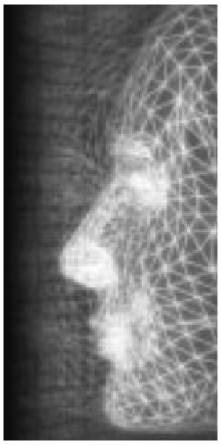
COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2010; **21**: 355–364
Published online 28 May 2010 in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/cav.351

Repairing topological inconsistency of mesh sequences



By Wei Feng, Hongxin Zhang, Jin Huang, Caoyu Wang and Hujun Bao*

We propose a novel approach for repairing topological inconsistency of mesh sequences with a few user interactions. The main idea of our approach is to leverage curve skeletons to detect the inconsistency of a mesh sequence. The skeleton of one mesh in the sequence is edited by user, which produces a prototype skeleton. We propagate this prototype using graph matching in-between frames. By using temporal coherence cues, the matching procedure can be dramatically accelerated. Finally, the mesh sequence is repaired according to the inconsistencies by comparing matched skeletons and original skeletons. As demonstrated in the results, our approach avoids manually editing in all meshes, and is able to output a mesh sequence with consistent topology. Copyright © 2010 John Wiley & Sons, Ltd.

KEY WORDS: topology repairing; mesh sequence; curve skeleton

Introduction

Thanks to recent fast scanning and vision techniques, sequence of 3D meshes can be produced more easily than before. However, topological problems, like unwanted small handles (i.e., rings and tunnels), frequently appear due to precision error or methodology bias. Besides that, topological inconsistency, i.e., meshes in a sequence with different genus, can cause applications to fail such as cross-parameterization and morphing operations. Therefore, repairing such problems for mesh sequences is vital in the pipeline of time-varying geometry processing. Unfortunately, to our knowledge, there is no practical approach which can help users efficiently detecting inconsistency of a given mesh sequence and then repairing them with few interactions.

Curve skeletons[1] are widely used to characterize topological structures as well as geometrical features of 3D objects. Using curve skeletons, the topology of corresponding meshes can be detected. In this paper, we detect the inconsistency of skeletons between different mesh frames by matching a prototype to all original skeletons extracted from meshes in a sequence. Using these inconsistencies, we overcome the inconsistency problems of mesh sequences.

The main contributions of our work are

- a novel approach is proposed to efficiently detect and correct topological inconsistency of a mesh sequence via skeleton matching;
- a method to increase matching efficiency by using temporal coherence.

Related Work

With the development of data acquisition techniques[2–5], time-varying data can be acquired more easily than before. As it is naturally more complex than reconstruction of single frames, sequence reconstruction encounters more problems in topology consistency. Our topology repairing method can serve as a post-processing step for them.

Repairing topology for static mesh is an active research topic in surface reconstruction recently. Wood *et al*.[6] uses Reeb Graphs to detect and remove unwanted topological handles of one given mesh. Meanwhile, Zhou and colleague[7,8] utilize adaptive volumetric presentation with its introduced skeletons to ensure the robustness of topology repairing. A similar work is presented in Reference[9]. All those approaches work well in single mesh, but they may require a great amount of human interactions when applying them to a long mesh sequence.

*Correspondence to: J. Huang, State Key Laboratory of CAD&CG, Zhejiang University, P.R. China.
E-mail: hj@cad.zju.edu.cn

