The Visual Computer manuscript No.
(will be inserted by the editor)

Stefanie Wuhrer · Alan Brunton

Segmenting Animated Objects Into Near-Rigid Components

Abstract We present a novel approach to solve the problem of segmenting a sequence of animated objects into near-rigid components based on k given poses of the same non-rigid object. We model the segmentation problem as a clustering problem in dual space and find near-rigid segments with the property that segment boundaries are located at regions of large deformation. The presented approach is asymptotically faster than previous approaches that achieve the same property and does not require any user-specified parameters. However, if desired, the user may interactively change the number of segments. We demonstrate the practical value of our approach using experiments.

1 Introduction

Mesh segmentation is an important tool in computer graphics. Different applications require mesh segmentations such as morphing, texture mapping, mesh simplification, and skeleton extraction. The type of mesh segmentation that is required depends strongly on the application. In the following, we assume that an animated object is given as triangular mesh and we focus on near-rigid mesh segmentations. For a recent survey on mesh segmentations in general, refer to Shamir [17].

Segmenting a mesh into near-rigid components based on a given set of deforming input meshes has various applications in geometry processing and computer graphics such as skeleton extraction [7] and morphing [7], [13], [21].

Most previous methods segment one given static mesh [9], [14], [15], [21]. Recently, several methods were proposed that consider segmenting a mesh based on a given set of deforming meshes with known point-to-point correspondence [8], [7], [16], [12]. Katz et al. [8] transform the given poses into a multi-dimensional space with the property that all of the poses are similar in this space. James and Twigg [7] consider the rotational sequences between corresponding triangles in different poses. Each rotational sequence is mapped to a point in a high-dimensional space and the near-rigid components are found using mean-shift clustering in this space. This clustering is used for skeleton extraction and animation. Although James and Twigg find near-rigid components, they do not find a segmentation of the mesh because some triangles in deforming regions do not belong to any cluster. Sattler et al. [16] pursue a similar approach. However, unlike James and Twigg, they do not analyze the motion of each triangle of the mesh, but the motion of each vertex of the mesh. Vertices with similar motions are clustered to obtain a segmentation. However, the experimental results show that a near-rigid segmentation is not always obtained. Lee et al. [12] propose an algorithm to find a near-rigid segmentation. The algorithm initially finds feature triangles on the mesh and uses those to grow clusters. The clusters are grown according to a distance metric between triangles based on a combination of geodesic distances and deformation distances. Although visually pleasing results are

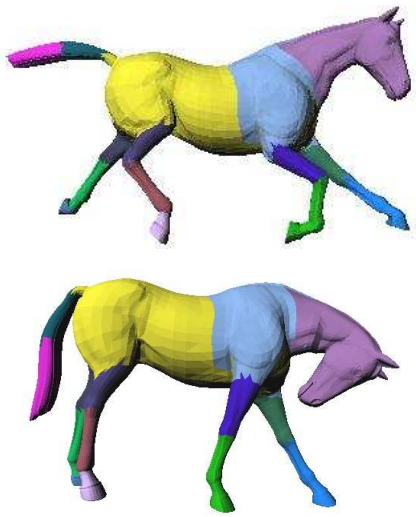

Fig. 1 *Two segmented poses of a horse model.*

Research supported in part by HPCVL.

National Research Council of Canada, Ottawa, Canada
E-mail: {stefanie.wuhrer, alan.brunton}@nrc-cnrc.gc.ca

obtained, the algorithm is too slow for practical purposes since all pairwise distances between triangles are computed. Furthermore, the algorithm requires user-specified parameters.

We propose a novel method to segment a mesh into near-rigid components. Given k poses of the same non-rigid object as triangular meshes $S^{(0)}, \ldots, S^{(k-1)}$ with known point-to-point correspondences, we aim to partition the mesh into near-rigid segments with the property that segment boundaries are located at regions of large deformation. Let the meshes $S^{(0)}, \ldots, S^{(k-1)}$ contain n vertices each. Our approach works for 2-manifold meshes of arbitrary topology.

We achieve this goal by modeling the segmentation problem as a clustering problem in dual space. Our algorithm runs in $O(k^2 n + n \log n)$ time, which is a significant improvement over the algorithm by Lee et al. that takes $\omega(n^2)$ time. Furthermore, our algorithm does not require any user-specified parameters. However, if desired, the user may interactively change the number of segments.

2 Segmentation Using Minimum Spanning Tree in Dual Space

This section shows how a near-rigid segmentation is computed using a clustering approach in dual space. The segmentation has the property that segment boundaries are located at regions of the largest deformation.

