current research interests include computer graphics, non-photorealistic rendering, image-based rendering, visualization, virtual reality, surgical simulation, distributed and collaborative virtual environment. He leads a Computer Graphics Group/Visual System Lab at National Cheng-Kung University (<http://couger.csie.ncku.edu.tw/~vr>). He is a member of the IEEE.



Ping-Hsien Lin received his B.S. degree in Mechanical Engineering and his Ph.D. in Computer Engineering from National Cheng-Kung University, Taiwan, in 1993 and 2004, respectively. He is currently an assistant professor in the Department of Computer Science and Information Engineering at National Changhua University of Education in Taiwan. His research interests include computer graphics, computer vision, image-based rendering.



Shaur-Uei Yan received his B.S. degree in Civil Engineering from National Taiwan University, Taiwan, in 2000. He is currently working toward his Ph.D. in the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interests include computer graphics and mesh parameterization.



Chun Hao Lin received his B.S. degree from his Department of Computer Science, National Chengchi University, Taipei, Taiwan, in 2000, and his M.S. degree from the Department of Computer Science and Information Engineering, National Cheng-Kuang University, Tainan, Taiwan, in 2004. And his research interests include computer graphics, computer vision and image processing.

Copyright of Computer Animation & Virtual Worlds is the property of John Wiley & Sons, Inc. / Engineering. The copyright in an individual article may be maintained by the author in certain cases. Content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.