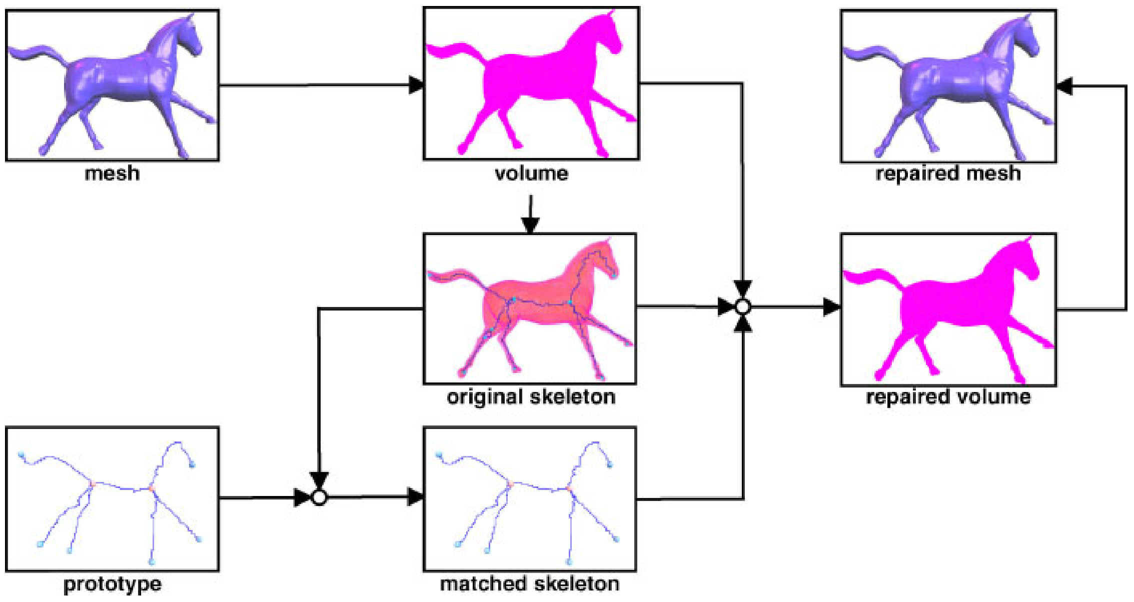


Figure 1. Overview of the topology repairing process for one frame.

In order to robustly repair the topology of meshes and ensure the consistency of a mesh sequence, we adopt volume-based method because they are known to be robust compared to mesh-based method [7]. Since we use curve skeletons to detect the topological inconsistency of meshes, we need to extract curve skeletons from volumes. This is still challenging [1], despite recent advances on curve skeleton extraction on meshes[10–12]. The extracted curve skeletons are noisy which challenges the robustness and efficiency of our matching step.

To match the skeletons so as to detect the topological inconsistency, we use graph matching technique because curve skeletons can be described as graph structures. Basically, graph matching is a pattern recognition technique that tries to find a mapping between nodes and edges of one graph to another. Graph matching obtains growing attention recently because of the increasing computation power of computers, which is capable to deal with large scale and complex graphs. However, as pointed out in the survey of Conte *et al.*[13], one of the key problems of graph matching is computational efficiency, thus, researches of graph matching algorithms try to balance the accuracy and the efficiency. In order to achieve fast response for users, we choose a greedy instance of graph matching algorithm for its efficiency. We also enhance the efficiency of graph matching in our approach.

Method overview

This work is inspired by the work on robust topology repairing a single mesh via sketching[8]. However, it is infeasible to apply their techniques to a mesh sequence since users have to edit in every frame.

Our main idea is to match all original skeletons from a user specified skeleton. Then, topological inconsistencies are detected by comparing the matched skeletons with original skeletons. These topology inconsistencies are then used to repair the mesh sequence. This user specified skeleton is edited from an arbitrarily selected frame with simple interactions and is called *prototype*. The matching propagates from the selected frame through all other frames. If one of the matching fails, user can specify the prototype at that frame and the matching procedure continues. By this matching, all the resulting skeletons have the same topology, so do the repaired meshes.

As illustrated in Figure 1, there are six steps in our approach:

1. Each mesh of input sequence is converted to a volume by using PolyMender[14].
2. Curve skeletons are extracted from the volumes.
3. Users edit one of the extracted skeletons to define a prototype.
4. We propagate the prototype to all skeletons by graph matching.

5. Volumes are repaired according to topological inconsistency by comparing original skeletons and matched ones.
6. A sequence of meshes with same topology are extracted from the repaired volume sequence.

In our approach, curve skeletons are extracted using a combination of previous work and the topology repairing is partially similar to Reference[8]. As detailed in the next section, the focus of our approach is to match curve skeletons from a prototype to original skeletons. The matched skeletons are temporal coherent with the prototype and more importantly have the same topology. We call the prototype and matched skeletons *consistent curve skeletons*.

Consistent Curve Skeleton

According to our requirements, the matched skeletons must have the same topological structure as the prototype, so that the repaired meshes will also maintain the same topology. However, there may exist multiple matches between the prototype and an original skeleton, and some of the matches may result in unwanted shape. Therefore, we further introduce the temporal coherence to constrain the matching, which intuitively implies that the curve skeletons smoothly change from frame to frame.

Formally speaking, let $\mathcal{C}^i$ be the curve skeleton of mesh frame i. It is a graph structure $\mathcal{C}^i = (\mathcal{N}^i, \mathcal{E}^i)$, with $\mathcal{N}^i$ and $\mathcal{E}^i$ denoting its node and edge set. Skeleton $\mathcal{C}^i$ is *consistent* to skeleton $\mathcal{C}^j$ if there exists a mapping of nodes and edges between the two skeletons satisfying