We start with k poses $S^{(0)}, \ldots, S^{(k-1)}$ of the same non-rigid object given as triangular meshes. We assume that the meshes $S^{(0)}, \ldots, S^{(k-1)}$ share the same underlying mesh structure M. Hence, we know the mesh structure M with k sets of ordered vertex coordinates $V^{(0)}, \ldots, V^{(k)}$ in $\mathbb{R}^3$. Let M contain n triangles.

To find a near-rigid segmentation of M, we make use of the *dual graph* $D(M)$ of M. The dual graph $D(M)$ has a node for each triangle of M. We denote the dual node corresponding to face f of M by $D(f)$. Two nodes of $D(M)$ are joined by an arc if the two corresponding triangles in M share an edge. We denote the dual arc corresponding to an edge e of M by $D(e)$. Note that the dual graph is merely an abstract graph that captures the connectivity information of M. The dual vertices $D(f)$ of $D(M)$ are not embedded.

We assign a weight to each edge e of $D(M)$. The weight of e is equal to the maximum difference in dihedral angle of the supporting planes of the two triangles of M corresponding to the two endpoints of e. That is, we compute the dihedral angle between the two supporting planes of the two triangles of M corresponding to the two endpoints of e for all the poses $S^{(0)}, \ldots S^{(k-1)}$. The weight of e is then set as the maximum difference between any pair of dihedral angles. This weight corresponds to the change in dihedral angle during the deformation. The weight can therefore be seen as a measure of rigidity. The smaller the weight, the smaller the change in dihedral angle between the two triangles during the deformation, and the more rigidly the two triangles move with respect to each other.

Next, we find a clustering in dual space. The goal is to find a segmentation with the property that segment boundaries are located at regions of the largest deformation. Note that regions of largest deformation in M correspond to edges with large weights in $D(M)$. Hence, we can find the sought segmentation by finding a partition of $D(M)$ consisting of d clusters that has the property that the smallest distance between any pair of clusters is largest among all partitions of $D(M)$ that consist of d clusters. We call a partition with this property a *farthest d-partition of $D(M)$*.

More formally, a farthest d-partition is defined as follows. Let $\delta_{p,q}$ denote the weight of the edge connecting p and q. A partition $\mathcal{P}$ of a set M into d disjoint clusters $C_1, \ldots, C_k$ is called a *d-partition of M*. A d-partition $\mathcal{P}$ is characterized by the *inter-cluster distance*

$$D_{int}(\mathcal{P}) = \min_{C_i \neq C_j \in \mathcal{P}} \min_{p \in C_i, q \in C_j} \delta_{p,q}.$$

A farthest d-partition is a d-partition with largest inter-cluster distance.

If the number d of clusters is known, then a farthest d-partition of $D(M)$ can be found as follows. First, a minimum spanning tree $T(M)$ of $D(M)$ is computed. Second, the $d-1$ edges of $T(M)$ that have the largest weights are deleted from $T(M)$. This partitions $T(M)$ into d clusters. Kleinberg and Tardos [10, p. 160] prove that this algorithm computes a farthest d-partition of $D(M)$.

As the number d of clusters is unknown in our case, we modify the second step of the algorithm by Kleinberg and Tardos to delete all the edges of $T(M)$ that have weights larger than a threshold t_1. This results in a farthest partition of $D(M)$. It remains to compute the threshold t_1.

To compute t_1 in a fully automatic manner, we analyzed the distribution of the edge weights for a given set of input meshes. We found that the distribution of the edge weights resembles an exponential distribution. The distribution for the Alien example discussed in Section 5 is shown in Figure 2. We use the known edge weights to learn the underlying exponential distribution via maximum likelihood estimation. We then set t_1 to the third quartile of the learned distribution. This way we are expected to keep 75% of the edges in $T(M)$.

This approach results in a farthest partition of $D(M)$. Once a partition of $D(M)$ is known, we can easily compute the corresponding segmentation for M. Each cluster of $D(M)$ corresponds to a connected set of triangles of M. These sets of triangles are the computed near-rigid segments of M. As the partition of $D(M)$ is farthest, the near-rigid segmentation of M has the property that segment boundaries are located at regions of the largest deformation.

As the algorithm only computes edge weights and a minimum spanning tree $T(M)$, the algorithm's running time is $O(k^2 n + n \log n)$.