1. topological constraints:

$$\begin{aligned} e^i(n_1^i, n_2^i) &\overset{h}{\rightleftharpoons} e^j(n_1^j, n_2^j) \\ n_k^i &\overset{h}{\rightleftharpoons} n_k^j \qquad k = 1, 2 \end{aligned} \tag{1}$$

   where $e^i \in \mathcal{E}^i, e^j \in \mathcal{E}^j, n_k^i \in \mathcal{N}^i, n_k^j \in \mathcal{N}^j$.
2. geometrical constraints:

$$\begin{aligned} \arg\min_h \ & \textstyle\sum_{n\in\mathcal{N}^i} \|p(n) - p(h(n))\|_2 \\ & + \textstyle\sum_{e\in\mathcal{E}^i} D_c(e, h(e)) \end{aligned} \tag{2}$$

   where $h(n) \in \mathcal{N}^j, h(e) \in \mathcal{E}^j$, $p(n)$ is the position for node n, $D_c(e_1, e_2)$ is the distance between edges e_1 and e_2.

The computational complexity of the matching depends on the number of nodes and edges in the skeletons. Therefore, it is desirable to reduce the number of nodes and edges during matching for efficiency consideration. As a step of pre-processing, we divide the nodes of curve skeleton into three categories: *end*, *link*, *joint*, if their valence is less than two, two, or more than two, respectively. We collapse all links and their incident edges into *curves* which are edge paths connecting ends and joints. Note that, the edges of Equation (2) are actually curves. In the rest of this paper, we call all elements of $\mathcal{E}$ curves without any confusion.

To compute Equations (2), we uniformly sample the curves and sum up the distance of corresponding sample points as $D_c(e_1, e_2)$. Thus, the distance of curves include all distance of nodes and Equation (2) can be reduced to

$$\arg\min_h \sum_{e\in\mathcal{E}^i} D_c(e, h(e)) \tag{3}$$

It is worthy of noticing that we do not specify a skeleton and calculate the deformation over time like what is done to traditional inverse kinetics skeletons because the deformation of curve skeletons is hard to define. Moreover, by matching we do not impose any restriction on the deformable model of the mesh sequence. As illustrated in the Octopus example (Figure 8), the tentacles are squashed and stretched arbitrarily.

Skeleton Matching

The skeleton matching begins from the user specified frame and prototype editing of this frame. This prototype is matched with skeleton of neighboring frame. When this matching succeeds, the matched skeleton becomes a new prototype and is used to match further neighbors. These matching procedures are performed until all curve skeletons are matched. If one matching fails at a particular frame, users could specify another prototype at that frame and the matching continues.

At each matching step, we find a mapping h from the prototype $\mathcal{C}^p = (\mathcal{N}^p, \mathcal{E}^p)$ to an original skeleton $\mathcal{C}^o = (\mathcal{N}^o, \mathcal{E}^o)$ according to Equations (1) and (3). This mapping h consists of pairs of nodes and curves in $\mathcal{C}^p$ and $\mathcal{C}^o$, respectively, which are *corresponded*. After successful matching, we prune nodes and curves of $\mathcal{C}^o$ which are not in h and obtain a new prototype that is consistent to $\mathcal{C}^p$.

Our matching procedure is an adaption of the standard graph matching algorithm. We use an efficient algorithm described in Reference[15] which uses classic branch and

```
1: find guesses of initial $S$
2: for all guesses of initial $S$ do
3:     retValue = GraphMatchingRecursion($S$)
4:     if retValue == success then
5:         prune extra node and curve from $C^s$
6:         return success
7:     end if
8: end for
9: return failure
```

Figure 2. Skeleton Matching.

```
1: if $S$ is complete then
2:     return SUCCEED
3: end if
4: Pick one unmatched node $n_c^p \in \mathcal{N}^p$
5: Find candidates $n_{ci}^o \in \mathcal{N}^o$,
   $i = 1, \ldots, m; m = \#candidates$
6: Sort all candidate pairs $C = \{(n_c^p, n_{ci}^o)\}$
7: for all $C_i = (n_c^p, n_{ci}^o) \in C$ do
8:     if $C_i$ satisfy topology constrain then
9:         $S' \leftarrow S + C_i$
10:        GraphMatchingRecursion($S'$)
11:    end if
12: end for
```

Figure 3. GraphMatchingRecursion.

bound scheme. This scheme iteratively adds or removes pair of nodes from a set *S* and the full mapping set of *S* gives *h*. The pseudo-code of this algorithm is listed in Figures 2 and 3.

Note that *S* only consists of mapped nodes. To ensure the mapping of curves, we check the topology constrain for each candidate as in Figure 3 line 8. This checking ensures that every curve in prototype $\mathcal{C}^p$ maps to a unique curve in $\mathcal{C}^o$ not running into other curves. With unique node mapping and this topology checking, we ensure that Equation (1) is satisfied.

To minimize the curve distance as in Equation (3), we calculate the distance for each incident curve of the processing node if the other end of the curve is in *S* (i.e., matched). Note that the curve in $\mathcal{C}^o$ is actually curve path and we select the smallest distance if there are multiple of them. We sort all candidates according to the distance and select the one with smallest distance as a first match. If no matched neighbor is available, node distance is used.

The matching starts with an initial *S*. To create *S*, we find each node of the prototype a nearest one in $\mathcal{C}^o$. Then we calculate the curve distances from incident curves using Equation (3) (averaged distance for joint). This set is sorted according to these distances. Then we choose a single pair starting from the lowest distance as initial *S*. The matching fails after all pairs are tried.

One could use more pairs as initial *S* and the matching time would decrease if these initial guesses are correct. However when guesses are bad, matching times would increase. In our experiment, we found that our current strategy works better on average. The initial guess could not be good enough because these selections use node distance rather than Equation (3). We thus relax the nodes after the successful matching in order to reduce the energy of Equation (3) (see subsection 3).

Improving Matching Efficiency

The graph matching problem is known to be NP-hard. Although we use the efficient strategy of greedy searching[15], the performance is still discouraging.

To accelerate the matching procedure, one possible way is to match the nodes of same type. This reduces the candidate numbers so do the searching space. But in our context, matching end does not hold. Please see the left example of Figure 4. Type matching for joint nodes could reduce a few candidates, but the improvement is limited. Using a valence criterion, that is, the valence of candidates from original skeleton should be no less than the valence of corresponding unmatched node in prototype, does not work, too. That is because our graph matching is inexact that node merging may be required during the matching process, as illustrated in the right-hand side of Figure 4.

We utilize temporal coherence to reduce the matching time. We use a threshold ϵ_t that indicates how much a node would maximally move between neighbor frame and we find two ways to use this temporal coherence. Firstly, we select all nodes whose distance to the cur-

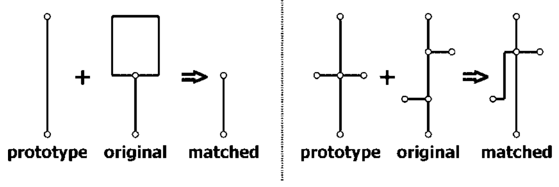


Figure 4. Two matching examples. The left shows that an end could match to a joint. The right shows that there might need a node merging to complete the matching.

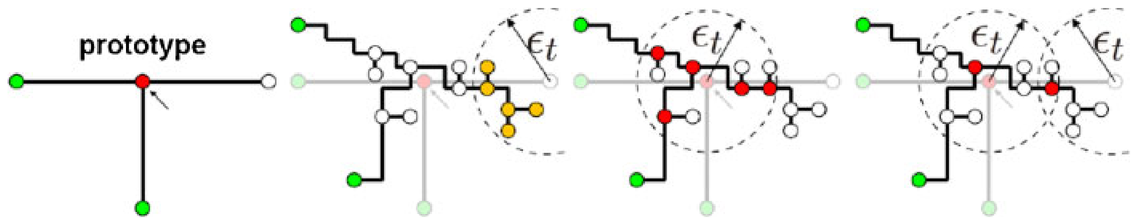


Figure 5. Using TC&TC. In first figure, the processing node is in red and two green nodes are matched nodes. The second figure shows all but processing nodes' potential nodes (both green and yellow nodes). The yellow nodes are potential nodes inside the ϵ_t range of white unmatched node in prototype and the green nodes are matched nodes. The third figure shows all possible candidates in red. The last figure illustrates remaining candidates after applying TC&TC.

rent node of the prototype are no larger than ϵ_t. This is straightforward that we do not want to search too far away. Secondly, by augmenting this temporal coherence criterion with type criterion of joint nodes (TC&TC for short), we further eliminate impropriate candidates as soon as possible, so that we search a greatly reduced solution space.

The TC&TC uses the criterion that a joint of $\mathcal{C}^p$ should be mapped only to a joint of $\mathcal{C}^o$. We perform TC&TC as follows: for each node n^p in prototype $\mathcal{C}^p$, if it is in S, we mark the corresponding node in $\mathcal{C}^o$ as *potential*, or if not in S, all nodes in $\mathcal{C}^o$ within distance ϵ_t are marked as potential. Then for each joint n^p in prototype $\mathcal{C}^p$, we try to prune branches starting from a node n^o of current skeleton $\mathcal{C}^o$ that is not connected to any potential node. If this node is no longer joint, we can safely remove this candidate (Figure 5). By using TC&TC, we can immediately eliminate a large amount of candidates for a joint node of $\mathcal{C}^p$ and greatly reduce matching time (Figure 11).