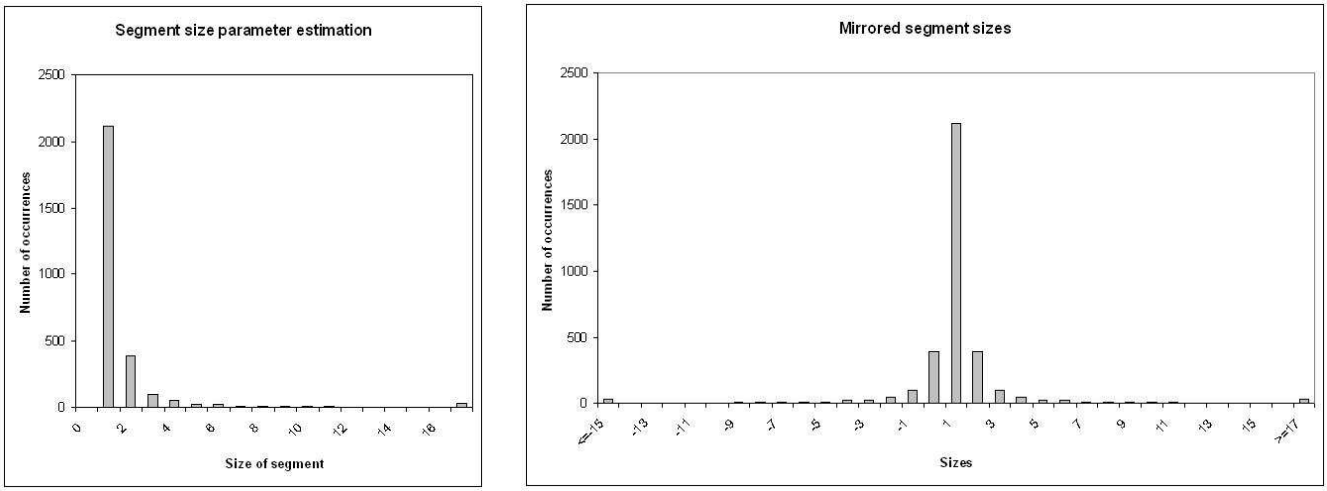

Fig. 3 *Distribution of the segment sizes. The left side shows the distribution. The right side shows the Gaussian obtained by mirroring the values along $x = 1$.*

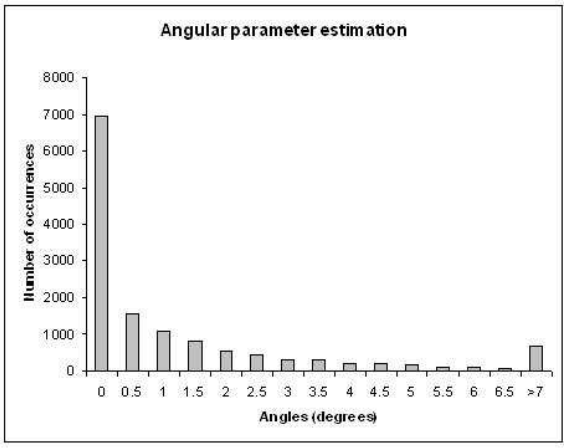

Fig. 2 *Distribution of the edge weights.*

3 Merging Small Segments

The segmentation corresponding to a farthest clustering of $D(M)$ may contain a large number of small segments. In many applications such as metamorphosis and skeleton extraction, small segments are undesirable. This section therefore shows how to merge small segments with neighboring segments in a fully automatic manner.

This section considers two subproblems: evaluating which segments are considered to be small, and merging small segments with neighboring segments. Assume that we are given a threshold t_2 such that any segment with less than t_2 elements is considered to be small and any segment with at least t_2 elements is not considered to be small. We will discuss later how to compute t_2 in a fully automatic way.

To merge small segments with neighboring segments, we find the edge in $T(M)$ with minimum weight that connects a small segment to a neighboring segment. Once the edge is found, we merge the two segments joined by the edge and repeat. The algorithm terminates once no segment is considered to be small. This algorithm maintains the property that segment boundaries are located at regions of large deformation because we merge along the least deforming edges that are available. More precisely, the resulting segmentation S has the property that S has the segment boundaries along edges of largest deformation among all segmentations with minimum segment size t_2.