The threshold ϵ_t is set to 10% of bounding box size in our experiments. If it is too small, matchings would fail for the corresponding nodes get too far away. To relieve this problem, we double ϵ_t when no candidate can be found, until we find at least one candidate or ϵ_t is bigger than bounding box size. Matching would still fail when ϵ_t is small and we find wrong candidates within its range while the right candidate is out of it. But this chance of failure is dramatically reduced. Setting bigger ϵ_t will get slightly better chance of success but result in increased matching time. In extreme case, when ϵ_t is set to bounding box size, the matching time will be 10 times larger. We leave to users to select a good ϵ_t. Through our experiments, when ϵ_t is within a good range (less than 30% bounding box size), matching times increase not much, so that a large bound of good ϵ_t can be selected.

Another way to reduce the matching time is to carefully select the next unmatched node of prototype. In priority we select end nodes with matched neighbor or

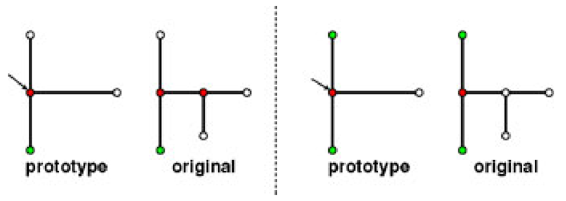


Figure 6. Picking a prototype node affects how many nodes can be rejected by topology constraint. Matched nodes are in green and red ones are the processing nodes. Second and forth columns are original skeletons with red valid candidates satisfying topology constraint.

joint nodes with no less than two matched neighbor because more candidates could be rejected by our topology constrain (1). One example is shown in Figure 6.

Node Relaxation

Our matching always satisfies Equation (1) and tries minimizing Equation (3). However, due to the greedy nature of our approach, our skeleton matching may not be the global minimal of Equation (3). On the other hand, the initial guess S is only computed via node distance rather than Equation (3) which can be further minimized. We use a relaxation process after matching succeeds by iterating each pair in S to see if we can find a new pair with less curve distance while satisfies Equation (1) to replace the original pair. This iteration stops until no pair can be replaced. This iteration is also greedy but guarantees convergence. One example is shown in Figure 7.

Implementation Details

Extracting Curve Skeletons We utilize curve skeletons to robustly repair the topology of the meshes just as in Reference[8]. So, we could not extract curve skele-

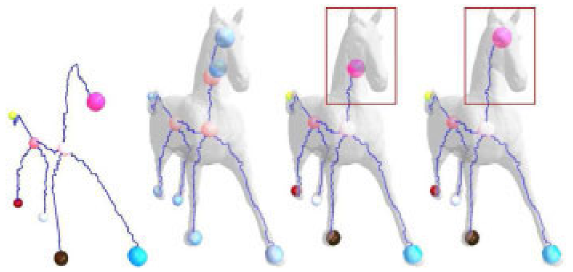


Figure 7. Node Relaxation. From left to right, the figures are prototype, the original skeleton, matching result before node relaxation, and matching result after node relaxation.

tons directly from meshes as in Reference[11] but have to convert meshes to volumes and extract curve skeletons directly from volumes. However, it is still a challenging work to extract curve skeletons from volumes. We are not trying to solve this problem in this paper but use a combination of previous works[7,16–18] to achieve plausible solutions. In our implementation, we use a two phases thinning algorithm. At the first phase, we thin a volume to its medial axis, and at the second phase, the medial axis is further thinned via a distance guided method[1].

Prototype Editing User specify prototype by editing one original curve skeleton. We do not use the sketching interface as in Reference[8] because users tend to sketch arbitrarily so that the node and curves may not be in the location where we could better use the temporal coherence. Initially users select one frame and edit corresponding curve skeleton. Another editing is needed only when matching fails.

We provide three basic operations for user to edit the curve skeletons: node deleting, node merging, and curve breaking. Node deleting is used to prune unwanted branches. Curve breaking can break handles and using node merging, we can force several close nodes to be mapped to a single node.

Volume Repairing and Surface Extraction We compare the matched curve skeletons with the original ones to detect the topological inconsistencies which are used for volume repairing. Repaired mesh sequence is then extracted from the repaired volumes. Both repairing and surface extraction methods are introduced in Reference[8].

Results

We have performed our topology repairing methods on two synthetic mesh sequences and two real-world ones. The horse sequence is selected from an original horse sequence[19] and then modified in a 3D modeling software called Blender. We clean the extracted skeletons to give better demonstration. The Octopus sequence is generated by deforming a static octopus model. The Benjamin Baton and the Antoine mesh sequences are downloaded from INRIA 4D datasets (http://4drepository.inrialpes.fr/index.php). All meshes are converted to an Octree representation using PolyMender. During

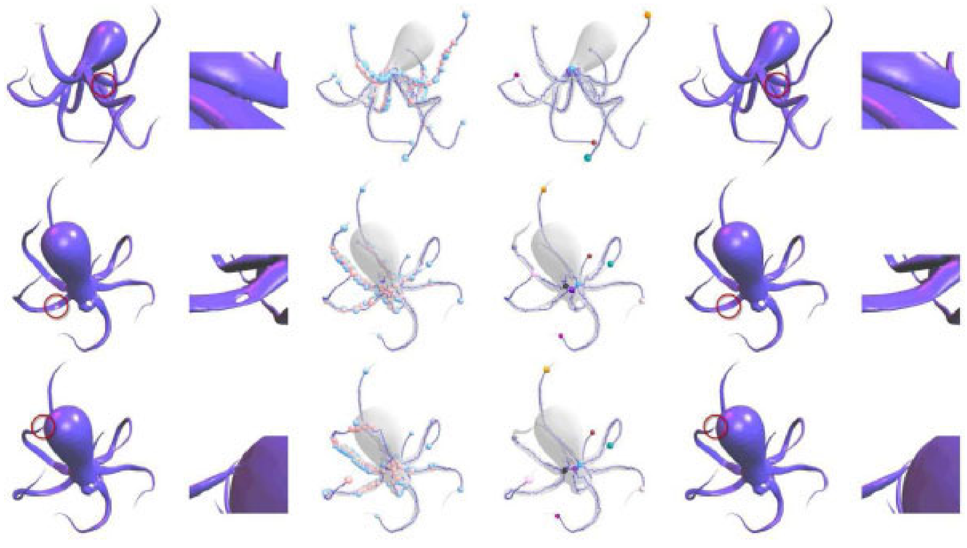


Figure 8. Frames 0 and 21, 23 of Octopus sequence. First column is the original meshes and second column is the zoomed view of marked areas. Third column is original curve skeletons. Forth column is matched curve skeletons and colors show the correspondences. Fifth column is meshes after our topology repairing and sixth column is the zoomed view.

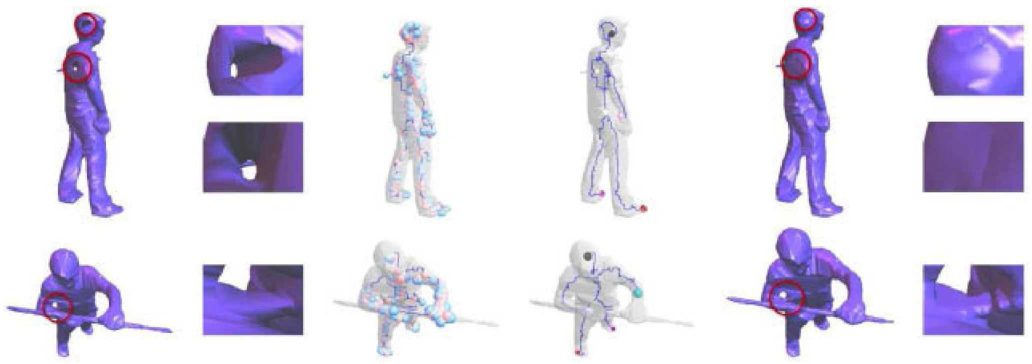


Figure 9. Frames 4 and 12 of Benjamin Baton sequence. First column is the original meshes and second column is the zoomed view of marked areas. Third column is original curve skeletons. Forth column is matched curve skeletons and colors show the correspondences. Fifth column is meshes after our topology repairing and sixth column is the zoomed view.

conversions, handles may be created or removed due to the lack of sampling rate in some areas or when some parts are too close to each other. The topology problems in the Octopus sequence are created this way. Some parts of input meshes may be discarded in the conversion, too, like the shortened bars in Benjamin Baton results. We repair the topological inconsistencies from both original meshes and these conversions. The selected results of the octopus and the Benjamin Baton are shown in Figures 8 and 9, respectively.

The prototypes of former results are actually trees, so they might be viewed as tree–graph matching. However, like other graph matching algorithms, we can handle graph–graph matching which is shown in Antoine results (Figure 10).