It remains to compute the threshold t_2. To compute t_2 in a fully automatic way, we analyzed the distribution of the segment sizes for a given set of input meshes. The distribution for the Cat example discussed in Section 5 is shown in Figure 3. The left side shows the distribution of the segment sizes. Note that each segment contains at least one element. The right side shows the distribution obtained by mirroring the values along $x = 1$. We see that the resulting distribution resembles a Gaussian distribution. Clearly, the obtained distribution is not a true Gaussian distribution because each segment size is an integer value. Nonetheless, we model the mirrored segment sizes as a Gaussian distribution. We use the known segment sizes to learn the underlying normal distribution (μ, σ^2) via maximum likelihood estimation. We then set t_2 to $\mu + 3\sigma$. This way, 99.7% of the segments are expected to be considered small.

We can implement this algorithm using a priority queue. Since we need to merge at most $n - 1$ times, the running time of the merge step is $O(n \log n)$.

Note that the approach taken in this paper first computes too many segments and merges them subsequently. The reason for this is that the correct number of segments is initially unknown. It is therefore not straightforward to tell where the over segmentation of the model will occur before an initial clustering is computed. Our two-step approach overcomes this problem in an elegant way.

4 Interactively Changing the Number of Segments

The previous sections outline how to segment a mesh into near-rigid components using a fully automatic approach that does not require any user-specified parameters. However, since different applications require different degrees of segmentation, it may be desirable to the user to interactively change the number of segments found by the algorithm. This section presents two ways how the user can change the number of segments in a time efficient way. In Section 5, we demonstrate that for medium-size datasets (containing up to 15000 triangles), the interaction is performed in real-time.

The number of segments computed by our two-step algorithm depends both on t_1 and t_2. Denote the number of segments obtained by cutting edges of the dual tree with weight above t_1 by d_1. Denote the number of segments obtained after merging all of the segments with at most t_2 triangles by d_2. Clearly, $d_2 \leq d_1$. We call the automatically computed segmentation containing d_2 segments the *base segmentation*.

We allow the user to efficiently change the segmentation in two ways. First, the user can set the number of segments to any number d between d_2 and d_1. We achieve this by storing the list L of edges that are added during the merge step in the order in which they are added during the merge step. When the algorithm terminates, d_2 segments are present in the base segmentation. If the user desires d segments, we remove the last $d - d_2$ edges in L from the dual graph. This yields d segments with the property that each segment in the new segmentation is a subset of a segment in the base segmentation. That is, no new segments are formed that contain triangles from two or more segments of the base segmentation. This is a desirable property as it allows the user to obtain a segmentation hierarchy. The disadvantage of this segmentation hierarchy is that small segments may occur.

Second, observe that by varying t_2 and by performing the merge step outlined in Section 3 with this updated parameter, we can adjust the number of segments. By increasing t_2 above the size of the smallest existing segment s in the current segmentation, s is merged with another segment and the number of segments decreases. By decreasing t_2 to the size of the largest segment l that was merged in the previous step, l is not merged to another segment and the number of segments increases. Hence, we adjust the number of segments by varying the parameter t_2 and by performing the merge step with this new parameter. In order to allow this update in an efficient way, we store the components c obtained after deleting edges from the dual tree $T(M)$ based on t_1 as outlined in Section 2. For each user interaction, we adjust the threshold t_2 and perform the merge step starting from the components c. Note that this adjustment does not yield a segmentation hierarchy. However, no small fragments occur since the size of the smallest segment is bounded.

5 Experiments

This section presents experiments using the algorithm presented in this paper. The experiments were conducted using an implementation in C++ on an Intel (R) Pentium (R) D with 3.5 GB of RAM. OpenMP was used to improve the efficiency of the algorithms. To compute the minimum spanning tree $T(M)$ and to find connected components of a graph, the boost graph library [19] was used.

We first present the segmentation obtained using our algorithm and subsequently compare our algorithm to previous methods.

5.1 Results

The first experiment shows the computed near-rigid segments of a set of alien models. The alien model is chosen from the Princeton Shape Benchmark[1] [18] and animated to obtain multiple postures with known correspondences using the automatic technique by Baran and Popović [2]. The models contain 13664 triangles. The given input poses to our algorithm with the final near-rigid segmentation are shown in the last four columns of Figure 4. The near-rigid segmentation before the merge step is shown in the first two columns of Figure 4. We can see that the merge step is necessary to prevent fragmented models. The final result captures the near-rigid correspondences learned from the given input poses. Note how the segmentation boundary along the waist separates distinct regions that deform non-rigidly. A similar segmentation boundary occurs between the hands and the arms. Note that since the alien does not bend its right knee in any of the given poses, the upper and lower right leg belong to the same segment.