In our experiments, users only edit one of the curve skeleton as initial prototype and the graph matching goes through all other frames successfully. The performances

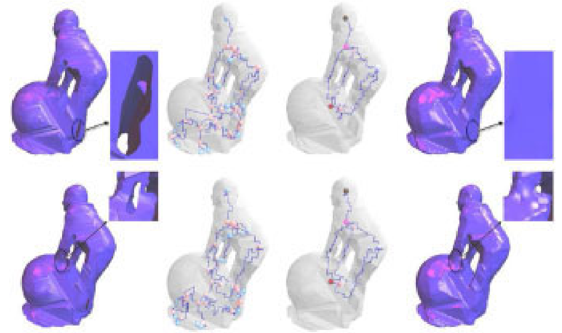


Figure 10. Antoine results of frames 1 and 2. The four columns are original meshes, original curve skeletons, matched curve skeletons and repaired meshes respectively.

of the matchings are shown in Figure 11. All timing are gathered on a PC with Intel Pentium D 2.8 GHz CPU and 3 GB memory. The matching time of each frame varies a lot, as it depends mostly on how many good guesses temporal coherence assumption can give and the quality of these guesses varies a lot. The maximal matching times show how slow it would be when bad guesses encounter and average times show how much time it would usually take to match a typical sequence like these in our experiments. We compare the results with or without TC&TC as we discussed in subsection 1 and matchings can be approximately 10 times faster with our TC&TC. Note in the Horse result, we do not have noise near the joints and TC&TC cannot eliminate any impropriate candidate, so the matching is slightly slower with TC&TC because we do extra checking. We use Octree of depth 8 for all our experiments and the depth only dramatically affect the time of extracting curve skeletons. This is our current bottleneck of topology repairing which may take up to 10 minutes for a single Octree of depth 8 and this time increase much when depth grows. The time in volume repairing and iso-surface extraction counts only a couple of seconds and is not increasing much when depth grows. See, e.g., Reference[8], for how these two procedures perform.

The repaired areas are smoothed so that we can clearly see the cut areas are split and filled areas are smooth to surrounding faces.

Discussion

In this paper we propose an approach to assist users in repairing topological inconsistency of mesh sequences

	$\#\mathcal{N}^p$	avg $\#\mathcal{N}^o$	without TC&TC		with TC&TC	
			max time	avg time	max time	avg time
Horse	8	10	2.137s	1.367s	2.638s	1.653s
Octopus	11	179	3592s	287.8s	596.6s	39.91s
Benjamin Baton	7	308	124.8s	27.41s	4.549s	1.127s
Antoine	6	54	289.9s	112.4s	7.538s	4.233s

Figure 11. Performance of graph matching. $\mathcal{N}^p$ is the node set of prototype, and $\mathcal{N}^o$ is the node set of an original skeleton.

with a few user interactions. Users only need to provide a prototype curve skeleton which describes the topology they want and the prototype is propagated through the sequence using graph matching. In seldom case, users may need to provide additional prototypes when the matching fails. However, our approach managed to match all curve skeletons in our experiments while users only give the initial prototypes.

In our applications, we would like to give fast responses, so we choose greedy strategies both in graph matching and node relaxation. At most of the time, the matching results can give desired results. But occasionally matching results may not be fully correct because our matching is suboptimal and these matching results may lead to undesired meshes. Fortunately, as we aim at assisting users as much as possible, users can easily give precise control over the repaired meshes by editing the curve skeletons as they do in prototype editing.

Our implementation is only tested in cases of one connecting component. However, it is quite straightforward to extend our approach to multiple components and apply our method in a component-wise way, if the prototype and original skeletons have the same number of components and the components do not intersect with each other. We currently do not consider the case when prototype and original skeletons have different component numbers.

In our current work, we have temporal coherence assumption but do not assume prior knowledge of deformations, so that we could apply our approach to wide sets of dynamic objects. However, the temporal coherence assumption could be weak when movements between consecutive frames are large, though we could still match many of them by consuming more running time. If users have other priors, e.g., the type of movement of the object, we could estimate the locations of nodes and curves of new prototype using existing prototypes. We leave this as future works when investigating specific type of objects.

The proposed approach produces consistent topology by reducing complexity of topology to the simplest one. We cannot add handles because it would require estimating curve positions when adding rings. Also our approach does not work when branches in the curve skeleton are missing. This may be caused by missing or merged geometry. This limitation can also possibly be solved by motion estimation which we will investigate in future.

ACKNOWLEDGEMENTS

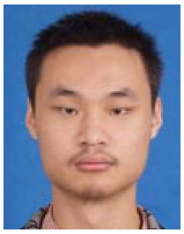
We gratefully thank Robert *et al.* for providing horse sequence and Inria Perception group for Benjamin Baton and Antoine sequences. We also thank Xavier Granier for his helpful proofreading. This work is supported by National Natural Science Foundation of China (no. 60703039, 60933007), 973 program of China (no. 2009CB320801), and the National High Technology Research and Development Program of China (no. 2009AA12Z229).