We demonstrate the accuracy of our approach by comparing our result to the ground truth for this experiment. The ground truth is found by assigning each vertex to the segment (or bone) that obtains the largest weight during the skinning phase by Baran and Popović's algorithm. The ground truth segmentation C' consists of 17 segments and is shown on the left of Figure 5. The segmentation C computed by our algorithm assigns each triangle to a segment. We assign each vertex of the mesh to the segment that contains most of its incident triangles. We evaluate the accuracy of our segmentation using the dice measure. That is, for each ground truth segment c' in C', we find the percentage of c' covered by the corresponding segment c in C. The corresponding segment c in C of a segment c' in C' is found as the segment of C that maximizes the number of overlapping vertices.

We show two ground truth tests. First, we compare the segmentation C shown in Figure 4 to the ground truth. The segmentation C consists of eight segments. Hence, one segment in C may correspond to more than one segment in C'. The dice measure averaged over all segments in C' is 81%. The best dice measure is 100% and the worst dice measure is 37%.

[1] http://shape.cs.princeton.edu/benchmark/

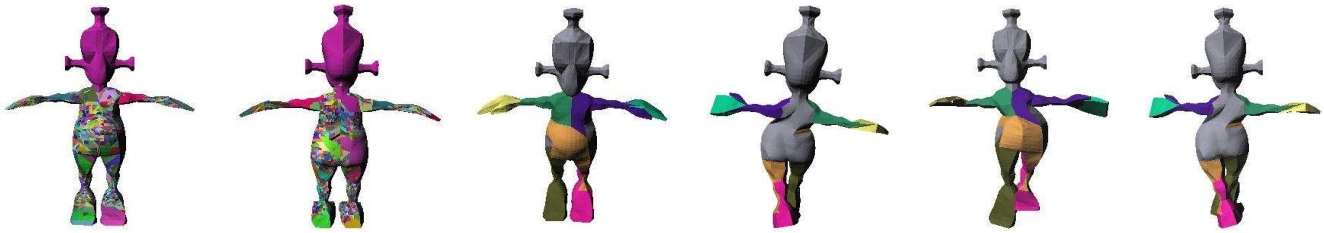

Fig. 4 *Alien model. First two columns show the segmentation before the merge step. Remaining columns show the final segmentation based on four input poses.*

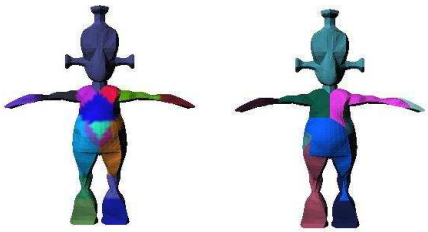

Fig. 5 *Ground truth experiment. Left shows the ground truth. Right shows result after interactively choosing 17 segments.*

Second, we interactively increase the number of segments in our segmentation to 17 using the first type of interaction outlined in Section 4. We then perform the same experiment as before. The segments are shown on the right of Figure 5. The dice measure averaged over all segments in C' is 81%. The best dice measure is 100% and the worst dice measure is 38%. This shows that a good segmentation is learned based on only four input poses.

The second experiment shows the computed near-rigid segments of a set of armadillo models. The models are chosen from the AIM@SHAPE repository [2]. The models contain 331904 triangles. The segmented input poses to our algorithm are shown in Figure 6. Note how the upper arm, lower arm, hands, and on the armadillo's left, the fingers, form near-rigid components.

Furthermore, we conducted experiments on a number of datasets created and used by Sumner et al. [20]. The datasets are obtained using deformation transfer. The results are shown in Figure 7. The horse model contains 16843 triangles, the elephant model contains 84638 triangles, the cat model contains 14410 triangles, and the flamingo model contains 52895 triangles. We can see that the segmentation captures the skeletal structure of the animals well in all of the examples. Note that parts of the mesh which deform smoothly such as the tail or the spine of the cat are correctly recognized as one segment. Although these parts of the mesh are partitioned into many small segments during the clustering step discussed in Section 2, they are correctly merged during the merge step discussed in Section 3.

Finally, we show some results of interactively changing the number of segments using both approaches presented in Section 4. First, we interactively change the number of segments using the first approach presented in Section 4. The result for the horse model is shown in Figure 8. The base segmentation shown on the left contains 15 segments. The figure shows segmentations containing 1193 and 3880 segments. Note that these segmentations are instances of a segmentation hierarchy and contain many small segments.

Second, we interactively update t_2 to increase the number of segments for the horse model using the second approach presented in Section 4. The result is shown in Figure 9. The base segmentation shown on the left contains 15 segments. The figure shows segmentations containing 22, 23, and 33 segments. Note that all of the segmentations show geometrically meaningful near-rigid components. For instance, when increasing the number of segments from 22 to 23, a segment corresponding to a bone of the horse's left back leg occurs. Furthermore, note that no small segments occur.

The running time of the algorithm is summarized in Table 1.

5.2 Comparison

We compare the quality of the segmentation found by our algorithm to the quality of the segmentation found by previous algorithms. We use the horse model for this comparison.

The methods by Katz et al. [8] and Katz and Tal [9] consider shape properties such as concavities instead of considering non-rigid deformation. These approaches therefore do not segment the tail of the horse. Therefore, these approaches are not suitable to decompose a shape into non-rigid components.

Figure 10 shows a comparison between our result and the segmentation result by Lee et al. [11]. Four similar poses are used in both experiments to find the segmentation. The poses are shown in the top row of Figure 10. The bottom of Figure 10 shows the results. The left side of Figure 10 shows the result by Lee et al., the middle of Figure 10 shows our result using the automatically computed parameters t_1 and t_2, and the right side of Figure 10 shows our result after interactively increasing the number of segments using the second approach discussed in Section 4. Unlike the approach by Lee et al., our approach does not place the head and the body of the horse in different segments. This is even true as the number of segments is increased interactively. As the segmentation is based on four input poses where the body

[2] http://shapes.aimatshape.net/releases.php

Fig. 6 *Armadillo model.*

	Number of triangles	Number of input poses	Time without merge	Total time	Interaction 1	Interaction 2
Armadillo	331904	5	31 s	349 s	13 s	320 s
Elephant	84638	10	17 s	46 s	3 s	30 s
Flamingo	52895	7	5 s	16 s	1 s	11 s
Horse	16843	10	3 s	4 s	2 ms	1 s
Cat	14410	10	3 s	4 s	$<$ 1 ms	$<$ 1 ms
Alien	13664	4	739 ms	2 s	$<$ 1 ms	$<$ 1 ms

Table 1 *Running time of the algorithm. Interaction 1 adjusts the number of segments by cutting edges. Interaction 2 adjusts t_2 and repeats the merge step.*

and the head move almost rigidly with respect to each other, it makes sense to treat the body and the head as one segment.

Figure 11 shows a comparison between our result and the segmentation result by James and Twigg [7]. The same poses are used in both experiments to find the segmentation. The top row shows some of the used poses. Note that in the result obtained by James and Twigg, segments can be disconnected. Furthermore, the triangles shown in black do not belong to any segment. Therefore, the partition is not suitable for many applications such as motion transfer. Our result yields a segmentation that partitions the model into connected segments where each triangle belongs to exactly one segment. We can see that, unlike the result obtained by James and Twigg, our result does not suffer from over segmenting the model.

Figure 12 shows a comparison between our result and the segmentation result by Lee et al. [12]. The same poses are used in both experiments to find the segmentation. The top row of Figure 11 shows some of the used poses. Note that although the segmentation results are different, both results capture the overall skeletal structure of the horse well. Our algorithm results in fewer segments than the algorithm by Lee et al. Namely, the body and the head are in the same segment. The reason is that in the given sequence, the body and the head move mainly rigidly with respect to each other.

The main advantage of our algorithm compared to Lee et al.'s work is its computational efficiency. To compute the segmentation of the horse based on 48 key frames, our algorithm takes 63 seconds. Hence, our algorithm is about 15 times faster than the algorithm by Lee et al. [12] for this example. Furthermore, we demonstrate the practicability of our algorithm by segmenting the large-scale armadillo model. Lee et al. only present experiments for models with up to 20000 triangles.

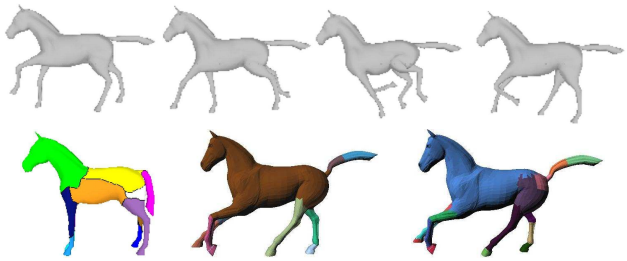

Fig. 10 *Top: Poses used to find the segmentation. Bottom: Comparison between our algorithm (middle and right) and the algorithm by Lee et al. [11] (left).*