References

1. Cornea ND, Siver D and Min P. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics* 2007; **13**(3): 530–548.
2. Wand M, Jenke P, Huang Q, Bokeloh M, Guibas L, Schilling A. Reconstruction of deforming geometry from time-varying point clouds. *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, 2007; 49–58.
3. Pekelny Y, Gotsman C. Articulated object reconstruction and markerless motion capture from depth video. *Computer Graphics Forum* 2008; **27**(10): 399–408.
4. Sharf A, Alcantara DA, Lewiner T, *et al*. Space-time surface reconstruction using incompressible flow. *ACM Transactions on Graphics* 2008; **27**(5): 1–10.
5. Wand M, Adams B, Ovsjanikov M, *et al*. Efficient reconstruction of nonrigid shape and motion from real-time 3d scanner data. *ACM Transactions on Graphics* 2009; **28**(2): 1–15.
6. Wood Z, Hoppe H, Desbrun M, Schröder P. Removing excess topology from isosurfaces. *ACM Transactions on Graphics* 2004; **23**(2): 190–208.
7. Zhou Q-Y, Ju T, Hu S-M. Topology repair of solid models using skeletons. *IEEE Transactions on Visualization and Computer Graphics* 2007; **13**(4): 675–685.

8. Ju T, Zhou Q-Y, Hu S-M. Editing the topology of 3d models by sketching. *ACM Transaction on Graphics* 2007; **26**(3): 42.
9. Sharf A, Lewiner T, Shklarski G, Toledo S, Cohen-Or D. Interactive topology-aware surface reconstruction. *ACM Transaction on Graphics* 2007; **26**(3): 43.
10. Sharf A, Lewiner T, Shamir A, Kobbelt L. On-the-fly curve-skeleton computation for 3d shapes. *Computer Graphics Forum* 2007; **26**(3): 323–328.
11. Au OKC, Tai C-L, Chu H-K, Cohen-Or D, Lee T-Y. Skeleton extraction by mesh contraction. *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, 2008; 1–10.
12. Tagliasacchi A, Zhang H, Cohen-Or D. Curve skeleton extraction from incomplete point cloud. *ACM Transaction on Graphics* 2009; **28**(3): .
13. Conte D, Foggia P, Sansone C, Vento M. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence* 2004; **18**(3): 265–298.
14. Ju T. Robust repair of polygonal models. *ACM Transaction on Graphics* 2004; **23**(3): 888–895.
15. Cordella LP, Foggia P, Sansone C, Vento M. An efficient algorithm for the inexact matching of arg graphs using a contextual transformational model. *ICPR '96: Proceedings of the International Conference on Pattern Recognition*, 1996; 180.
16. Couprie M, Coeurjolly D, Zrour R. Discrete bisector function and euclidean skeleton in 2d and 3d. *Image and Vision Computing* 2007; **25**(10): 1543–1556.
17. Remy E, Thiel E. Exact medial axis with euclidean distance. *Image and Vision Computing* 2005; **23**(2): 167–175.
18. Meijster A, Roerdink JBTM, Hesselink WH. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its Applications to Image and Signal Processing*, Springer, New York, US, 2000; 331–340.
19. Sumner RW, Popović J. Deformation transfer for triangle meshes. *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, 2004; 399–405.

Authors' biographies:



Wei Feng is a PhD student in the State Key Laboratory of CAD&CG at Zhejiang University, PR China. He received his BS degree from the University of Electronic Science and Technology of China in 2005. His research interests include geometry processing and geometry modeling.

Hongxin Zhang is an associate professor of the State Key Laboratory of CAD&CG at Zhejiang University, PR China. He received BS and PhD degrees in applied mathematics from Zhejiang University. His research interests include geometric modeling, texture synthesis, and machine learning.

Jin Huang is an associate professor in the State Key Laboratory of CAD&CG at Zhejiang University, PR China. He received his PhD degree in Computer Science from Zhejiang University in 2007 with Excellent Doctoral Dissertation Award of China Computer Federation (CCF). His research interests include geometric deformation and physical-based simulation. He has served as reviewer for ACM SIGGRAPH, IEEE CAD/Graphics, PSIVT, and JCST.

Caoyu Wang is an undergraduate student in the State Key laboratory of CAD&CG at Zhejiang University. He is now pursuing his Bachelor's Degree in the Department of Computer Science, Chu Kochen Honors College. His research interests include geometry processing and geometry modeling.

Hujun Bao received his Bachelor and PhD in Applied Mathematics from Zhejiang University in 1987 and 1993, respectively. His research interests include modeling and rendering techniques for large scale of virtual environments and their applications. He is currently the Director of State Key Laboratory of CAD&CG of Zhejiang University. He is also the principal investigator of the virtual reality project sponsored by Ministry of Science and Technology of China.