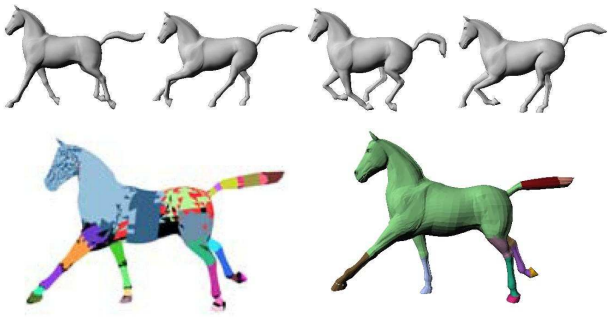

Fig. 11 *Comparison between our algorithm (right) and the algorithm by James and Twigg [7] (left).*

6 Conclusion

We presented an approach to efficiently solve the problem of segmenting a deforming triangular mesh into near-rigid components based on k given poses of the same non-rigid object. The efficiency of the approach was demonstrated in

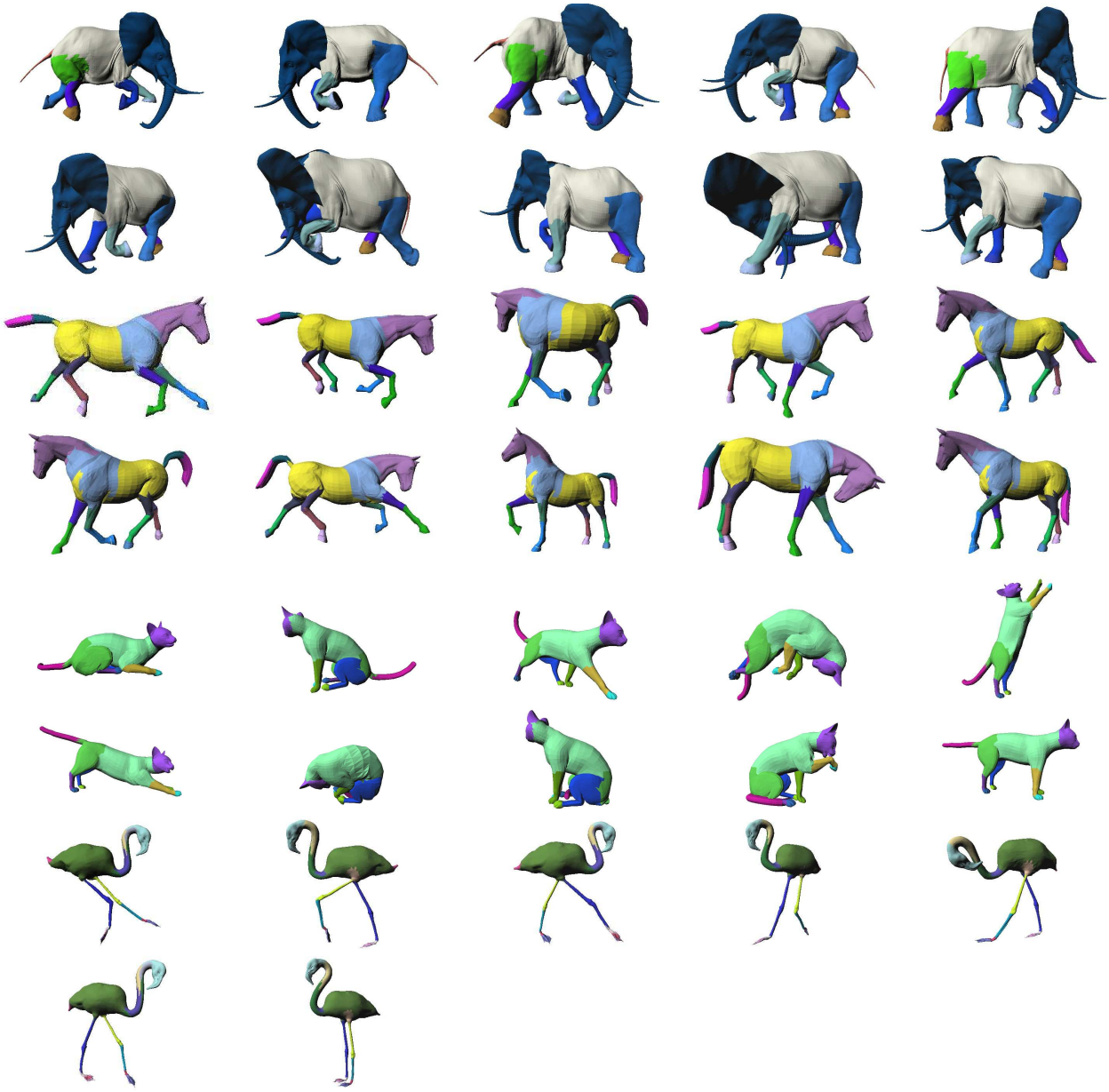

Fig. 7 *Segmentations of the elephant model, the horse model, the cat model, and the flamingo model.*

Fig. 12 *Comparison between our algorithm (right) and the algorithm by Lee et al. [12] (left).*

We assume that the point-to-point correspondences of the input meshes are known. An interesting direction for future work is to automatically compute the point-to-point correspondences prior to segmenting the meshes. This is a challenging task that can be approached by either using statistical shape models [4], by repeatedly computing the point-to-point correspondences for pairs of shapes [3], [5], [6], [22], or by registering one template mesh to all poses [1].

experimental results. The approach does not require any user-specified parameters.

Fig. 8 *Interactively updating the number of segments of the horse model.*

Fig. 9 *Interactively updating t_2 to increase the number of segments of the horse model.*

Acknowledgments

We thank Doug James and Dong-Yee Lee for their kind permission to use their images. This work was partially supported by OGS.

References

1. Dragomic Anguelov, Daphne Koller, Hoi-Cheung Pang, Praveen Srinivasan and Sebastian Thrun. Recovering Articulated Object Models from 3D Range Data. In *Uncertainty in Artificial Intelligence Conference*, 2004.
2. Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Transactions on Graphics*, 26(3), 2007. Proceedings of SIGGRAPH.
3. Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Calculus of non-rigid surfaces for geometry and texture manipulation. *IEEE Transactions of Visualization and Computer Graphics*, 13(5):902–913, 2007.
4. Rhodri H. Davies, Carole J. Twining, Tim F. Cootes, John C. Waterton, and Chris J. Taylor. 3d statistical shape models using direct optimization of description length. *Lecture Notes in Computer Science*, 2352:3–20, 2002.
5. Qixing Huang, Bart Adams, Martin Wicke, and Leonidas J. Guibas. Non-rigid registration under isometric deformations. *Computer Graphics Forum (Special Issue of Symposium on Geometry Processing 2008)*, 27(5), 2008.
6. Varun Jain, Hao Zhang, and Oliver van Kaick. Non-rigid spectral correspondence of triangle meshes. *International Journal on Shape Modeling*, Special Issue of SMI 2006, to appear.
7. Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Transactions on Graphics*, 24(3):399–407, 2005. Proceedings of SIGGRAPH.
8. Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8–10):865–875, 2005.
9. Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics*, 22(3):954–96, 2003.
10. Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley, 2005.
11. Tong-Yee Lee, Ping-Hsien Lin, Shaur-Uei Yan, and Chun-Hao Lin. Mesh decomposition using motion information from animation sequences: Animating geometrical models. *Computer Animation and Virtual Worlds*, 16(3-4):519–529, 2005.
12. Tong-Yee Lee, Yu-Shuen Wang, and Tai-Guang Chen. Segmenting a deforming mesh into near-rigid components. *The Visual Computer*, 22(9):729–739, 2006.
13. John Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. 2000.
14. Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition of polygons. *Computational Geometry: Theory and Applications*, 35(1):100–123, 2006.
15. Rong Liu and Hao Zhang. Mesh segmentation via spectral embedding and contour analysis. *Computer Graphics Forum (Special Issue of Eurographics 2007)*, 26:385–394, 2007.
16. Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Simple and efficient compression of animation sequences. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217, 2005.
17. Ariel Shamir. A survey on mesh segmentation techniques. *Computer graphics forum*, to appear.
18. Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The princeton shape benchmark. In *Proceedings of Shape Modeling International*, 2004.
19. Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The boost graph library: user guide and reference manual*. Addison-Wesley Longman Publishing Co., Inc., 2002.
20. Robert W. Sumner and Jovan Popović. Deformation transfer for triangle meshes. *ACM Transactions on Graphics*, 23(3):399–405, 2004. Proceedings of SIGGRAPH.
21. Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi. Invariant high level reeb graphs of 3D polygonal meshes. In *3rd IEEE International Symposium on 3D Data Processing Visualization Transmission*, 2006.
22. Hao Zhang, Alla Sheffer, Daniel Cohen-Or, Qingnan Zhou, Oliver van Kaick, and Andrea Tagliasacchi. Deformation-driven shape correspondence. *Computer Graphics Forum (Special Issue of Symposium on Geometry Processing 2008)*, 27(5), 2008.